برمجة الجوئي باستخدام لغة جافا

GUI Programming Via Java

عن كتاب :

Java Software Solution: foundations of program design

ترجمة : عبد الفتاح عبد الرب المشرقي

مركز الحاسوب وتقنية المعلومات – جامعة إب – الجمهورية اليمنية

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الإهداء

إلى الأرواح الزكية من شهداء ثورات الربيع العربي (٢٠١١)
مدنيين وعسكريين في تونس ومصر واليمن وليبيا
والحبيبة .. سوريا ، الذين قضوا نحبهم وفي قلوبهم
الكبيرة تسكن أوطاننا وآلامنا وأفراحنا.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

المقدمة

أصبح مفهوم واجهات المستخدم الرسومية *Graphical User Interface*، أو ما يطلق عليها اختصاراً "جوئي" *GUI*، واحداً من المواضيع التي لاقت اهتماماً كبيراً من قبل مهندسي البرمجيات، وذلك كونها تمثل همزة الوصل بين المستخدم النهائي *end user* وخدمات التطبيقات الحاسوبية المختلفة. بل أن المستخدم ينظر للواجهة التي يتفاعل بها مع البرنامج على أنها البرنامج كله. ومن خلالها يبني تقييماً في ذهنه عن فاعلية البرمجية وقدراتها المختلفة، حيث أنه لا يهتم بالتفاصيل التي قد يهتم به المبرمجون والمختصون.

ومعظم أنظمة التشغيل والبرامج التطبيقية والبرامج المساعدة مزودة بواجهات مستخدم صديقة، بمعنى أنها تساعد مستخدميها على إتمام مهامهم الروتينية بكفاءة وسهولة. وهذا، بلا شك، يزيد من عدد المستخدمين للبرمجية، ويزيد من كفاءتها الإنتاجية والزمنية.

وبما أن جزءاً من المسئولية الأساسية في بناء البرمجيات تقع على عاتق لغات البرمجة وإمكاناتها، فقد زودت الكثير من لغات البرمجة الحديثة مقدرات التعامل مع المكونات الرسومية التي تستخدم في تصميم واجهات المستخدم. ومن هذه اللغات لغة جافا، التي تزودنا بالعديد من مكونات الجوئي التي تمتاز بالقوة وسهولة الاستخدام، مثل مربعات الإدخال والإخراج الحوارية، وأزرار الضغط، والحقول النصية، ومربعات الاختيار، وأزرار الراديو، ومربع الخيارات المنسدلة، والقوائم .. الخ. وبإمكاننا استخدام هذه المكونات وغيرها لجعل البرامج أكثر جاذبية وفعالية.

وهذا الكتاب هو ترجمة متواضعة للأقسام المتعلقة بالجوئي في نهاية كل فصل من كتاب "الحلول البرمجية باستخدام لغة جافا: أساسيات تصميم البرنامج" (*Java Software Solutions: foundations of program design*) للمؤلفين: *John Lewis* ، و *William Loftus*، الإصدار الخامس. وقد اخترت هذا الكتاب لما فيه من البساطة في طرح والمفاهيم، والاختصار الغير المخل، والتركيز على أهم المفاهيم المتعلقة بالجوئي. ولعله من الحري بي في هذا المقام أن أتوجه بالشكر للمؤلفين لما بذلاه من جهد واضح في إعداد هذا الكتاب بالصورة التي جعلتني وكثير من القراء والمبرمجين نلمس فيه استفادة كبيرة.

تحوي هذه الترجمة على المفاهيم الأساسية المتعلقة بالجوئي، والعديد من الأمثلة التطبيقية التي تغطي مفاهيم كل فصل. وهو مدعم بالرسوم التوضيحية وشاشات التنفيذ. وأرجو أن يجد فيه القارئ والمبرمج الفائدة المرجوة، وأرجو كذلك إرسال أي ملاحظات على عنوان البريد الإلكتروني الخاص بي. وأطلب من الجميع الدعاء.

والسلام عليكم ورحمة الله وبركاته ...

عبد الفتاح عبد الرب المشرقي
٨ / ٣ / ٢٠١٢م

الفصل الأول

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

١.١ الرسومات GRAPHICS

تلعب الرسومات دوراً مهماً في أنظمة الحاسوب. ومن خلال هذا الكتاب سنقوم باستكشاف مجالات الرسومات المختلفة وسنناقش كيفية عملها.

مفهوم أساسي Key Concept

يتم تمثيل البيانات الرسومية عن طريق تقسيمها إلى عدد من الأجزاء الصغيرة تسمى بكسلات pixels.

إن الصورة مثلها مثل كل المعلومات التي يتم تخزينها في الحاسوب، يجب إن تحول إلى الشكل الرقمي *digital* عن طريق تقسيم المعلومات إلى أجزاء وتمثيل هذه الأجزاء كأرقام. وفي حالة الصور، يتم تحويل الصورة إلى بكسلات *pixels* (*picture elements*). يمكن النظر للبكسل على أنه نطاق صغير جداً من الصورة. ويتم تخزين الصورة كاملة عن طريق تخزين لون كل بكسل على حدة. يمكن إعادة إنتاج الصورة الرقمية عند الحاجة عن طريق إعادة تجميع البكسلات الخاصة بها، وكلما زاد عدد البكسلات المستخدمة لتمثيل الصورة، زادت درجة واقعيتها عند إعادة إنتاجها. ويسمى عدد البكسلات المستخدمة لتمثيل الصورة بـ دقة الصورة *picture resolution*، ويسمى عدد البكسلات التي يمكن عرضها على الشاشة بـ دقة الشاشة *monitor resolution*.

يمكن تخزين الصورة ذات اللونين الأسود والأبيض عن طريق تمثيل كل بكسل باستخدام بت واحد (*single bit*). حيث أنه إذا كانت البت تساوي *0* يكون لون البكسل أبيضاً، أما إذا كانت البت تساوي *1* فإن لون البكسل يكون أسوداً. ويعرض الشكل ١.١ صورة أبيض وأسود تم تخزينها رقمياً وكذلك جزءاً مكبراً من تلك الصورة، والتي يتضح فيها البكسلات المفردة.

الشكل رقم ١.١ صورة رقمية مع جزء صغير تم تكبيره

أنظمة الإحداثيات Coordinate Systems

عند رسم أي بكسل في الصورة فإنه يتم ربطه ببكسل على الشاشة. ويعمل أي نظام حاسوبي ولغة برمجة على تعريف نظام إحداثيات يمكننا من خلاله التعامل مع بكسلات معينة.

يمتلك نظام الإحداثيات الكارتيزي التقليدي ثنائي البعد محورين يلتقيان في نقطة الأصل. يمكن إن تكون القيم على كلا المحورين سالبة أو موجبة. وتمتلك لغة جافا نظام إحداثيات بسيط نسبياً تكون فيه كل الإحداثيات المرئية موجبة. ويقارن الشكل ٢.١ بين نظام الإحداثيات التقليدي ونظام إحداثيات جافا.

مفهوم أساسي Key Concept

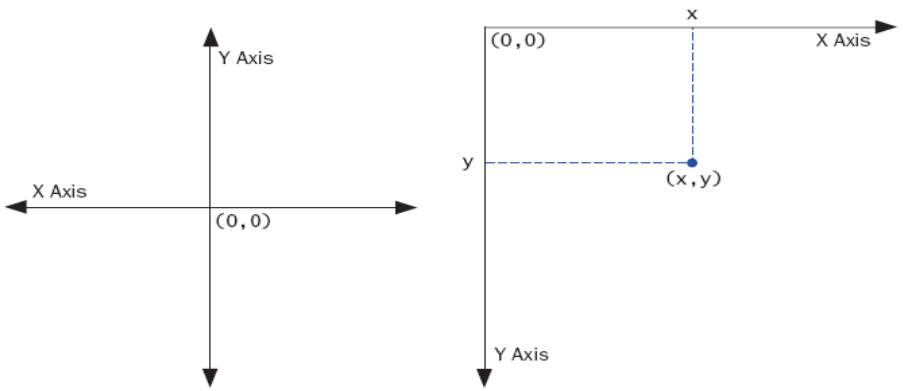
تقع نقطة الأصل في نظام إحداثيات جافا في الزاوية العليا الشمالية وتكون كل الإحداثيات المرئية موجبة.

يتم تمثيل أي نقطة في نظام إحداثيات جافا باستخدام زوج من القيم (x, y). وإحداثيات الزاوية العليا من منطقة رسم جافا هي $(0,0)$. يكبر الإحداثي السيني *x-axis* كلما اتجهنا جهة اليمين، ويكبر الإحداثي الصادي *y-axis* كلما تحركنا جهة الأسفل.

عند كتابة برامج جافا الرسومية فإن كل مكون رسومي في البرنامج يملك نظام الإحداثيات الخاص به، حيث نقطة الأصل هي(0,0) تقع في الزاوية العليا الشمالية. إن هذا المبدأ الثابت يجعل من السهل نسبياً إدارة العناصر الرسومية المختلفة.

تمثيل اللون Representing Color

يتم تقسيم الصورة الملونة إلى بكسلات، كما هو الحال في الصور الأسود والأبيض. بالرغم من ذلك، وبسبب أن لون كل بكسل يمكن أن يكون لوناً من عدة ألوان ممكنة، فإنه ليس كافياً أن يتم تمثيل أي بكسل فقط باستخدام بت واحدة. وهناك عدة طرق لتمثيل لون البكسل. دعنا نناقش باختصار إحدى التقنيات المشهورة.



الشكل رقم ٢.١ نظام الإحداثيات التقليدي ونظام إحداثيات جافا

مفهوم أساسي Key Concept

يتم تمثيل الألوان في لغة جافا باستخدام قيمة RGB - وفيها ثلاثة قيم لتمثيل الألوان الرئيسية الأحمر red، الأخضر green، والأزرق blue.

يتم تمثيل كل لون كخليط من ثلاثة ألوان رئيسية: الأحمر *red*، والأخضر *green*، والأزرق *blue*. وفي لغة جافا، كما في عدة لغات برمجية أخرى، يتم تحديد الألوان عن طريق ثلاثة ألوان يتم الإشارة إلى مجموعها بقيمة *RGB*، حيث *RGB* هي اختصار لـ *Red-Green-Blue*. ويمثل أي رقم نصيب كل لون من هذه الألوان الأساسية، ويتم استخدام واحد بايت *1 byte* (٨ بتات *8 bits*) لخزن أياً من الأرقام الثلاثة، وهذا يجعل هذه الأرقام تتراوح بين *0* و *255*. ويعمل مستوى أي لون أساسي على تحديد اللون الكلي. مثلاً، القيم العالية للونين الأحمر والأخضر يتم تركيبها مع مستوى منخفض من اللون الأزرق لإنتاج ظلال اللون الأصفر.

مفهوم أساسي Key Concept

يحتوي الكلاس Color على ألوان عديدة معرفة مسبقاً وتستخدم بشكل شائع، ويمكن استخدامها لتعريف ألوان أخرى متعددة.

في لغة جافا، يستخدم المبرمج الكلاس *Color*، والذي يعتبر جزءاً من الحزمة *java.awt*، لتعريف و إدارة الألوان. يمثل أي كائن من الكلاس *Color* لوناً مفرداً. ويحتوي هذا الكلاس على العديد من الكائنات للتزويد بمجموعة من الألوان المعرفة مسبقاً. وفي الشكل ٣.١ سرد للألوان المعرفة مسبقاً في الكلاس *Color*. ويحتوي هذا الكلاس أيضاً على دوال *methods* لتعريف وإدارة العديد من الألوان الأخرى.

اللون Color	الكائن Object	قيمة آر جي بي RGB Value
الأسود black	`Color.black`	0, 0, 0
الأزرق blue	`Color.blue`	0, 0, 255
الأزرق المائي cyan	`Color.cyan`	0, 255, 255
الرمادي gray	`Color.gray`	128, 128, 128
الرمادي الغامق dark gray	`Color.darkGray`	64, 64, 64
الرمادي الفاتح light gray	`Color.lightGray`	192, 192, 192
الأخضر green	`Color.green`	0, 255, 0
الأرجواني magenta	`Color.magenta`	255, 0, 255
البرتقالي orange	`Color.orange`	255, 200, 0
الوردي pink	`Color.pink`	255, 175, 175
الأحمر red	`Color.red`	255, 0, 0
الأبيض white	`Color.white`	255, 255, 255
الأصفر yellow	`Color.yellow`	255, 255, 0

الشكل رقم ٣.١ الألوان المعرفة مسبقاً في الكلاس Color

٢.١ الآبلتات APPLETS

مفهوم أساسي Key Concept

الآبلتات هي برامج جافا يتم عادة نقلها عبر شبكة وتنفيذها باستخدام مستعرض الوب.

هناك نوعان من برامج جافا: آبلتات الجافا *Java applets* وتطبيقات الجافا *Java applications* . آبلت الجافا هو عبارة عن برنامج جافا يُنوى تضمينه داخل ملف *HTML*، ونقله عبر شبكة، وتنفيذه باستخدام مستعرض الوب. أما تطبيق الجافا فهو برنامج قائم بذاته يمكن تنفيذه باستخدام مفسر لغة جافا.

يتمكن المستخدمون عن طريق الوب من إرسال واستقبال أنواع عديدة من الوسائط، مثل النصوص *text*، الرسومات *graphics*، والصوت *sound*، واستخدام واجهة التأشير والنقـــر *point-and-click interface* و التي تعتبر ملائمة بشكل كبير جداً وسهلة الاستخدام. كان آبلت الجافا هو أول نوع من البرامج التنفيذية التي يمكن الحصول عليها باستخدام مستعرض الوب. ويمكن اعتبار آبلتات جافا على أنها نوع آخر من الوسائط يمكن تبادلها عبر الوب.

وبالرغم من أن آبلتات الجافا معدة عموماً للنقل عبر شبكة، فليس بالضرورة أن يتم ذلك. حيث يمكن استعراضها محلياً باستخدام مستعرض الوب. ولهذا السبب، فإنه ليس من الضروري حتى تنفيذها من خلال مستعرض الوب مطلقاً. فهناك أداة ضمن حزمة التطوير البرمجية للغة جافا المقدمة من شركة Sun تسمى *appletviewer* يمكن استخدامها لتفسير وتنفيذ الآبلت. يمكن استخدام *appletviewer* لعرض الآبلتات في كتابنا هذا. ومع ذلك، فإنه من المعتاد أن يتم عمل آبلتات جافا ليتم عمل ارتباط لها *link* على صفحة وب والسماح بأن يتم الحصول عليها وتنفيذها من قبل مستخدمي الوب عبر العالم.

يتم ربط الجافا بايت كود *Java bytecode* (وليس كود جافا المصدري) بملف الـ *HTML* وإرساله عبر الوب. وهناك إصدار من مفسر جافا يأتي مضمناً في مستعرض الوب ويستخدم لتنفيذ الآبلت عند وصوله إلى وجهته. ويتم ترجمة آبلت الجافا إلى شكل بايت كود قبل أن نتمكن من استخدامه على الوب.

هناك بعض الفروق المهمة بين بنية آبلت الجافا وبنية تطبيق الجافا. فبسبب أن مستعرض الوب الذي ينفذ أي آبلت يكون في حالة تشغيل مسبقاً، فيمكن أن ننظر إلى الآبلت على أنه جزء من برنامج أكبر، وبالتالي فهو لا يملك دالة *main* يبدأ منها التنفيذ. هناك دالة في الآبلت تسمى *paint* يتم استدعائها تلقائياً بواسطة الآبلت. ولنأخذ بعين الاعتبار المثال ١.١ والذي تم فيه استخدام الدالة *paint* لرسم بعض الأشكال وكتابة تعليق لألبـــرت اينشتاين *Albert Einstein* على الشاشة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

تشير جملتا *import* في بداية البرنامج وبشكل صريح إلى الحزم *packages* المستخدمة في البرنامج. وفي هذا المثال، نحتاج الكلاس *JApplet*، والذي يعد جزءاً من الحزمة المسماة *javax.swing*، ونحتاج كذلك لمقدرات رسومية مختلفة معرفة في الحزمة *java.awt*.

يقوم أي كلاس يتم تعريف آبلت فيه بوراثة الكلاس *JApplet*، كما هو واضح في أول سطر في تعريف الكلاس. وفي هذه العملية يتم استخدام مفهوم من مفاهيم البرمجة كائنيه المنحنى *Object Oriented Programming* وهو الوراثة *Inheritance*، وسنناقشها بشكل أكبر في قادم الصفحات. أيضاً، يجب التصريح عن كلاسات الآبلت على أنها *public*.

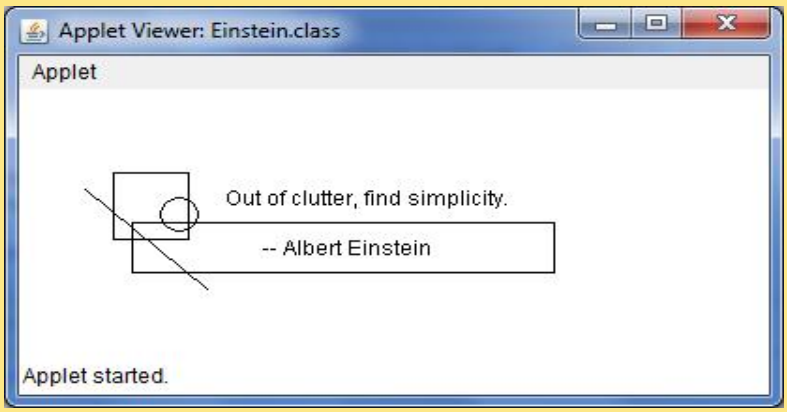
تعتبر الدالة *paint* واحدة من عدة دوال خاصة بالآبلت والتي تمتلك أهمية خاصة، حيث تستدعى تلقائياً عندما تكون هناك حاجة لرسم أي عنصر رسومي على الشاشة، مثل عملية بدء تشغيل الآبلت أو عند تحريك نافذة كانت تغطي على الآبلت.

لأحظ أن الدالة *paint* تستقبل كائن من الكلاس *Graphics* كوسيط لها. ويعمل هذا الكائن على تعريف سياق رسومي خاص *graphics context* يمكننا التفاعل معه. يعمل هذا السياق الرسومي الممرر إلى الدالة *paint* على تمثيل كامل نافذة الآبلت. إن أي سياق رسومي يملك النظام الإحداثي الخاص به. وفي الأمثلة اللاحقة، سيكون لدينا مكونات متعددة، لكل منها سياق رسومي خاص.

يمكننا كائن الكلاس *Graphics* من رسم عدة أشكال باستخدام دوال مثل *drawRect*، و *drawLine*، و *drawString*. وتعمل الوسائط الممررة إلى دوال الرسم هذه على تحديد إحداثيات وأحجام الأشكال التي يتم رسمها. وسنقوم باستكشاف هذه الدوال ودوال أخرى لرسم الأشكال في القسم التالي.

مثال رقم ١.١

```
//********************************************************************
//  Einstein.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال آبلت أساسي
//********************************************************************

import javax.swing.JApplet;
import java.awt.*;

public class Einstein extends JApplet
{
   //-----------------------------------------------------------------
   //  يرسم مقولة لآلبرت إينشتاين وسط بعض الأشكال
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.drawRect (50, 50, 40, 40);     // مربع
      page.drawRect (60, 80, 225, 30);    // مستطيل
      page.drawOval (75, 65, 20, 20);     // دائرة
      page.drawLine (35, 60, 100, 120);   // خط

      page.drawString ("Out of clutter, find simplicity.", 110, 70);
      page.drawString ("-- Albert Einstein", 130, 100);
   }
}
```



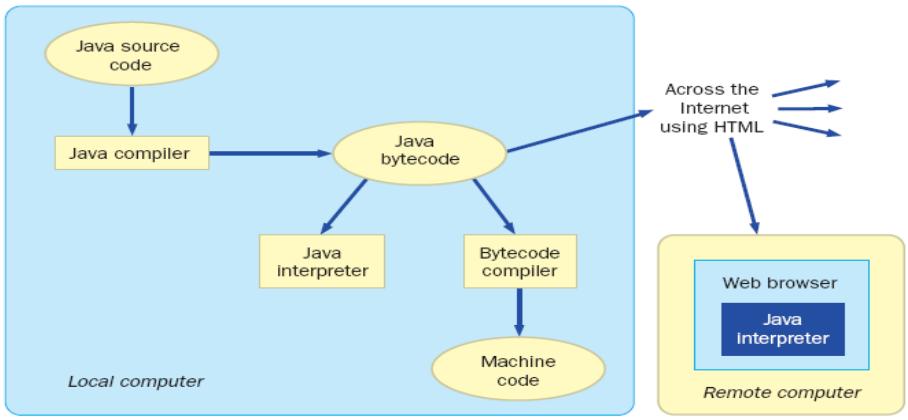
تنفيذ الآبلتات باستخدام الوب Executing Applets Using the Web

لكي يتم نقل الآبلت عبر الوب وتنفيذه بواسطة المستعرض، يجب أن يتم الإشارة إليه في مستند لغة *HTML* (*Hyper Text Markup Language*). ويحتوي مستند *HTML* على وسومات *tags* تحدد تعليمات التهيئة و الأنواع الخاصة للوسائط التي يتم تضمينها في المستند. ويعتبر برنامج الجافا نوعاً خاصاً من الوسائط مثله مثل النصوص *text*، والرسومات *graphics* والصوت *sound* .

يتم كتابة الـ *HTML tag* ضمن أقواس زوايا. ويعد الكود التالي مثالاً على الـ *applet tag* :

```
<applet code = "Einstein.class" width = "350" height = "175">
</applet>
```

هذا الوسم يوضح أن البايت كود *byte code* المخزن في الملف المسمى *Einstein.class* يجب أن ينقل عبر الشبكة وينفذ على جهاز كمبيوتر يحتاج لعرض مستند *HTML* معين. يشير وسم الآبلت *applet tag* أيضاً إلى عرض وارتفاع الآبلت.



الشكل رقم ٤.١ عملية ترجمة وتنفيذ الجافا، تشمل الآبلتات

هناك أيضاً وسوم *tags* أخرى يمكن استخدامها للإشارة إلى الآبلت في ملف *HTML*، وهي تشمل الوسم *<object>* والوسم *<embed>*. إن الوسم *<object>* هو الوسم الذي يجب أن يستخدم فعلاً، وذلك وفقاً لنادي الوب العالمي *World Wide Web Consortium* أو *(W3C)*. ومع ذلك فإن المستعرض الذي يدعم الوسم *<object>* غير ثابت. والى حد الآن يعتبر الحل الأكثر اعتماداً هو الوسم *<applet>*.

لاحظ أن وسم الآبلت يشير إلى ملف البايت كود للآبلت *Einstein*، وليس إلى ملف الشفرة المصدرية. وقبل أن نتمكن من نقل الآبلت باستخدام الوب، يجب ترجمته إلى صيغة البايت كود. ثم بعد ذلك، وكما هو واضح في الشكل ٤.١، فإنه يمكن تحميل المستند باستخدام مستعرض الوب والذي يقوم تلقائياً بتفسير وتنفيذ الآبلت.

٣.١ رسم الأشكال DRAWING SHAPES

تزودنا مكتبة الكلاسات القياسية للغة جافا *Java standard class library* بالعديد من الكلاسات التي تسمح لنا بتقديم ومعالجة المعلومات الرسومية. ويعتبر الكلاس *Graphics* أساسياً لكل عملية من هذا النوع.

الكلاس جرافيكس The Graphics Class

الكلاس *Graphics* معرف في الحزمة المسماة *java.awt*. وهو يحتوي على دوال مختلفة تمكننا من رسم الأشكال، بما في ذلك الخطوط *lines*، والمستطيلات *rectangles*، والأشكال البيضاوية *ovals*. وفي الشكل ٥.١ سرد لبعض دوال الرسم الأساسية في الكلاس *Graphics*. لاحظ أن هذه الدوال تسمح لنا أيضاً برسم الدوائر *circles* والمربعات *squares*، والتي تعتبر أشكالاً خاصة من الأشكال البيضاوية *ovals* والمستطيلات *rectangles*، على التوالي. وسنناقش دوال رسم إضافية في الكلاس *Graphics* لاحقاً في هذا الكتاب.

مفهوم أساسي Key Concept

يمكن رسم أغلب الأشكال بطريقة التعبئة filled أو opaque أو بدون تعبئة unfilled أو outline.

تمكننا دوال الكلاس *Graphics* من تحديد فيما إذا كنا نريد شكلاً بتعبئة *filled* أو بدون تعبئة *unfilled*. يظهر الشكل بدون تعبئة المخطط الخارجي *outline* للشكل فقط، ولهذا السبب فهو شفاف *transparent* (حيث يمكننا رؤية أي شكل أسفل منه). أما الشكل ذو التعبئة فهو مصمت بين حدوده ويغطي على أي رسومات أسفل منه.

كل هذه الدوال تعتمد نظام إحداثيات جافا، والذي ناقشناه سابقا في هذا الفصل. ولنتذكر أن النقطة *(0,0)* تقع في الزاوية الشمالية العليا، بحيث تزيد قيم *x* كلما تحركنا يميناً، وتزيد قيم *y* كلما تحركنا للأسفل. إن أي أشكال ترسم في إحداثيات خارج المساحة المرئية لن تكون مرئية.

مفهوم أساسي Key Concept

يستخدم مستطيل الإحاطة bounding rectangle لتعريف موقع وحجم الأشكال المنحنية مثل الأشكال البيضاوية ovals.

تعتبر العديد من دوال الرسم في الكلاس *Graphics* ذاتية التفسير، لكن بعضها يتطلب نقاشاً أكثر قليلاً. لاحظ، مثلاً، أن الشكل البيضاوي المرسوم بواسطة الدالة *drawOval* تعرف عن طريق إحداثيات الزاوية الشمالية العليا والأبعاد التي تحدد العرض والطول لمستطيل الإحاطة *bounding rectangle*. غالباً ما يتم تعريف الأشكال ذوات المنحنيات، مثل الأشكال البيضاوية *ovals*، بواسطة مستطيل يطوق حدود المحيط لهذه الأشكال. ويوضح الشكل ٥.١ مستطيل الإحاطة للشكل البيضاوي.

يمكن أن ننظر للقوس *arc* على أنه قطاع من الشكل البيضاوي *ovals*. ولكي نرسم القوس، نقوم بتحديد الشكل البيضاوي الذي يكون القوس جزءاً منه والجزء من الشكل البيضاوي الذي يهمنا. ويتم تعريف نقطة البداية للقوس بواسطة زاوية البدء *start angle* ونقطة نهاية القوس تعرف بواسطة زاوية القوس *arc angle*. إن زاوية القوس لا تشير إلى نهاية القوس، ولكنها تحدد مداه. وتقاس زاويتا البدء و القوس بالدرجات *degrees*. وأصل زاوية البدء هو خط أفقي وهمي يمر عبر مركز الشكل البيضاوي ويمكن الإشارة إلى ذلك بالزاوية *0°*، كما هو واضح في الشكل ٦.١.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
void drawLine(int x1, int y1, int x2, int y2)
```
ترسم سطراً من النقطة (x1, y1) إلى النقطة (x2, y2).

```
void drawRect(int x, int y, int width, int height)
```
ترسم مستطيلاً زاويته العليا الشمالية هي (x, y) وأبعاده هي width و height.

```
void drawOval(int x, int y, int width, int height)
```
ترسم شكلاً بيضاوياً محاط بمستطيل زاويته العليا الشمالية هي (x, y) وأبعاده هي width و height.

```
void drawString(String str, int x, int y)
```
ترسم السلسلة النصية str ابتداءً من النقطة (x, y)، وتمتد جهة اليمين.

```
void drawArc(int x, int y, int width, int height,
 int startAngle, int arcAngle)
```
ترسم قوساً عبر الشكل البيضاوي المحاط بالمستطيل المعرف بـ x، y، width، و height. يبدأ القوس بالزاوية startAngle ويمتد للمسافة المعرفة بالزاوية arcAngle.

```
void fillRect(int x, int y, int width, int height)
```
تشبه أختها draw، باستثناء أنها تعبي الشكل باللون الأمامي الحالي.

```
void fillOval(int x, int y, int width, int height)
```
تشبه أختها draw، باستثناء أنها تعبي الشكل باللون الأمامي الحالي.

```
void fillArc(int x, int y, int width, int height,
 int startAngle, int arcAngle)
```
تشبه أختها draw، باستثناء أنها تعبي الشكل باللون الأمامي الحالي.

```
Color getColor( )
```
ترجع اللون الأمامي للسياق الرسومي الحالي.

```
void setColor(Color color )
```
تضبط اللون الأمامي للسياق الرسومي الحالي على اللون المحدد color.

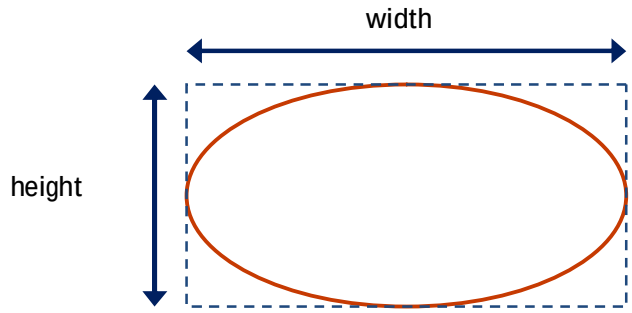
الشكل رقم ٥.١ بعض دوال الكلاس Graphics

مفهوم أساسي Key Concept

القوس arc **هو قطاع من الشكل البيضاوي** ovals **يبدأ بزاوية بدء محددة ويمتد لمسافة محددة بواسطة زاوية القوس.**

لكل سياق رسومي لون أمامي حالي *foreground color* **يستخدم عند رسم الأشكال** *shapes* **والسلاسل النصية** *strings*. **وكل سطح يمكن الرسم عليه له لون خلفي** *background color*. **يتم ضبط قيمة اللون الأمامي باستخدام الدالة** *setColor* **التابعة للكلاس** *Graphics*، **ويتم ضبط قيمة اللون الخلفي باستخدام الدالة** *setBackground* **للمكون** *component* **الذي نرسم عليه، مثل الآبلت.**

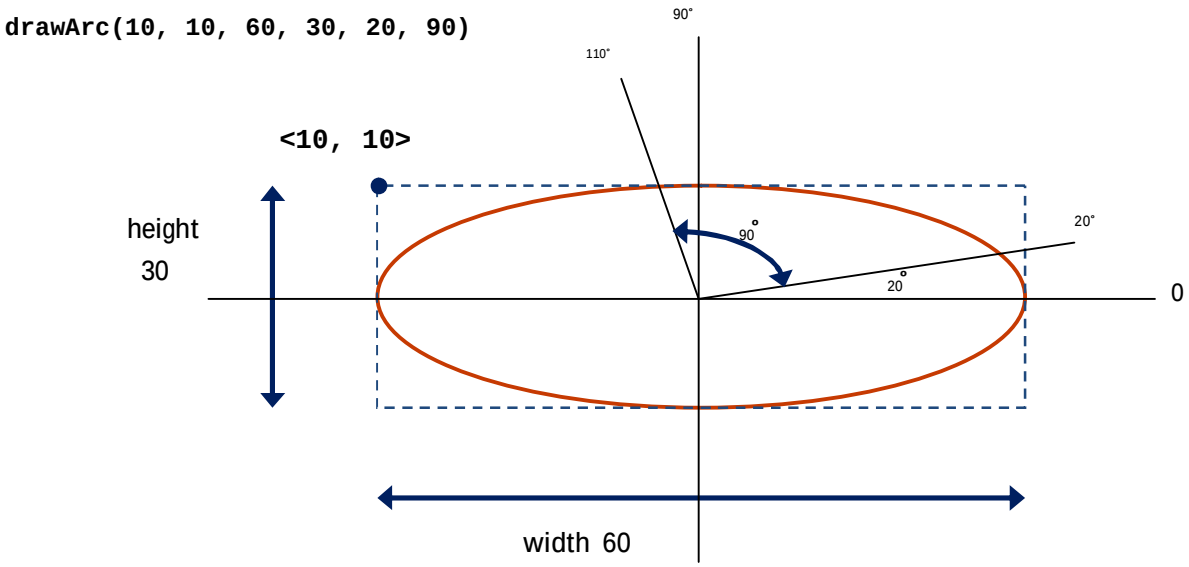
Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



الشكل رقم ٦.١ شكل بيضاوي ومستطيل الإحاطة الخاص به

يوضح المثال ٢.١ آبلت يسمى *Snowman* (رجل الثلج). وهو يستخدم دوال رسم وتلوين متنوعة لرسم مشهد شتوي يظهر فيه رجل الثلج. قم بمراجعة الشفرة جيداً لتلاحظ كيفية رسم كل شكل بغرض توليد الصورة الكلية.

لاحظ أن شكل رجل الثلج يعتمد على قيمتين ثابتتين تسميان *MID* و *TOP*، وهاتان النقطتان تعرفان نقطة الوسط لرجل الثلج (من الشمال إلى اليمين) وقمة رأس رجل الثلج. ويتم رسم شكل رجل الثلج كاملاً تناسباً مع هذه القيم. إن استخدامك لمثل هذه القيم الثابتة يجعل من السهل إنشاء رجل الثلج وعمل التعديلات لاحقاً. مثلاً، إذا أردنا إزاحة رجل الثلج إلى جهة اليمين أو إلى جهة الشمال في هذه الصورة، فسنحتاج فقط لتغيير التصريح عن واحدة من القيم الثابتة.



الشكل رقم ٧.١ قوس معرف بواسطة شكل بيضاوي، وزاوية بدء، وزاوية قوس

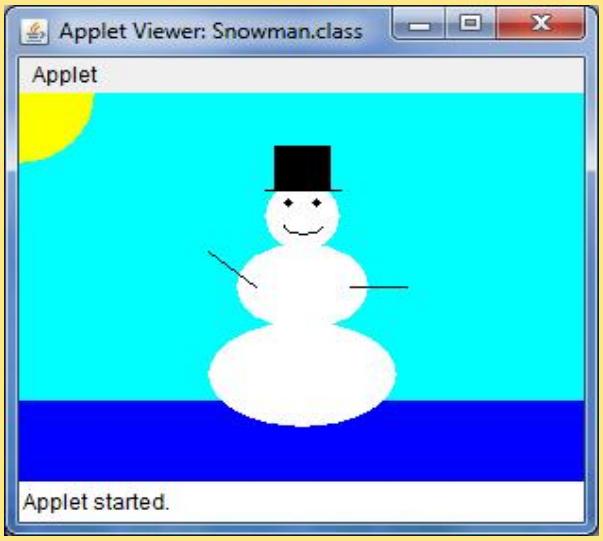
مثال رقم ٢.١

```
//*************************************************************************
//  Snowman.java       المؤلف: Lewis/Loftus    المترجم : Almashraqi
//
//  يشرح هذا البرنامج دوال الرسم الأساسية واستخدام التلوين
//*************************************************************************

import javax.swing.JApplet;
import java.awt.*;
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
public class Snowman extends JApplet
{
   //-----------------------------------------------------------------
   //  ترسم الدالة التالية رجل ثلج
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      final int MID = 150;
      final int TOP = 50;

      setBackground (Color.cyan);

      page.setColor (Color.blue);
      page.fillRect (0, 175, 300, 50);  // الأرض


      page.setColor (Color.yellow);
      page.fillOval (-40, -40, 80, 80);  // الشمس

      page.setColor (Color.white);
      page.fillOval (MID-20, TOP, 40, 40);      // الرأس
      page.fillOval (MID-35, TOP+35, 70, 50);   // الجزء الأعلى من الجسم
      page.fillOval (MID-50, TOP+80, 100, 60);  // الجزء الأسفل من الجسم

      page.setColor (Color.black);
      page.fillOval (MID-10, TOP+10, 5, 5);   // العين اليسرى
      page.fillOval (MID+5, TOP+10, 5, 5);    // العين اليمنى

      page.drawArc (MID-10, TOP+20, 20, 10, 190, 160);   // الابتسامة

      page.drawLine (MID-25, TOP+60, MID-50, TOP+40);  // الذراع اليسرى
      page.drawLine (MID+25, TOP+60, MID+55, TOP+60);  // الذراع اليمنى

      page.drawLine (MID-20, TOP+5, MID+20, TOP+5);  // أسفل القبعة
      page.fillRect (MID-15, TOP-20, 30, 25);        // أعلى القبعة
   }
}
```

العرض Display



الفصل الثاني

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

تكلمنا في الفصل الأول عن مقدرات جافا في رسم الأشكال باستخدام الكلاسات *Graphics* و *Color* في مكتبة الكلاسات القياسية للغة جافا. قمنا أيضاً بتعريف مفهوم الآبلت *applet* والذي هو عبارة عن برنامج جافا معد ليتم تضمنيه في صفحة وب ويتم تنفيذه عن طريق مستعرض الوب. ولتتذكر أن تطبيقات جافا، بخلاف الآبلتات، هي برامج قائمة بذاتها لا يتم تنفيذها من خلال الوب.

هناك الكثير من برامج جافا عبارة عن تطبيقات *applications*. وتحديداً، هي تطبيقات جافا تنفذ عبر سطر الأوامر (شاشة الدوز) *command-line applications*، حيث أن الطريقة الوحيدة للتفاعل مع المستخدم تكون فقط من خلال رسائل نصية تحفيزية. ويمكن أن يمتلك تطبيق الجافا مكونات رسومية. وسنقوم باستكشاف مقدرات جافا في توليد برامج ذات واجهات مستخدم رسومية *graphical user interfaces(GUIs)*. وفي هذا الفصل سنتكلم عن القضايا الأساسية المتعلقة بالتطبيقات المعتمدة على الرسومات.

إن مكون الجوئي *GUI component* هو عبارة عن كائن يمثل عنصر في الشاشة يستخدم لعرض المعلومات والسماح للمستخدم بالتفاعل مع البرنامج بطريقة ما. ومن الأمثلة على مكونات الجوئي المعنونات *labels*، والأزرار *buttons*، وحقول النص *text fields*، وأشرطة التمرير *scroll bars*، والقوائم *menus*.

إن كلاسات مكونات جافا وكلاسات الجوئي ذات العلاقة معرفة بشكل أساسي في حزمتين هما *java.awt* و *javax.swing*، (لاحظ الحرف *x* في *javax.swing*). لقد كانت أداة النوافذ المجردة *Abstract Windowing Toolkit (AWT)* تمثل حزمة الجوئي الأصلية في لغة جافا. ومازالت هذه الحزمة تحوى العديد من الكلاسات المهمة، مثل الكلاس *Color* الذي استخدمناه في الفصل الأول. أما الحزمة *Swing* فقد تم إضافتها حديثاً وقد زودتنا بمكونات تعتبر أكثر تنوعاً من تلك الموجودة في الحزمة *AWT*. وسنحتاج لكلا الحزمتين لعمليات تطوير الجوئي، ولكننا سنلجأ لمكونات *Swing* كلما كان هناك فرصة.

يعتبر الحاوي *container* نوعاً خاصاً من المكونات يستخدم لضم وتنظيم المكونات الأخرى. ومن الأمثلة على الحاويات في لغة جافا الإطارات *frames* واللوحات *panels*. دعنا الآن نقوم باستكشاف هذين الحاويين بشكل أكثر تفصيلاً.

مفهوم أساسي Key Concept

الحاويات *containers* هي مكونات جوئي خاصة تضم وتنظم مكونات أخرى.

الإطارات واللوحات Frames and Panels

يعتبر الإطار *frame* حاوياً يستخدم لعرض تطبيقات الجافا المعتمدة على الجوئي ويتم عرض الإطار كنافذة منفصلة لها شريط عنوان *title bar* خاص بها. ويمكن تغيير موضع الإطار على الشاشة وتغيير حجمه عند الحاجة عن طريق سحبه باستخدام الماوس. ويحتوي الإطار على أزرار صغيرة في زاويته تسمح بتصغير الإطار وتكبيره وإغلاقه. ويتم تعريض الإطار من خلال الكلاس *JFrame* .

تعتبر اللوحة *panel* أيضا حاوياً. مع ذلك وبخلاف الإطار، فإنه لا يمكن عرضها بشكل مستقل. حيث يجب إضافتها إلى حاوي آخر ليتم عرضها عليه. عموماً لا يمكن تحريك اللوحة ما لم يتم تحريك الحاوي الذي يضمها. والدور الرئيسي للوحة هو المساعدة في تنظيم المكونات الأخرى في الجوئي. ويتم تعريف اللوحة من خلال الكلاس *JPanel*.

مفهوم أساسي Key Concept

يتم عرض الإطار *frame* كنافذة منفصلة، لكن اللوحة *panel* يمكن عرضها فقط كجزء من حاوي آخر.

يمكننا تصنيف الحاويات *containers* إلى حاويات ذات وزن ثقيل *heavyweight* وحاويات خفيفة الوزن *lightweight*. الحاوي ذو الوزن الثقيل يتم إدارته من قبل نظام التشغيل الذي يتم تشغيل البرنامج عليه، بينما الحاوي خفيف الوزن فيدار بواسطة برنامج الجافا نفسه. وبين الحين والآخر سيكون هذا الفرق مهماً عندما نقوم باستكشاف تطوير الجوئي. يعتبر الإطار *frame* مكوناً ثقيل الوزن، بينما تعتبر اللوحة *panel* مكوناً خفيف الوزن.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

تعتبر المكونات ثقيلة الوزن أكثر تعقيداً من المكونات خفيفة الوزن بشكل عام. فالإطار، مثلاً، له العديد من المقاطع *panes* والتي تكون مسئولة عن الخصائص المختلفة لنافذة الإطار. ويتم عرض كل العناصر المرئية لواجهة الجافا في مقطع المحتوى *content pane* الخاص بالإطار.

عموماً، يمكننا توليد تطبيقات جافا معتمدة على الجوئي عن طريق توليد إطار يتم من خلاله عرض واجهة البرنامج. يتم غالباً تنظيم الواجهة في داخل اللوحة الأساسية، والتي يتم إضافتها إلى مقطع المحتوى *content pane* الخاص بالإطار. وغالباً ما يتم تنظيم مكونات اللوحة الأساسية باستخدام لوحات أخرى حسب الحاجة.

وبشكل عام يمكن القول أن الحاويات *containers* ليست مهمة ما لم تساعدنا في تنظيم وعرض المكونات الأخرى. دعنا الآن نستكشف مكون جوئي أساسي آخر. هذا المكون هو المعنون *label* وهو مكون يعرض سطر نصي في الجوئي. يمكن أن يقوم المعنون أيضاً بعرض صورة *image* وهذا موضوع سيتم مناقشته لاحقاً في هذا الفصل. عادة ما يتم استخدام المعنونات لعرض المعلومة أو تحديد مكونات أخرى في الجوئي. يمكن أن توجد المعنونات تقريباً في كل برنامج يعتمد على الجوئي.

دعنا الآن ننظر إلى مثال يستخدم الإطارات *frames*، واللوحات *panels*، والمعنونات *labels*. عندما يتم تنفيذ البرنامج في المثال ١.٢، ستظهر نافذة جديدة على الشاشة تقوم بعرض عبارة (مقولة). يتم عرض نص العبارة باستخدام اثنين من المعنونات. ويتم تنظيم المعنونات من خلال لوحة، ويتم عرض اللوحة على مقطع المحتوى *content pane* الخاص بالإطار.

يقوم الباني *constructor* الخاص بالكلاس *JFrame* بأخذ سلسلة نصية *String* كوسيط، حيث يتم عرض هذا النص في شريط العنوان الخاص بالإطار. و يعمل استدعاء الدالة *setDefaultCloseOperation* على تحديد ما الذي سيحدث عند ضغط زر الإغلاق (الزر X) في زاوية الإطار. وفي أغلب الحالات سنقوم ببساطة بجعل هذا الزر يقوم بإنهاء البرنامج، كما هو مشار إليه من خلال القيمة الثابتة *EXIT_ON_CLOSE*.

ويتم إنشاء اللوحة *panel* عن طريق الكلاس *JPanel*. ويتم ضبط قيمة لون الخلفية للوحة باستخدام الدالة *setBackground*. أما الدالة *setPreferredSize* فتستقبل كائن من الكلاس *Dimension* كوسيط لها، وهو يستخدم لتحديد عرض وارتفاع المكون بالبكسلات. ويمكن بمثل هذه الطريقة ضبط حجم العديد من المكونات، وهناك مكونات عديدة أخرى تملك الدوال *setMinimumSize* و *setMaximumSize* للمساعدة في التحكم بمظهر الواجهة.

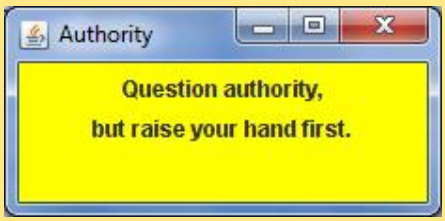
ويتم إنشاء المعنونات *labels* عن طريق الكلاس *JLabel* حيث يتم تمرير نص المعنون إلى الباني الخاص بهذا الكلاس. وفي برنامجنا هذا تم إنشاء اثنين من المعنونات. تمتلك الحاويات *containers* دالة تسمى *add* تسمح بإضافة مكونات أخرى إلى هذه الحاويات. حيث تم إضافة كلا المعنونين إلى اللوحة الأساسية، ومن هنا فهما يعتبران جزءاً من هذه اللوحة. ويعتبر الترتيب الذي يتم به إضافة المكونات إلى الحاوي أمراً مهماً، ففي هذه الحالة يحدد أياً من المعنونات يظهر فوق الآخر. أخيراً، يتم الحصول على مقطع المحتوى *content pane* الخاص بالإطار باستخدام الدالة *getContentPane*، وهذا يتم مباشرة بعد إن يتم استدعاء الدالة *add* الخاصة بمقطع المحتوى *content pane* والتي تقوم بإضافة اللوحة. أما الدالة *pack* الخاصة بالإطار فتعمل على ضبط حجم الإطار بشكل مناسب اعتماداً على محتواه. في هذه الحالة يتم ضبط حجم الإطار بحيث يتلائم مع حجم اللوحة التي يحتويها. وهذا أسلوب أفضل من محاولة ضبط حجم الإطار بشكل صريح، حيث أنه يجب أن يتغير حجم الإطار تبعاً لتغير حجم مكوناته. ثم إن استدعاء الدالة *setVisible* يعمل على عرض الإطار على الشاشة. ولا يعتبر البرنامج *Authority* برنامجاً تفاعلياً. وعموماً، فإن المعنونات لا تسمح للمستخدم بالتفاعل مع البرنامج. وسوف نقوم بدراسة مكونات الجوئي التفاعلية في الفصل التالي. مع ذلك، فإن بإمكانك التفاعل مع الإطار نفسه بطرق متنوعة. حيث يمكنك تحريك كامل الإطار إلى نقطة أخرى على سطح المكتب عن طريق النقر على شريط العنوان الخاص بالإطار وسحبه بالماوس. يمكنك أيضاً تغيير حجم الإطار عن طريق سحب الزاوية اليمنى السفلية من الإطار. ولاحظ ما يحدث عندما يصبح الإطار أوسع: سيقوم المعنون الثاني بالارتفاع إلى جانب المعنون الأول.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مثال رقم ١.٢

```
//*****************************************************************
//  Authority.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الإطارات، اللوحات، والمعنونات
//*****************************************************************

import java.awt.*;
import javax.swing.*;

public class Authority
{
   //--------------------------------------------------------------
   //  تعرض الدالة الرئيسية بعض المقولات الحكيمة
   //--------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Authority");

      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JPanel primary = new JPanel();
      primary.setBackground (Color.yellow);
      primary.setPreferredSize (new Dimension(250, 75));

      JLabel label1 = new JLabel ("Question authority,");
      JLabel label2 = new JLabel ("but raise your hand first.");

      primary.add (label1);
      primary.add (label2);

      frame.getContentPane().add(primary);
      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مفهوم أساسي Key Concept

يتم إدارة أي حاوي *container* بواسطة مدير التخطيط *layout manager*.

يتم إدارة أي حاوي *container* بواسطة كائن يسمى مدير التخطيط *layout manger*، والذي يكون مسئولاً عن كيفية تخطيط المكونات داخل الحاوي، ويتم استدعاء مدير التخطيط عن حدوث أشياء مهمة للواجهة، مثلاً تغيير حجم الإطار.

إذا لم تقم بتحديد طريقة تخطيط المكونات، فإن مكونات اللوحة ستحاول ترتيب نفسها بجانب بعضها البعض في شكل صف، وينتقل المكون إلى الصف التالي فقط عندما لا يصبح عرض اللوحة كافياً لاحتوائه في هذا الصف. جرب هذا البرنامج لترى كيف أن مدير التخطيط يعمل على تغيير تنظيم المعنونات كلما تغير حجم النافذة. وسنقوم بمناقشة موضوع مدراء التخطيط *layout mangers* بتفصيل أكثر في الفصل الخامس.

٢.٢ اللوحات المتداخلة NESTED PANELS

رأينا في القسم السابق، مثالاً تم فيه احتواء معنونين داخل لوحة واحدة والتي بدورها كانت محتواة داخل إطار . مثل هذه العلاقة تشكل ما يسمى هرم الاحتواء *containment hierarchy* للواجهة، والذي يمكن أن يكون معقداً حسب الحاجة لتوليد التأثيرات المرئية المطلوبة.

مفهوم أساسي Key Concept

يمكن عمل لوحات متداخلة *nested panels* لتوليد هرم احتواء معقد للمكونات.

بشكل خاص، من الشائع أن يكون لدينا عدة طبقات *layers* من اللوحات المتداخلة لتنظيم وتجميع المكونات بطرق متنوعة. وفي حين أنه لا يتوجب عليك تضمين المكونات الغير ضرورية في هرم الاحتواء، فلا تكن مغرماً بتضمين مكونات إضافية غير ضرورية عند إنشاء الواجهة للمساعدة في الحصول على التأثير المطلوب.

يقوم البرنامج في المثال ٢.٢، *NestedPanel*، بإنشاء لوحتين فرعيتين *subpanels* كلٌ منهما تحتوي على معنون. وكلا اللوحتان الفرعيتان تم وضعهما داخل لوحة أخرى، والتي أضيفت بعد ذلك إلى مقطع المحتوى *content pane* الخاص بالإطار.

لاحظ أن اللوحة الأساسية في البرنامج لم يتم ضبطها بشكل صريح، فهي تضبط حجمها بنفسها كلما دعت الحاجة كي تتلائم مع اللوحتين المحتواتين بداخلها.

أيضاً لاحظ أن اللوحات الفرعية يحيط بها مساحة فارغة من خلالها يمكن رؤية اللون الأزرق للوحة الأساسية. مثل هذا الفراغ يعد من اختصاصات مدير التخطيط *layout manager* والذي يستخدم للتحكم بالحاوي، ومجموعة الخصائص للمكونات نفسها. وسيتم التطرق بشكل أكثر تفصيلاً لمثل هذه القضايا في الفصول القادمة.

وكما قمت باختبار المثال السابق، قم الآن بتنفيذ واختبار هذا المثال. قم بتغيير حجم الإطار لرؤية التأثير الذي يحدث للمكونات. لاحظ أن حجم اللوحات الفرعية يظل ثابتاً ، وأن اتجاه كلا اللوحتين يتغير تبعاً لتغير عرض اللوحة الأساسية (والتي تتمدد بتمدد الإطار).

بعد أن ترتاح لطريقة تخطيط المكونات مع بعضها البعض، قم بتغيير لون الخلفية لكل اللوحات بحيث يكون لها نفس اللون (ولنقل أنه اللون الأخضر) وذلك لكي ترى كيف أن الفرق بين اللوحات يمكن أن يكون غير مرئياً إذا تم تصميم الواجهة وفقاً لذلك.

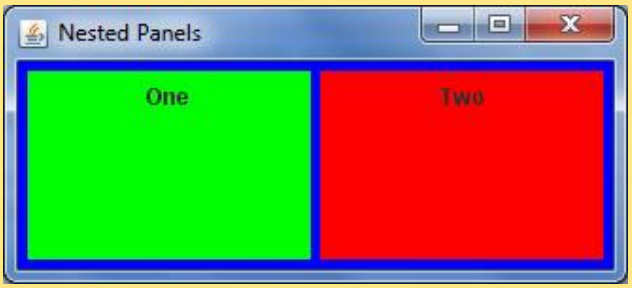
مثال رقم ٢.٢

```
//*****************************************************************
//  NestedPanels.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال الهرمية الأساسية للمكون
//*****************************************************************

import java.awt.*;
import javax.swing.*;

public class NestedPanels
{
   //-----------------------------------------------------------------
   //  تعرض الدالة الرئيسية لوحتين ملونتين متداخلتين داخل لوحة ثالثة
   //-----------------------------------------------------------------
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
    public static void main (String[] args)
    {
       JFrame frame = new JFrame ("Nested Panels");
       frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

       // تهيئة اللوحة فرعية الأولى
       JPanel subPanel1 = new JPanel();
       subPanel1.setPreferredSize (new Dimension(150, 100));
       subPanel1.setBackground (Color.green);
       JLabel label1 = new JLabel ("One");
       subPanel1.add (label1);

       // تهيئة اللوحة فرعية الثانية
       JPanel subPanel2 = new JPanel();
       subPanel2.setPreferredSize (new Dimension(150, 100));
       subPanel2.setBackground (Color.red);
       JLabel label2 = new JLabel ("Two");
       subPanel2.add (label2);

       // تهيئة اللوحة الرئيسية
       JPanel primary = new JPanel();
       primary.setBackground (Color.blue);
       primary.add (subPanel1);
       primary.add (subPanel2);

       frame.getContentPane().add(primary);
       frame.pack();
       frame.setVisible(true);
    }
}
```

العرض Display



٣.٢ الصور IMAGES

تلعب الصور *images* **غالباً دوراً مهماً في البرمجيات المعتمدة على الرسومات. وتمتلك لغة جافا القدرة على استخدام صور** *JPEG* **و** *GIF* **بطرق متنوعة. حيث يحتوي الكلاس** *Graphics* **على الدالة** *drawImage* **والتي تسمح لك برسم الصورة تماماً مثلما ترسم الشكل أو الحروف النصية. يمكن أيضاً أن تدمج الصورة** *image* **مع معنون** *label*. **دعنا الآن نستكشف العلاقة بين الصور والمعنونات بتفصيل أكثر.**

كما رأينا سابقاً، يمكن استخدام معنون معرف من الكلاس *JLabel* ليزود بمعلومة للمستخدم أو لوصف مكونات في الواجهة. يمكن أن يحتوي المعنون *JLabel* أيضاً على صورة، أي أنه يمكن أن يتكون المعنون من نص *text* أو صورة *image* أو كليهما.

مفهوم أساسي Key Concept

يمكن للمعنون *label* أن يحتوي على نص *text* أو صورة *image* أو كليهما.

يستخدم الكلاس *ImageIcon* لتمثيل صورة مضمنة داخل معنون، ويأخذ باني الكلاس *ImageIcon* اسم ملف الصورة ويعمل على تحميله في الكائن. يمكن عمل كائنات الكلاس *ImageIcon* إما من خلال صور *JPEG* أو صور *GIF*.

يمكننا بشكل صريح ضبط محاذاة النص والصورة داخل المعنون، وذلك إما باستخدام باني الكلاس *JLabel* أو باستخدام دوال مخصصة. بشكل مشابه، يمكننا ضبط موقع النص تناسباً مع الصورة.

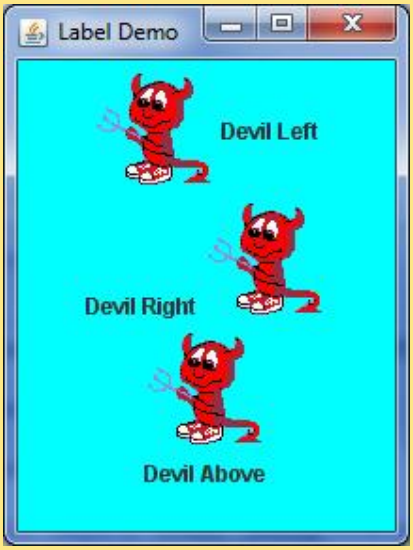
يقوم البرنامج في المثال ٣.٢ بعرض عدة معنونات، وكل معنون يقوم بعرض نص وصورة بعدة اتجاهات.

مثال رقم ٣.٢

```
//*************************************************************
//  LabelDemo.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام أيقونات الصور في المعنونات
//*************************************************************

import java.awt.*;
import javax.swing.*;

public class LabelDemo
{
   //-----------------------------------------------------------------
   //  تنشئ الدالة الرئيسية وتعرض إطار التطبيق الأساسي
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Label Demo");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      ImageIcon icon = new ImageIcon ("devil.gif");

      JLabel label1, label2, label3;

      label1 = new JLabel ("Devil Left", icon, SwingConstants.CENTER);

      label2 = new JLabel ("Devil Right", icon, SwingConstants.CENTER);
      label2.setHorizontalTextPosition (SwingConstants.LEFT);
      label2.setVerticalTextPosition (SwingConstants.BOTTOM);

      label3 = new JLabel ("Devil Above", icon, SwingConstants.CENTER);
      label3.setHorizontalTextPosition (SwingConstants.CENTER);
      label3.setVerticalTextPosition (SwingConstants.BOTTOM);
```

```
        JPanel panel = new JPanel();
        panel.setBackground (Color.cyan);
        panel.setPreferredSize (new Dimension (200, 250));
        panel.add (label1);
        panel.add (label2);
        panel.add (label3);

        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

العرض Display



هناك وسيط ثالث يتم تمريره إلى باني الكلاس *JLabel* ويستخدم لتعريف الموضع الأفقي للمعنون في الفراغ المسموح به لهذا المعنون داخل اللوحة. تحتوي الواجهة *interface* المسماة *SwingConstants* على العديد من الثوابت التي تستخدم بواسطة مكونات *Swing* مختلفة وهذا يسهل الإشارة إليها.

يتم بشكل صريح ضبط اتجاه نص المعنون وصورته باستخدام الدالتين *setHorizontalTexPosition* و *setVerticalTextPosition*. وكما هو واضح في حالة المعنون الأول، فإن الموضع الأفقي الافتراضي للنص هو جهة اليمين (بحيث تكون الصورة على الشمال)، وكذلك فإن الموضع العمودي الافتراضي للنص هو التوسيط بالتناسب مع الصورة.

لا تخلط بين الموضع الأفقي للمعنون في الحاوي مع ضبط الاتجاه بين النص والصورة. حيث يقوم الوسيط الثالث في الباني بتحديد الموضع الأفقي، في حين أن النداءات الصريحة للدالة تحدد ضبط الاتجاه بين النص والصورة .

وبوضعنا للصورة في المعنون، تصبح الصورة جزءاً من مكون يتم تخطيطه مع كامل المكونات الأخرى في الحاوي، بدلاً من أن يتم رسمه في مكان مخصص. إن رسم الصورة باستخدام الدالة *drawImage* الموجودة في الكلاس *Graphics* أو باستخدام معنون لعرض الصورة يعتبر قراراً تصميمياً بامتياز. ثم إن اختيارك لأي من هذين الخيارين يجب أن يعتمد على الاحتياجات الخاصة ببرنامجك.

الفصل الثالث

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

تمتلك بعض الكائنات تمثيلاً رسومياً، مما يعني أن حالتها وسلوكها يتضمنان معلومات عن المظهر المرئي للكائن. ويمكن أن يحتوي الكائن الرسومي على بيانات عن الحجم *size* واللون *color* مثلاً، ويمكن أن يحتوي أيضاً على دالة لرسمه.

لقد قمنا في الفصل الثاني بتوليد واستخدم مكونات رسومية مثل الإطارات *frames*، واللوحات *panels*، والمعنونات *labels*، والصور *images*. ومن المؤكد أنه يمكن اعتبار هذه المكونات مكونات رسومية. وفي هذا القسم سندرسها بتفصيل أكثر ونتعلم كيف يمكننا تعريف كائنات خاصة بنا، بحيث يكون لها خصائص رسومية.

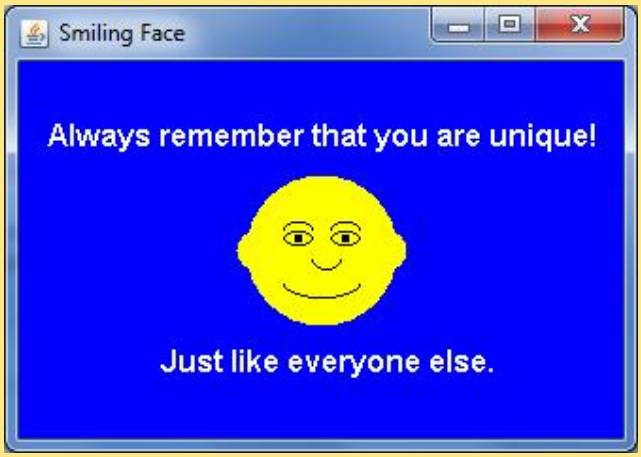
يعرض البرنامج في المثال ١.٣ وجهاً مبتسماً ونصاً مرافقاً. ونلاحظ أن الدالة *main* الموجودة في الكلاس *SmilingFace* لا تتعامل مع كل تلك التفاصيل. فبدلاً من ذلك، تعمل الدالة *main* على تهيئة إطار البرنامج واستخدامه لعرض كائن من الكلاس *SmilingFacePanel*.

الكلاس *SmilingFacePanel* موضح من خلال المثال ٢.٣ حيث تم فيه تعريف ثابتين يعتمد عليهما الرسم وهما *(BASEY, BASEX)*، وكذلك تم تعريف أهم مكونات اللوحة، وكذا تم تعريف الدالة *paintComponent* لرسم الوجه الذي نراه عند تنفيذ البرنامج. في هذه الحالة، بدلاً من إضافة مكونات الجوئي إلى هذه اللوحة، نقوم ببساطة بالرسم عليها.

مثال رقم ١.٣

```
//*************************************************************
//  SmilingFace.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام كلاس للوحة منفصلة
//*************************************************************

import javax.swing.JFrame;

public class SmilingFace
{
   //-----------------------------------------------------------
   //  تنشئ الدالة الرئيسية إطار البرنامج
   //-----------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Smiling Face");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      SmilingFacePanel panel = new SmilingFacePanel();

      frame.getContentPane().add(panel);

      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



مثال رقم ٢.٣

```
//********************************************************************
//  SmilingFacePanel.java  المؤلف: Lewis/Loftus  المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام كلاس للوحة منفصلة
//********************************************************************

import javax.swing.JPanel;
import java.awt.*;

public class SmilingFacePanel extends JPanel
{
   private final int BASEX = 120, BASEY = 60; // النقطة الأساسية للرأس

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة الخصائص الرئيسية لهذه اللوحة
   //-----------------------------------------------------------------
   public SmilingFacePanel ()
   {
      setBackground (Color.blue);
      setPreferredSize (new Dimension(320, 200));
      setFont (new Font("Arial", Font.BOLD, 16));
   }

   //-----------------------------------------------------------------
   //  الدالة التالية ترسم الوجه
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);

      page.setColor (Color.yellow);
      page.fillOval (BASEX, BASEY, 80, 80);  // الرأس
      page.fillOval (BASEX-5, BASEY+20, 90, 40);  // الأذنان
```

```
        page.setColor (Color.black);
        page.drawOval (BASEX+20, BASEY+30, 15, 7);  // العينان
        page.drawOval (BASEX+45, BASEY+30, 15, 7);

        page.fillOval (BASEX+25, BASEY+31, 5, 5);   // بؤبؤ العينان
        page.fillOval (BASEX+50, BASEY+31, 5, 5);

        page.drawArc (BASEX+20, BASEY+25, 15, 7, 0, 180); // حاجب العينان
        page.drawArc (BASEX+45, BASEY+25, 15, 7, 0, 180);

        page.drawArc (BASEX+35, BASEY+40, 15, 10, 180, 180);  // الأنف
        page.drawArc (BASEX+20, BASEY+50, 40, 15, 180, 180);  // الفم

        page.setColor (Color.white);
        page.drawString ("Always remember that you are unique!",
                         BASEX-105, BASEY-15);
        page.drawString("Just like everyone else.", BASEX-45, BASEY+105);
    }
}
```

لاحظ أن الكلاس *SmilingFacePanel* يرث الكلاس *JPanel*. وكما ذكرنا في الفصل الأول عند نقاشنا للآبلتات، فإننا نستخدم كلمة *extends* لتمثيل علاقة الوراثة *inheritance*. وعندما نقول أن الكلاس *SmilingFacePanel* يرث خصائص الكلاس *JPanel*، فإن ذلك يعني أن *SmilingFacePanel* هو عبارة عن *JPanel*. والى حد الآن سيكون هذا كل ما نحتاج أن نعرفه عن الوراثة، وسنناقش التفاصيل في الفصل السابع.

يقوم الباني الخاص بالكلاس *SmilingFacePanel* بضبط لون الخلفية والحجم المفضل للوحة، وكذلك ضبط الخط الافتراضي للوحة. لاحظ أن هذه النداءات لا تعمل مع بعض الكائنات الأخرى، تماماً كما حصل معنا في الفصل الثاني عندما قمنا بإنشاء كائن *JPanel* منفصل. عندما لا يتم استدعاء الدالة من خلال كائن معين، فيمكن النظر إليها على أنها كائن "يتحدث إلى نفسه". فالنداءات في الباني تتم للكائن الذي يمثل الكلاس *SmilingFacePanel*.

تستقبل الدالة *paintComponent* كائناً من الكلاس *Graphics* كوسيط لها، والذي يمثل، كما ناقشنا في الفصل الأول، السياق الرسومي للمكون. يتم رسم الرسومات على اللوحة عن طريق استدعاء السياق الرسومي الخاص بها (الوسيط *Page*). يمتلك كل كائن من الكلاس *JPanel* دالة تسمى *paintComponent* يتم استدعاءها تلقائياً لرسم اللوحة. في هذه الحالة قمنا بإضافات إلى تعريف الدالة *paintComponent* لنخبرها أنه بالإضافة إلى الرسم خلفية اللوحة، فإننا نريدها أن نرسم الوجه والكلمات المعرفة في النداءات المختلفة التي تتم داخل الدالة *paintComponent*. السطر الأول من الدالة *paintComponent* هو استدعاء لـ *super.paintComponent*، والتي تمثل الإصدار الاعتيادي من الدالة *paintComponent* داخل الكلاس *JPanel*، والتي تقوم برسم خلفية اللوحة. وسنقوم دائماً باستخدام هذا السطر كسطر أول في الدالة *paintComponent* .

دعنا الآن نأخذ مثالاً آخر. يحتوي الكلاس *Splat* الموضح في المثال ٣.٣ على دالة *main* تقوم بإنشاء وعرض إطار البرنامج. من حيث المظهر، يقوم هذا البرنامج ببساطة برسم بعض الدوائر الممتلئة. والشيء المثير للاهتمام في هذا البرنامج ليس ما يعمله، بل كيفية عمله - حيث أن كل دائرة يتم رسمها في البرنامج تمثل بكائن خاص بها.

تقوم الدالة *main* بإنشاء كائن من الكلاس *SplatPanel* وتقوم بإضافته إلى الإطار. والكلاس *SplatPanel* موضح من خلال المثال ٤.٣ . وبشكل مشابه للكلاس *SmilingFacePanel* في المثال السابق، فإن الكلاس *SplatPanel* أيضاً مشتق من الكلاس *JPanel*. ويتم فيه توليد بيانات خمس كائنات دائرية داخل الباني.

تقوم الدالة *paintComponent* الخاصة بالكلاس *SplatPanel* برسم اللوحة عن طريق استدعاء دالة *draw* لكل دائرة. وبشكل أساسي، يطلب الكلاس *SplatPanel* من كل دائرة رسم نفسها.

الكلاس *Circle* موضح في المثال ٥.٣ . وهو يقوم بتعريف بيانات لحفظ حجم الدائرة، وموقع (x, y) الخاص بها، ولونها. يتم ضبط هذه القيم باستخدام الباني، ويحتوي الكلاس على كل دوال الإرجاع *accessors* والضبط *mutators* اللازمة. تقوم الدالة *draw* الخاصة بالكلاس *Circle* ببساطة برسم الدائرة اعتماداً على قيم البيانات المحددة.

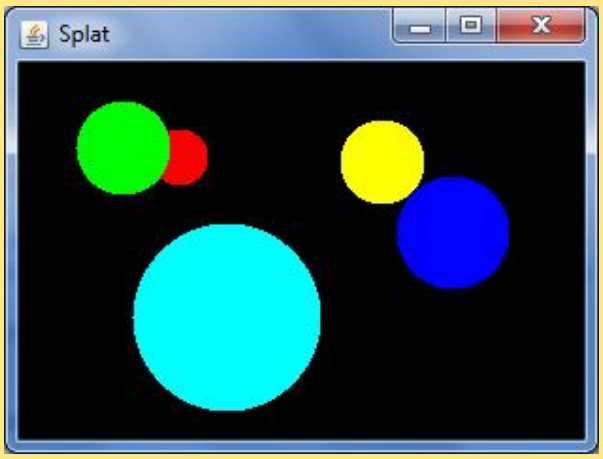
Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

يتضمن برنامج *Splat* فهماً أساسياً للبرمجة كائنيه التوجه *Object-oriented programming*. حيث أن كل دائرة تقوم بإدارة نفسها وتقوم برسم نفسها بأي سياق رسومي تمرره لها. إن أي كائن من الكلاس *Circle* يحتفظ بحالته الخاصة. ونلاحظ أن الكلاس *Circle* معرف بطريقة تمكننا من استخدامه في مواقف أو برامج أخرى.

مثال رقم ٣.٣

```
//********************************************************************
//  Splat.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الكائنات الرسومية
//********************************************************************

import javax.swing.*;
import java.awt.*;

public class Splat
{
   //-----------------------------------------------------------------
   //  تعرض الدالة الرئيسية مجموعة من الدوائر
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Splat");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new SplatPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مثال رقم ٤.٣

```
//********************************************************************
//  SplatPanel.java       : Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الكائنات الرسومية
//********************************************************************

import javax.swing.*;
import java.awt.*;

public class SplatPanel extends JPanel
{
   private Circle circle1, circle2, circle3, circle4, circle5;

   //-----------------------------------------------------------------
   //  الباني: يقوم بإنشاء خمس كائنات دائرية
   //-----------------------------------------------------------------
   public SplatPanel()
   {
      circle1 = new Circle (30, Color.red, 70, 35);
      circle2 = new Circle (50, Color.green, 30, 20);
      circle3 = new Circle (100, Color.cyan, 60, 85);
      circle4 = new Circle (45, Color.yellow, 170, 30);
      circle5 = new Circle (60, Color.blue, 200, 60);

      setPreferredSize (new Dimension(300, 200));
      setBackground (Color.black);
   }

   //-----------------------------------------------------------------
   //  ترسم الدالة التالية هذه اللوحة حيث تطلب من كل دائرة رسم نفسها
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent(page);

      circle1.draw(page);
      circle2.draw(page);
      circle3.draw(page);
      circle4.draw(page);
      circle5.draw(page);
   }
}
```

مثال رقم ٥.٣

```
//********************************************************************
//  Circle.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا المثال دائرة لها موقع، وحجم، ولون مخصص
//********************************************************************

import java.awt.*;

public class Circle
{
   private int diameter, x, y;
   private Color color;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة هذه الدائرة باستخدام القيم المحددة
   //-----------------------------------------------------------------
   public Circle (int size, Color shade, int upperX, int upperY)
   {
      diameter = size;
      color = shade;
      x = upperX;
      y = upperY;
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية برسم هذه الدائرة بالسياق الرسومي المحدد
   //-----------------------------------------------------------------
   public void draw (Graphics page)
   {
      page.setColor (color);
      page.fillOval (x, y, diameter, diameter);
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية بتهيئة قيمة القطر
   //-----------------------------------------------------------------
   public void setDiameter (int size)
   {
      diameter = size;
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية بتهيئة اللون
   //-----------------------------------------------------------------
   public void setColor (Color shade)
   {
      color = shade;
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية بتهيئة قيمة x
   //-----------------------------------------------------------------
   public void setX (int upperX)
   {
      x = upperX;
   }
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
    //-----------------------------------------------------------------
    //  تقوم الدالة التالية بتهيئة قيمة y
    //-----------------------------------------------------------------
    public void setY (int upperY)
    {
       y = upperY;
    }

    //-----------------------------------------------------------------
    //  تقوم الدالة التالية بإرجاع قيمة القطر
    //-----------------------------------------------------------------
    public int getDiameter ()
    {
       return diameter;
    }

    //-----------------------------------------------------------------
    //  تقوم الدالة التالية بإرجاع قيمة اللون
    //-----------------------------------------------------------------
    public Color getColor ()
    {
       return color;
    }

    //-----------------------------------------------------------------
    //  تقوم الدالة التالية بإرجاع قيمة x
    //-----------------------------------------------------------------
    public int getX ()
    {
       return x;
    }

    //-----------------------------------------------------------------
    //  تقوم الدالة التالية بإرجاع قيمة y
    //-----------------------------------------------------------------
    public int getY ()
    {
       return y;
    }
}
```

٢.٣ واجهات المستخدم الرسومية GRAPHICAL USER INTERFACE

في الفصلين الأول والثاني تكلمنا عن بعض المكونات الأساسية المساعدة في تصميم البرامج المعتمدة على الرسومات. وما نحتاجه الآن هو التفاعل الحقيقي من قبل المستخدم، وهذا هو قلب "واجهة المستخدم الرسومية *"Graphical User Interface "GUI"*. وسنقوم في هذا الفصل بالتقديم للمفاهيم التي نحتاجها لتوليد برامج تفاعلية معتمدة على الجوئي. ويعتبر هذا الفصل أرضية عمل ستبنى عليها نقاشات الجوئي في الفصول اللاحقة.

نحتاج على الأقل إلى ثلاثة أنواع من الكائنات لإنشاء واجهة مستخدم رسومية باستخدام لغة جافا:

- المكونات *components*.
- الأحداث *events*.
- المستمعات *listeners*.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

كما ذكرنا في الفصل الثاني، فإن مكون الجوئي *GUI component* هو كائن يقوم بتعريف عنصر شاشة لعرض معلومة أو للسماح للمستخدم بالتفاعل مع البرنامج بطريقة معينة. ومن الأمثلة على مكونات الجوئي أزرار الضغط *push buttons*، والحقول النصية *text fields*، والمعنونات *labels*، وأشرطة التمرير *scroll bars* والقوائم *menus*. ويعد الحاوي *container* نوعاً خاصاً من المكونات يستخدم لاحتواء وتنظيم مكونات أخرى. وقد قمنا سابقاً باستخدام حاويات مثل الإطارات *frames* واللوحات *panels*. وكذلك تطرقنا إلى استخدام المعنونات *labels*.

أما الحدث *event* فهو عبارة عن كائن يمثل حدوث شيء ما نهتم به. غالباً، ترتبط الأحداث *events* بأفعال المستخدم *user actions*، مثل ضغط زر الماوس، أو ضغط زر في لوحة المفاتيح. تقوم معظم مكونات الجوئي بتوليد أحداث تدل على فعل معين من المستخدم ذو علاقة بالمكون. مثلاً، مكون الزر *button component* يقوم بتوليد حدث يشير إلى أنه تم ضغطه. يسمى البرنامج المصمم بالجوئي، والذي يستجيب لأحداث المستخدم بالبرنامج المتفاعل مع الأحداث *event-driven*.

مفهوم أساسي Key Concept

يتكون الجوئي من مكونات، وأحداث تمثل حركات المستخدم ومستمعات تستجيب لهذه الأحداث.

فيما يخص المستمع *listener* فهو عبارة عن كائن "ينتظر" حدوث حدث معين *event* ويستجيب لهذا الحدث بطريقةٍ ما. ويتوجب علينا أن نقوم بعناية فائقة بفهم العلاقات بين المستمع *listener*، والحدث الذي يستمع إليه *event*، والمكون *component* الذي يولد الحدث.

وسنقوم في أغلب الحالات باستخدام مكونات وأحداث معرفة مسبقاً *predefined* بواسطة الكلاسات الموجودة في مكتبة كلاسات جافا. وسوف نقوم بتكييف سلوك المكونات، ولكن مع إدراك أن دورهم الأساسي قد تم تأسيسه. مع ذلك، سوف نقوم بكتابة كلاسات المستمع *listener classes* لإنجاز الأفعال التي نريد ظهورها عند حدوث الإحداث.

بشكلٍ خاص، يتوجب علينا عند إنشاء برنامج جافا يستخدم الجوئي ما يلي:

- توليد وتهيئة المكونات الضرورية،
- تنفيذ كلاسات المستمع *listener classes* التي تقوم بتعريف ما الذي يحدث عند حدوث حدثٍ ما، و
- تأسيس العلاقة بين المستمعات والمكونات التي تولد الأحداث التي نهتم بها.

في بعض الأحيان، عندما يكون لديك الفهم الأساسي للبرمجة المتفاعلة مع الأحداث، فإن ما يتبقى هو فقط تفاصيل. هناك العديد من أنوع المكونات التي يمكنك استخدامها والتي تولد أنواعاً متعددة من الأحداث التي قد تريد تعلمها، ولكنها كلها تعمل بنفس الطريقة الأساسية. فكلها تمتلك نفس العلاقات الأساسية مع بعضها البعض.

وسنقوم من خلال الأقسام التالية بتوضيح بعض المكونات الإضافية ونقدم أمثلةً على برامج معتمدة على الجوئي تسمح بتفاعل حقيقي مع المستخدم.

٣.٣ الأزرار BUTTONS

يقدم برنامج الـ *PushCounter* الموجود في المثال ٦.٣ للمستخدم زر ضغط وحيد (معنون بالعبارة "*Push Me!*"). وفي كل مرة يتم فيها ضغط الزر، يتم تحديث قيمة العداد *counter* من ثم عرضها. تتضمن المكونات المستخدمة في هذا البرنامج الزر *button*، والمعنون *label* المستخدم لعرض العدد، واللوحة *panel* المستخدمة لتنظيم الجوئي، والإطار *frame* المستخدم لعرض اللوحة. ويتم تعريف اللوحة من خلال الكلاس *PushCounterPanel*، والموضح في المثال ٧.٣.

مثال رقم ٦.٣

```
//********************************************************************
//  PushCounter.java       المؤلف: Lewis/Loftus    المترجم : Almashraqi
//
//  يشرح هذا المثال واجهة مستخدم رسومية ومستمع حدث
//********************************************************************

import javax.swing.JFrame;
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
public class PushCounter
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء الإطار الرئيسي للبرنامج
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Push Counter");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new PushCounterPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display

مثال رقم ٧.٣

```
//********************************************************************
//  PushCounterPanel.java   المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال واجهة مستخدم رسومية ومستمع حدث
//********************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PushCounterPanel extends JPanel
{
   private int count;
   private JButton push;
   private JLabel label;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة الجوئي
   //-----------------------------------------------------------------
   public PushCounterPanel ()
   {
      count = 0;

      push = new JButton ("Push Me!");
      push.addActionListener (new ButtonListener());

      label = new JLabel ("Pushes: " + count);

      add (push);
      add (label);
```

```
      setPreferredSize (new Dimension(300, 40));
      setBackground (Color.cyan);
   }

   //*****************************************************************
   //  الكلاس التالي يمثل مستمعاً للأحداث الفعلية للزر
   //*****************************************************************
   private class ButtonListener implements ActionListener
   {
      //--------------------------------------------------------------
      // الدالة التالية تقوم بتحديث قيمة العداد والمعنون عندما يضغط
       //المستخدم على الزر
      //--------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         count++;
         label.setText("Pushes: " + count);
      }
   }
}
```

يعتبر زر الضغط *push button* مكوناً يسمح للمستخدم بتوليد حدث عند الضغط عليه بزر الماوس. وهناك أنواع أخرى من المكونات الزرية *button components* سنتعرف عليها في الفصول التالية. يتم تعريف زر الضغط من خلال الكلاس *JButton*.

يقوم باني الكلاس *PushCounterPanel* بتهيئة الجوئي. حيث نلاحظ أن استدعاء باني الكلاس *JButton* يأخذ وسيطاً من نوع *String* يحدد النص الذي يعرض على الزر. يتم في هذا الباني أيضاً إضافة الزر والمعنون إلى اللوحة.

إن الحدث الوحيد الذي نهتم به في البرنامج هو حدث الضغط على الزر بالماوس وللاستجابة لهذا الحدث، يجب إنشاء كائن مستمع *listener object* لهذا الحدث، لذلك يجب علينا كتابة كلاس يمثل المستمع.

يقوم الكلاس *JButton* بتوليد حدث فعلي *action event* عند الضغط عليه. لذلك فإن الكلاس المستمع الذي سوف نكتبه سيكون مستمع حدث فعلي *action event listener*. في هذا البرنامج، قمنا بتعريف كلاس يدعى *ButtonListener* لتمثيل المستمع لهذا الحدث.

مفهوم أساسي Key Concept

تعرف المستمعات غالباً ككلاسات داخلية وذلك بسبب العلاقة الجوهرية بين المستمع ومكونات الجوئي.

يمكننا كتابة الكلاس *ButtonListener* في ملف خاص به، أو حتى في نفس الملف لكن خارج الكلاس *PushCounterPanel*. مع ذلك، فإنه يتوجب علينا تهيئة طريقة الاتصال بين المستمع ومكونات الجوئي الذي يقوم هذا المستمع بتحديثها. ولذلك فسنقوم بتعريف الكلاس *ButtonListener* ككلاس داخلي *inner class*، والذي هو عبارة عن كلاس معرف داخل كلاس آخر. وعليه، فإن هذا الكلاس الداخلي يتمكن تلقائياً من الوصول إلى أعضاء الكلاس الذي يحويه. وعليك أن تعرف أنه يتوجب إنشاء كلاسات داخلية فقط في المواقف التي توجد فيها علاقة وثيقة بين الكلاسين وأيضاً في حالة أننا لا نريد الوصول إلى الكلاس الداخلي من كلاس آخر. إن العلاقة بين المستمع ومكونات الجوئي الخاصة به هي واحدة من الحالات القليلة التي يعتبر فيها الكلاس الداخلي مناسباً.

يتم كتابة الكلاسات المستمعة عن طريق تنفيذ واجهة *interface*، والتي هي عبارة عن قائمة من الدوال التي يتوجب على الكلاس المنفّذ تعريفها. تحتوي مكتبة كلاسات جافا القياسية على واجهات لأنواع عديدة من الأحداث. يتم توليد مستمع الفعل *action listener* عن طريق تنفيذ الواجهة المسماة *ActionListener*، لذلك نقوم بتضمين عبارة *implement* في الكلاس *ButtonListener*. وسيتم مناقشة الواجهات *interfaces* بتفصيل أكثر في الفصل الخامس.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الدالة الوحيدة الموجودة داخل الواجهة *ActionListener* هي الدالة المسماة *actionPerformed*، ولذلك فهي الدالة الوحيدة التي يتوجب على الكلاس *ButtonListener* تنفيذها. سيقوم المكون الذي يولد الحدث (في هذه الحالة الزر) باستدعاء الدالة *actionPerformed* عند ظهور الحدث، ويتم تمرير كائن من الكلاس *ActionEvent* الذي يمثل الحدث. في بعض الأحيان سنستخدم كائن الحدث، وأحياناً أخرى يكون ببساطة كافياً أن الحدث ظهر فعلاً . في هذه الحالة لن نكون بحاجة للتفاعل مع كائن الحدث. وعند ظهور الحدث، يقوم المستمع بزيادة المتغير *count* بمقدار واحد و يعيد ضبط نص المعنون باستخدام الدالة *setText*.

تذكر أننا لسنا بحاجة إلى إنشاء مستمع للحدث فحسب، بل يجب علينا أيضاً تهيئة العلاقة بين المستمع والمكون الذي سيولد الحدث. ولعمل ذلك، نقوم بإضافة المستمع إلى المكون عن طريق استدعاء الدالة المناسبة. حيث نقوم في الباني الخاص بالكلاس *PushCounterPanel* بنداء الدالة *addActionListener*، والتي نمرر لها كائن جديد من الكلاس *ButtonListener*.

قم بمراجعة هذا المثال بشكل جيد، ولاحظ كيف تتم الثلاث الخطوات الرئيسية لإنشاء برنامج تفاعلي معتمد على الجوئي. فهذا المثال يقوم بإنشاء وتهيئة مكونات الجوئي، وكذلك إنشاء المستمع المناسب للحدث الذي نريده، و أيضاً تهيئة العلاقة بين المستمع والمكون الذي يولد الحدث.

٤.٣ الحقول النصية TEXT FIELDS

دعنا الآن ننظر في مثال آخر يستخدم مكوناً آخر: الحقل النصي *text field*. يوضح المثال ٨.٢ البرنامج المسمى *Fahrenheit* والذي يتضمن حقلاً نصياً يتمكن المستخدم من خلاله من إدخال الدرجة الفهرنهايتية. وعندما يقوم المستخدم بضغط المفتاح *Enter* (أو *RETURN*)، فسيتم عرض الدرجة المئوية *Celsius* المكافئة.

ويتم تهيئة الواجهة الخاصة بالبرنامج *Fahrenheit* من خلال الكلاس *FahrenheitPanel*. يعتبر الحقل النصي كائناً من الكلاس *JTextField*. ويستقبل الباني الخاص بالكلاس *JTextField* وسيطاً يحمل قيمة صحيحة تحدد حجم الحقل عن طريق عدد الحروف اعتماداً على نوع الخط الافتراضي الحالي.

تم إضافة حقل نصي ومعنونات مختلفة إلى اللوحة من أجل عرضها. تذكر إن اللوحة يتم التحكم بها عن طريق مدير التخطيط *layout manager* المسمى *Flow layout*، والذي يعمل على وضع أكبر قدر ممكن من المكونات في سطر واحد. لذلك فإنك إذا قمت بتغيير حجم الإطار، فقد يتم تغيير اتجاه بعض المعنونات والحقل النصي. وسنقوم بدراسة أكثر تفصيلاً لمدراء التخطيط في الفصل الخامس، مما يتيح خيارات أكثر للتحكم بتخطيط المكونات.

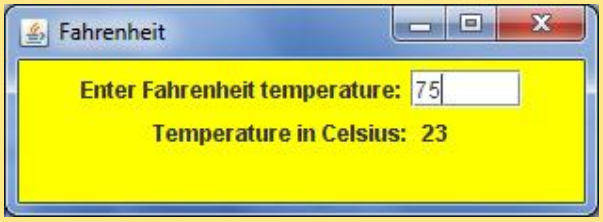
مثال رقم ٨.٣

```
//********************************************************************
//  Fahrenheit.java       المؤلف : Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الحقول النصية
//********************************************************************

import javax.swing.JFrame;

public class Fahrenheit
{
   //-----------------------------------------------------------------
   // تقوم الدالة الرئيسية بإنشاء وعرض الجوئي الخاص بمحول درجة الحرارة
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Fahrenheit");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      FahrenheitPanel panel = new FahrenheitPanel();
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مثال رقم ٩.٣

```
//********************************************************************
//  FahrenheitPanel.java  المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الحقول النصية
//********************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FahrenheitPanel extends JPanel
{
   private JLabel inputLabel, outputLabel, resultLabel;
   private JTextField fahrenheit;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة مكونات الجوئي الأساسية
   //-----------------------------------------------------------------
   public FahrenheitPanel()
   {
      inputLabel = new JLabel ("Enter Fahrenheit temperature:");
      outputLabel = new JLabel ("Temperature in Celsius: ");
      resultLabel = new JLabel ("---");

      fahrenheit = new JTextField (5);
      fahrenheit.addActionListener (new TempListener());

      add (inputLabel);
      add (fahrenheit);
      add (outputLabel);
      add (resultLabel);

      setPreferredSize (new Dimension(300, 75));
      setBackground (Color.yellow);
   }
```

```
   //*****************************************************************
   //  يمثل الكلاس التالي مستمع حدث لحقل إدخال درجة الحرارة
   //*****************************************************************
   private class TempListener implements ActionListener
   {
      //-----------------------------------------------------------
      //  تقوم الدالة التالية بإتمام التحويل عندما يتم الضغط على مفتاح
      //  الإدخال في الحقل النصي
      //-----------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         int fahrenheitTemp, celsiusTemp;

         String text = fahrenheit.getText();

         fahrenheitTemp = Integer.parseInt (text);
         celsiusTemp = (fahrenheitTemp-32) * 5/9;

         resultLabel.setText (Integer.toString (celsiusTemp));
      }
   }
}
```

إذا كان المؤشر حالياً موجود في الحقل النصي، فإنه يقوم بتوليد حدث فعلي عند ضغط المفتاح *Enter* أو *Return*. لذلك فنحن بحاجة لتهيئة كائن مستمع للاستجابة لهذا الحدث. وكما عملنا في برنامج *PushCounter* في القسم السابق، سنقوم هنا بتعريف مستمع كصف داخلي ينفذ الواجهة المسماة *ActionListener*.

يقوم مكون الحقل النصي باستدعاء الدالة *actionPerformed* عندما يقوم المستخدم بالضغط على المفتاح *Enter*. تقوم الدالة أولاً بجلب النص من الحقل النصي عن طريق استدعاء الدالة *getText*، والتي ترجع سلسلة نصية. يتم بعد ذلك تحويل النص إلى عدد صحيح باستخدام الدالة *parseInt* الموجودة في الكلاس المغلف المسمى *Integer*. ثم تقوم الدالة بإتمام العملية الحسابية التي من خلالها يتم تحديد الدرجة المئوية المكافئة وضبط النص للمعنون المناسب على هذه النتيجة.

لاحظ إن زر الضغط *push button* والحقل النصي *text field* يولدان نفس نوع الحدث: وهو الحدث الفعلي *action event*. لذلك فان البدائل التصميمية لبرنامج *Fahrenheit* هي إضافة كائن *JButton* إلى الجوئي يعمل على إتمام عملية التحويل عند الضغط بزر الماوس عليه. ولإتمام هذا الأمر، يمكن استخدام نفس كائن الاستماع للاستماع إلى مكونات متعددة في الوقت نفسه. ولذلك فإنه بإمكاننا إضافة المستمع إلى كل من الحقل النصي والزر، معطيين بهذا حق الاختيار للمستخدم. حيث إن ضغط الزر أو المفتاح *Enter* سيسببان في إتمام عملية التحويل المطلوبة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الفصل الرابع

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

تعمل العبارات الشرطية *conditionals* و الحلقات *loops* على تحسين قدرتنا على توليد رسومات جذابة بشكل كبير جداً.

يرسم البرنامج *Bullseye* الموضح في المثال ١.٤ هدفاً *target*. ويظهر الرسم فعلياً في الكلاس المسمى *BullseyePanel* الموضح في المثال ٢.٤ . تستخدم الدالة *paintComponent* الموجودة في الكلاس *BullseyePanel* جملة *if* للتبديل بين اللونين الأبيض والأسود.

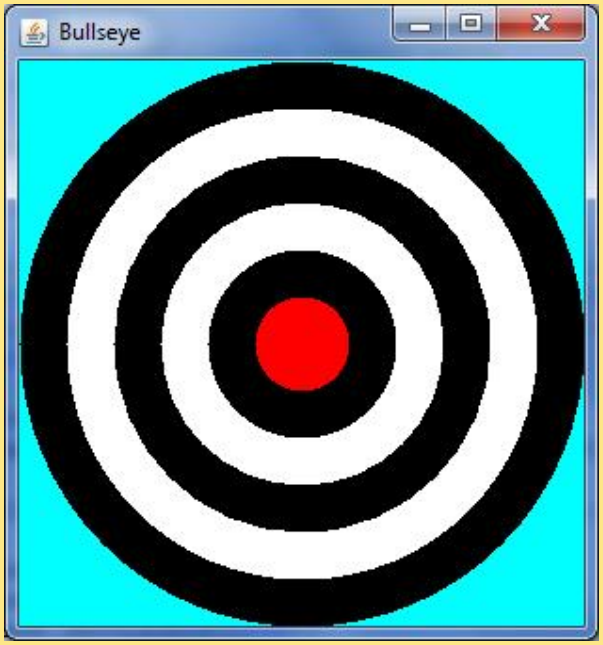
لاحظ أن كل حلقة *ring* ترسم فعلياً كدائرة ممتلئة *filled circle* (والدائرة عبارة عن شكل بيضاوي *oval* يتساوى فيه العرض مع الطول). ولأننا نقوم برسم الدوائر فوق بعضها، فإن الدوائر الداخلية تغطي الجزء الداخلي من الدوائر الكبيرة، مولدة بذلك الشكل الحلقي. وفي النهاية، يتم رسم الدائرة الحمراء النهائية والتي تمثل قلب الهدف.

دعنا نرى مثالاً آخر. يوضح المثال ٣.٤ الكلاس *Boxes*، والذي يعمل على تعريف كائن من الكلاس *BoxesPanel*، والموضح في المثال ٤.٤ . إن الهدف من هذا البرنامج هو رسم مستطيلات عديدة ذات حجم عشوائي وفي مواقع عشوائية. إذا كان عرض المستطيل أقل من سمك محدد (وهو ٥ بكسل)، فإنه يتم ملء هذا المستطيل باللون الأصفر. وإذا كان ارتفاع المستطيل أقل من نفس السمك المحدد، فيتم ملئه باللون الأخضر. فيما عدا ذلك، يرسم المستطيل بدون تعبئة وباللون الأبيض.

مثال رقم ١.٤

```
//********************************************************************
//  Bullseye.java       المؤلف : Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الحلقات للرسم
//********************************************************************

import javax.swing.JFrame;

public class Bullseye
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء الإطار الأساسي للبرنامج
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Bullseye");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      BullseyePanel panel = new BullseyePanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



مثال رقم ٢.٤

```
//********************************************************************
//  BullseyePanel.java      المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الجمل الشرطية والحلقات لإتمام الرسم
//********************************************************************

import javax.swing.JPanel;
import java.awt.*;

public class BullseyePanel extends JPanel
{
   private final int MAX_WIDTH = 300, NUM_RINGS = 5, RING_WIDTH = 25;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة لوحة مربع الرمي
   //-----------------------------------------------------------------
   public BullseyePanel ()
   {
      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300,300));
   }
```

```
   //-----------------------------------------------------------------
   //  تقوم الدالة التالية برسم دوائر مربع الرمي
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);

      int x = 0, y = 0, diameter = MAX_WIDTH;

      page.setColor (Color.white);

      for (int count = 0; count < NUM_RINGS; count++)
      {
         if (page.getColor() == Color.black)  // تبديل الألوان
            page.setColor (Color.white);
         else
            page.setColor (Color.black);

         page.fillOval (x, y, diameter, diameter);

         diameter -= (2 * RING_WIDTH);
         x += RING_WIDTH;
         y += RING_WIDTH;
      }

      // رسم الدائرة الحمراء التي في المركز
      page.setColor (Color.red);
      page.fillOval (x, y, diameter, diameter);
   }
}
```

٢.٤ تحديد مصادر الحدث DETERMINING EVENT SOURCES

بدأنا في الفصل الثالث باستكشاف عملية توليد برامج ذات واجهة مستخدم رسومية تفاعلية حقيقية (*GUI* أو جوئي). وستتذكر أن الجوئي التفاعلية تتطلب أن نقوم بإنشاء كائنات مستمعة *listener objects* وتهيئة العلاقة بين المستمعات *listeners*، والمكونات *components* التي تولد الأحداث المطلوبة.

دعنا الآن نرى مثالاً يتم فيه استخدام كائن مستمع واحد للاستماع لمكونين مختلفين. هذا البرنامج ممثل في الكلاس *LeftRight*، والموضح في المثال ٥.٤، والذي يعرض معنوناً واحداً *1 label* وزرين *2. buttons* عندما يتم الضغط على الزر الأيسر *left button*، يقوم المعنون بعرض كلمة *Left*، وعندما يتم الضغط على الزر الأيمن *right button*، يقوم المعنون بعرض كلمة *Right*.

أما الكلاس *LeftRightPanel*، الموضح في المثال ٦.٤، فيعمل على إنشاء كائن من الكلاس *ButtonListener*، ثم يضيف هذا المستمع إلى كلا الزرين. ولذلك، فإنه عند ضغط أي من الزرين، يتم استدعاء الدالة *actionPerformed* التابعة للكلاس *ButtonListener*.

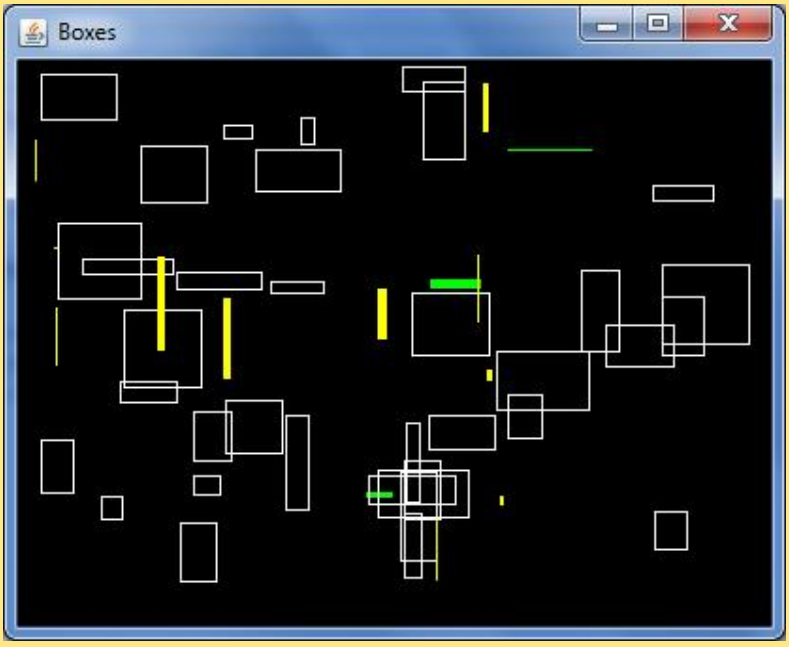
في كل عمليه استدعاء لها، تستخدم الدالة *actionPerformed* عبارة *if – else* لتحديد أياً من الزرين هو الذي قام بتوليد الحدث. ويتم استدعاء الدالة *getSource* عن طريق كائن من الكلاس *ActionEvent* والذي مرره الزر إلى الدالة *actionPerformed*. وترجع الدالة *getSource* عنوان المكون الذي قام بتوليد الحدث. ويقوم الشرط في عبارة *if* بمقارنة مصدر الحدث *event source* مع عنوان الزر الأيسر. إذا لم يتطابقا، فإن الحدث سيكون حتماً قد تولد بواسطة الزر الأيمن.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مثال رقم ٣.٤

```
//********************************************************************
//  Boxes.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الحلقات للرسم
//********************************************************************

import javax.swing.JFrame;

public class Boxes
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء الإطار الأساسي للمكونات
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Boxes");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      BoxesPanel panel = new BoxesPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مثال رقم ٤.٤

```
//*******************************************************************
//  BoxesPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام الجمل الشرطية والحلقات لإتمام الرسم
//*******************************************************************

import javax.swing.JPanel;
import java.awt.*;
import java.util.Random;

public class BoxesPanel extends JPanel
{
   private final int NUM_BOXES = 50, THICKNESS = 5, MAX_SIDE = 50;
   private final int MAX_X = 350, MAX_Y = 250;
   private Random generator;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة لوحة الرسم
   //-----------------------------------------------------------------
   public BoxesPanel ()
   {
      generator = new Random();

      setBackground (Color.black);
      setPreferredSize (new Dimension(400, 300));
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية برسم مربعات ذات عرض وارتفاع عشوائي
   //  في مواقع مختلفة
   //  وقد تم تعليم المربعات الضيقة أو القصيرة عن طريق ملئها باللون
   //-----------------------------------------------------------------
   public void paintComponent(Graphics page)
   {
      super.paintComponent (page);

      int x, y, width, height;

      for (int count = 0; count < NUM_BOXES; count++)
      {
         x = generator.nextInt(MAX_X) + 1;
         y = generator.nextInt(MAX_Y) + 1;

         width = generator.nextInt(MAX_SIDE) + 1;
         height = generator.nextInt(MAX_SIDE) + 1;

         if (width <= THICKNESS)  // فحص المربعات الضيقة
         {
            page.setColor (Color.yellow);
            page.fillRect (x, y, width, height);
         }
         else
            if (height <= THICKNESS)  // فحص المربعات القصيرة
            {
               page.setColor (Color.green);
               page.fillRect (x, y, width, height);
            }
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
            else
            {
               page.setColor (Color.white);
               page.drawRect (x, y, width, height);
            }
         }
      }
   }
```

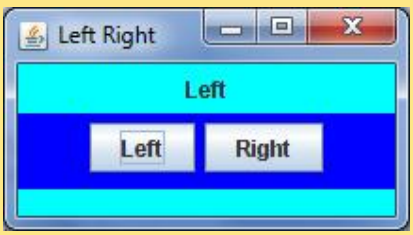
كان بإمكاننا أن نقوم بإنشاء كلاسين مستمعين منفصلين، أحدهما للاستماع إلى الزر الأيسر والآخر للاستماع إلى الزر الأيمن. وفي هذه الحالة لن نكون محتاجين في الدالة *actionPerformed* أن نقوم بتحديد مصدر الحدث. إن اختيارنا لخيار المستمعات المتعددة أو لخيار تحديد مصدر الحدث عند ظهوره يعتبر قراراً تصميمياً يجب اتخاذه بناءاً على الوضع الذي يتم معالجته.

لاحظ أن الزرين وضعا في نفس اللوحة المسماة *buttonPanel*، وهي لوحة منفصلة عن اللوحة الممثلة في الكلاس *LeftRightPanel*. أن وضعنا لكلا الزرين في لوحة واحدة، يضمن العلاقة المرئية لهما حتى لو تم تغيير حجم الإطار بطرق متنوعة. ويعتبر ذلك مهماً بالنسبة للزرين المعنونين بـ *Left* و *Right*.

مثال رقم ٥.٤

```
//********************************************************************
//  LeftRight.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام مستمع واحد لعدة أزرار
//********************************************************************

import javax.swing.JFrame;
public class LeftRight
{
   //-----------------------------------------------------------------
   //  الدالة الرئيسية تقوم بإنشاء الإطار الأساسي للبرنامج
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Left Right");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new LeftRightPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مثال رقم ٦.٤

```
//********************************************************************
//LeftRightPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام مستمع واحد لعدة أزرار
//********************************************************************
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LeftRightPanel extends JPanel
{
   private JButton left, right;
   private JLabel label;
   private JPanel buttonPanel;
   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة الجوئي
   //-----------------------------------------------------------------
   public LeftRightPanel ()
   {
      left = new JButton ("Left");
      right = new JButton ("Right");
      ButtonListener listener = new ButtonListener();
      left.addActionListener (listener);
      right.addActionListener (listener);

      label = new JLabel ("Push a button");
      buttonPanel = new JPanel();
      buttonPanel.setPreferredSize (new Dimension(200, 40));
      buttonPanel.setBackground (Color.blue);
      buttonPanel.add (left);
      buttonPanel.add (right);

      setPreferredSize (new Dimension(200, 80));
      setBackground (Color.cyan);
      add (label);
      add (buttonPanel);
   }
   //*****************************************************************
   //  الكلاس التالي يمثل مستمعاً لكلا الزرين
   //*****************************************************************
   private class ButtonListener implements ActionListener
   {
      //--------------------------------------------------------------
      //  الدالة التالية تحدد أي من الأزرار تم ضغطه وتضبط نص المعنون
      //  وفقاً لذلك
      //--------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         if (event.getSource() == left)
            label.setText("Left");
         else
            label.setText("Right");
      }
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

إن المكون المسمى بالمربع الحواري *dialog box* يمكن أن يكون مفيداً في المساعدة في معالجة الجوئي. ويعتبر المربع الحواري نافذة رسومية تظهر فوق كل النوافذ النشطة بحيث يتمكن المستخدم من التفاعل معها. ويمكن أن يخدمنا المربع الحواري في تحقيق عدة أغراض، مثل إرسال بعض المعلومات، والتأكيد على فعل ما، أو السماح للمستخدم بإدخال بعض المعلومات. عادة ما يكون الغرض من المربع الحواري وحيداً، ويكون تفاعل المستخدم معه مختصراً.

تحتوي الحزمة *Swing* في مكتبة كلاسات جافا على كلاس يسمى *JOptionPane* يعمل على تبسيط إنشاء واستخدام المربعات الحوارية الأساسية. ويسرد الشكل ١.٤ بعضاً من الدوال الموجودة داخل الكلاس *JOptionPane*.

يمكن تصنيف الأشكال الأساسية لمربعات *JOptionPane* الحوارية إلى ثلاثة أصناف. مربع الرسالة الحواري *message dialog box* والذي يقوم ببساطة بعرض مخرجات نصية *output string*. ومربع الإدخال الحواري *input dialog box* والذي يمثل رسالة حث *prompt* وحقلاً نصياً *text field* للإدخال يقوم فيه المستخدم بإدخال بيانات نصية. ومربع التأكيد الحواري *confirm dialog box* والذي يعرض للمستخدم سؤال نعم/لا *yes-or-no question*.

دعنا الآن نرى برنامجاً يستخدم كل هذه الأنواع من المربعات الحوارية. يوضح المثال ٧.٤ برنامجاً يظهر في البداية مربع إدخال حواري للمستخدم يطلب من المستخدم إدخال عدد صحيح. وبعد أن يقوم المستخدم بضغط الزر *OK* في مربع الإدخال الحواري، سيظهر مربع حواري ثاني (في هذه المرة مربع رسالة حواري)، وذلك لإعلام المستخدم فيما إذا كان الرقم المدخل زوجياً *even* أم فردياً *odd*. وبعد أن يترك المستخدم هذا المربع، يظهر مربع حواري ثالث، وذلك من أجل تحديد فيما إذا كان المستخدم يريد اختبار رقم آخر أم لا. إذا قام المستخدم بضغط الزر *YES*، فسيتم تكرار ظهور سلسلة المربعات الحوارية. وفيما عدا ذلك يتم إنهاء البرنامج.

```
static String showInputDialog(Object msg)
```
تعرض مربعاً حوارياً يحتوي على الرسالة المحددة وحقل إدخال نصي. ويتم إرجاع محتويات الحقل النصي.

```
static int showConfirmDialog(Component parent, Object msg)
```
تعرض مربعاً حوارياً يحتوي على الرسالة المحددة وخيارات زرية من نوع نعم / لا. إذا كان المكون الأب null، فهذا يعني أن المربع سيظهر في وسط الشاشة.

```
static void showMessageDialog(Component parent, Object msg)
```
تعرض مربعاً حوارياً يحتوي على الرسالة المحددة. إذا كان المكون الأب null، فهذا يعني أن المربع سيظهر في وسط الشاشة.

الشكل رقم ١.٤ بعض دوال الكلاس JOptionPane

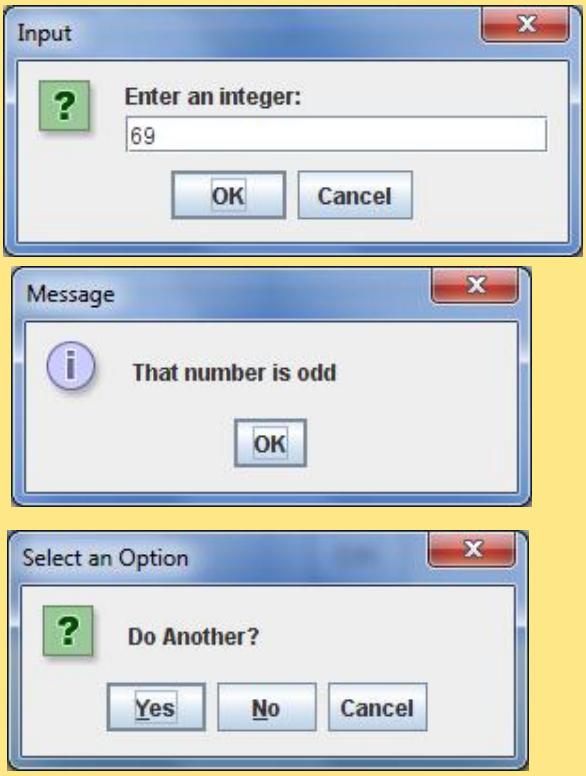
مثال رقم ٧.٤

```
//*************************************************************
//  EvenOdd.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  JOptionPane يشرح هذا المثال استخدام الكلاس
//*************************************************************

import javax.swing.JOptionPane;
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
public class EvenOdd
{
   //-----------------------------------------------------------------
   //  تحدد الدالة الرئيسية إذا كانت القيمة المدخلة من قبل المستخدم
   //  زوجية أم فردية وذلك باستخدام عدة مربعات حوارية للتفاعل
   //  مع المستخدم
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String numStr, result;
      int num, again;

      do
      {
         numStr = JOptionPane.showInputDialog ("Enter an integer: ");

         num = Integer.parseInt(numStr);

         result = "That number is " + ((num%2 == 0) ? "even" : "odd");

         JOptionPane.showMessageDialog (null, result);

         again = JOptionPane.showConfirmDialog (null, "Do Another?");
      }
      while (again == JOptionPane.YES_OPTION);
   }
}
```

العرض Display



يعمل الوسيط الأول في الدالتين *showMessageDialog* و *showConfirmDialog* على تحديد المكون الأعلى الذي يتحكم بالمربع الحواري. واستخدام المرجع *null* في هذا الوسيط يسبب في ظهور المربع الحواري في وسط الشاشة تماماً.

هناك العديد من دوال الكلاس *JOptionPane* التي تسمح للمبرمج بتكييف مكونات المربع الحواري وفق الحاجة. بالإضافة إلى ذلك، يمكن استخدام الدالة *showOptionDialog* لإنشاء مربعات حوارية فيها خصائص مركبة من الثلاثة الأشكال الأساسية وذلك بغرض إضفاء تفاعلات أكثر تعقيداً.

٤.٤ المزيد من المكونات الزرية MORE BUTTON COMPONENTS

يعتبر زر الضغط *push button* المعرف بواسطة الكلاس *JButton* فقط أحد أنواع الأزرار التي يمكن استخدمها مع جوئي الجافا. وهناك نوعان آخران هما مربعات الاختيار *check boxes* وأزرار الراديو *radio buttons*. وسنقوم باستعراض هذين المكونين في هذا القسم بالتفصيل.

مربعات الاختيار Check Boxes

يمكن تعريف مربع الاختيار *check box* بأنه زر يمكن تبديل حالته بين قيمتين هما *off* و *on* باستخدام الماوس، وذلك في إشارة إلى تحقق شرط منطقي معين أو عدم تحققه. مثلاً، قد نستخدم مربع اختيار عنوانه *Collate* ليوضح فيما إذا كان يجب مقارنة مخرجات عمل طباعي. وبالرغم من أنه يمكن أن يكون لدينا مجموعة من مربعات الاختيار تدل على مجموعة خيارات، فإن أياً من هذه المربعات سيعمل بشكل مستقل، أي أنه يمكن ضبط حالة أي مربع على *on* أو *off* بدون أن تؤثر حالته على المربعات الأخرى.

يعرض البرنامج في المثال ٨.٤ اثنين من مربعات الاختيار ومعنون واحد. تحدد مربعات الاختيار فيما إذا كان سيتم عرض العنوان بخط عريض *bold* أو مائل *italic*، أو كليهما، أو بدون أي منهما، وتعتبر أي تركيبة للعريض *bold* والمائل *italic* صحيحة. مثلاً يمكن ضبط كلا المربعين على الحالة (*on*)، في حالة أردنا عرض النص بخط عريض ومائل في آن واحد. أما إذا لم يتم اختيار أياً منهما، فسيتم عرض نص العنوان كنص اعتيادي *plain text*.

مثال رقم ٨.٤

```
//*************************************************************
//  StyleOptions.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام مربعات الاختيار
//*************************************************************
import javax.swing.JFrame;
public class StyleOptions
{
   //-----------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء وعرض إطار البرنامج
   //-----------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Style Options");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);



      StyleOptionsPanel panel = new StyleOptionsPanel();
      frame.getContentPane().add (panel);

      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مثال رقم ٩.٤

```
//********************************************************************
//  StyleOptionsPanel.java  المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام مربعات الاختيار
//********************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class StyleOptionsPanel extends JPanel
{
   private JLabel saying;
   private JCheckBox bold, italic;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة اللوحة باستخدام معنون وبعض مربعات الاختيار
   //  التي تتحكم في نمط خط المعنون
   //-----------------------------------------------------------------
   public StyleOptionsPanel()
   {
      saying = new JLabel ("Say it with style!");
      saying.setFont (new Font ("Helvetica", Font.PLAIN, 36));

      bold = new JCheckBox ("Bold");
      bold.setBackground (Color.cyan);
      italic = new JCheckBox ("Italic");
      italic.setBackground (Color.cyan);

      StyleListener listener = new StyleListener();
      bold.addItemListener (listener);
      italic.addItemListener (listener);



      add (saying);
      add (bold);
      add (italic);

      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300, 100));
   }
```

```
   //*************************************************************
   //  الكلاس التالي يمثل مستمعاً لمربعي الاختيار كليهما
   //*************************************************************
   private class StyleListener implements ItemListener
   {
      //-------------------------------------------------------------
      //  الدالة التالية تقوم بتحديث نمط الخط للمعنون
      //-------------------------------------------------------------
      public void itemStateChanged (ItemEvent event)
      {
         int style = Font.PLAIN;

         if (bold.isSelected())
            style = Font.BOLD;

         if (italic.isSelected())
            style += Font.ITALIC;

         saying.setFont (new Font ("Helvetica", style, 36));
      }
   }
}
```

تم تجسيد الجوئي الخاصة بالبرنامج *StyleOptions* في الكلاس المسمى *StyleOptionsPanel* الموضح في المثال ٩.٤، حيث تم تمثيل مربع الاختيار بواسطة الكلاس *JCheckBox*. وعندما تتغير حالة مربع الاختيار من الحالة مختار (*selected - checked*) إلى الحالة غير مختار (*deselected - unchecked*) أو العكس، فإنه يولد حدث عنصر *item event*. وتحتوي الواجهة *ItemListener* على دالة واحدة تدعى *itemStateChanged*. في هذا المثال استخدمنا نفس كائن الاستماع للتعامل مع مربعي الاختيار كليهما.

يستخدم هذا البرنامج الكلاس *Font* والذي يمثل خط حرفي معين. ويتم تعريف الكائن *Font* عن طريق اسم الخط *font name* ونمط الخط *font style* وحجم الخط *font size* ويؤسس اسم الخط الخصائص المرتبة العامة للحروف. وفي هذا البرنامج قمنا باستخدام الخط *Helvetica*. أما نمط الخط في جافا فيمكن أن يكون بسيطاً *plain* أو عريضاً *bold* أو مائلاً *italic* أو عريضاً ومائلاً في نفس الوقت. وقد تم تهيئة مربعات الاختيار في الجوئي هنا بحيث تعمل على تغيير خصائص نمط الخط.

يتم تمثيل نمط الخط كعدد صحيح *integer*، ويتم استخدام ثوابت صحيحة معرفة في الكلاس *Font* لتمثيل الجوانب المختلفة لأنماط الخط فالثابت *PLAIN* يستخدم لتمثيل الخط البسيط، والثابتان *BOLD* و *ITALIC* يستخدمان لتمثيل النمطين العريض والمائل على التوالي، وجمع النمطين *BOLD* و *ITALIC* يشير إلى نمط جديد هو النمط العريض المائل.

تقوم دالة المستمع المسماة *itemStateChanged* بتحديد ما هو الخط المعدل الذي يجب تطبيقه في حالة أن تم تغيير حالة أحد مربعات الاختيار. وتقوم هذه الدالة مبدئياً بضبط النمط على النوع البسيط، ثم يتم بعد ذلك اختيار كل مربع اختيار باستخدام الدالة *isSelected* والتي ترجع قيمة منطقية. وفي البداية، إذا تم اختيار مربع الاختيار *bold*، فسيتم ضبط النمط على النمط العريض ثم، إذا تم اختيار مربع الاختيار *italic* فسيتم إضافة الثابت *ITALIC* إلى المتغير *style*. وأخيراً، يتم ضبط خط المعنون على الخط الجديد بالنمط المعدل.

لاحظ أنه لا مشكلة في أي من مربعات الاختيار سيولد الحدث إذا ما عرفنا طريقة كتابة المستمع في هذا البرنامج. حيث أنه هنا تم معالجة كلاً من مربعي الاختيار في نفس المستمع. كذلك لا مشكلة في تغير حالة مربع الاختيار من *selected* إلى *unselected* أو العكس، حيث أنه سيتم اختبار حالة كلاً من مربعي الاختيار إذا تغيرت حالة أحدهما.

مفهوم أساسي Key Concept

تعمل أزرار الراديو *radio buttons* كمجموعة، مزودة بذلك بمجموعة خيارات حصرية تبادلية.

يستخدم زر الراديو *radio button* مع أزرار راديو أخرى لتزودنا بمجموعة خيارات حصرية تبادلية. ونلاحظ أنه بخلاف مربع الاختيار *check box*، فإن زر الراديو لا يعتبر مفيداً بنفسه، حيث أنه لا معنى فعلي له إلا عند استخدامه مع واحد أو أكثر من أزرار الراديو. ويعتبر واحد فقط من الخيارات هو الخيار الصحيح. أي أنه في أي لحظة زمنية يتم اختيار زر واحد فقط من مجموعة أزرار الراديو وضبطه على الحالة (*on*) وعندما يتم ضغط أي زر راديو من المجموعة، فإن أي زر آخر في المجموعة حالته *on* يتم تبديله تلقائيا إلى الحالة *off*.

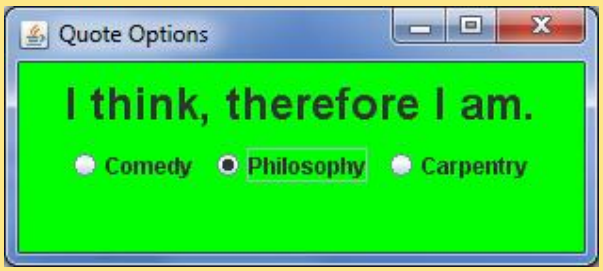
وإذا أردنا معرفة أصل المصطلح (أزرار الراديو *radio buttons*) فإنه جاء أصلاً من الطريقة التي كانت تعمل بها الأزرار في راديو السيارات قديمة الطراز. حيث أنه في أي لحظة يتم فيها ضغط أي زر لتحديد الاختيار الحالي للقناة، في حال كون زر آخر مضغوط، فإن الزر الحالي يعود إلى حالته قبل الضغط بشكل تلقائي.

يعرض البرنامج *QuoteOptions* الموضح في المثال ١٠.٤، معنوناً ومجموعة من أزرار الراديو. تقوم أزرار الراديو هذه بتحديد ما هي المقولة التي ستعرض على المعنون. وقد استخدمنا أزرار الراديو في برنامجنا هذا، لأنه لا يمكن في المرة الواحدة إلا عرض مقولة واحدة باستخدام المعنون. مثلاً، إذا تم اختيار زر الراديو *comedy*، فسيتم عرض مقولة كوميدية على المعنون، وإذا تم بعد ذلك ضغط الزر *philosophy*، فسيتم تلقائيا تحويل حالة زر الراديو *comedy* من *on* إلى *off* ويتم استبدال المقولة الكوميدية بأخرى فلسفية.

ويعمل الكلاس *QuoteOptionsPanel* الموضح في المثال ١١.٤ على تهيئة مكونات الجوئي للبرنامج. حيث يتم تمثيل زر الراديو عن طريق الكلاس *JRadioButton*. ولأن أزرار الراديو التابعة لمجموعة واحدة تعمل معاً، يتم استخدام الكلاس *ButtonGroup* لتعريف مجموعة من أزرار الراديو المترابطة.

مثال رقم ١٠.٤

```
//********************************************************************
//  QuoteOptions.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام أزرار الراديو
//********************************************************************

import javax.swing.JFrame;

public class QuoteOptions
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء وعرض إطار البرنامج
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Quote Options");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      QuoteOptionsPanel panel = new QuoteOptionsPanel();
      frame.getContentPane().add (panel);

      frame.pack();
      frame.setVisible(true);
   }
}
```



مثال رقم ١١.٤

```
//********************************************************************
//QuoteOptionsPanel.java   المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام أزرار الراديو
//********************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class QuoteOptionsPanel extends JPanel
{
   private JLabel quote;
   private JRadioButton comedy, philosophy, carpentry;
   private String comedyQuote, philosophyQuote, carpentryQuote;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة اللوحة بمعنون وكذا ضبط أزرار الراديو
   //  التي تتحكم بنمس المعنون
   //-----------------------------------------------------------------
   public QuoteOptionsPanel()
   {
      comedyQuote = "Take my wife, please.";
      philosophyQuote = "I think, therefore I am.";
      carpentryQuote = "Measure twice. Cut once.";

      quote = new JLabel (comedyQuote);
      quote.setFont (new Font ("Helvetica", Font.BOLD, 24));

      comedy = new JRadioButton ("Comedy", true);
      comedy.setBackground (Color.green);
      philosophy = new JRadioButton ("Philosophy");
      philosophy.setBackground (Color.green);
      carpentry = new JRadioButton ("Carpentry");
      carpentry.setBackground (Color.green);

      ButtonGroup group = new ButtonGroup();
      group.add (comedy);
      group.add (philosophy);
      group.add (carpentry);
```

```
        QuoteListener listener = new QuoteListener();
        comedy.addActionListener (listener);
        philosophy.addActionListener (listener);
        carpentry.addActionListener (listener);

        add (quote);
        add (comedy);
        add (philosophy);
        add (carpentry);

        setBackground (Color.green);
        setPreferredSize (new Dimension(300, 100));
    }

    //*****************************************************************
    //  الكلاس التالي يمثل مستمعاً لكل أزرار الراديو
    //*****************************************************************
    private class QuoteListener implements ActionListener
    {
        //-----------------------------------------------------------
        //  الدالة التالية تقوم بضبط نص المعنون اعتمادا على
        //  زر الراديو الذي تم ضغطه
        //-----------------------------------------------------------



        public void actionPerformed (ActionEvent event)
        {
            Object source = event.getSource();

            if (source == comedy)
                quote.setText (comedyQuote);
            else
                if (source == philosophy)
                    quote.setText (philosophyQuote);
                else
                    quote.setText (carpentryQuote);
        }
    }
}
```

لاحظ أنه يتم إضافة أي زر إلى مجموعة الأزرار ، وكذلك يتم إضافة أي زر بشكل انفرادي إلى اللوحة. إن الكائن *ButtonGroup* لا يعتبر حاوياً بحيث يمكن استخدامه في تنظيم وعرض المكونات، إنه ببساطة طريقة لتعريف أزرار الراديو التي تعمل معاً لتشكيل مجموعة أزرار راديو تعمل معاً لتشكيل مجموعة الخيارات الغير منفصلة. يعمل الكائن *ButtonGroup* على ضمان أن زر الراديو المختار حالياً ستتغير حالته من *on* إلى *off* عندما يتم ضغط زر آخر في المجموعة.

يقوم زر الراديو بتوليد حدث فعلي *action event* وتقوم الدالة *actionPerformed* الخاصة بالمستمع أولاً باسترجاع مصدر الحدث باستخدام الدالة *getSource*، ومن ثم مقارنته مع كل من أزرار الراديو الثلاثة. واعتماداً على الزر الذي تم ضغطه، يتم ضبط نص المعنون على المقولة المناسبة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

لاحظ أنه، وبخلاف أزرار الضغط *push buttons*، فإن كلاً من مربعات الاختيار *check boxes* وأزرار الراديـــو *radio buttons*، هي أزرار تبديلية *toggle buttons*، بمعنى أنها وفي أي لحظة تكون إما في حالة *on* أو *off*. والفرق يكمن في كيفية استخدامها، يتم التحكم بالخيارات المستقلة عن بعضها البعض (اختيار أي شيء) باستخدام مربعات الاختيار، ويتم التحكم بالاختيارات الغير مستقلة عن بعضها البعض (اختيار واحد من المجموعة) باستخدام أزرار الراديو. أما إذا كان هناك فقط خيار واحد فيمكن استخدام مربع اختيار واحد فقط. وكما ذكرنا سابقاً، فإن زر الراديو يكون له فائدة فقط عند ترابطه مع واحد أو أكثر من أزرار الراديو.

أيضاً لاحظ أن مربعات الاختيار وأزرار الراديو تولد نوعين مختلفين من الأحداث. حيث يقوم مربع الاختيار بتوليد حدث عنصر *item event* بينما يقوم زر الراديو بتولي حدث فعلي *action event*. إن استخدام أنواع أحداث مختلفة مرتبط بالفروقات الوظيفة للأزرار. حيث يقوم مربع الاختيار بتوليد حدث عند اختياره *selected* أو إلغاء اختياره *deselected*. ويقوم المستمع بتوضيح الفرق إذا أردنا. أما زر الراديو فيولد حدثاً فقط عند اختياره *selected*، ويتم تلقائياً إلغاء اختيار الزر الحالي المختار في المجموعة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الفصل الخامس

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

١.٥ تصميم الجوئي GUI DESIGN

بينما نكون في غمرة التركيز على التفاصيل التي تسمح لنا بإنشاء الجوئي، قد نفقد في بعض الأحيان المنظر العام للصورة. وكلما تقدمنا في استكشاف إنشاء الجوئي، يجب أن نضع في أذهاننا أن هدفنا هو حل المشكلة. وعلى وجه التحديد، نحن نريد إنشاء برمجية تكون مفيدة. إن معرفة تفاصيل المكونات، والأحداث وعناصر اللغة الأخرى يعطينا الأدوات التي تمكننا من تركيب المكونات الهرمية مع بعضها البعض، لكن يجب علينا معرفة الأفكار الرئيسية للتصميم الجيد للجوئي:

- اعرف المستخدم *Knows the user*.
- تجنب أخطاء المستخدم *Prevent user errors*.
- اجعل قدرات المستخدم في المستوى الأمثل *Optimize user abilities*.
- كن ثابتاً *Be consistent*.

مفهوم أساسي Key Concept

يجب أن يلتزم تصميم أي جوئي بالمحددات الأساسية المتعلقة بالثبات وقابلية الاستخدام.

يجب على المصمم البرمجيات *software designer* معرفة احتياجات المستخدم والأنشطة الرئيسية تمهيداً لبناء واجهة تخدم المستخدم بشكل جيد. لا تنسى أن الواجهة تمثل بالنسبة للمستخدم البرمجية كلها. حيث أنها تمثل الطريق الوحيد الذي يتفاعل من خلاله المستخدم مع النظام. وعليه، فإن الواجهة يجب أن تلبي احتياجات المستخدم.

يجب علينا كلما أمكن تصميم واجهات تقلل من المواقف التي يحتمل فيها وقوع أخطاء. في الكثير من المواقف، يكون لدينا مرونة في اختيار واحد من مكونات عدة لإتمام مهمة ما. يجب علينا أن نحاول دائماً اختيار المكونات التي ستمنع الأفعال الغير مناسبة وتجنب الإدخال الخاطئ. مثلاً، إذا كان لدينا قيمة يجب إدخالها من بين مجموعة من القيم المحددة، فيجب علينا استخدام مكونات تسمح لنا باختيار قيمة صحيحة واحدة فقط. بمعنى، حصر المستخدم في قليل من الخيارات الصحيحة، مثلاً، استخدام مجموعة من أزرار الراديو يعتبر أفضل من السماح للمستخدم بطباعة بيانات عشوائية وربما غير صحيحة في حقل نصي. وسنقوم في هذا الفصل بتغطية مكونات إضافية مناسبة لمواقف محددة.

ليس كل المستخدمين متشابهون. فبعضهم أكثر مهارة من الآخرين في استخدام مكونات جوئي معينة. يجب أن لا نصمم بالحد الأدنى من المكونات الشائعة التي نعرفها. مثلاً، يجب علينا أن نعمل اختصارات عندما يكون ذلك مناسباً. بمعنى أنه وبالإضافة إلى سلسلة الأحداث الطبيعية التي ستسمح للمستخدم بإتمام مهمة ما، يجب علينا أيضاً التزويد بطرق متعددة لإنجاز نفس المهمة. ويعتبر استخدام اختصارات لوحة المفاتيح *(mnemonics)* مثالاً جيداً على ذلك. وفي بعض الأحيان تكون هذه الآليات الإضافية أقل من حيث مستوى الإدراك والفهم، ولكنها قد تكون أسرع بالنسبة للمستخدم ذي الخبرة.

وأخيراً، يعتبر الثبات *consistency* أمراً مهماً عند التعامل مع الأنظمة الكبيرة أو الأنظمة المتعددة في البيئة المشتركة. قد يكون المستخدم متعوداً على تنظيم معين أو نمط لوني معين. وهذه الاعتبارات يجب أن لا تتغير بشكل عشوائي.

٢.٥ مدراء التخطيط LAYOUT MANAGERS

مفهوم أساسي Key Concept

يحدد مدير التخطيط *layout manager* الخاص بالحاوي كيفية ظهور العرض المرئي للمكونات.

بالإضافة إلى المكونات *components*، والأحداث *events*، والمستمعات *listeners* التي تشكل العمود الفقري للجوئي، فإن النشاط الأكثر أهمية في تصميم الجوئي هو استخدام مدراء التخطيط *layout managers*. يعتبر مدير التخطيط كائن يتحكم بكيفية ترتيب المكونات داخل الحاوي. فهو يحدد حجم وموضع كل مكون ويمكن أن يضع في اعتباره العديد من العوامل الأخرى. ويملك كل حاوي مدير تخطيط افتراضي، ومع ذلك يمكننا استبداله إذا فضلنا مدير تخطيط آخر.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مفهوم أساسي Key Concept

عند حدوث تغييرات، فإن مكونات الحاوي تعيد تنظيم نفسها وفقاً لسياسة مدير التخطيط.

يتم استدعاء مدير تخطيط للحاوي عندما يكون هناك احتمال الحاجة إلى تغيير المظهر المرئي لمكوناته. مثلاً، عندما يتغير حجم الحاوي، يتم استدعاء مدير التخطيط لتحديد كيفية ظهور كل مكونات الحاوي بعد تغير حجمه. وفي كل مرة يتم فيها إضافة مكون إلى الحاوي، يقوم مدير التخطيط بتحديد كيف تؤثر هذه الإضافة في كل المكونات الموجودة.

يسرد الشكل ١.٥العديد من مدراء التخطيط المعرفين مسبقاً في مكتبة الكلاسات القياسية في لغة جافا.

يمتلك كل مدير تخطيط خصائص معنية وقواعد تتحكم بتخطيط المكونات. ففي بعض مدراء التخطيط، يؤثر ترتيب إضافة المكونات على موضعها، بينما هناك مدراء آخرين يعطونا تحكماً أكثر تخصيصاً. بعض مدراء التخطيط تأخذ في الاعتبار الحجم المفضل للمكون أو المحاذاة، بينما هناك مدراء آخرين لا يهتموا بهذا الأمر. ولتطوير جوئيات جديدة بلغة جافا، من المهم أن تصبح معتاداً على مزايا وخصائص مدراء التخطيط المختلفين.

يمكننا استخدام الدالة *setLayout* الخاصة بالحاوي، لتغيير مدير التخطيط له. وقد قمنا بذلك عدة مرات في الأمثلة السابقة. مثلاً، تعمل الشفرة التالية على ضبط مدير التخطيط لـ *JPanel*، والتي تملك مدير التخطيط التدفقي *flow layout* كمدير افتراضي، ولهذا ستستخدم مدير التخطيط الحدودي *border layout* بدلاً من ذلك:

```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
```

مفهوم أساسي Key Concept

يمكن تغيير مدير التخطيط لكل حاوي بشكل صريح.

دعنا الآن نستكشف بعضاً من مدراء التخطيط بشكل أكثر تفصيلاً. وسنركز على مدراء التخطيط الأكثر شهرة هنا، وهم: التدفقي *flow*، والحدودي *border*، والصندوقي *box*، والشبكي *grid*. يحتوي الكلاس الموجود في المثال ١.٥ على الدالة *main* الخاصة بالتطبيق الذي يشرح استخدام وتأثيرات مدراء التخطيط.

يبين البرنامج *LayoutDemo* استخدام المقطع المبوب *tabbed pane*، والذي هو عبارة عن حاوي يسمح للمستخدم باختيار واحدة من اللوحات المتعددة المرئية حالياً (عن طريق الضغط على المفتاح *tab*). ويتم تعريف المقاطع المبوبة عن طريق الكلاس *JTabbedPane*. ويتم إنشاء تبويب جديد عن طريق الدالة *addTab*، وفيها يتم تحديد الاسم الذي يظهر على التبويب وكذا المكون الذي سيعرض على التبويب عندما يتم تفعيله عن طريق "إحضاره إلى المقدمة" ويصبح مرئياً للمستخدم.

مدير التخطيط *Layout manager*	الوصف *Description*
التخطيط الحدودي *Border layout*	ينظم المكونات في خمس مناطق (شمالية *North*، جنوبية *south*، شرقية *East*، غربية *West*، وسطية *center*).
التخطيط الصندوقي *Box layout*	ينظم المكونات في وصف واحد *one row* أو عمود واحد *one column*.
تخطيط البطاقة *Card layout*	ينظم المكونات في منطقة واحدة بحيث يتم عرض مكون واحد فقط في اللحظة الواحدة.
التخطيط التدفقي *Flow layout*	ينظم المكونات من الشمال إلى اليمين، والبدء بصفوف جديدة عند الضرورة.
التخطيط الشبكي *Grid layout*	ينظم المكونات في شكل شبكة *grid* مكونة من صفوف *rows* وأعمدة *columns*.
تخطيط الحقيبة الشبكي *Grid Bag layout*	ينظم المكونات في شكل شبكة من الخلايا *cells*، ويسمح للمكونات بأن تتوسع لأكثر من خلية.

الشكل رقم ١.٥ بعض مدراء التخطيط المعرفين مسبقاً في جافا

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مثال رقم ١.٥

```
//*****************************************************************
//  LayoutDemo.java       المؤلف : Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام التخطيطات التدفقية، الحدودية، الشبكية
//  والصندوقية
//*****************************************************************

import javax.swing.*;

public class LayoutDemo
{
   //-----------------------------------------------------------------
   //  يقوم الباني بتهيئة الإطار الذي يحوي مقطع مبوب. وتوضح اللوحة
   //  في كل تبويب مدير تخطيط مختلف
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Layout Manager Demo");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JTabbedPane tp = new JTabbedPane();
      tp.addTab ("Intro", new IntroPanel());
      tp.addTab ("Flow", new FlowPanel());
      tp.addTab ("Border", new BorderPanel());
      tp.addTab ("Grid", new GridPanel());
      tp.addTab ("Box", new BoxPanel());

      frame.getContentPane().add(tp);
      frame.pack();
      frame.setVisible(true);
   }
}
```

والمثير للاهتمام، أن هناك تداخلاً وظيفياً بين المقاطع المبوبة *tabbed panes* ومدير تخطيط البطائق *card layout manger*. فتخطيط البطائق يشبه المقطع المبوب في أنه يسمح بتعريف عدة طبقات *Layers*، ويتم فقط عرض واحدة من هذه البطاقات في المرة الواحدة. وبالرغم من ذلك، فإن الحاوي الذي يدار من قبل مدير البطائق يمكن موائمته فقط تحت سيطرة البرنامج، بينما المقاطع المبوبة تسمح للمستخدم بالإشارة المباشرة إلى أي من التبويبات يجب عرضه.

في هذا المثال، يحوي كل تبويب من المقاطع المبوبة على لوحة يتم التحكم بها بواسطة مدير تخطيط مختلف. فالتبويب الأول يحتوي ببساطة على لوحة فيها رسالة توضيحية، كما هو مبين في المثال ٢.٥. وكلما قمنا باستكشاف تفاصيل أي مدير تخطيط، فسنقوم باختبار الكلاس الذي يعرف اللوحة المعنية بهذا البرنامج، وسنناقش التأثير المرئي لها.

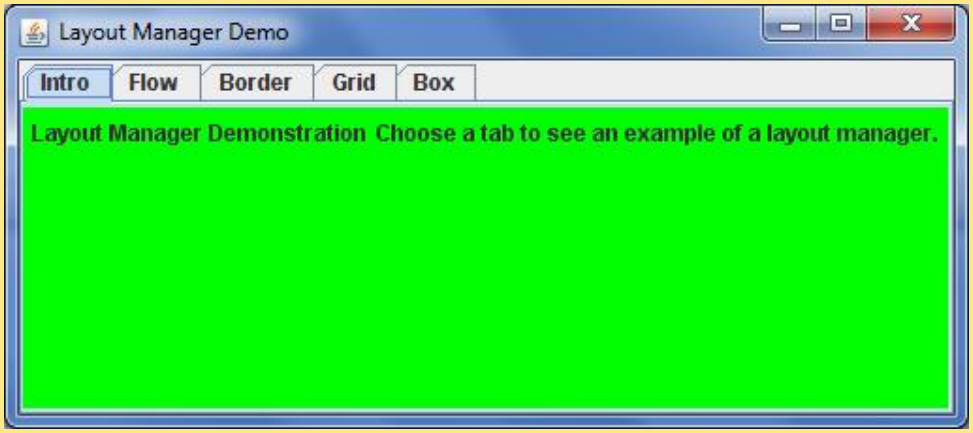
مثال رقم ٢.٥

```
//*****************************************************************
//  IntroPanel.java       المؤلف : Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل اللوحة التقديمية لبرنامج شرح التخطيط
//*****************************************************************
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
import java.awt.*;
import javax.swing.*;

public class IntroPanel extends JPanel
{
   //-----------------------------------------------------------------
   //  يقوم الباني بتهيئة اللوحة بمعنونين
   //-----------------------------------------------------------------
   public IntroPanel()
   {
      setBackground (Color.green);

      JLabel l1 = new JLabel ("Layout Manager Demonstration");
      JLabel l2 = new JLabel ("Choose a tab to see an example of " +
                              "a layout manager.");

      add (l1);
      add (l2);
   }
}
```

العرض Display



التخطيط التدفقي Flow Layout

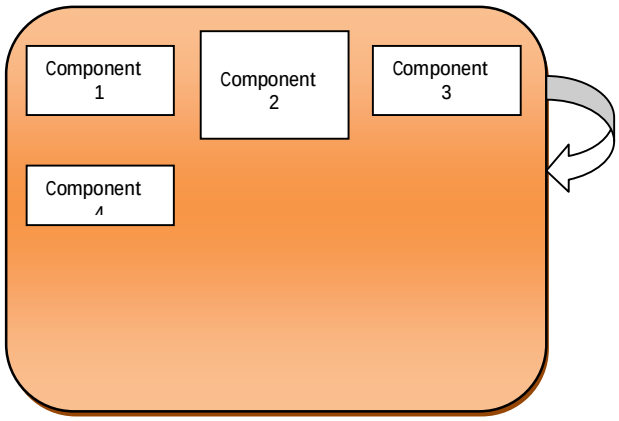
يعتبر التخطيط التدفقي *flow layout* واحد من أبسط مدراء التخطيط المستخدمة. حيث أن الكلاس *JPanel* يستخدم التخطيط التدفقي بشكل افتراضي. يعمل التخطيط التدفقي على وضع أكبر قدر ممكن من المكونات في صف واحد، وفقاً للحجم المفضل للمكون. وعندما لا يمكن موائمة المكون في الصف، يتم وضعه في الصف التالي. ويتم إضافة القدر اللازم الذي نحتاجه من الصفوف لملائمة كل المكونات التي تم إضافتها إلى الحاوي. ويوضح الشكل ٢.٥ حاوياً يتم التحكم به عن طريق مدير التخطيط التدفقي.

يبين الكلاس في المثال ٣.٥ اللوحة التي توضح التخطيط التدفقي في البرنامج *LayoutDemo*، فهو يقوم بشكل صريح بضبط التخطيط على النوع التدفقي (مع ذلك، وفي حالتنا هذه، يعتبر هذا غير ضرورياً لأن التخطيط التدفقي هو التخطيط الافتراضي لـ *JPanel*). ثم يتم بعد ذلك إنشاء الأزرار وإضافتها إلى اللوحة.

يتم تكبير كل زر بشكل كافٍ بحيث يتلائم مع حجم المعنون الموجود داخله. وكما ذكرنا سابقاً، يعمل التخطيط التدفقي على وضع أكبر قدر ممكن من هذه الأزرار في صف واحد داخل اللوحة، ومن ثم يبدأ بوضع المكونات في صف آخر. وعندما يتم زيادة عرض الإطار (عن طريق سحب الزاوية اليمنى السفلية بالماوس، مثلاً)، فإن اللوحة تكبر كذلك، ويزداد عدد الأزرار التي يمكن موائمتها في صف واحد. عندما يتم تغير حجم الإطار، يتم

استدعاء مدير التخطيط الذي يعمل على إعادة تنظيم المكونات تلقائياً. لاحظ أنه في كل صف يتم توسيط المكونات داخل النافذة بشكل افتراضي.

هنالك عدة إصدارات من الباني الخاص بالكلاس *FlowLayout* تسمح للمستخدم بالتحكم بخصائص مدير التخطيط. ففي كل صف، إما أن يتم محاذاة المكونات في الوسط، أو إلى جهة الشمال، أو إلى جهة اليمين. والوضع الافتراضي للمحاذاة هو وسط *center*. يمكن أيضاً تحديد حجم الفجوة الأفقية والعمودية بين المكونات عند إنشاء مدير التخطيط. كذلك يمتلك الكلاس *FlowLayout* دوالاً لضبط المحاذاة وأحجام الفجوة بعد إنشاء مدير التخطيط.



الشكل ٢.٥ التخطيط التدفقي يضع أكبر قدر من المكونات في صف

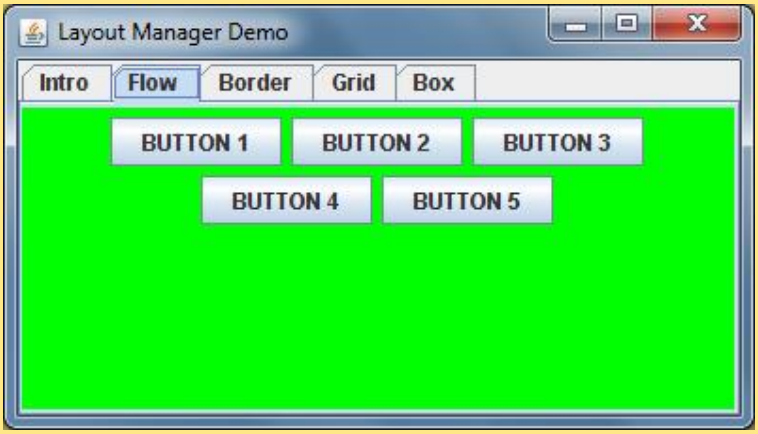
مثال رقم ٣.٥

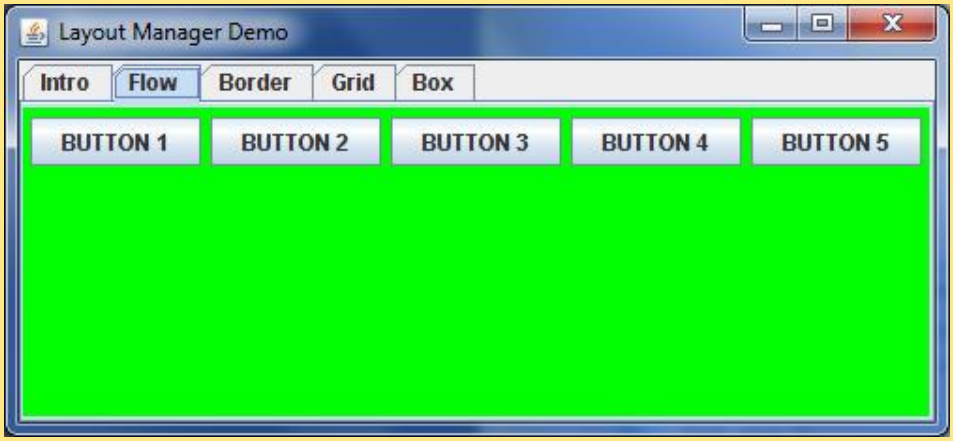
```
//*******************************************************************
//  FlowPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل اللوحة في برنامج شرح التخطيط التي توضح
//  مدير التخطيط التدفقي
//*******************************************************************

import java.awt.*;
import javax.swing.*;

public class FlowPanel extends JPanel
{
   //-----------------------------------------------------------------
   //  يهيئ الباني اللوحة ببعض الأزرار لتوضيح كيف أن التخطيط التدفقي
   //  يؤثر في مواقعها
   //-----------------------------------------------------------------
   public FlowPanel ()
   {
      setLayout (new FlowLayout());

      setBackground (Color.green);

      JButton b1 = new JButton ("BUTTON 1");
      JButton b2 = new JButton ("BUTTON 2");
      JButton b3 = new JButton ("BUTTON 3");
      JButton b4 = new JButton ("BUTTON 4");
      JButton b5 = new JButton ("BUTTON 5");
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
        add (b1);
        add (b2);
        add (b3);
        add (b4);
        add (b5);
    }
}
```

العرض Display

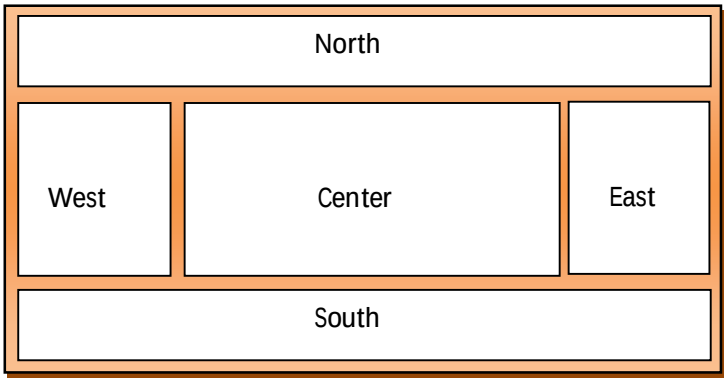




التخطيط الحدودي Border layout

يمتلك التخطيط الحدودي *border layout* خمس مناطق يمكن إضافة المكونات إليها، هي: الشمالية *North*، الجنوبية *South*، الشرقية *East*، الغربية *West*، والوسطية *Center*. ولهذه المناطق علاقات موضعية محددة مع بعضها البعض كما هو موضوع في الشكل ٣.٥.

تصبح الأربع المناطق الخارجية كبيرة بما فيه الكفاية لتتلائم مع المكونات التي تحتويها. أما إذا لم يتم إضافة مكونات إلى المنطقة الشمالية أو الجنوبية أو الشرقية أو الغربية، فإن هذه المناطق لن تحتل أي مساحة في المخطط الكلي. ثم إن المنطقة الوسطى (المركزية) ستتسع لملئ أي فراغ متاح.



الشكل ٣.٥ التخطيط الحدودي ينظم المكونات في خمس مناطق

قد يستخدم حاوٍ ما مناطق قليلة فقط، اعتماداً على الطبيعة الوظيفية للنظام. مثلاً، قد يستخدم البرنامج المناطق المركزية والجنوبية والغربية فقط.هذا التنوع يجعل من التخطيط الحدودي مديراً تخطيطياً مفيداً جداً.

تأخذ الدالة *add*، الخاصة بالحاوي والمحكومة بواسطة التخطيط الحدودي، المكون المراد إضافته كوسيط أول لها. أما الوسيط الثاني فيدل على المنطقة التي سيضاف إليها هذا المكون. ويتم تحديد المنطقة باستخدام ثوابت معرفه في الكلاس *BorderLayout*. ويبين المثال ٤.٥ اللوحة المستخدمة من قِبل برنامج *LayoutDemo* والتي توضح التخطيط الحدودي.

في الباني الخاص بالكلاس *BorderPanel*، تم بشكل صريح ضبط مدير التخطيط للوحة بحيث يكون تخطيطاً حدودياً. ثم بعد ذلك تم إنشاء الأزرار وإضافتها إلى مناطق محددة في اللوحة. يتم بشكل افتراضي جعل كل زر عريضاً بما فيه الكفاية لملائمة المعنون الخاص به، وطويلاً بما فيه الكفاية لملء المساحة التي تم إسناده إليها. كلما تم تغيير حجم الإطار (واللوحة)، فإن حجم كل زر يتغير وفقاً لذلك، ويملأ الزر الذي في المنطقة الوسطية أي مساحة غير مستخدمة.

تقوم أي منطقة من التخطيط الحدودي بعرض مكون واحد فقط. أي أنه، يتم إضافة مكون واحد فقط لأي منطقة في أي تخطيط حدودي. ومن الأخطاء الشائعة إضافة مكونين إلى المنطقة المخصصة من التخطيط الحدودي، حيث في هذه الحالة يتم استبدال المكون الأول المضاف بالمكون الثاني، وسيكون المكون الثاني هو المرئي فقط عند عَرض الحاوي. ولكي نتمكن مِن إضافة مكونات متعددة لمنطقةٍ ما ضمن تخطيط حدودي، نقوم أولاً بإضافة المكونات إلى حاوٍ آخر، مثل الـ *JPanel*، ثم إضافة اللوحة إلى المنطقة المعينة التي نريدها.

لاحظ أنه وبالرغم من أن اللوحة المستخدمة لعرض الأزرار لها لون خلفيه أخضر، فإنه لا لون أخضر يظهر في العرض في المثال ٤.٥ . بشكل افتراضي، ليس هناك فجوات أفقية أو عمودية بين مناطق التخطيط الحدودي. يمكن ضبط قيم هذه الفجوات بواسطة باني إضافي أو بواسطة دوال صريحة في الكلاس *BorderLayout*. إذا تم زيادة قيم الفجوات، فإن اللوحة السفلية ستظهر من خلالها.

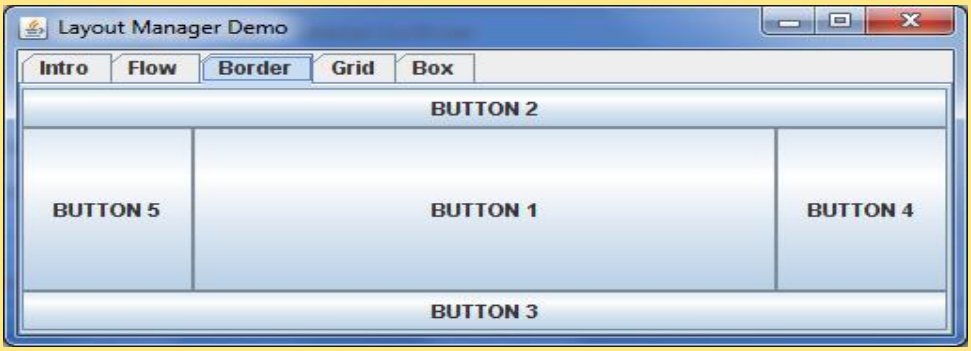
مثال رقم ٤.٥

```
//********************************************************************
//  BorderPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا البرنامج اللوحة التي في برنامج شرح التخطيط التي تشرح
//  مدير التخطيط الحدودي
//********************************************************************

import java.awt.*;
import javax.swing.*;
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
public class BorderPanel extends JPanel
{

   //-----------------------------------------------------------------
   //  تهيئة هذه اللوحة بزر في كل منطقة من مناطق مدير التخطيط
   //  لتوضيح كيف أنه يؤثر في موقعها، وشكلها، وحجمها
   //-----------------------------------------------------------------
   public BorderPanel()
   {
      setLayout (new BorderLayout());

      setBackground (Color.green);

      JButton b1 = new JButton ("BUTTON 1");
      JButton b2 = new JButton ("BUTTON 2");
      JButton b3 = new JButton ("BUTTON 3");
      JButton b4 = new JButton ("BUTTON 4");
      JButton b5 = new JButton ("BUTTON 5");

      add (b1, BorderLayout.CENTER);
      add (b2, BorderLayout.NORTH);
      add (b3, BorderLayout.SOUTH);
      add (b4, BorderLayout.EAST);
      add (b5, BorderLayout.WEST);
   }
}
```

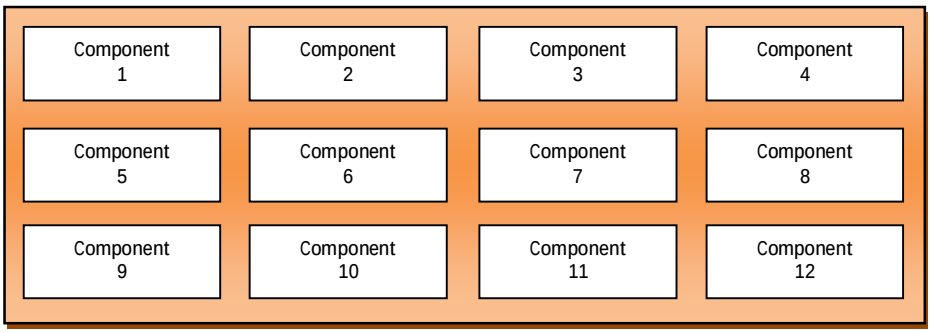
العرض Display



التخطيط الشبكي Grid Layout

يعرض التخطيط الشبكي *grid layout* مكونات الحاوي في شبكةٍ مستطيلة من الصفوف والأعمدة. ويتم وضع مكون واحد في كل خلية في الشبكة، ويكون حجم كل الخلايا متساوٍ. يبين الشكل ٤.٥ التنظيم العام للتخطيط الشبكي.

يتم تحديد عدد الصفوف والأعمدة في التخطيط الشبكي باستخدام وسطاء للباني عند إنشاء مدير التخطيط. ويبين الكلاس في المثال ٥.٥ اللوحة المستخدمة من قِبل البرنامج *LayoutDemo* لتوضيح التخطيط الشبكي. ففيه تم تحديد أن اللوحة تُدار بواسطة شبكة مكونة من صفين وثلاثة أعمدة.



الشكل ٤.٥ التخطيط الشبكي يُنشئ شبكة مستطيلة ذات خلايا

متى ما تم إضافة الأزرار إلى الحاوي، فإنها تملئ الشبكة (بشكل افتراضي) من الشمال إلى اليمين ومن الأعلى إلى الأسفل. ولا توجد هناك طريقة يمكن من خلالها الإسناد الصريح لمكونات معينة بحيث ترتبط بموقع معين في الشبكة باستثناء الترتيب التي يتم به إضافة هذه المكونات إلى الحاوي.

يتم تحديد حجم كل خلية عن طريق الحجم الكلي للحاوي. وعندما يتم تغيير حجم الإطار، فإن حجم كل الخلايا يتغير تبعاً لذلك بتناسب طردي بحيث تملئ الحاوي.

إذا كانت القيمة المستخدمة لتحديد عدد الصفوف أو عدد الأعمدة هي صفر، فإن الشبكة تتمدد حسب الحاجة في ذلك البعد، بحيث تستوعب عدد المكونات المضافة إلى الحاوي. ولا يمكن أن تكون قيم عدد الصفوف وعدد الأعمدة صفراً معاً في نفس الوقت.

وبشكل افتراضي، لا يوجد هناك فجوات أفقية أو عمودية بين خلايا الشبكة. ويمكن تحديد أحجام الفجوات باستخدام الباني الإضافي أو باستخدام دوال *GridLayout* مخصصة.

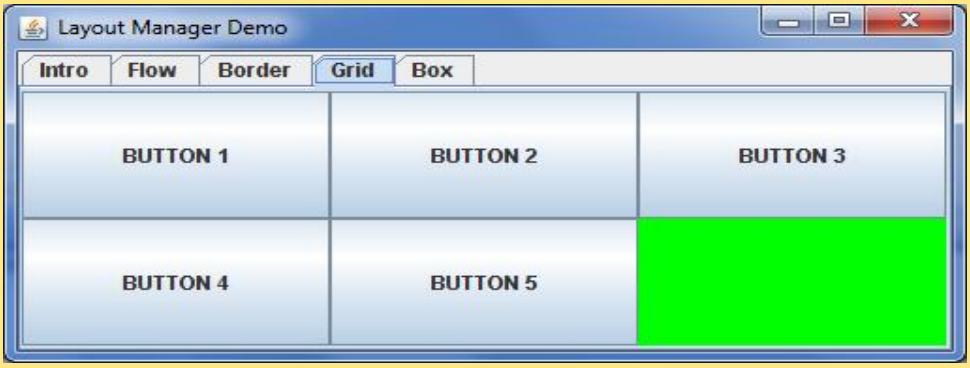
مثال رقم ٥.٥

```
//********************************************************************
//  GridPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل اللوحة في برنامج شرح التخطيط التي توضح
//  مدير التخطيط الشبكي
//********************************************************************

import java.awt.*;
import javax.swing.*;

public class GridPanel extends JPanel
{
   //-----------------------------------------------------------------
   //  يقوم الباني بتهيئة بعض الأزرار لتوضيح كيف أن التخطيط الشبكي
   //  يؤثر في مواقعها، وشكلها، وحجمها
   //-----------------------------------------------------------------
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

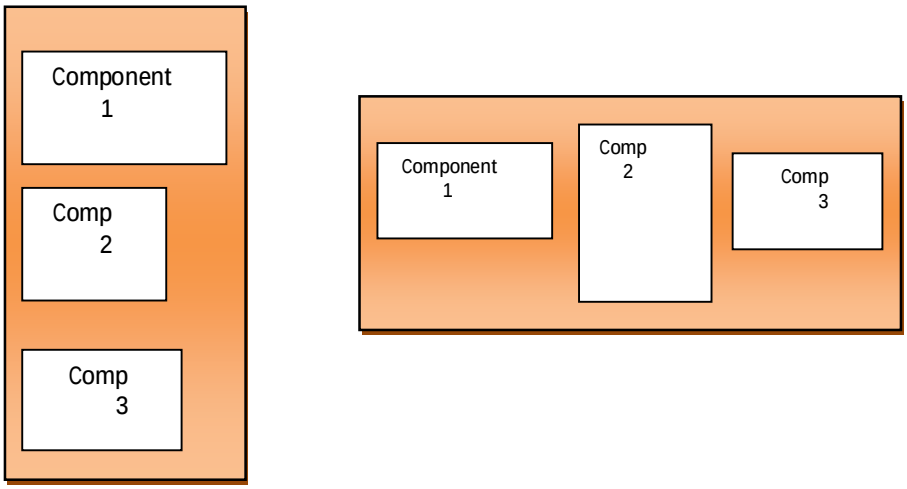
```
    public GridPanel()
    {
       setLayout (new GridLayout (2, 3));

       setBackground (Color.green);

       JButton b1 = new JButton ("BUTTON 1");
       JButton b2 = new JButton ("BUTTON 2");
       JButton b3 = new JButton ("BUTTON 3");
       JButton b4 = new JButton ("BUTTON 4");
       JButton b5 = new JButton ("BUTTON 5");

       add (b1);
       add (b2);
       add (b3);
       add (b4);
       add (b5);
    }
}
```

العرض Display



التخطيط الصندوقي Box Layout

يعمل التخطيط الصندوقي *box layout* على تنظيم المكونات إما عمودياً أو أفقياً في صفٍ واحد أو في عمودٍ واحد، كما هو موضح في الشكل ٥.٥ . وهذا التخطيط سهل الاستخدام، حتى عند تركيبه مع تخطيطات صندوقية أخرى، فبإمكانه توليد تصميمات جوئي معقدة مشابهة لتلك التي يمكن عملها باستخدام الـ *GridLayout*، والذي يعتبر بشكل عام أكثر صعوبة من حيث التحكم.



الشكل ٥.٥ التخطيط الصندوقي ينظم المكونات إما عمودياً وإما أفقياً

عندما يتم إنشاء كائن *BoxLayout*، نقوم بتحديد فيما إذا كان سيتبع المحور *X* (أفقي) أو المحور *Y* (عمودي)، باستخدام ثوابت معرفه في الكلاس *BoxLayout*. ونلاحظ أنه، بخلاف مدراء التخطيط الأخرى، فإن باني الكلاس *BoxLayout* يستقبل المكون الذي يحكمه كوسيط أول. ولذلك فإنه يجب توليد كائن *BoxLayout* لكل مكون. ويوضح المثال ٦.٥ اللوحة المستخدمة بواسطة البرنامج *LayoutDemo* لتوضيح التخطيط الصندوقي.

يتم تنظيم مكونات الحاوي المحكوم بمدير التخطيط الصندوقي (من أعلى إلى أسفل ومن الشمال إلى اليمين) بنفس الترتيب الذي تم به إضافتها إلى الحاوي.

لا يوجد هناك فجوات بين المكونات في التخطيط الصندوقي. وبخلاف مدراء التخطيط السابقين، نلاحظ أن التخطيط الصندوقي يمتلك فجوة عمودية أو أفقية يمكن تحديدها للحاوي بأكمله. وبدلاً من ذلك، يمكننا إضافة مكونات غير مرئية إلى الحاوي لتمثل المساحة الموجودة بين المكونات. يحتوي الكلاس *Box*، والذي يعد جزءاً من مكتبة كلاسات جافا القياسية، على دوال ساكنة *static* يمكن استخدامها لتوليد هذه المكونات الغير مرئية.

والنوعان المستخدمان من المكونات الغير مرئية في الكلاس *BoxPanel* هما المناطق القاسية *rigid areas*، والتي تملك حجماً ثابتاً، والغراء *glue*، والذي يحدد أين ستكون المساحة الزائدة من الحاوي. يتم توليد المنطقة القاسية *rigid area* باستخدام الدالة *createRigidArea* التابعة للكلاس *Box*، والتي تستقبل كائن *Dimension* كوسيط لها لتعريف حجم المنطقة الغير مرئية. أما الغراء *glue* فيتم توليده باستخدام الدالة *createHorizontalGlue* أو الدالة *createVerticalGlue*، حسب الحاجة.

لاحظ أنه، في مثالنا هذا، يظل الفراغ بين الأزرار المفصولة بالمنطقة القاسية ثابتاً حتى إذا تم تغيير حجم الحاوي. من ناحية أخرى يتمدد الغراء أو ينكمش حسب الحاجة لملئ الفراغ.

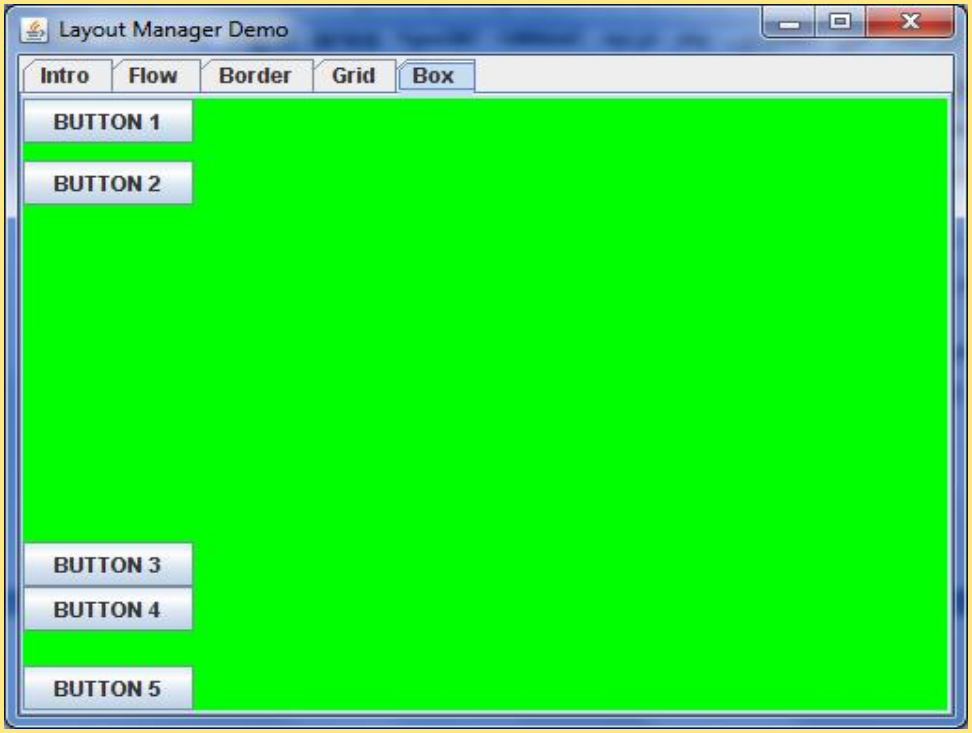
ومن الجدير ذكره أن التخطيط الصندوقي، وبشكلٍ أكبر من مدراء التخطيط الآخرين، يهتم بالمحاذاة *alignment* والأحجام المفضلة *preferred sizes* والأحجام الصغرى والكبرى للمكونات التي يحكمها. لذلك، فإن ضبط خصائص المكونات الداخلة في الحاوي يعتبر طريقة أخرى للتحكم بالتأثير المرئي.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مثال رقم ٦.٥

```
//********************************************************************
//  BoxPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل اللوحة التي في برنامج شرح التخطيط التي توضح
//  مدير التخطيط الصندوقي
//********************************************************************

import java.awt.*;
import javax.swing.*;

public class BoxPanel extends JPanel
{
   //-----------------------------------------------------------------
   //  يقوم الباني بتهيئة بعض الأزرار لتوضيح كيف أن التخطيط الصندوقي
   //  العمودي (والمكونات الغير مرئية) تؤثر على مواقع الأزرار
   //-----------------------------------------------------------------
   public BoxPanel()
   {
      setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));

      setBackground (Color.green);

      JButton b1 = new JButton ("BUTTON 1");
      JButton b2 = new JButton ("BUTTON 2");
      JButton b3 = new JButton ("BUTTON 3");
      JButton b4 = new JButton ("BUTTON 4");
      JButton b5 = new JButton ("BUTTON 5");

      add (b1);
      add (Box.createRigidArea (new Dimension (0, 10)));
      add (b2);
      add (Box.createVerticalGlue());
      add (b3);
      add (b4);
      add (Box.createRigidArea (new Dimension (0, 20)));
      add (b5);
   }
}
```

العرض Display



٣.٥ الحـــدود BORDERS

مفهوم أساسي Key Concept

يمكن تطبيق الحدود *borders* على مكونات *Swing* لتجميع الكائنات أو لتركيز الاهتمام عليها.

تزود جافا بإمكانية وضع حد *border* حول أي مكون *Swing*. ولا يعتبر الحد مكوناً بحد ذاته ولكنه يعرِّف كيفية رسم حواف أي مكون، وله تأثير مهم في تصميم الجوئي. ويزودنا الحد بالملامح المرئية مثل كيفية تنظيم مكونات الجوئي، ويمكن استخدامه لإعطاء عناوين للمكونات. ويسرد الشكل ٦.٥ الحدود المعرفة مسبقاً في مكتبة كلاسات جافا القياسية.

الحد *Border*	الوصف *Description*
الحد الفارغ *Empty Border*	يضع مساحة حاجزة حول حافة المكون، ولكن ليس له أي تأثير مرئي.
الحد الخطي *Line Border*	خط بسيط يحيط بالمكون.
الحد المحفور *Etched Border*	ينشأ تأثيراً يشبه الأخدود المحفور حول المكون.
الحد المشطوف *Bevel Border*	ينشأ تأثيراً يجعل المكون كأنه طافٍ على سطح أو غاطس تحته.
الحد المعنون *Titled Border*	يتضمن عنواناً نصياً على الحد أو حوله.
الحد المت *Matte Border*	يسمح بتحديد حجم كل حافة. يستخدم إما لوناً قاسياً أو صورة.
الحد المركب *Compound Border*	خليط من حدين.

الشكل ٦.٥ حدود المكونات

ويعتبر الكلاس *BorderFactory* مهماً في إنشاء حدود للمكونات. حيث أنه يملك عدة دوال لإنشاء أنواع مخصصة من الحدود. ويتم تطبيق الحد على المكون باستخدام الدالة *setBorder* الخاصة بالمكون.

يشرح البرنامج في المثال ٧.٥ الأنواع المتعددة للحدود. حيث إنه يقوم ببساطة بإنشاء لوحات متعددة، ويضبط كل لوحة على نوع مختلف من الحدود، ومن ثم يعرض هذه اللوحات في لوحة أكبر باستخدام تخطيط شبكي.

دعنا الآن ننظر في كل نوع من الحدود التي يولدها هذا البرنامج. الحد الفارغ *empty border*: يتم تطبيقه على اللوحة الكبيرة التي تحمل بقية اللوحات الأخرى، وذلك من أجل إنشاء فراغ إضافي حول الحافة الخارجية للإطار. ويتم تحديد أحجام الحواف العليا، والشمالية، والسفلى، واليمنى للحد الفارغ للبكسلات. الحد الخطي *line border*: يتم إنشاؤه باستخدام لون محدد وبتحديد سمك الخط بالبكسلات (٣ بكسلات في هذه الحالة). أما إذا ترك سمك الخط بدون تحديد فإن القيمة الافتراضية له هي ١ بكسل. الحد المحفور *etched border*: يتم توليده في هذا البرنامج باستخدام ألوان افتراضية لإضاءة وظل الأحفورة، ومع ذلك يمكن ضبط الإضاءة والظل بشكل صريح إذا أردنا ذلك.

الحد المشطوف *bevel border*: يمكن أن يكون إما مرتفعاً أو منخفضاً. وفي هذا البرنامج تم استخدام التلوين الافتراضي، ومع ذلك يمكننا تخصيص تلوين كل جانب من الشطف حسب الطلب، بما في ذلك الإضاءة الخارجية، والإضاءة الداخلية، والظل الخارجي، والظل الداخلي. ويمكن أن يكون كل جانب من هذه الجوانب ذو لون مختلف إذا أردنا ذلك.

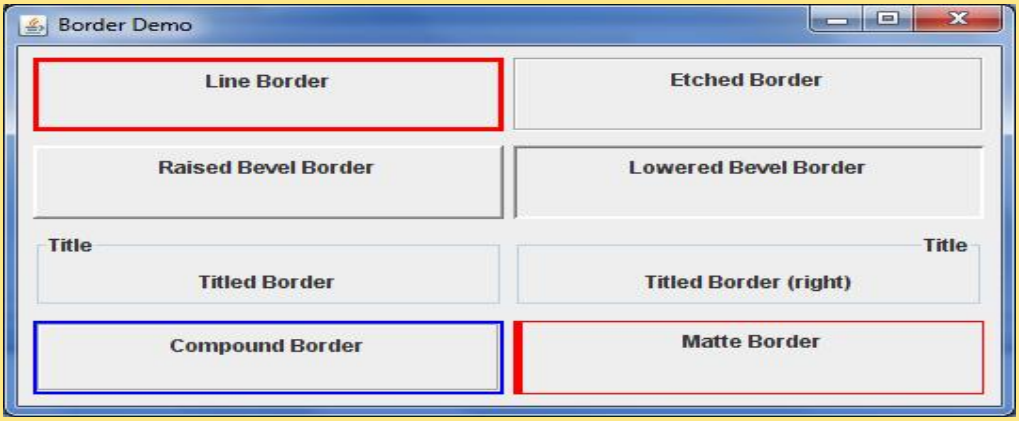
الحد المعنون *titled border*: يعمل على وضع عنوان على أو حول الحد. والموضع الافتراضي للعنوان هو على الحد في الحافة العليا الشمالية. ويمكن تغيير هذا الموضع إلى عدة أماكن أخرى فوق الحد، أو تحت الحد، أو على الحد، أو على شمال الحد، أو إلى يمين الحد، أو في وسطه، وذلك باستخدام الدالة *setTitleJustification* التابعة للكلاس *TitleBorder*.

مثال رقم ٧.٥

```
//*****************************************************************
//  BorderDemo.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح استخدام الأنواع المختلفة من الحدود
//*****************************************************************
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
public class BorderDemo
{
   //--------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء لوحات محاطة بحدود ومن ثم تعرضها
   //--------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Border Demo");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JPanel panel = new JPanel();
      panel.setLayout (new GridLayout (0, 2, 5, 10));
      panel.setBorder (BorderFactory.createEmptyBorder (8, 8, 8, 8));

      JPanel p1 = new JPanel();
      p1.setBorder (BorderFactory.createLineBorder (Color.red, 3));
      p1.add (new JLabel ("Line Border"));
      panel.add (p1);

      JPanel p2 = new JPanel();
      p2.setBorder (BorderFactory.createEtchedBorder ());
      p2.add (new JLabel ("Etched Border"));
```

```
        panel.add (p2);
        JPanel p3 = new JPanel();
        p3.setBorder (BorderFactory.createRaisedBevelBorder ());
        p3.add (new JLabel ("Raised Bevel Border"));
        panel.add (p3);

        JPanel p4 = new JPanel();
        p4.setBorder (BorderFactory.createLoweredBevelBorder ());
        p4.add (new JLabel ("Lowered Bevel Border"));
        panel.add (p4);

        JPanel p5 = new JPanel();
        p5.setBorder (BorderFactory.createTitledBorder ("Title"));
        p5.add (new JLabel ("Titled Border"));
        panel.add (p5);

        JPanel p6 = new JPanel();
        TitledBorder tb = BorderFactory.createTitledBorder ("Title");
        tb.setTitleJustification (TitledBorder.RIGHT);
        p6.setBorder (tb);
        p6.add (new JLabel ("Titled Border (right)"));
        panel.add (p6);

        JPanel p7 = new JPanel();
        Border b1 = BorderFactory.createLineBorder (Color.blue, 2);
        Border b2 = BorderFactory.createEtchedBorder ();
        p7.setBorder (BorderFactory.createCompoundBorder (b1, b2));
        p7.add (new JLabel ("Compound Border"));
        panel.add (p7);

        JPanel p8 = new JPanel();
        Border mb = BorderFactory.createMatteBorder (1, 5, 1, 1,
                                                     Color.red);
        p8.setBorder (mb);
        p8.add (new JLabel ("Matte Border"));
        panel.add (p8);

        frame.getContentPane().add (panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Display العرض



الحد المركب *compound border*: هو عبارة عن تركيبة من اثنين أو أكثر من الحدود. يقوم المثال في هذا البرنامج بإنشاء حد مركب باستخدام الحد الخطي والحد المحفور. تستقبل الدالة *createCompoundBorder* حدين كوسيطين، وتجعل الوسيط الأول هو الحد الخارجي، والوسيط الثاني هو الحد الداخلي. أما التركيبة لثلاثة حدود أو أكثر فيتم إنشاؤها أولاً بإنشاء حد مركب باستخدام حدين، ثم عمل حد مركب آخر باستخدامه وواحد آخر.

الحد المت *matte border*: يتم فيه تحديد أحجام الحواف العليا والشمالية والسفلى واليمنى للحد بالبكسلات. يمكن تكوين هذه الحواف من لون واحد، كما هو في مثالنا هذا، أو يمكن استخدام أيقونة صورية.

يجب استخدام الحدود بعناية. يمكن أن تكون الحدود مفيدة في رسم الانتباه للأجزاء المناسبة في الجوئي الخاص بك. ويمكنها عمل رابط معنوي للعناصر المتصلة مع بعضها البعض. ومع ذلك، إذا ما استخدمت الحدود بشكل غير مناسب، فإنها يمكن أن تقلل من أناقة العرض. يتوجب أن تعمل الحدود على تحسين الواجهة، وليس تعقيدها أو التنافس معها.

٤.٥ هرميات الاحتواء CONTAINMENT HIERARCHIES

تنشأ الطريقة التي يتم بها تجميع المكونات في الحاويات، والطريقة التي تتداخل بها الحاويات مع بعضها البعض ما يسمى بهرم الاحتواء للجوئي *containment hierarchy*. وقد قدمنا الحديث حول هذا المفهوم في الفصل الثاني. وإذا ما اعتنينا بتصميم هرم الاحتواء، فان ذلك ينعكس على دقة التأثير المرئي للجوئي.

يوجد لأي برنامج جافا بشكل عام حاوي أساسي واحد، يسمى حاوي المستوى الأعلى *top level container*، مثل الإطار *frame*، أو الآبلت *applet*. وغالباً يحتوي حاوي المستوى الأعلى على واحد أو أكثر من الحاويات، مثل اللوحات *panels*. وهذه اللوحات يمكن أن تحتوي لوحات أخرى لتنظيم المكونات الأخرى حسب الطلب.

مفهوم أساسي Key Concept

إن مظهر الجوئي هو أحد وظائف هرم الاحتواء *containment hierarchy* ومدراء التخطيط *layout managers* لكل حاوي.

يمكن أن يملك كل حاوي مدير التخطيط الخاص به. والذي يكمل المظهر النهائي للجوئي هو مدراء التخطيط المختارة لكل حاو، وكذلك تصميم هرم الاحتواء. يوجد العديد من التركيبات الممكنة، ونادراً ما يكون هناك خيار واحد هو الأفضل. وكما هو الحال دائماً، يجب أن نلتزم بأهداف النظام المطلوب والإرشادات العامة لتصميم الجوئي.

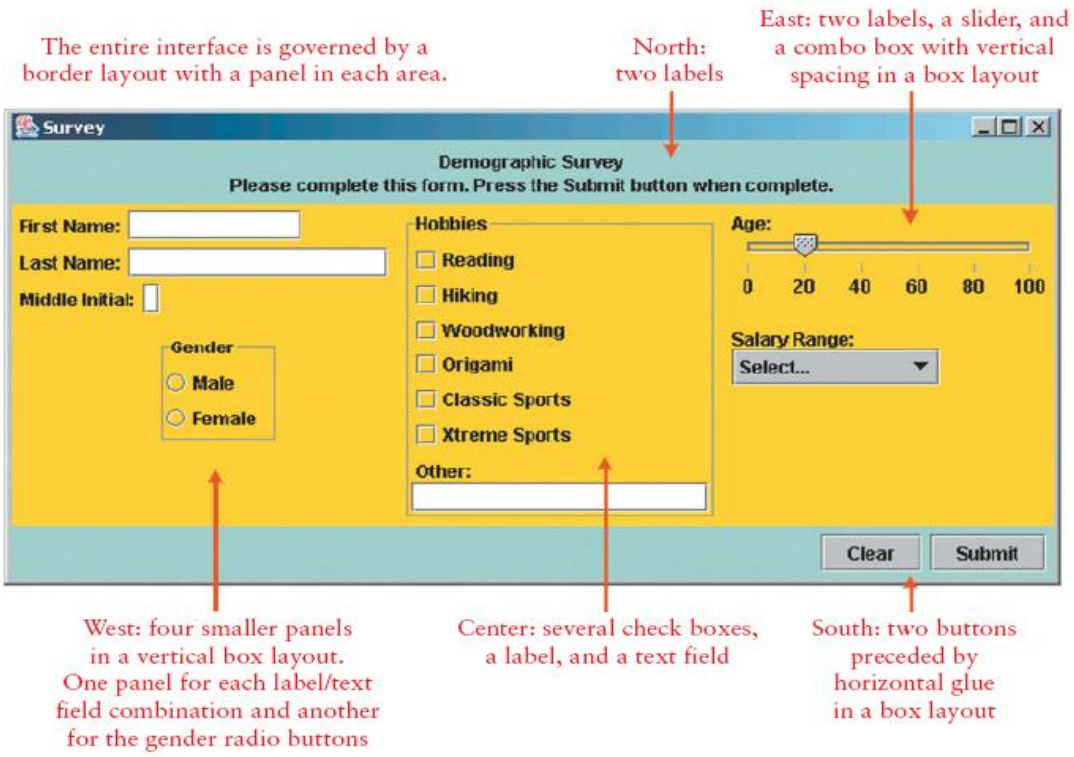
الشكل ٧.٥ يبين تطبيق جوئي تم في حاشيته وصف هرمه الاحتوائي. وقد تم مناقشة العديد من المكونات المستخدمة في هذا البرنامج سابقاً في هذا الكتاب، وهناك مكونات أخرى ستناقش في الفصول القادمة.

ولاحظ أنه في العديد من الحالات، يكون استخدام بعض الحاويات غير واضح بمجرد النظر إلى الجوئي. فاللوحة على وجه التحديد، تكون غير مرئية ما لم نرسم الانتباه إليها بطريقة ما بين المكونات. وتكون هذه المكونات كلها جزءاً من هرم الاحتواء، حتى مع كونها غير مرئية بالنسبة للمستخدم.

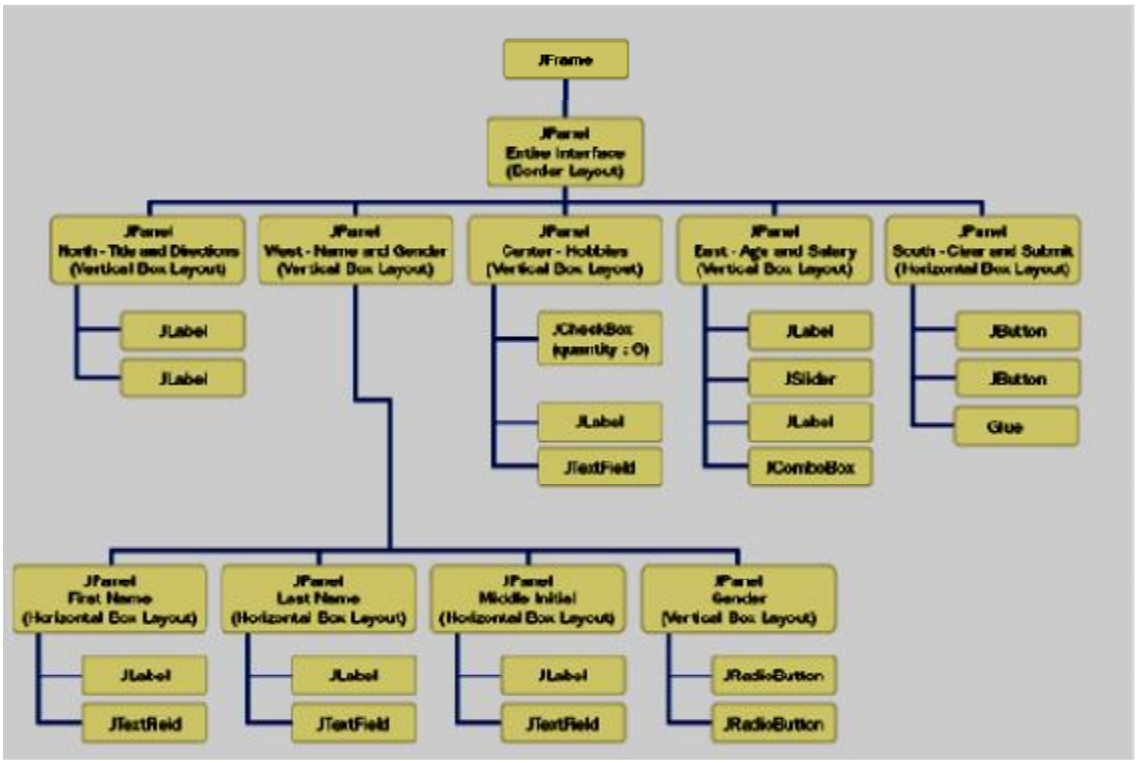
ويمكن تمثيل هرم احتواء لبرنامج معين كهيكل شجرة، كما هو مبين في الشكل ٨.٥ . حيث أن جذر الشجرة يمثل الحاوي ذو المستوى الأعلى. وكل مستوى في الشجرة يظهر الحاويات والمكونات الموجودة في الحاويات للمستوى الأعلى.

عندما يتم عمل تغييرات قد تؤثر على التخطيط المرئي لمكونات البرنامج، يتم استدعاء مدراء التخطيط تبعاً لذلك. حيث أن التغيير في إحداها قد يؤثر على الآخرين. ويتم عكس هذه التغييرات على هرم الاحتواء حسب الحاجة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



الشكل ٧.٥ هرم الاحتواء لجوئي



الشكل ٨.٥ شجرة هرم الاحتواء

الفصل السادس

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

تعتبر المصفوفات *arrays* مفيدة عند رسم الأشكال المعقدة. فالمضلع *polygon*، على سبيل المثال، شكل متعدد الجوانب يتم تعريفه في لغة جافا باستخدام سلسلة من النقاط *(x, y)* التي تمثل نقاط المضلع *vertices*. وغالباً ما تستخدم المصفوفات لتخزين قائمة الإحداثيات.

يتم رسم المضلعات باستخدام دوال موجودة في الكلاس *Graphics*، بشكل مشابه لطريقة رسم المستطيلات والأشكال البيضاوية. فمثل هذه الأشكال الأخرى، يمكن رسم مضلع ممتلئ *filled*، أو غير ممتلئ *unfilled*. والدوال المستخدمة لرسم مضلع هي *drawPolygon* و *fillPolygon*. وكل من هاتين الدالتين لهما إصدارات متعددة. فأحد الإصدارات يستخدم مصفوفات من أعداد صحيحة لتعريف المضلع، والإصدار الآخر يستخدم كائن من الكلاس *Polygon* لتعريف المضلع. وسنناقش الكلاس *Polygon* في وقت لاحق في هذا القسم.

في الإصدار الذي يستخدم المصفوفات، تستقبل الدالتان *drawPolygon* و *fillPolygon* ثلاثة وسائط. الأول هو مصفوفة صحيحة تمثل إحداثيات *x* لنقاط المضلع، والثاني هو مصفوفة صحيحة تمثل إحداثيات *y* لهذه النقاط، والثالث هو عدد صحيح يمثل عدد النقاط المستخدمة لكل مصفوفة. وإذا ما أخذنا الوسيطين الأوليـــن معاً، نجد أنهما يمثلان الإحداثيات *(x, y)* لنقاط المضلع.

دائماً يكون الشكل المضلع مغلقاً. حيث يتم دائماً رسم خط مستقيم من آخر نقطة في القائمة إلى أول نقطة في القائمة.

مفهوم أساسي Key Concept

الشكل متعدد الخطوط *polyline* هو شكل يشبه المضلع *polygon* إلا أن الشكل متعدد الخطوط ليس شكلاً مغلقاً.

وبشكل مشابه للمضلع، يحتوي الشكل متعدد الخطوط *polyline* على سلسلة من النقاط المتصلة بواسطة مقاطع خطية. وتختلف الأشكال متعددة الخطوط عن المضلعات في أن الإحداثيات الأولى والأخيرة لا تتصل تلقائياً عند رسمها. وبما أن الشكل متعدد الخطوط ليس مغلقاً، فإنه لا يمكن ملؤه. ولذلك ليس هناك سوى دالة واحدة تستخدم لرسم الخطوط المتعددة، وهي *drawPolyline*.

وكما هو الحال مع الدالة *drawPolygon*، فإن أول وسيطين من الدالة *drawPolyline* عبارة عن مصفوفات صحيحة. وإذا ما أخذناهما معاً، فإن الوسيطين الأولين يمثلان إحداثيات *(x, y)* للنقاط الطرفية للمقاطع الخطية التي تكون الشكل متعدد الخطوط. أما الوسيط الثالث فهو عدد النقاط في قائمة الإحداثيات.

يوضح البرنامج الموضح في المثال ١.٦ استخدام المضلعات لرسم صاروخ *rocket*. وفي الكلاس *RocketPanel*، والموضح في المثال ٢.٦، تعمل المصفوفتان المسميتان *xRocket* و *yRocket* على تعريف نقاط المضلع التي تشكل الجسم الرئيسي للصاروخ. وتمثل أول نقطة في المصفوفتين النقطة العليا من الصاروخ، والنقاط التالية تمثل نقاط الصاروخ التالية باعتبار أننا نتحرك في اتجاه عقارب الساعة ابتداءً من نقطة الرأس. وتعمل المصفوفتان *xWindow* و *yWindow* على تحديد نقاط المضلع التي تشكل نافذة الصاروخ. وقد تم رسم كل من الصاروخ والنافذة كمضلعات مملوءة.

وتعمل المصفوفتان *xFlame* و *yFlame* على تحديد النقاط الشكل متعدد الخطوط الذي يستخدم لرسم صورة اللهب المنطلق من ذيل الصاروخ. وبسبب رسمنا للهب كشكل متعدد الخطوط، وليس كشكل مضلع، فإن هذا اللهب ليس مغلقاً ولا ممتلئاً.

الكلاس بوليجون The Polygon Class

يمكننا أيضاً تعريف مضلع بشكل صريح باستخدام كائن من الكلاس *Polygon*، والمعرف في حزمة *java.awt* التابعة لمكتبة كلاسات جافا القياسية. ويوجد إصداران من الدوال *drawPolygon* و *fillPolygon* تستقبلان كائن واحد من الكلاس *Polygon* كوسيط لها.

يقوم الكائن *Polygon* بتغليف إحداثيات جوانب المضلع. ويسمح باني الكلاس *Polygon* بإنشاء شكل مضلع فارغ مبدئياً. أو شكل مضلع معرف بواسطة مصفوفات صحيحة تمثل الإحداثيات النقطية. ويحتوي الكلاس *Polygon* على دوال تستخدم لإضافة نقاط إلى المضلع وتحديد ما إذا كانت نقطة معينة محتواة داخل شكل مضلع. كما أنه يحتوي

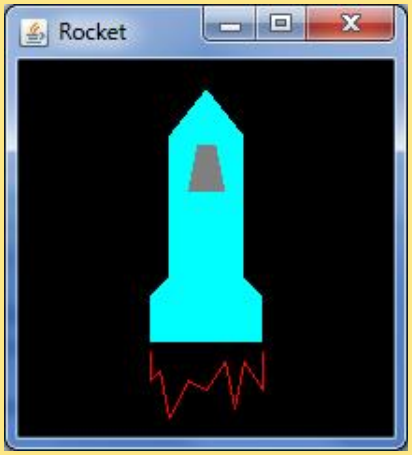
Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

أيضاً على دوال تمكننا من الحصول على تمثيل لمستطيل الإحاطة الخاص بالمضلع، وكذلك دالة لنقل جميع النقاط في المضلع إلى موضع آخر. ويسرد الشكل ١.٦ هذه الدوال.

مثال رقم ١.٦

```
//********************************************************************
//  Rocket.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام المضلعات ومتعددات الخطوط
//********************************************************************

import javax.swing.JFrame;

public class Rocket
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء الإطار الرئيسي للبرنامج
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Rocket");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      RocketPanel panel = new RocketPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مثال رقم ٢.٦

```
//********************************************************************
//  RocketPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرماج استخدام المضلعات ومتعددات الخطوط
//********************************************************************


import javax.swing.JPanel;
import java.awt.*;

public class RocketPanel extends JPanel
{
   private int[] xRocket = {100, 120, 120, 130, 130, 70, 70, 80, 80};
   private int[] yRocket = {15, 40, 115, 125, 150, 150, 125, 115, 40};

   private int[] xWindow = {95, 105, 110, 90};
   private int[] yWindow = {45, 45, 70, 70};

   private int[] xFlame = {70, 70, 75, 80, 90, 100, 110, 115, 120,
                           130, 130};
   private int[] yFlame = {155, 170, 165, 190, 170, 175, 160, 185,
                           160, 175, 155};

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة الخصائص الأساسية لهذه اللوحة
   //-----------------------------------------------------------------
   public RocketPanel()
   {
      setBackground (Color.black);
      setPreferredSize (new Dimension(200, 200));
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية برسم الصاروخ باستخدام المضلعات ومتعددات
   //  الخطوط
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);

      page.setColor (Color.cyan);
      page.fillPolygon (xRocket, yRocket, xRocket.length);

      page.setColor (Color.gray);
      page.fillPolygon (xWindow, yWindow, xWindow.length);

      page.setColor (Color.red);
      page.drawPolyline (xFlame, yFlame, xFlame.length);
   }
}
```

`Polygon()`

الباني: يقوم بإنشاء مضلع فارغ.

`Polygon(int[] xpoints, int[] ypoints, int npoints)`

الباني: يقوم بإنشاء مضلع باستخدام زوج الإحداثيات (x, y) في المدخلات المتناظرة في xpoints و ypoints.

`void addPoint(int x, int y)`

تضيف النقطة المحددة إلى هذا المضلع.

`boolean contains(int x, int y)`

ترجع true إذا كانت النقطة المحددة محتواة في هذا المضلع.

`boolean contains(Point p)`

ترجع true إذا كانت النقطة المحددة محتواة في هذا المضلع.

`Rectangle getBounds()`

ترجع مستطيل الإحاطة الخاص بهذا المضلع.

`void translate(int deltaX, int deltaY)`

تنقل نقاط هذا المضلع بإزاحة مقدارها deltaX في المحور السيني ومقدارها deltaY في المحور الصادي.

الشكل رقم ١.٦ بعض دوال الكلاس بوليجون Polygon.

٢.٦ أحداث الماوس MOUSE EVENTS

دعنا الآن ننظر في الأحداث التي يتم إنشاؤها عند استخدام الماوس. تقسم لغة جافا هذه الأحداث إلى فئتين: أحداث الماوس *mouse events* وأحداث حركة الماوس *mouse motion events*. ويعرف الجدول الموضح في الشكل ٢.٦ هذه الأحداث.

عندما تنقر بزر الماوس على عنصر من عناصر جوئي الجافا، فسيتم توليد ثلاثة أحداث: واحد عندما يتم الضغط على زر الماوس (*mouse pressed*) واثنين عندما تحرر الزر (*mouse released* و *mouse clicked*). ويعرف النقر بالماوس على أنه عملية الضغط والتحرير لزر الماوس في نفس الموقع. أما إذا ضغطت زر الماوس لأسفل، وحركته، ثم حررته، فلن يتم إنشاء حدث النقر بالماوس.

ويقوم المكون بتوليد حدث دخول الماوس (*mouse entered*) عندما يمر مؤشر الماوس فوق المنطقة الرسومية لهذا المكون. وبالمثل، يتم توليد حدث خروج الماوس (*mouse exited*) عندما يترك مؤشر الماوس المنطقة الرسومية للمكون.

مفهوم أساسي Key Concept

إن تحريك الماوس أو ضغط أزراره يولد أحداثاً يمكن للبرنامج الاستجابة لها.

أما أحداث حركة الماوس (*mouse motion events*)، كما هو واضح من اسمها، فتظهر طالما كان الماوس في حالة حركة. فحدث حركة الماوس (*mouse moved*) يشير ببساطة إلى أن الماوس في حالة حركة. وحدث سحب الماوس (*mouse dragged*) يتم توليده عندما يقوم المستخدم بضغط زر الماوس للأسفل وتحريكه دون تحريره. ويتم توليد أحداث حركة الماوس عدة مرات، وبسرعة، طالما أن الماوس في حالة حركة.

وقد نكون في بعض المواقف المخصصة، مهتمين فقط بحدث أو حدثين من أحداث الماوس. وسيعتمد ما نستمع إليه من أحداث على ما نحاول أن ننجزه.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

حدث الماوس *Mouse Event*	الوصف *Description*
ضغط الماوس *mouse pressed*	عندما يتم ضغط زر الماوس للأسفل.
تحرير الماوس *mouse released*	عندما يتم تحرير زر الماوس.
نقر الماوس *mouse clicked*	عندما يتم ضغط زر الماوس للأسفل وتحريره بدون تحريكه بين هاتين العمليتين.
دخول الماوس *mouse entered*	عندما يتحرك مؤشر الماوس على (فوق) المكون.
خروج الماوس *mouse exited*	عندما يتحرك مؤشر الماوس بعيداً (خارج) المكون.

حدث حركة الماوس *Mouse Motion Event*	الوصف *Description*
حركة الماوس *mouse moved*	عندما يتحرك الماوس
سحب الماوس *mouse dragged*	عندما يتحرك الماوس في حين يكون زره مضغوطاً للأسفل.

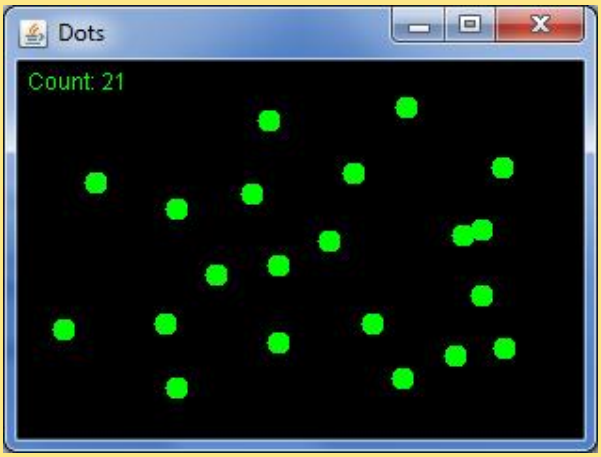
الشكل ٢.٦ أحداث الماوس mouse events وأحداث حركة الماوس mouse motion events

يقوم البرنامج *Dots*، الموضح في المثال ٣.٦، بالاستجابة لحدث واحد من أحداث الماوس. فهو، تحديداً، يقوم برسم نقطة خضراء في موقع مؤشر الماوس متى ما تم ضغطه.

مثال رقم ٣.٦

```
//********************************************************************
//  Dots.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يقوم هذا البرنامج بشرح أحداث الماوس
//********************************************************************

import javax.swing.JFrame;

public class Dots
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء وعرض إطار التطبيق
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Dots");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new DotsPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



تقوم الدالة الرئيسية *main* في الكلاس *Dots* بإنشاء الإطار وإضافة اللوحة إليه. ويتم تعريف اللوحة من خلال الكلاس *DotsPanel* الموضح في المثال ٤.٦ .

مثال رقم ٤.٦

```
//********************************************************************
//  DotsPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا البرنامج اللوحة الأساسية لبرنامج النقاط
//********************************************************************

import java.util.ArrayList;
import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class DotsPanel extends JPanel
{
   private final int SIZE = 6;  // radius of each dot

   private ArrayList<Point> pointList;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة هذه اللوحة للاستماع لأحداث الماوس
   //-----------------------------------------------------------------
   public DotsPanel()
   {
      pointList = new ArrayList<Point>();

      addMouseListener (new DotsListener());

      setBackground (Color.black);
      setPreferredSize (new Dimension(300, 200));
   }
```

```
   //-----------------------------------------------------------------
   //  ترسم هذه الدالة كل النقاط المخزنة في القائمة
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent(page);

      page.setColor (Color.green);

      for (Point spot : pointList)
         page.fillOval (spot.x-SIZE, spot.y-SIZE, SIZE*2, SIZE*2);

      page.drawString ("Count: " + pointList.size(), 5, 15);
   }

   //*****************************************************************
   //  يمثل هذا الكلاس مستمعاً لأحداث الماوس
   //*****************************************************************
   private class DotsListener implements MouseListener
   {
      //--------------------------------------------------------------
      //  تقوم هذه الدالة بإضافة النقطة الحالية لقائمة النقاط وتعيد رسم
      //  اللوحة كلما تم ضغط زر الماوس
      //--------------------------------------------------------------
      public void mousePressed (MouseEvent event)
      {
         pointList.add(event.getPoint());
         repaint();
      }

      //--------------------------------------------------------------
      //  التزويد بتعريفات فارغة لدوال الأحداث الغير مستخدمة
      //--------------------------------------------------------------
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
   }
}
```

تحتفظ اللوحة *DotsPanel* بقائمة من كائنات *Point* التي تمثل جميع المواقع التي قام المستخدم بنقر زر الماوس عندها. يمثل الكلاس *Point* إحداثيات *(x, y)* لنقطة معينة في الفضاء ثنائي الأبعاد. وهذا الكلاس يوفر وصولاً عاماً إلى المتغيرات *x* و *y* لنقطة. وفي كل مرة يتم فيها رسم لوحة، يتم رسم جميع النقاط المخزنة في القائمة.

يتم حفظ قائمة من كائنات الـ *Point* ككائن من الكلاس *ArrayList*. وكي نكون دقيقين أكثر، فإن نوع الكائن *pointList* هو *ArrayList<Point>*، مع العلم أنه يمكن تخزين كائنات الـ *Point* فقط في الـ *ArrayList*. ولرسم النقاط، نستخدم حلقة *for* لتدور على كافة النقاط المخزنة في القائمة.

يتم تعريف مستمع حدث ضغط الماوس ككلاس داخلي خاص ينفذ الواجهة *MouseListener*. ثم يتم استدعاء الدالة *mousePressed* بواسطة اللوحة في كل مرة يضغط المستخدم على زر الماوس عندما يكون فوق اللوحة.

يظهر حدث الماوس دائماً في نقطة ما في الفضاء ثنائي الأبعاد، ويحتفظ الكائن الذي يمثل هذا الحدث بذلك الموقع. ويمكننا في مستمع الماوس، أن نحصل على هذه النقطة أو أن نستخدمها كلما دعت الحاجة إلى ذلك. وفي برنامج النقاط *Dots*، يتم في كل مرة تستدعى فيها الدالة *mousePressed*، الحصول على موقع الحدث باستخدام الدالة *getPoint* الخاصة بالكائن *MouseEvent*. يتم تخزين هذه النقطة في الـ *ArrayList*، ويتم إعادة رسم اللوحة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مفهوم أساسي Key Concept

يجب أن نزود في المستمع بتعريفات دوال فارغة للأحداث الغير ضرورية لتلبية متطلبات الواجهة.

لاحظ أنه، وبخلاف واجهات المستمع التي استخدمناها في الأمثلة السابقة التي تحتوي على دالة واحدة في كل واحدة منها، فإن الواجهة *MouseListener* تحتوي على خمس دوال. وفي هذا البرنامج، كان الحدث الوحيد الذي يهمنا هو حدث ضغط الماوس. ولذلك، فإن الدالة الوحيدة التي تهمنا هي *mousePressed*. ومع ذلك، فإن تنفيذ الواجهة يعني وجوب التزويد بتعريفات لكل دوال الواجهة. ولذلك قمنا بالتزويد بتعريفات فارغة للدوال المرتبطة بالأحداث الأخرى. وعندما تتولد هذه الأحداث، يتم استدعاء الدوال الفارغة، لكن لا يوجد شفرة للتنفيذ. وسنناقش في الفصل السابع تقنية إنشاء مستمعات تجنبنا إنشاء مثل هذه التعريفات الفارغة.

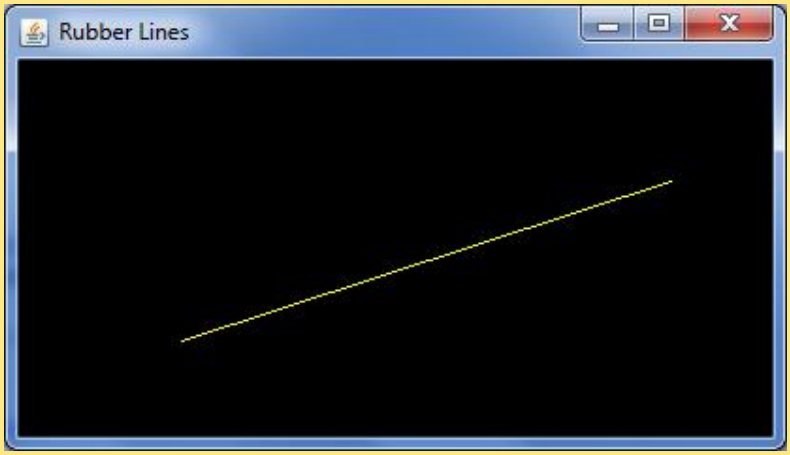
دعنا ننظر في هذا المثال والذي يستجيب لحدثين موجهين من قبل الماوس. فالبرنامج *RubberLines* الموضح في المثال ٥.٦ يرسم خطاً بين نقطتين. النقطة الأولى تحدد بالموضع الذي تم فيه ضغط الماوس أولاً. والنقطة الثانية تتغير كلما سحبنا الماوس مع استمرار الضغط على زره. عندما يتم تحرير الماوس، يظل الخط ثابتاً بين النقطة الأولى والثانية. وعندما يتم الضغط مجدداً على زر الماوس، فإن خطاً جديداً سيتولد.

ويمثل البرنامج *RubberLinesPanel* الموضح في المثال ٦.٦ اللوحة التي يتم عليها رسم الخطوط. نحتاج هنا لمستمع يستجيب لحدث الضغط على الماوس وكذا حدث سحب الماوس، وذلك لأن هذين الحدثين داخلان في بناء هذا البرنامج. لاحظ أن الكلاس المستمع في هذا المثال ينفذ واجهتين، هما: *MouseListener* و *MouseMotionListener*. وذلك فإنه يتوجب علينا هنا تنفيذ كل الدوال في كلا الواجهتين. إن الدالتين اللتين نحن مهتمين بهما هما: *mousePressed* و *mouseDragged*، ولذا تم تنفيذهما لتحقيق الهدف من البرنامج، أما بقية الدوال فقد أعطيت تعريفات فارغة لتحقيق متطلبات الواجهة.

مثال رقم ٥.٦

```
//********************************************************************
//  RubberLines.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يقوم هذا البرنامج بشرح أحداث الماوس والرباط المطاطي
//********************************************************************

import javax.swing.JFrame;

public class RubberLines
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء وعرض إطار التطبيق
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Rubber Lines");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new RubberLinesPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



مثال رقم ٦.٦

```
//********************************************************************
//   RubberLinesPanel.java          المؤلف: Lewis/Loftus     المترجم : Almashraqi
//
//  يمثل هذا البرنامج لوحة الرسم الأساسية لبرنامج الخطوط المطاطية
//********************************************************************

import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class RubberLinesPanel extends JPanel
{
   private Point point1 = null, point2 = null;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة هذه اللوحة للاستماع لأحداث الماوس
   //-----------------------------------------------------------------
   public RubberLinesPanel()
   {
      LineListener listener = new LineListener();
      addMouseListener (listener);
      addMouseMotionListener (listener);

      setBackground (Color.black);
      setPreferredSize (new Dimension(400, 200));
   }
```

```
   //-----------------------------------------------------------------
   // الابتدائية تقوم الدالة التالية برسم الخط الحالي من نقطة ضغط الماوس
   //  إلى الموضع الحالي للماوس
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);

      page.setColor (Color.yellow);
      if (point1 != null && point2 != null)
         page.drawLine (point1.x, point1.y, point2.x, point2.y);
   }
   //*****************************************************************
   //  يمثل هذا الكلاس مستمعاً لكل أحداث الماوس
   //*****************************************************************
   private class LineListener implements MouseListener,
                                         MouseMotionListener
   {
      //--------------------------------------------------------------
      //  تقوم هذه الدالة بالتقاط الموضع الابتدائي الذي تم فيه ضغط
      //  زر الماوس
      //--------------------------------------------------------------
      public void mousePressed (MouseEvent event)
      {
         point1 = event.getPoint();
      }
      //--------------------------------------------------------------
      //  تقوم هذه الدالة بالحصول على الموضع الحالي للماوس عند سحبه
      //  وتعيد رسم الخط لإنشاء ما يشبه تأثير الرباط المطاطي
      //--------------------------------------------------------------
      public void mouseDragged (MouseEvent event)
      {
         point2 = event.getPoint();
         repaint();
      }
      //--------------------------------------------------------------
      //  التزويد بتعريفات فارغة لدوال الأحداث الغير مستخدمة
      //--------------------------------------------------------------
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
      public void mouseMoved (MouseEvent event) {}
   }
}
```

مفهوم أساسي Key Concept

الربط المطاطي *rubberbanding* هو التأثير الرسومي الذي يحدث عندما يبدو أن الشكل يتمدد عند سحبنا للماوس.

عندما يتم استدعاء الدالة *mousePressed*، يتم ضبط قيمة المتغير *point1*. ثم، في أثناء سحب الماوس، يتم باستمرار ضبط قيمة المتغير *point2* وإعادة رسم اللوحة. لذلك فإن الخط يتم رسمه بشكل ثابت في أثناء تحرك الماوس، مما يعطينا مظهراً فيه نرى خطاً واحداً يتمدد بين نقطة ثابتة وأخرى متحركة. وهذا التأثير يسمى الرباط المطاطي *rubberbanding* وهو شائع في البرامج الرسومية.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

لاحظ أنه، في باني الكلاس *RubberLinesPanel*، تم إضافة كائن الاستماع إلى اللوحة مرتين: مرة كمستمع لأحداث الماوس، وأخرى كمستمع لأحداث حركة الماوس. ويجب أن تتوافق الدالة التي يتم نداؤها لإضافة المستمع مع الكائن الممر كوسيط. في هذا الحالة، لدينا كائن واحد يعمل كمستمع لكلا النوعين من الأحداث. ويمكن أن ننشأ كلاسين مستمعين لو أردنا: أحدهما للاستماع لأحداث الماوس والآخر للاستماع لأحداث حركة الماوس. ويمكن للمكون الواحد أن يملك مستمعات متعددة للأنواع المختلفة من الأحداث.

كذلك لاحظ أن هذا البرنامج يرسم خطاً واحداً في كل مرة. بمعنى أنه عندما يبدأ المستخدم برسم خط آخر بنقرة جديدة على الماوس، فإن الخط الأول يختفي. وهذا يحدث بسبب أن الدالة *paintComponent* تعيد رسم خلفيتها، ملغية الخط السابق في كل مرة. أما إذا أردنا رؤية الخطوط السابقة، فيجب أن نحتفظ بهم، ربما باستخدام *ArrayList* كما فعلنا في برنامج النقاط *Dots*. وسنترك هذا التعديل على برنامج الخطوط المطاطية *RubberLines* كمشروع برمجي يقوم الطالب بعمله.

٣.٦ أحداث المفتاح KEY EVENTS

مفهوم أساسي Key Concept

تسمح أحداث المفتاح *key events* للبرنامج بالاستجابة الفورية للمستخدم عند ضغطه لمفاتيح لوحة المفاتيح.

يتم توليد حدث المفتاح *key event* عندما يتم الضغط على مفتاح في لوحة المفاتيح. تسمح أحداث المفتاح للبرنامج أن يستجيب فوراً للمستخدم عندما يقوم بالطباعة أو بالضغط على مفاتيح أخرى في لوحة المفاتيح، مثل مفاتيح الأسهم *arrows keys*. عندما يتم معالجة أحداث المفتاح، فإن البرنامج يمكنه الاستجابة للحدث في نفس الوقت الذي تم فيه الضغط على المفتاح؛ أي أنه ليس هناك حاجة للانتظار حتى يتم ضغط المفتاح *Enter* أو انتظار تفعيل أي مكون آخر (مثل الزر).

يستجيب البرنامج *Direction* الموضح في المثال ٧.٦ لأحداث المفتاح. ففيه يتم عرض صورة سهم وتتحرك هذه الصورة عبر الشاشة كلما ضغطنا على مفاتيح الأسهم. وفعلياً، تم استخدام أربع صور مختلفة، بحيث توجد صورة لكل سهم يمثل الاتجاهات الأساسية (فوق *up*، تحت *down*، يمين *right*، شمال *left*).

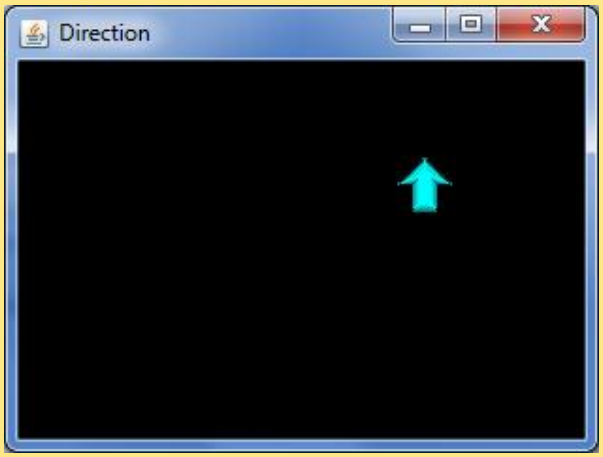
ويمثل الكلاس *DirectionPanel*، الموضح في المثال ٨.٦، اللوحة التي يتم عليها عرض صورة السهم. يقوم الباني بتحميل صور الأسهم الأربعة، وأحد هذه الصور يعتبر دائماً الصورة الحالية *current image* (أي الصورة المعروضة). ويتم ضبط قيمة متغير الصورة الحالية اعتماداً على أحدث ضغطة سهم تم القيام بها. مثلاً، إذا تم ضغط السهم الأعلى *up*، فسيتم عرض صورة السهم الذي يشير إلى الأعلى. أما إذا تم ضغط أحد الأسهم بشكل مستمر، فستتحرك الصور المناسبة في الاتجاه المناسب.

مثال رقم ٧.٦

```
//********************************************************************
//  Direction.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج أحداث المفاتيح
//********************************************************************

import javax.swing.JFrame;

public class Direction
{
   //-----------------------------------------------------------------
   //  تقوم الدالة الرئيسية بإنشاء وعرض إطار التطبيق
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Direction");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
```

```
      frame.getContentPane().add (new DirectionPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



مثال رقم ٨.٦

```
//********************************************************************
// DirectionPanel.java      المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا البرنامج لوحة العرض الأساسية لبرنامج الاتجاه
//********************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DirectionPanel extends JPanel
{
   private final int WIDTH = 300, HEIGHT = 200;
   private final int JUMP = 10;  // increment for image movement

   private final int IMAGE_SIZE = 31;

   private ImageIcon up, down, right, left, currentImage;
   private int x, y;

   //-----------------------------------------------------------------
   //  يقوم الباني بتهيئة هذه اللوحة وتحميل الصور
   //-----------------------------------------------------------------
   public DirectionPanel()
   {
      addKeyListener (new DirectionListener());

      x = WIDTH / 2;
      y = HEIGHT / 2;
```

```
      up = new ImageIcon ("arrowUp.gif");
      down = new ImageIcon ("arrowDown.gif");
      left = new ImageIcon ("arrowLeft.gif");
      right = new ImageIcon ("arrowRight.gif");

      currentImage = right;

      setBackground (Color.black);
      setPreferredSize (new Dimension(WIDTH, HEIGHT));
      setFocusable(true);
   }

   //-----------------------------------------------------------------
   //  تقوم هذه الدالة برسم الصور في الموقع الحالي
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);
      currentImage.paintIcon (this, page, x, y);
   }

   //*****************************************************************
   //  تمثل هذه الدالة مستمعاً لنشاط لوحة المفاتيح
   //*****************************************************************
   private class DirectionListener implements KeyListener
   {
      //--------------------------------------------------------------
      //  تستجيب هذه الدالة لضغط المستخدم لمفاتيح الأسهم
      //  وتعيد ضبط الصورة وموقعها وفقاً لذلك
      //--------------------------------------------------------------
      public void keyPressed (KeyEvent event)
      {
         switch (event.getKeyCode())
         {
            case KeyEvent.VK_UP:
               currentImage = up;
               y -= JUMP;
               break;
            case KeyEvent.VK_DOWN:
               currentImage = down;
               y += JUMP;
               break;
            case KeyEvent.VK_LEFT:
               currentImage = left;
               x -= JUMP;
               break;
            case KeyEvent.VK_RIGHT:
               currentImage = right;
               x += JUMP;
               break;
         }

         repaint();
      }
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
    //-------------------------------------------------------------
    //  التزويد بتعريفات فارغة لدوال الحدث الغير مستخدمة
    //-------------------------------------------------------------
    public void keyTyped (KeyEvent event) {}
    public void keyReleased (KeyEvent event) {}
  }
}
```

تم توليد صور الأسهم ككائنات *ImageIcon*. ففي هذا المثال، يتم رسم الصورة باستخدام الدالة *paintIcon* في كل مرة يتم فيها إعادة رسم اللوحة. تستقبل الدالة *paintIcon* أربعة وسائط: مكون يمثل مراقب الصورة *image observer*، والسياق الرسومي الذي به يتم رسم الصورة، والإحداثيات *(x, y)* لتحديد مكان رسم الصورة. ومراقب الصورة هو مكون يعمل على إتمام مهمة تحميل الصورة؛ وفي هذه الحالة استخدمنا اللوحة *panel* كمراقب للصورة.

تم تهيئة الكلاس الداخلي الخاص المسمى *DirectionListener* بحيث يستجيب لأحداث المفتاح. فهو ينفذ الواجهة *KeyListener*، والتي تعرف ثلاث دوال يمكننا استخدامها للاستجابة لنشاط لوحة المفاتيح. والشكل ٣.٦ يسرد هذه الدوال.

`void keyPressed(KeyEvent event)`	تستدعى عندما يتم ضغط مفتاح.
`void keyReleased(KeyEvent event)`	تستدعى عندما يتم تحرير مفتاح.
`void keyTyped(KeyEvent event)`	تستدعى عندما يقوم مفتاح مضغوط أو مفتاح مركب بتوليد حرف مفتاحي.

الشكل رقم ٣.٦ دوال الواجهة KeyListener.

بشكل خاص، يستجيب البرنامج *Direction* لأحداث ضغط المفتاح *key pressed events*. وبسبب أن الكلاس المستمع يجب أن ينفذ كل الدوال المعرفة في الواجهة، فقد قمنا بالتزويد بدوال فارغة للأحداث الأخرى.

يمكن استخدام الكائن *KeyEvent* الذي يتم تمريره إلى الدالة *keyPressed* الخاصة بالمستمع في تحديد أي من المفاتيح تم ضغطه. ففي المثال، قمنا باستدعاء الدالة *getKeyCode* الخاصة بكائن الحدث للحصول على كود رقمي يمثل المفتاح الذي قمنا بالضغط عليه. وقد استخدمنا عبارة *switch* لتحديد المفتاح الذي تم الضغط عليه و الاستجابة تبعاً لذلك. و يحتوي الكلاس *KeyEvent* على ثوابت مرتبطة بكل كود رقمي يتم استرجاعه من الدالة *getKeyCode*. وإذا تم ضغط أي مفتاح غير مفتاح السهم، فإنه يتم تجاهله.

يتم إطلاق أحداث المفاتيح كلما تم الضغط على مفتاح، ولكن معظم الأنظمة تمكن من تطبيق مفهوم تكرار المفتاح *key repetition*. وذلك يحدث عندما يتم الضغط على مفتاح مع الاستمرار، كما لو أننا قمنا بضغط ذلك المفتاح بشكل متكرر وبسرعة. يتم توليد أحداث المفاتيح بنفس الطريقة. ففي برنامج الاتجاه *Direction*، يمكن للمستخدم الضغط باستمرار على مفتاح السهم ومراقبة تنقل الصور عبر الشاشة بسرعة.

إن المكون الذي يولد أحداث المفاتيح هو ذلك الذي عليه تركيز لوحة المفاتيح حالياً *keyboard focus*. وعادة ما يتم الحصول على تركيز لوحة المفاتيح من قبل المكون الأساسي "النشط". وعادة يحصل المكون على تركيز لوحة المفاتيح عندما ينقر المستخدم عليه بالماوس. وعليه فإن استدعاء الدالة *setFocusable* في باني اللوحة يسبب في جعل تركيز لوحة المفاتيح على اللوحة.

لا يضع برنامج الاتجاه *Direction* حدوداً لصورة السهم، بحيث يمكن تحريكها خارج النافذة المرئية، ثم تعود مرة أخرى إذا لزم ذلك. يمكنك إضافة شفرة إلى المستمع لوقف الصورة عندما تصل إلى أحد حدود النافذة. وسنترك لك هذا الأمر كتطبيق برمجي.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الفصل السابع

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

١.٧ هرم كلاس المكونات THE COMPONENT CLASS HIERARCHY

تعتبر كل كلاسات الجافا التي تعرف مكونات الجوئي جزءاً من هرم الكلاس المبين جزئياً في الشكل ١.٧. حيث يتم اشتقاق معظم مكونات الجوئي التي نوعها *Swing* من الكلاس المسمى *JComponent*، والذي يعرف كل المكونات العاملة بشكل عام. والكلاس *JComponent* مشتق من الكلاس *Container*، وهو بدوره مشتق من الكلاس *Component*.

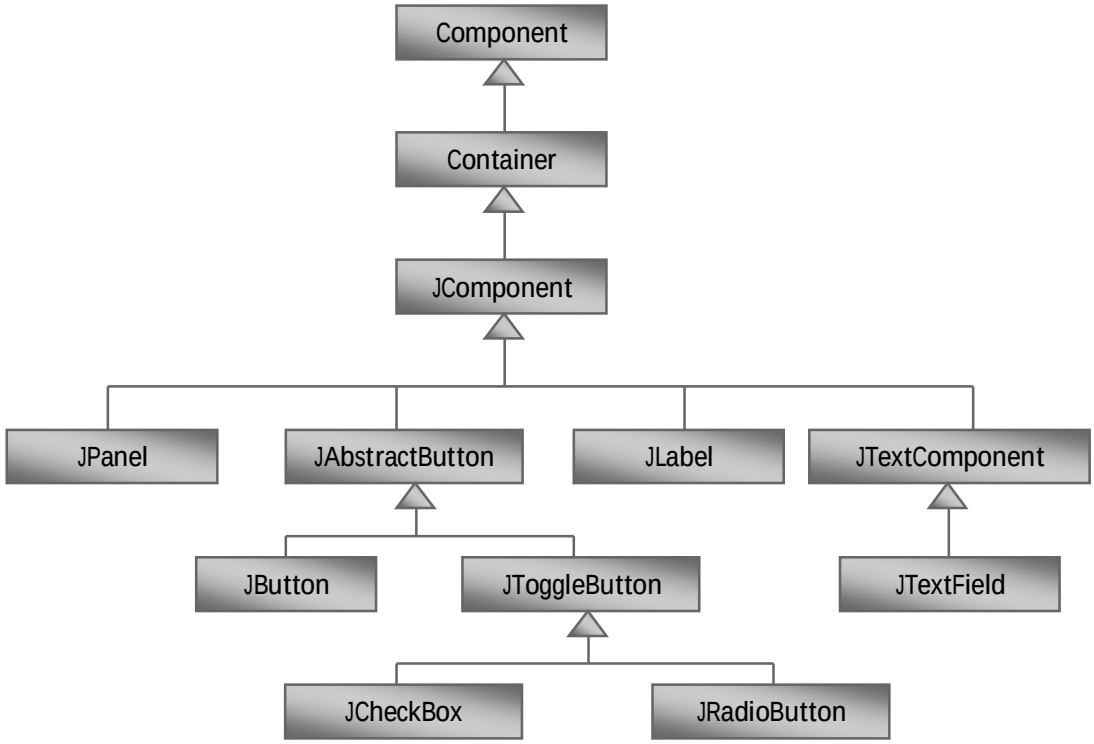
مفهوم أساسي Key Concept

يتم تنظيم الكلاسات التي تمثل مكونات الجوئي في لغة الجافا في شكل هرم كلاس *class hierarchy*.

ولعلك تتذكر أن هناك حزمتان للجوئي الأساسية في لغة جافا، هما: أداة النوافذ المجردة *Abstract Windowing Toolkit (AWT)* وكلاسات *Swing*. وتمثل *AWT* المجموعة الأصلية للكلاسات الرسومية في لغة جافا. أما كلاسات *Swing*، والتي تحدثنا عنها سابقاً، فهي تضيف مكونات تزودنا بأداء وظيفي أكثر من مكونات *AWT* المناظرة. وسنقوم في أمثلة هذا الكتاب باستخدام مكونات *Swing*. ونلاحظ في هرم الكلاس للمكون، أن بعض كلاسات *Swing* في الأساس مشتقة من كلاسات *AWT*.

كلا الكلاسين *Container* و *Component* يعتبران من كلاسات *AWT* الأصلية. ويحتوي الكلاس *Component* على كثير من الوظائف العامة التي تنطبق على كل مكونات الجوئي، مثل وظائف الرسم الأساسية ومعالجة الأحداث. لذلك فإنه بالرغم من أننا قد نفضل استخدام بعض مكونات *Swing* المخصصة، إلا أنها معتمدة على مفاهيم *AWT* الأساسية وتستجيب لنفس الأحداث التي تستجيب لها مكونات *AWT*. وبسبب كون العديد من مكونات *Swing* مشتقة من الكلاس *Container* فإنها يمكن أن تستخدم كحاويات، ومع ذلك فإن هذه الإمكانات في أغلب الظروف محدودة. مثلاً، رأينا أن الكائن *JLabel* يمكن أن يحوي صورة *image*، ولكنه لا يمكن أن يستخدم كحاوي عمومي يمكن إضافة المكونات إليه.

تم تعريف العديد من المزايا التي تنطبق على كل مكونات *JComponent* وتم توريث هذه المزايا إلى الكلاسات الأبناء. مثلاً، لدينا إمكانية وضع حد *border* على أي مكون *Swing* (كما رأينا في الفصل الخامس). وهذه الإمكانية عرفت مرة واحدة في الكلاس *JComponent* وتم توريثها لكل كلاس مشتق من هذا الكلاس، سواء بشكل مباشر أو غير مباشر.



الشكل رقم ١.٧ جزء من هرم الكلاس لمكونات الجوئي.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

بعض كلاسات المكونات، مثل *JPanel* و *JLabel* مشتقة مباشرة من الكلاس *JComponent*، بينما هناك كلاسات مكونات أخرى تتواجد في مستويات أدنى من بنية الهرم الوراثي. مثلاً، الكلاس *JAbstractButton* هو كلاس مجرد يعرف الوظائف التي تنطبق على كل أنواع أزرار الجوئي. حيث يتم اشتقاق الكلاس *JButton* منه. مع ذلك، لاحظ أن الكلاسين *JCheckBox* و *JRadioButton* كلاهما مشتقان من الكلاس المسمى *JToggleButton*، والذي يجسد الخصائص المشتركة للأزرار التي يمكن أن تكون في حالة واحدة من حالتين. وتوضح مجموعة الكلاسات التي تعرف أزرار الجوئي مرة أخرى كيف أن الخصائص المشتركة توضع بشكل مناسب في المستويات الأعلى من هرم الكلاسات بدلاً من تكرار هذه الخصائص في كل الكلاسات في المستويات الأدنى لهرم الكلاسات.

إن عالم المكونات النصية يشرح هذا الأمر بشكل جيد. فالكلاس *JTextField* الذي استخدمناه في أمثلة سابقة هو واحد من العديد من مكونات الجوئي في لغة الجافا والذي يدعم إدارة البيانات النصية. وهذا الكلاس يندرج في الهرم تحت الكلاس المسمى *JTextComponent*. لا تنس أن هناك العديد من كلاسات مكونات الجوئي لم تظهر في المخطط الموضح بالشكل ١.٧.

قمنا في الفصول السابقة بتوسيع هرم كلاس المكونات بشكل أكبر عن طريق تعريفات كلاسات لوحات وكلاسات آبلتات باستخدامنا الخاص لمفهوم الوراثة. أننا بوراثة الكلاس *JPanel* أو الكلاس *JApplet*، نقوم بإنشاء كلاسات خاصة بنا تملك تلقائياً خصائص تلك المكونات. وفي بعض الأحيان كنا نقوم بإعادة تعريف دالة، مثل الدالة *paintComponent*، وذلك كي نجعلها تسلك سلوكاً مخصصا يناسب برنامجنا.

إن إنشاءنا لكلاسات لوحات وآبلتات خاصة بنا هو استخدام كلاسيكي لمفهوم الوراثة *inheritance*، وهذا المفهوم يسمح للكلاس الأب *parent class* أن يأخذ على عاتقه المسئوليات التي تنطبق على كل الأبناء *descendants*. فمثلاً، تم تصميم الكلاس *JApplet* مسبقاً للتعامل مع كل التفاصيل المتعلقة بإنشاء الآبلت وتنفيذه. يتفاعل الآبلت مع مستعرض الإنترنت *browser*، ويمكنه استقبال وسطاء من خلال شفرة *HTML*، ولكنه يكون مقيداً بقيود حماية معينة. يهتم الكلاس *JApplet* مسبقاً بهذه التفاصيل بطريقة عمومية تنطبق على كل الآبلتات. ويكون كلاس الآبلت الذي نكتبه (والمشتق من الـ *JApplet*) جاهزاً للتركيز على الغرض أو الهدف من ذلك البرنامج المخصص. وبكلمات أخرى يمكننا القول أن القضايا الوحيدة التي تطرقنا لها في كود الآبلت الخاص بنا هي تلك القضايا التي تجعله مختلفا عن الآبلتات الأخرى.

٢.٧ وراثة الكلاسات المكيفة EXTENDING ADAPTER CLASSES

في الأمثلة السابقة المعتمدة على الأحداث، قمنا بإنشاء الكلاسات المستمعة عن طريق تنفيذ واجهة مستمع مخصصة. على سبيل المثال، لإنشاء كلاس يستمع لأحداث الماوس، قمنا بإنشاء كلاس مستمع ينفذ الواجهة *MouseListener*. وكما رأينا في البرنامج *Dots* والبرنامج *RubberLines* في الفصل السادس، فإن واجهة المستمع غالباً ما تحتوي على دوال أحداث ليست مهمة لبرنامج معين، وفي هذه الحالة نقوم بالتزويد بتعريفات فارغة لهذه الدوال لتحقيق متطلبات الواجهة.

مفهوم أساسي Key Concept

يمكن إنشاء الكلاس المستمع عن طريق اشتقاقه من كلاس مكيف الحدث *event adapter*.

ومن التقنيات البديلة لإنشاء كلاس مستمع وراثة كلاس مكيف الحدث *event adapter*. إن أي واجهة مستمع تحتوي أكثر من دالة تملك كلاساً يحتوي مسبقاً على تعريفات فارغة لكل دوال هذه الواجهة. ولكي نقوم بإنشاء المستمع، يمكننا اشتقاق كلاس مستمع جديد من الكلاس المكيف المناسب وإعادة تعريف أي دوال من دوال الأحداث التي نكون مهتمين بها. وباستخدامنا لهذه التقنية، لن نكون بحاجة للتزويد بتعريفات فارغة للدوال الغير مستخدمة.

يعرض البرنامج المبين في المثال ١.٧ لوحة تستجيب لأحداث ضغط الماوس. فمتى ما تم ضغط زر الماوس فوق هذه اللوحة، يتم رسم خط *line* من موقع مؤشر الماوس إلى مركز اللوحة. ويتم عرض المسافة التي يمثلها هذا الخط بالبكسلات.

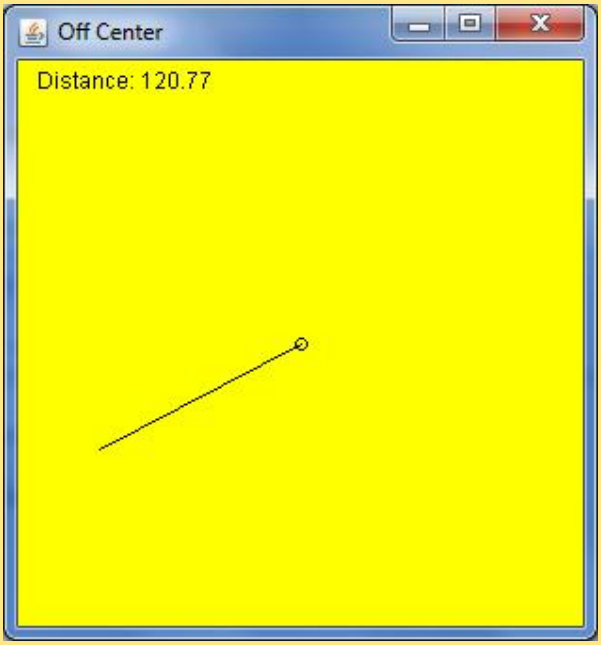
يتم تنفيذ الكلاس المستمع ككلاس داخلي *inner class* يوجد داخل الكلاس *OffCenterPanel*، والموضح في المثال ٢.٧ . فبدلاً من التنفيذ المباشر للواجهة *MouseListener* كما عملنا في الأمثلة السابقة، يقوم هذا المستمع

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مثال رقم ١.٧

```
//********************************************************************
//  OffCenter.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام كلاس مكيف الحدث
//********************************************************************

import javax.swing.*;

public class OffCenter
{
   //-----------------------------------------------------------------
   //  تنشأ الدالة الرئيسية الإطار الرئيسي للبرنامج
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Off Center");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new OffCenterPanel());
      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display



بوراثة الكلاس *MouseAdapter*، والمعرف داخل الحزمة *java.awt.event* التابعة لمكتبة الكلاسات القياسية الخاصة بلغة جافا. يقوم الكلاس *MouseAdapter* بتنفيذ الواجهة *MouseListener* ويحتوي على تعريفات فارغة لكل دوال أحداث الماوس. وفي الكلاس المستمع الخاص بنا، نقوم فقط بإعادة تعريف الدالة *mouseClicked* والملائمة لاحتياجاتنا. ولن نكون مضطرين للتزويد بتعريفات فارغة لبقية دوال أحداث الماوس، وذلك بسبب وراثتنا للتعريفات الفارغة لهذا الدوال من الكلاس المكيف.

بسبب مفهوم الوراثة، أصبح لنا الخيار الآن في إنشاء مستمعات الأحداث. فبإمكاننا تنفيذ واجهة مستمع الحدث، أو يمكننا وراثة الكلاس المكيف للحدث. ويعتبر هذا قراراً تصميماً يجب أخذه بعين الاعتبار. والتقنية الأفضل تعتمد على الموقف الذي نتعامل به.

مثال رقم ٢.٧

```
//********************************************************************
// OffCenterPanel.java      المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا البرنامج لوحة الرسم الأساسية لبرنامج أوف سنتر
//********************************************************************

import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;
import javax.swing.*;

public class OffCenterPanel extends JPanel
{
   private final int WIDTH=300, HEIGHT=300;

   private DecimalFormat fmt;
   private Point current;
   private int centerX, centerY;
   private double length;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة اللوحة والبيانات الضرورية
   //-----------------------------------------------------------------
   public OffCenterPanel()
   {
      addMouseListener (new OffCenterListener());

      centerX = WIDTH / 2;
      centerY = HEIGHT / 2;

      fmt = new DecimalFormat ("0.##");

      setPreferredSize (new Dimension(WIDTH, HEIGHT));
      setBackground (Color.yellow);
   }

   //-----------------------------------------------------------------
   //  تقوم هذه الدالة برسم خط من عند مؤشر الماوس إلى النقطة المركزية
   //  للآبلت وتعرض المسافة
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);

      page.setColor (Color.black);
      page.drawOval (centerX-3, centerY-3, 6, 6);
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
      if (current != null)
         {
            page.drawLine (current.x, current.y, centerX, centerY);
            page.drawString ("Distance: " + fmt.format(length), 10, 15);
         }
      }

      //*****************************************************************
      //  يمثل الكلاس التالي مستمعاً لأحداث الماوس. ويشرح
      //  القدرة على وراثة كلاس مكيف
      //*****************************************************************
      private class OffCenterListener extends MouseAdapter
      {
         //-----------------------------------------------------------
         //  تحسب الدالة التالية المسافة من مؤشر الماوس إلى
         //  النقطة المركزية للآبلت
         //-----------------------------------------------------------
         public void mouseClicked (MouseEvent event)
         {
            current = event.getPoint();
            length = Math.sqrt(Math.pow((current.x-centerX), 2) +
                               Math.pow((current.y-centerY), 2));
            repaint();
         }
      }
}
```

٣.٧ الكلاس المؤقت THE TIMER CLASS

يمكننا اعتبار كائن المؤقت *Timer*، المنشئ من الكلاس *Timer* والتابع للحزمة *javax.swing* على أنه مكون جوئي. مع ذلك، وبخلاف المكونات الأخرى، فإنه ليس لهذا المكون تمثيل مرئي يظهر على الشاشة. فبدلاً من ذلك، وكما هو واضح من الاسم، فإنه يساعدنا في إدارة نشاط ما خلال فترة زمنية.

يقوم كائن المؤقت بتوليد حدث فعلي في فترات زمنية منتظمة. و لعمل رسمة متحركة *animation*، نقوم بتهيئة مؤقت لتوليد حدث فعلي بشكل دوري، ثم نقوم بتحديث الرسومات المتحركة في المستمع الفعلي. ويسرد الشكل ٢.٧ دوال الكلاس*Timer*.

يعرض البرنامج الموضح في المثال ٣.٧ صورة وجه مبتسم يظهر وهو ينزلق عبر نافذة البرنامج بزاوية معينة، ويرتد عند اصطدامه بحواف النافذة.

مفهوم أساسي Key Concept

يولد كائن الكلاس *Timer* أحداثاً فعلية في فترات زمنية منتظمة، ويمكن استخدامه للتحكم برسم متحرك *animation*.

يقوم باني الكلاس *ReboundPanel*، والموضح في المثال ٤.٧، بإنشاء كائن من الكلاس *Timer*. حيث الوسيط الأول لباني الـ *Timer* هو التأخير *delay* بالملي ثانية، والوسيط الثاني للباني هو المستمع الذي يعالج الأحداث الفعلية للمؤقت. يقوم هذا الباني أيضاً بتهيئة الموضع الابتدائي للصورة وعدد البكسلات التي سوف تتحركها، في كلا الاتجاهين العمودي والأفقي، وفي كل مرة يتم إعادة رسم الصورة.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
Timer (int delay, ActionListener listener)
```
الباني: يقوم بإنشاء مؤقت يولد حدثاً فعلياً في فترات زمنية منتظمة، محددة بالقيمة delay.
ويتم معالجة الحدث بواسطة المستمع المحدد.

```
void addActionListener(ActionListener listener)
```
تضيف مستمع حدث إلى المؤقت

```
boolean isRunning()
```
ترجع true إذا المؤقت شغالاً.

```
void setDelay(int delay)
```
تضبط زمن التأخير للمؤقت.

```
void start()
```
تجعل المؤقت يبدأ في العمل، مما يسبب في تولد الأحداث الفعلية.

```
void stop()
```
تجعل المؤقت يتوقف عن العمل، مما يسبب في توقف تولد الأحداث الفعلية.

الشكل رقم ٢.٧ بعض دوال الكلاس المؤقت Timer.

تقوم الدالة *actionPerformed* الخاصة بالمستمع بتحديث قيم الإحداثي x و y الحاليين، ثم تقوم بفحص هذه القيم للتأكد فيما إذا كانت ستسبب في دخول الصورة في حافة اللوحة. إذا حدث ذلك، يتم تعديل الحركة بحيث تكون الحركات المستقبلية للصورة في الاتجاه المعاكس أفقياً أو عمودياً أو كلاهما. لاحظ أن هذه العلمية الحسابية تأخذ حجم الصورة بعين الاعتبار.

سرعة الرسم المتحرك في هذا البرنامج تتأثر بعاملين هما: فترة التوقف بين الأحداث الفعلية والمسافة التي تنزاح بها الصورة في كل مرة. في مثالنا هذا، تم ضبط المؤقت بحيث يولد حدثاً فعلياً كل ٢٠ مللي ثانية، ويتم إزاحة الصورة ٣ بكسلات في كل مرة يتم تحديثها. يمكنك التحكم بهذه القيم لتغيير سرعة الرسم المتحرك. ويجب أن يكون الهدف هو توليد حركة خادعة تكون مريحة للعين.

مثال رقم ٣.٧

```
//********************************************************************
//  Rebound.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//   يوضح هذا البرنامج رسماً متحركاً واستخدام الكلاس المؤقت
//********************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Rebound
{
   //-----------------------------------------------------------------
   //  تعرض الدالة الرئيسية الإطار الأساسي للبرنامج
   //-----------------------------------------------------------------
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Rebound");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new ReboundPanel());
      frame.pack();
      frame.setVisible(true);
   }
}
```

العرض Display

مثال رقم ٤.٧

```
//*****************************************************************
//  ReboundPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا البرنامج اللوحة الأساسية لبرنامج ريباوند
//*****************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ReboundPanel extends JPanel
{
   private final int WIDTH = 300, HEIGHT = 100;
   private final int DELAY = 20, IMAGE_SIZE = 35;

   private ImageIcon image;
   private Timer timer;
   private int x, y, moveX, moveY;

   //-----------------------------------------------------------------
   //   يقوم الباني بتهيئة اللوحة، بما في ذلك المؤقت
   //   الخاص بالرسم المتحرك
   //-----------------------------------------------------------------
   public ReboundPanel()
   {
      timer = new Timer(DELAY, new ReboundListener());

      image = new ImageIcon ("happyFace.gif");

      x = 0;
      y = 40;
      moveX = moveY = 3;
```

```
      setPreferredSize (new Dimension(WIDTH, HEIGHT));
      setBackground (Color.black);
      timer.start();
   }

   //-----------------------------------------------------------------
   //  تقوم هذه الدالة برسم الصورة في الموقع الحالي
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);
      image.paintIcon (this, page, x, y);
   }

   //*****************************************************************
   //  يمثل الكلاس التالي مستمع حدث للمؤقت
   //*****************************************************************
   private class ReboundListener implements ActionListener
   {
      //--------------------------------------------------------------
      //  تقوم هذه الدالة بتحديث موقع الصورة واتجاه
      //  الحركة متى ما قام المؤقت بإطلاق حدث فعلي
      //--------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         x += moveX;
         y += moveY;

         if (x <= 0 || x >= WIDTH-IMAGE_SIZE)
            moveX = moveX * -1;

         if (y <= 0 || y >= HEIGHT-IMAGE_SIZE)
            moveY = moveY * -1;

         repaint();
      }
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الفصل الثامن

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

١.٨ معالجة الحدث EVENT PROCESSING

دعنا نقوم بمراجعة مفهوم معالجة الحدث لجوئي الجافا ونرى ما هي علاقته بتعدد الأشكال *polymorphism*. كما رأينا عده مرات في الأمثلة السابقة، أنه لكي نستجيب لحدث ما، يجب تأسيس علاقة بين كائن مستمع الحدث والمكون المعين الذي قد يطلق الحدث. وكنا نقوم بتأسيس هذه العلاقة بين المستمع والمكون عن طريق استدعاء الدالة التي تضيف المستمع إلى المكون. مثلاً، لنفترض أن لدينا كلاس اسمه *MyButtonListener* يمثل مستمعاً لفعل. لكي نجعل هذا المستمع يستجيب لكائن *JButton*، يمكننا أن نقوم بما يلي:

```
JButton  button = new JButton();
button.addActionListener(new MyButtonListener());
```

متى ما تم تأسيس هذه العلاقة، سيستجيب المستمع عندما يطلق الزر حدثاً فعلياً (بسبب ضغط المستخدم عليه). الآن فكر ملياً بالدالة *addActionListener*. إنها دالة تابعة الكلاس *JButton*، والذي كتبها شخص ما يعمل لدى شركة صن مايكروسيستمز *Sun Microsystems* منذ سنوات مضت. من جهة أخرى، قد نكون قمنا بكتابة الكلاس *MyButtonListener* اليوم. لذلك كيف لدالة كتبت قبل سنوات مضت أن تأخذ وسيطاً من كلاس تم كتابته للتو؟

مفهوم أساسي Key Concept

إن عملية تأسيس علاقة بين المستمع والمكون الذي يستمع له يتم عن طريق مفهوم تعدد الأشكال *polymorphism*.

تمكن الإجابة في مفهوم تعدد الأشكال *polymorphism*. إذا ما قمت بإمعان النظر في الكود المصدري للدالة *addActionListener*، فستكتشف أنها تستقبل وسيطاً من نوع الواجهة *ActionListener*. لذلك، فهذه الدالة لا تستقبل وسيطاً من نوع كائني واحد، لكنها تستطيع أن تستقبل أي كائن من أي كلاس ينفذ الوجهة *ActionListener*. وكل دوال إضافة المستمعات الباقية تعمل بنفس الطريقة.

لا يعرف الكائن *JButton* أي شيء محدد عن الكائن الممرر إلى الدالة *addActionListener*، باستثناء حقيقة أن هذا الكائن ينفذ الوجهة *ActionListener* (وبدون ذلك لن يتم ترجمة الشفرة). يقوم الكائن *JButton* ببساطة بخزن الكائن المستمع ونداء الدالة *actionPerformed* الخاصة به عند ظهور الحدث.

لقد ناقشنا في الفصل السابع أنه بإمكاننا أيضاً إنشاء مستمع عن طريق وراثة كلاس مكيف. حسناً، مع كل هذا فقد رأينا أن استخدام كلاس مكيف ليس طريقة جديدة لإنشاء مستمع. فكل كلاس مكيف قد تم كتابته لتنفيذ واجهة مستمع مناسبة، يتم فيه التزويد بدوال فارغة لكل معالجات الأحداث. لذلك فإنه بوراثتنا لكلاس مكيف، سيقوم الكلاس المستمع الجديد تلقائياً بتنفيذ واجهة المستمع المعينة. وذلك هو في الحقيقة ما يجعله مستمعاً يمكن تمريره إلى دالة إضافة مستمع مناسبة.

وعليه، فأياً كانت طريقة إنشاء كائن الاستماع، فإننا بذلك نستخدم مفهوم تعدد الأشكال عبر الواجهات *polymorphism via interfaces* لإنشاء العلاقة بين المستمع والمكون الذي يستمع إليه. وتعتبر أحداث الجوئي مثالاً رائعاً على القوة والتنوع اللذان نتحصل عليهما عبر تعدد الأشكال.

٢.٨ منتقيات الملف FILE CHOOSERS

تكلمنا في الفصل الرابع عن مربعات الحوار *dialog boxes*. وقد استخدمنا الكلاس *JOptionPane* لإنشاء مربعات حوار متعددة لعرض المعلومة، واستقبال المدخلات، وتأكيد الأفعال.

مفهوم أساسي Key Concept

إن عملية تأسيس علاقة بين المستمع والمكون الذي يستمع له يتم عن طريق مفهوم تعدد الأشكال *polymorphism*.

يمثل الكلاس *JFileChooser* نوعاً آخر من مربعات الحوار، يسمى منتقى الملف *file chooser*، والذي يسمح للمستخدم باختيار ملف *file* من القرص الصلب *hard disk* أو أي وسيط تخزين آخر. ومن المحتمل أنك قد قمت بتشغيل العديد من البرامج التي تسمح لك بفتح ملف باستخدام مربع حواري مشابه.

يستخدم البرنامج الموضح في المثال ١.٨ مربع *JFileChooser* حواري لاختيار ملف. يوضح هذا البرنامج استخدام مكون جوئي آخر، وهو المنطقة النصية *text area*، والتي تشبه الحقل النصي *text field* مع فارق أنها تستطيع عرض

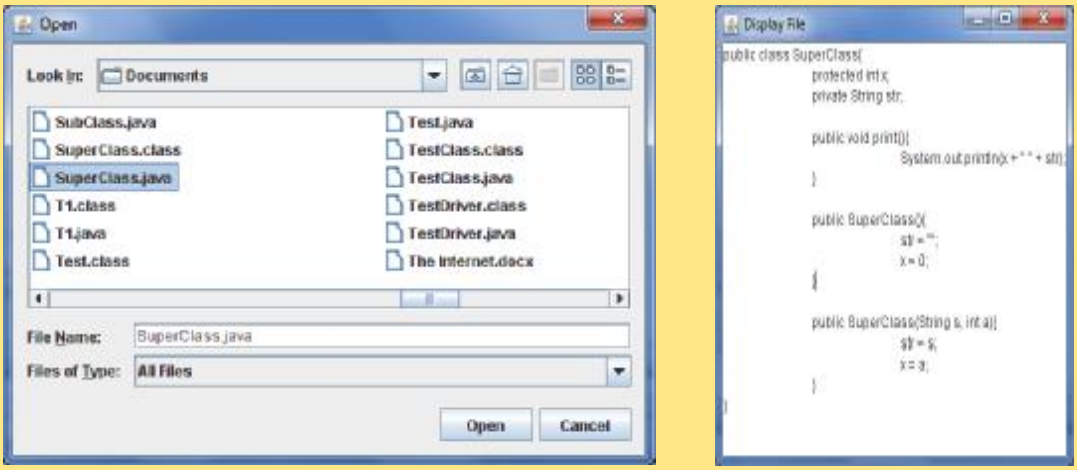
Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

أسطر متعددة من النص في اللحظة الواحدة. بعد أن يقوم المستخدم باختيار ملف باستخدام مربع منتقي الملف الحواري، يتم عرض النص الموجود داخل هذا الملف على المنطقة النصية.

مثال رقم ١.٨

```
//********************************************************************
//  DisplayFile.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام منتقي الملف والمنطقة النصية
//********************************************************************

import java.util.Scanner;
import java.io.*;
import javax.swing.*;

public class DisplayFile
{
   //-----------------------------------------------------------------
   //  في الدالة الرئيسية يتم فتح مربع حوار منتقي الملف، وقراءة الملف
   //  المختار، وتحميله إلى المنطقة النصية
   //-----------------------------------------------------------------
   public static void main (String[] args) throws IOException
   {
      JFrame frame = new JFrame ("Display File");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JTextArea ta = new JTextArea (20, 30);
      JFileChooser chooser = new JFileChooser();

      int status = chooser.showOpenDialog (null);

      if (status != JFileChooser.APPROVE_OPTION)
         ta.setText ("No File Chosen");
      else
      {
         File file = chooser.getSelectedFile();
         Scanner scan = new Scanner (file);

         String info = "";
         while (scan.hasNext())
            info += scan.nextLine() + "\n";

         ta.setText (info);
      }

      frame.getContentPane().add (ta);
      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



يظهر مربع منتقي الملف الحواري عند استدعاء الدالة *showOpenDialog*. حيث تقوم هذه الدالة تلقائياً بعرض قائمة الملفات الموجودة ضمن مجلد مخصص. ويمكن للمستخدم استخدام أدوات التحكم الموجودة على المربع الحواري للانتقال إلى مجلدات أخرى، وتغيير طريقه عرض الملفات، وتحديد نوعية الملفات التي نريد عرضها.

ترجع الدالة *showOpenDialog* عدداً صحيحاً يمثل حالة العملية، ويمكن فحص هذا العدد من خلال ثوابت معرفة في الكلاس *JFileChooser*. وفي برنامجنا هذا، إذا لم يتم اختيار ملف (ربما بضغطنا على زر الإلغاء *Cancel*)، سيتم عرض رسالة افتراضية في المنطقة النصية. أما إذا اختار المستخدم ملفاً، فسيتم فتحه وقراءة محتوياته باستخدام الكلاس *Scanner*. لاحظ أن هذا البرنامج يفترض أن الملف الذي سيتم اختياره يحتوي على نص *text*، ولا يقوم بالتعامل مع أي أخطاء استثنائية *exceptions*. ولذلك، فإنه إذا قام المستخدم باختيار ملف غير مناسب، فسيتم إنهاء البرنامج بمجرد تولد الاستثناء.

يتم تعريف المنطقة النصية *text area* عن طريق الكلاس *JTextArea* . قمنا في هذا البرنامج، بتمرير وسيطين إلى الباني الخاص بهذا الكلاس، يحددان حجم المنطقة النصية بدلالة عدد الحروف (الصفوف والأعمدة) التي يجب عرضها. ويتم تهيئة النص المراد عرضه باستخدام الدالة *setText*. وكما هو الحال في الحقل النصي، يمكن تهيئة مكون المنطقة النصية بحيث يكون قابل للتعديل أو غير قابل للتعديل. ويمكن للمستخدم تغيير محتويات المنطقة النصية القابلة للتعديل عن طريق نقر المنطقة النصية بالماوس ومن ثم الطباعة. أما إذا كانت المنطقة النصية غير قابلة للتعديل، فإنها تستخدم فقط لعرض النص. والوضع الافتراضي لمكون *JTextArea* أنه قابل للتعديل.

يعمل مكون *JFileChooser* على تسهيل السماح للمستخدم بتحديد ملف مخصص لغرض استخدامه. وهناك نوع آخر من المربعات الحوارية المخصصة – يسمح للمستخدم باختيار لون معين – وسنناقشه في القسم التالي.

٣.٨ منتقيات الألوان COLOR CHOOSERS

مفهوم أساسي Key Concept

يسمح منتقى اللون *color chooser* للمستخدم أن يقوم باختيار لون من لوحة الألوان أو استخدام قيم *RGB*.

في كثير من المواقف قد نريد إعطاء مستخدم البرنامج مقدرة اختيار لون. يمكن إنجاز هذه المهمة بطرق مختلفة. مثلاً، يمكننا أن نزود بقائمة الألوان باستخدام مجموعة من أزرار الراديو. مع ذلك، وبسبب التشكيلة الواسعة من الألوان المتوفرة، فيستحسن أن يكون هناك تقنية أكثر سهولة ومرونة لإنجاز هذه المهمة الشائعة. ولهذا الغرض، نستخدم مربعاً حوارياً مخصصاً يشار إليه غالباً بمنتقى اللون *color chooser*، والذي يعد أحد المكونات الرسومية في لغة *Java*.

والكلاس الذي يمثل منتقى اللون هو *JColorChooser*. ويمكن استخدامه لعرض مربع حواري يسمح للمستخدم بالضغط على لون يختاره من لوحة لونية معدة لهذا الغرض. ويمكن للمستخدم كذلك تحديد اللون باستخدام قيم *RGB* أو أي تقنيات تمثيل أخرى.

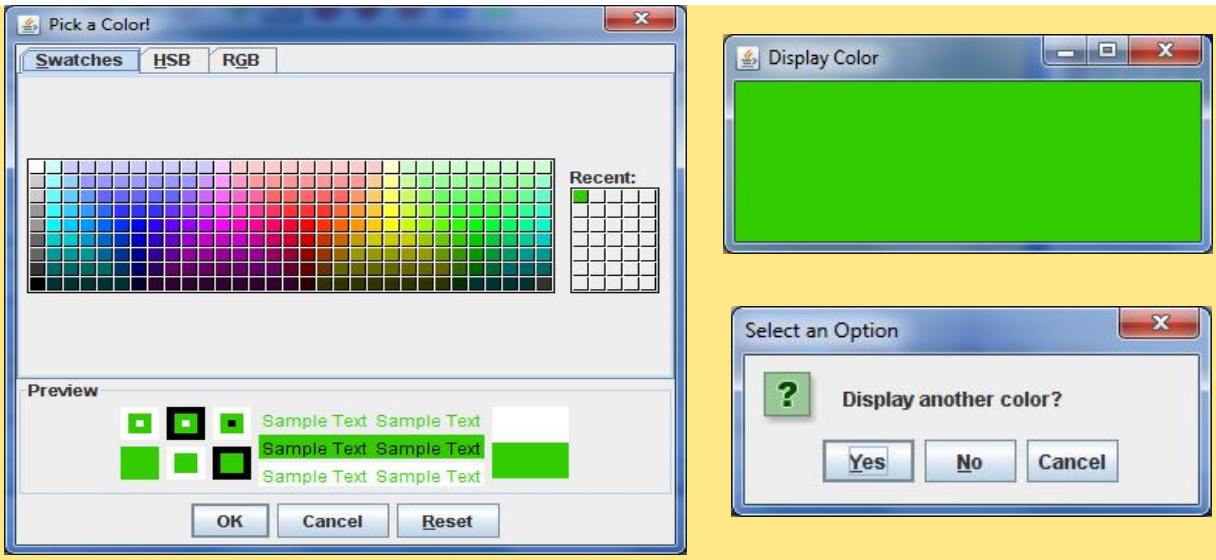
يستخدم البرنامج الموضح في المثال ٢.٨ مربعاً حوارياً لانتقاء اللون يستخدم لتحديد لون اللوحة والتي تعرض على إطار منفصل.

بعد أن يتم انتقاء لون، سيتم عرض هذا اللون الجديد على الإطار الرئيسي، ثم يظهر مربع حواري آخر (وهو منشئ باستخدام الكلاس *JOptionPane* كما ناقشنا في الفصل الرابع)، وهو يستخدم لتحديد ما إذا كان المستخدم يريد تغيير اللون مرة أخرى أم لا. إذا كان يريد ذلك، سيتم عرض مربع حواري آخر لانتقاء اللون. وتستمر هذه الدورة طالما أن المستخدم يرغب في ذلك.

مثال رقم ٢.٨

```
//********************************************************************
//  DisplayColor.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام منتقي اللون
//********************************************************************
import javax.swing.*;
import java.awt.*;

public class DisplayColor
{
   //-----------------------------------------------------------------
   //  تعرض الدالة الرئيسية إطاراً ذو لوحة ملونة، ثم تسمح للمستخدم
   //  بتغيير اللون عدة مرات باستخدام منتقي اللون
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Display Color");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JPanel colorPanel = new JPanel();
      colorPanel.setBackground (Color.white);
      colorPanel.setPreferredSize (new Dimension (300, 100));

      frame.getContentPane().add (colorPanel);
      frame.pack();
      frame.setVisible(true);

      Color shade = Color.white;
      int again;

      do
      {
         shade = JColorChooser.showDialog (frame, "Pick a Color!",
                                           shade);

         colorPanel.setBackground (shade);

         again = JOptionPane.showConfirmDialog (null,
            "Display another color?");
      }
      while (again == JOptionPane.YES_OPTION);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



إن استدعاء الدالة الساكنة *showDialog*، والتابعة للكلاس *JColorChooser*، يسبب في ظهور مربع حوار انتقاء اللون. وتحدد وسائط هذه الدالة المكون الأب لهذا المربع الحواري، وكذلك العنوان الذي سيظهر على إطار المربع الحواري، وأيضاً اللون الابتدائي الذي يتم عرضه على منتقى اللون. وباستخدامنا للمتغير *shade*، كوسيط ثالث، فإن اللون الابتدائي الذي يظهر على منتقى اللون عند ظهوره لأول مرة سيتطابق مع اللون الحالي للوحة.

٤.٨ الزلاجات SLIDERS

مفهوم أساسي Key Concept

تسمح الزلاجات *sliders* للمستخدم أن يقوم بتحديد قيمة رقمية ضمن مدى محدد.

الزلاجة *slider* هي مكون يسمح للمستخدم أن يقوم بتحديد قيمة رقمية ضمن مدى محدد. يمكن تمثيل الزلاجة إما عمودياً أو أفقياً، ويمكن أن يكون لها علامات تدريج اختيارية ومعنونات تحدد مدى القيم.

المثال ٣.٨ يوضح برنامجاً يسمى *SlideColor*. يمكن النظر إلى هذا البرنامج من إحدى زواياه على أنه تطوير لبرنامج *Display* الذي استعرضناه في القسم السابق، وذلك كونه يسمح للمستخدم أن يقوم بشكل ثابت بتغيير اللون المعروض بدون استخدام منتقى الألوان في كل مرة. يعرض هذا البرنامج ثلاث زلاجات تتحكم بمكونات *RGB* للون. ويتم إظهار اللون المحدد بواسطة قيم هذه الزلاجات في مربع معروض على يمين الزلاجات.

تستخدم اللوحة المسماة *ColorPanel* المعرفة في الدالة *main* لعرض اللون المحدد بواسطة الزلاجات، وذلك عن طريق ضبط لون خلفياتها على اللون المحدد. في الحالة الابتدائية تم ضبط كل الزلاجات على القيمة صفر، وهذا يسبب في ظهور اللون الأسود كلون ابتدائي.

أما الكلاس المسمى *SlideColorPanel* الموضح في المثال ٤.٨ فيمثل لوحة لعرض الزلاجات الثلاث. وقد تم إنشاء الزلاجات باستخدام الكلاس *JSlider*، والذي يستقبل أربعة وسطاء. حيث يقوم الوسيط الأول بتحديد اتجاه الزلاجات باستخدام واحد من الثابتين التابعين للكلاس *JSlider* وهما (*HORIZONTAL* أو *VERTICAL*). أما الوسيطان الثاني والثالث فيحددان القيم العظمى والصغرى للزلاجة، وهي في هذا المثال مضبوطة على القيمة *0, 255* لكل زلاجة. أما الوسيط الأخير في باني هذا الكلاس فيحدد القيمة الابتدائية للزلاجة. وفي مثالنا هذا، تم ضبط القيمة الابتدائية لكل زلاجة على القيمة صفر، وهذا يجعل مؤشر الزلاجة في أقصى الشمال عند بدء تنفيذ البرنامج.

يمتلك الكلاس *JSlider* دوال عديدة تسمح للمبرمج بالتحكم بمظهر الزلاجة. حيث يمكن ضبط تدريج الزلاجة الكبير على فترات مخصصة باستخدام الدالة *setMajorTickSpacing*. وأما تدريج الزلاجة الصغير الوسطي فيمكن ضبطه باستخدام الدالة *setMinorTickSpacing*. مع ذلك، فلن يتم عرض هذه التدريجات إلا إذا تم استدعاء الدالة *setPaintTicks*، بالقيمة الوسيطية *true*. وأما المعنونات التي تدل على قيم التدريج الكبير فيتم عرضها فقط إذا تم استدعاء الدالة *setPaintLabels*.

لاحظ أنه في هذا المثال، تم ضبط التدريجات الكبيرة على القيمة *50*. وحيث أنها تبدأ بصفر، يمكن عنونة كل تدريج من مضاعفات الـ *50*، ولذلك فإن العنوان الأخير سيكون *250*، بالرغم من أن قيمة الزلاجة تستطيع الوصول إلى القيمة *255*.

تولد الزلاجة حدث تغيير *change event*، في إشارة إلى تغير موضع الزلاجة والقيمة التي تمثلها. تحتوي الواجهة *ChangeListener* على دالة وحيدة تسمى *stateChanged*. ونلاحظ أنه في برنامج الـ *SlideColor* تم استخدم نفس كائن الاستماع للزلاجات الثلاث. ويتم في الدالة *stateChanged* التي يتم نداءها عند إعادة ضبط أي زلاجة، الحصول على قيمة الزلاجة، وتحديث معنونات الزلاجات الثلاث، وكذلك تعديل لون الخلفية للوحة العرض. وفعلياً نرى أننا فقط بحاجة إلى تعديل معنون واحد فقط (وهو ذلك المرتبط بالزلاجة التي تغيره). ومع ذلك، فإن المجهود الذي يبذل لتحديد أي من الزلاجات تم تغييرها ليس مضموناً. لذلك فإنه من السهل – وربما الأكفأ – أن نقوم بتحديث المعنونات الثلاثة كلها. وكبديل آخر، يمكن عمل كائن استماع لكل زلاجة على حدة، بحيث لا نحتاج لمجهود كتابة الكود الزائد.

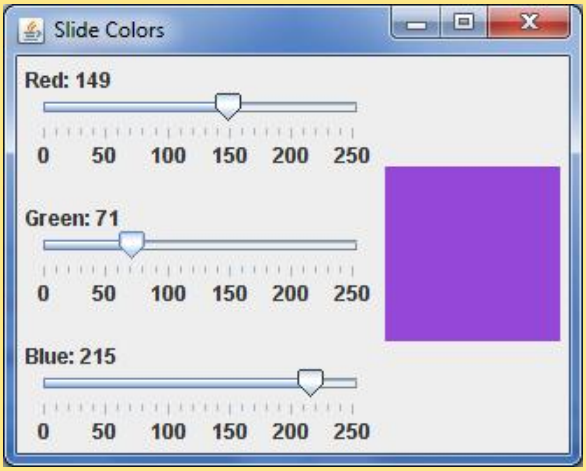
غالباً ما تكون الزلاجة خياراً جيداً عندما يكون هناك مدى واسع من القيم الممكنة لكنها محددة ببداية ونهاية. وإذا ما قارناها بالحقل النصي *text field*، نلاحظ أن الزلاجة تعطي المستخدم معلومات أكثر وتجنبنا من مشاكل أخطاء الإدخال.

مثال رقم ٣.٨

```
//********************************************************************
//  SlideColor.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام مكونات الزلاجة
//********************************************************************

import java.awt.*;
import javax.swing.*;

public class SlideColor
{
   //-----------------------------------------------------------------
   //  تعرض الدالة الرئيسية إطاراً فيه لوحة تحكم ولوحة
   //  لتغيير اللون تبعاً لتغيير الزلاجة
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Slide Colors");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new SlideColorPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

العرض Display



مثال رقم ٤.٨

```
//********************************************************************
//   SlideColorPanel.java          المؤلف: Lewis/Loftus     المترجم : Almashraqi
//
//  يمثل هذا البرنامج لوحة التحكم بالزلاجة لبرنامج زلاجة اللون
//********************************************************************

import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class SlideColorPanel extends JPanel
{
   private JPanel controls, colorPanel;
   private JSlider rSlider, gSlider, bSlider;
   private JLabel rLabel, gLabel, bLabel;

   //-----------------------------------------------------------------
   //  يقوم البناء بتهيئة الزلاجات ومعنوناتها، ومحاذاتهم على طول
   //  حافتهم اليسارية باستخدام التخطيط الصندوقي
   //-----------------------------------------------------------------
   public SlideColorPanel()
   {
      rSlider = new JSlider (JSlider.HORIZONTAL, 0, 255, 0);
      rSlider.setMajorTickSpacing (50);
      rSlider.setMinorTickSpacing (10);
      rSlider.setPaintTicks (true);
      rSlider.setPaintLabels (true);
      rSlider.setAlignmentX (Component.LEFT_ALIGNMENT);

      gSlider = new JSlider (JSlider.HORIZONTAL, 0, 255, 0);
      gSlider.setMajorTickSpacing (50);
      gSlider.setMinorTickSpacing (10);
      gSlider.setPaintTicks (true);
      gSlider.setPaintLabels (true);
      gSlider.setAlignmentX (Component.LEFT_ALIGNMENT);
```

```
      bSlider = new JSlider (JSlider.HORIZONTAL, 0, 255, 0);
      bSlider.setMajorTickSpacing (50);
      bSlider.setMinorTickSpacing (10);
      bSlider.setPaintTicks (true);
      bSlider.setPaintLabels (true);
      bSlider.setAlignmentX (Component.LEFT_ALIGNMENT);

      SliderListener listener = new SliderListener();
      rSlider.addChangeListener (listener);
      gSlider.addChangeListener (listener);
      bSlider.addChangeListener (listener);

      rLabel = new JLabel ("Red: 0");
      rLabel.setAlignmentX (Component.LEFT_ALIGNMENT);
      gLabel = new JLabel ("Green: 0");
      gLabel.setAlignmentX (Component.LEFT_ALIGNMENT);
      bLabel = new JLabel ("Blue: 0");
      bLabel.setAlignmentX (Component.LEFT_ALIGNMENT);

      controls = new JPanel();
      BoxLayout layout = new BoxLayout (controls, BoxLayout.Y_AXIS);
      controls.setLayout (layout);
      controls.add (rLabel);
      controls.add (rSlider);
      controls.add (Box.createRigidArea (new Dimension (0, 20)));
      controls.add (gLabel);
      controls.add (gSlider);
      controls.add (Box.createRigidArea (new Dimension (0, 20)));
      controls.add (bLabel);
      controls.add (bSlider);

      colorPanel = new JPanel();
      colorPanel.setPreferredSize (new Dimension (100, 100));
      colorPanel.setBackground (new Color (0, 0, 0));

      add (controls);
      add (colorPanel);
   }

   //*****************************************************************
   //  يمثل هذا الكلاس مستمعاً لكل الزلاجات الثلاث
   //*****************************************************************
   private class SliderListener implements ChangeListener
   {
      private int red, green, blue;

      //-----------------------------------------------------------------
      //  تقوم الدالة التالية بالحصول على قيمة كل زلاجة، ثم تقوم بتحديث
      //  المعنونات ولوحة اللون
      //-----------------------------------------------------------------
      public void stateChanged (ChangeEvent event)
      {
         red = rSlider.getValue();
         green = gSlider.getValue();
         blue = bSlider.getValue();
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
            rLabel.setText ("Red: " + red);
            gLabel.setText ("Green: " + green);
            bLabel.setText ("Blue: " + blue);

            colorPanel.setBackground (new Color (red, green, blue));
        }
    }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الفصل التاسع

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مفهوم أساسي Key Concept

تعمل أدوات التوضيح *tool tips* وحروف الاختصار *mnemonics* على تحسين الأداء الوظيفي لواجهة المستخدم الرسومية.

دعنا نأخذ نظرة عن بعض الخصائص التي يمكن استخدامها مع أي مكون *Swing*. إن التطبيق الصحيح لهذه المكونات يحسن واجهة المستخدم ويسهل استخدام المكونات. يشرح هذا القسم استخدام أدوات التوضيح *tool tips* وحروف الاختصار *mnemonics*، كذلك يشرح إمكانية إلغاء تفعيل المكونات، ثم يستكشف مثالاً يستخدم هذه الخصائص.

يمكننا إسناد أداة التوضيح لأي مكون *Swing*، وهي عبارة عن سطر نصي قصير يظهر عند وضع مؤشر الماوس لبرهة من الزمن على المكون. عادة ما يتم استخدام أدوات التوضيح لتوضيح معلومة للمستخدم عن المكون، مثل الغرض من الزر.

يمكن إسناد أداة التوضيح باستخدام الدالة *setToolTipText* الخاصة بمكونات *Swing*، مثلاً:

```
JButton button = new JButton ("Compute");
button.setToolTipText("Calculates the area above the curve.") ;
```

عندما يتم إضافة الزر إلى الحاوي وعرضه، فإنه يظهر طبيعياً. ولكن عندما يقوم المستخدم بوضع مؤشر الماوس فوق الزر، ويظل فوقه لبرهة، فسيتم ظهور نص أداة التوضيح. وسيختفي هذا النص، إذا أزاح المستخدم مؤشر الماوس عند المكون.

حرف الاختصار *mnemonic* هو حرف يسمح للمستخدم بالضغط على زر أو اختيار خيار في قائمة باستخدام لوحة المفاتيح بجانب إمكانية استخدام الماوس. مثلاً، إذا قمنا بتعريف حرف اختصار لزر، يمكن أن يقوم المستخدم بالضغط المستمر على المفتاح *Alt* وضغط حرف الاختصار لتفعيل الزر. إن استخدام حرف الاختصار في تفعيل الزر يؤدي إلى أن يسلك النظام كما لو أن المستخدم استخدم الماوس في ضغط الزر.

عند اختيارنا لحرف الاختصار يجب أن يكون أحد حروف معنون الزر أو عنصر القائمة. ومتى ما تم إنشاء حرف الاختصار باستخدام الدالة *setMnemonic*، فسيتم وضع خط تحت حرف الاختصار في المعنون للإشارة إلى أنه يمكن استخدامه كاختصار. أما إذا تم اختيار حرف ليس ضمن حروف المعنون، فلن يتم وضع خط تحت أي حرف، ولن يعرف المستخدم كيف سيستخدم الاختصار. ويمكنك ضبط حرف الاختصار كما يلي:

```
JButton button = new JButton("Calculate");
button.setMnemonic('C');
```

عندما يتم عرض الزر، فإن الحرف *C* في كلمة *calculate* سيوضع تحته خط في معنون الزر، وعندما يقوم المستخدم بضغط *ALT-C*، فسيتم تفعيل الزر كما لو أننا ضغطنا عليه باستخدام الماوس.

يمكن إلغاء تفعيل *disable* بعض المكونات إذا كان يتوجب عدم استخدامها. وسيظهر المكون الغير مفعل رمادياً، ولا شيء سيحدث إذا ما حاول المستخدم التفاعل معه. ولإلغاء تفعيل *disable* أو تفعيل *enable* المكونات، نقوم باستدعاء الدالة *setEnabled* الخاصة بالمكون، ونمرر لها قيمة بوليانية تشير فيما إذا كان المكون يجب أن لا يُفعّل *(false)* أو أن يُفعّل *(true)*. مثلاً :

```
JButton button = new JButton();
button.setEnabled (false);
```

يعتبر إلغاء تفعيل المكونات فكرة جيدة عندما لا نريد السماح للمستخدمين أن يستخدموا وظيفية مكون ما. ويعتبر المظهر الرمادي للمكون الغير مفعل إشارة إلى أن استخدام المكون في هذه اللحظة غير مناسب (وفي الحقيقة، هو مستحيل) في الوقت الحالي. وبالإضافة إلى أن المكونات الغير مفعلة خاطئة تعلم المستخدم بما هي الأحداث المناسبة وما هي الأحداث الغير مناسبة، فهي أيضاً تمنع ظهور مواقف الخطأ.

دعنا الآن ننظر في هذا المثال الذي يستخدم أدوات التوضيح، وحروف الاختصار، وإلغاء تفعيل المكونات. يوضح البرنامج في المثال ١.٩ صورة مصباح ضوئي ويزودنا بزرين لتشغيل المصباح *on* وإيقاف تشغيله *off*.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

مفهوم أساسي Key Concept

يجب إلغاء تفعيل المكونات عندما يكون استخدامها غير مناسب.

يوجد فعلياً صورتان للمصباح الضوئي: الصورة الأولى تظهر المصباح وهو في حالته المضيئة، والثانية تظهر المصباح وهو في حالة إيقاف. ويتم جلب هذه الصور باستخدام كائنات *ImageIcon*. تعمل الدالة *setIcon* الخاصة بالمعنون الذي يعرض الصورة على تهيئة الصورة المناسبة، اعتماداً على الحالة الحالية. ويتم التحكم بهذه المعالجة في الكلاس *LightBulbPanel* الموضح في المثال ٢.٩.

يمثل الكلاس *LightBulbControls* الموضح في المثال ٣.٩ اللوحة التي تحتوي الأزرار *on* و *off*. ويوجد لكل من هذين الزرين أداة توضيح مرتبطة به. وكلاهما أيضاً يملكان مفتاح اختصار. كذلك، نرى أنه إذا تم تفعيل أحد الأزرار فإن الآخر يلغي تفعيله، والعكس صحيح. فعندما يكون المصباح الضوئي في حالة *on*، فليس هناك حاجة إلى تفعيل الزر *off*.

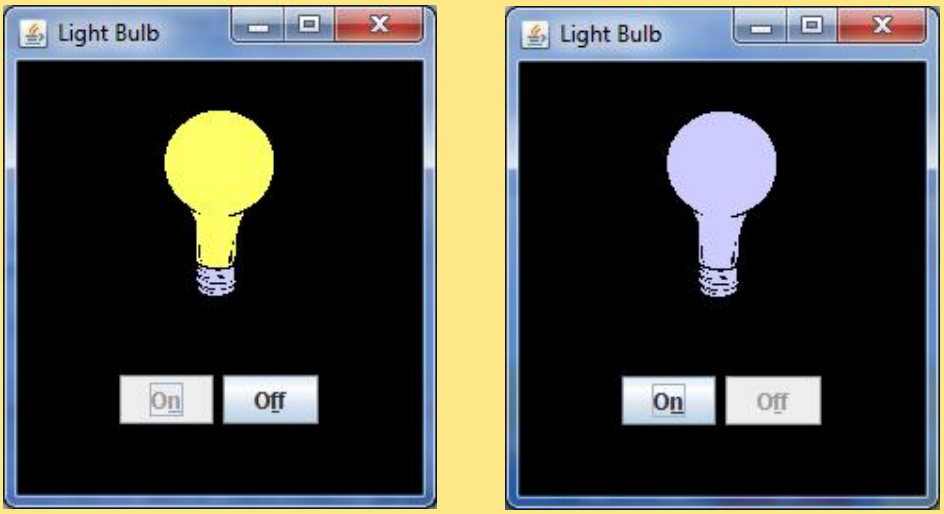
يملك كل زر كلاس مستمع خاص به. وتعمل الدالة *actionPerformed* الخاصة بكل زر على ضبط حالة المصباح، وتبديل حالة التفعيل لكلا الزرين، وإعادة رسم الصورة على اللوحة.

لاحظ أن حروف الاختصار المستخدمة لكل زر يوضع تحتها خط أثناء العرض. عندما تقوم بتشغيل البرنامج، لاحظ أن أدوات التوضيح تتضمن تلقائياً إشارة إلى حرف الاختصار الذي يمكن استخدامه مع الزر.

مثال رقم ١.٩

```
//********************************************************************
//  LightBulb.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال حروف الاختصار وأدوات التوضيح
//********************************************************************
import javax.swing.*;
import java.awt.*;
public class LightBulb
{
   //-----------------------------------------------------------------
   //  الدالة الرئيسية: تقوم بتهيئة الإطار الذي يعرض صور المصباح الضوئي
   //  الذي يمكن تشغيله وإيقاف تشغيله
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Light Bulb");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      LightBulbPanel bulb = new LightBulbPanel();
      LightBulbControls controls = new LightBulbControls (bulb);

      JPanel panel = new JPanel();
      panel.setBackground (Color.black);
      panel.setLayout (new BoxLayout(panel, BoxLayout.Y_AXIS));
      panel.add (Box.createRigidArea (new Dimension (0, 20)));
      panel.add (bulb);
      panel.add (Box.createRigidArea (new Dimension (0, 10)));
      panel.add (controls);
      panel.add (Box.createRigidArea (new Dimension (0, 10)));

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



مثال رقم ٢.٩

```
//********************************************************************
// LightBulbPanel.java      المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//   LightBulb يمثل هذا الكلاس الصورة لبرنامج
//********************************************************************

import javax.swing.*;
import java.awt.*;

public class LightBulbPanel extends JPanel
{
   private boolean on;
   private ImageIcon lightOn, lightOff;
   private JLabel imageLabel;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة الصور والحالة الإبتدائية
   //-----------------------------------------------------------------
   public LightBulbPanel()
   {
      lightOn = new ImageIcon ("lightBulbOn.gif");
      lightOff = new ImageIcon ("lightBulbOff.gif");

      setBackground (Color.black);

      on = true;
      imageLabel = new JLabel (lightOff);
      add (imageLabel);
   }
```

```
   //-----------------------------------------------------------------
   //  دالة رسم اللوحة باستخدام الصورة المناسبة
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent(page);

      if (on)
         imageLabel.setIcon (lightOn);
      else
         imageLabel.setIcon (lightOff);
   }

   //-----------------------------------------------------------------
   //  دالة ضبط حالة المصباح الضوئي
   //-----------------------------------------------------------------
   public void setOn (boolean lightBulbOn)
   {
      on = lightBulbOn;
   }
}
```

مثال رقم ٣.٩

```
//*****************************************************************
// LightBulbControls.java   المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//   LightBulb يمثل هذا الكلاس لوحة التحكم لبرنامج
//*****************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class LightBulbControls extends JPanel
{
   private LightBulbPanel bulb;
   private JButton onButton, offButton;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة لوحة التحكم بالمصباح الضوئي
   //-----------------------------------------------------------------
   public LightBulbControls (LightBulbPanel bulbPanel)
   {
      bulb = bulbPanel;

      onButton = new JButton ("On");
      onButton.setEnabled (false);
      onButton.setMnemonic ('n');
      onButton.setToolTipText ("Turn it on!");
      onButton.addActionListener (new OnListener());

      offButton = new JButton ("Off");
      offButton.setEnabled (true);
      offButton.setMnemonic ('f');
      offButton.setToolTipText ("Turn it off!");
      offButton.addActionListener (new OffListener());
```

```
        setBackground (Color.black);
        add (onButton);
        add (offButton);
    }

    //*****************************************************************
    //   On يمثل هذا الكلاس المستمع للزر
    //*****************************************************************
    private class OnListener implements ActionListener
    {
        //-----------------------------------------------------------
        //  تقوم هذه الدالة بتشغيل المصباح وتعيد رسم لوحة المصباح
        //-----------------------------------------------------------
        public void actionPerformed (ActionEvent event)
        {
            bulb.setOn (true);
            onButton.setEnabled (false);
            offButton.setEnabled (true);
            bulb.repaint();
        }
    }

    //*****************************************************************
    //  Off يمثل هذا الكلاس المستمع للزر
    //*****************************************************************
    private class OffListener implements ActionListener
    {
        //-----------------------------------------------------------
        //  تقوم هذه الدالة بإيقاف تشغيل المصباح وتعيد رسم لوحة المصباح
        //-----------------------------------------------------------
        public void actionPerformed (ActionEvent event)
        {
            bulb.setOn (false);
            onButton.setEnabled (true);
            offButton.setEnabled (false);
            bulb.repaint();
        }
    }
}
```

٢.٩ صناديق الخيارات المنسدلة COMBO BOXES

مفهوم أساسي Key Concept

يعمل صندوق الخيارات المنسدلة *combo box* على تزويد المستخدم بقائمة منسدلة من الخيارات.

يسمح صندوق الخيارات المنسدلة *combo box* للمستخدم باختيار واحد من خيارات متعددة من قائمة منسدلة *drop down menu*. فعندما يقوم المستخدم بضغط صندوق الخيارات المنسدلة باستخدام الماوس، يتم عرض قائمة خيارات يمكن للمستخدم الاختيار من بينها. يمكننا إنشاء صندوق الخيارات المنسدلة باستخدام الكلاس *JComboBox*.

ويمكن أن يكون صندوق الخيارات المنسدلة في واحدة من حالتين: إما قابل للتعديل *editable* أو غير قابل للتعديل *uneditable*. والحالة الافتراضية لصندوق الخيارات المنسدلة هو أنه غير قابل للتعديل. وعليه فلا يمكنك تغيير قيمة صندوق الخيارات المنسدلة الغير قابل للتعديل إلا عن طريق اختيار أحد عناصر القائمة. أما إذا كان صندوق

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الخيارات المنسدلة قابلاً للتعديل، فإن ذلك يعني أن المستخدم يمكنه تغيير قيمته إما عن طريق اختيار أحد عناصر القائمة أو عن طريق طباعة قيمة مخصصة في الصندوق.

يمكن إنشاء خيارات صندوق الخيارات المنسدلة بإحدى طريقتين. فإما أن نقوم بإنشاء مصفوفة من السلاسل النصية ومن ثم تمريرها إلى الباني الخاص بالكلاس *JComboBox*. وأما في الطريقة الثانية، فيمكننا استخدام الدالة *addItem* لإضافة عنصر إلى صندوق الخيارات المنسدلة بعد إنشائه. يمكن أيضاً عرض كائنات *ImageIcon* كخيارات في القائمة.

يبين البرنامج *JukeBox* الموضح في المثال ٤.٩ استخدام صندوق الخيارات المنسدلة. يقوم المستخدم باختيار الأغنية التي يريد تشغيلها باستخدام صندوق الخيارات المنسدلة، ومن ثم يضغط الزر *Play* من أجل بدء تشغيل الأغنية. ويمكن ضغط الزر *Stop* في أي وقت لتوقيف الأغنية. كذلك يمكن إيقاف الأغنية الحالية عن طريق اختيار أغنية جديدة، فهذا يعمل على إيقاف الأغنية القديمة.

ويمثل الكلاس *JukeBoxControls* الموضح في المثال ٥.٩ لوحة تحتوي المكونات التي تشكل الجوئي الخاص بالـ *jukebox*. يقوم باني هذا الكلاس أيضاً بتحميل المقاطع الصوتية التي سيتم تشغيلها. يتم توليد المقطع الصوتي أولاً عن طريق إنشاء كائن *URL* ينسجم مع ملف *wav* أو ملف *au* الذي يعرف المقطع. ويجب أن يكون أول وسيطين في باني الـ *URL* هما الملف "*file*" والمضيف المحلي "*localhost*"، على التوالي، وذلك في حالة أن المقطع الصوتي مخزن في نفس الجهاز الذي يتم تنفيذ البرنامج عليه. إن إنشاء كائنات *URL* يمكن أن يؤدي إلى تولد استثناء مفحوص *checked exception*؛ ولهذا يتم إنشاء هذه الكائنات داخل قالب *try*. مع ذلك، فإن هذا البرنامج يفترض أنه سيتم تحميل المقاطع الصوتية بنجاح ولذلك فإنه لم يعمل شيئاً في حالة حدوث الاستثناء.

متى ما تم إنشاء كائنات الـ *URL* ، فسيتم استخدامها لإنشاء كائنات *Audio clip* باستخدام الدالة الساكنة المسماة *newAudioClip* التابعة للكلاس *JApplet*. يتم تخزين المقاطع الصوتية في مصفوفة. ويكون المدخل الأول في المصفوفة، في الفهرس *0*، مضبوطاً على القيمة *null*. وهذا المدخل مرتبط بالخيار الأول في صندوق الخيارات المنسدلة، وهو ببساطة يشجع المستخدم على القيام باختيار مقطع صوتي.

مثال رقم ٤.٩

```
//********************************************************************
//  JukeBox.java       المؤلف : Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا الكلاس استخدام صندوق الاختيار المنسدل
//********************************************************************

import javax.swing.*;

public class JukeBox
{
   //-----------------------------------------------------------------
   //  الدالة الرئيسية: تنشئ وتعرض عناصر التحكم للجوك بوكس
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Java Juke Box");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JukeBoxControls controlPanel = new JukeBoxControls();

      frame.getContentPane().add(controlPanel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

مثال رقم ٥.٩

```
//*************************************************************************
//  JukeBoxControls.java    المؤلف: Lewis/Loftus    المترجم : Almashraqi
//
//  يمثل هذا الكلاس لوحة التحكم بالجوك بوكس
//*************************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.AudioClip;
import java.net.URL;

public class JukeBoxControls extends JPanel
{
   private JComboBox musicCombo;
   private JButton stopButton, playButton;
   private AudioClip[] music;
   private AudioClip current;

   //-----------------------------------------------------------------
   //  الباني: يقوم بتهيئة الجوئي الخاص بالجوك بوكس
   //-----------------------------------------------------------------
   public JukeBoxControls()
   {
      URL url1, url2, url3, url4, url5, url6;
      url1 = url2 = url3 = url4 = url5 = url6 = null;

      // Obtain and store the audio clips to play
      try
      {
         url1 = new URL ("file", "localhost", "westernBeat.wav");
         url2 = new URL ("file", "localhost", "classical.wav");
         url3 = new URL ("file", "localhost", "jeopardy.au");
         url4 = new URL ("file", "localhost", "newAgeRythm.wav");
         url5 = new URL ("file", "localhost", "eightiesJam.wav");
         url6 = new URL ("file", "localhost", "hitchcock.wav");
      }
      catch (Exception exception) {}
```

```
        music = new AudioClip[7];
        music[0] = null;  //  "Make Selection..." للاستجابة للخيار
        music[1] = JApplet.newAudioClip (url1);
        music[2] = JApplet.newAudioClip (url2);
        music[3] = JApplet.newAudioClip (url3);
        music[4] = JApplet.newAudioClip (url4);
        music[5] = JApplet.newAudioClip (url5);
        music[6] = JApplet.newAudioClip (url6);

        JLabel titleLabel = new JLabel ("Java Juke Box");
        titleLabel.setAlignmentX (Component.CENTER_ALIGNMENT);

        // إنشاء قائمة نصية لخيارات صندوق القائمة المنسدلة
        String[] musicNames = {"Make A Selection...", "Western Beat",
                "Classical Melody", "Jeopardy Theme", "New Age Rythm",
                "Eighties Jam", "Alfred Hitchcock's Theme"};

        musicCombo = new JComboBox (musicNames);
        musicCombo.setAlignmentX (Component.CENTER_ALIGNMENT);

        //  تهيئة الأزرار
        playButton = new JButton ("Play", new ImageIcon ("play.gif"));
        playButton.setBackground (Color.white);
        playButton.setMnemonic ('p');
        stopButton = new JButton ("Stop", new ImageIcon ("stop.gif"));
        stopButton.setBackground (Color.white);
        stopButton.setMnemonic ('s');

        JPanel buttons = new JPanel();
        buttons.setLayout (new BoxLayout (buttons, BoxLayout.X_AXIS));
        buttons.add (playButton);
        buttons.add (Box.createRigidArea (new Dimension(5,0)));
        buttons.add (stopButton);
        buttons.setBackground (Color.cyan);

        //  تهيئة هذه اللوحة
        setPreferredSize (new Dimension (300, 100));
        setBackground (Color.cyan);
        setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
        add (Box.createRigidArea (new Dimension(0,5)));
        add (titleLabel);
        add (Box.createRigidArea (new Dimension(0,5)));
        add (musicCombo);
        add (Box.createRigidArea (new Dimension(0,5)));
        add (buttons);
        add (Box.createRigidArea (new Dimension(0,5)));

        musicCombo.addActionListener (new ComboListener());
        stopButton.addActionListener (new ButtonListener());
        playButton.addActionListener (new ButtonListener());

        current = null;
    }
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
   //*****************************************************************
   //  الكلاس التالي يمثل مستمع الحدث لصندوق الخيارات المنسدلة
   //*****************************************************************
   private class ComboListener implements ActionListener
   {
      //-----------------------------------------------------------------
      //  الدالة التالية توقف الاختيار المشغل حالياً (إن كان موجوداً) و
      //  وتضبط الاختيار الحالي على المقطع الذي تم اختياره
      //-----------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         if (current != null)
            current.stop();

         current = music[musicCombo.getSelectedIndex()];
      }
    }

   //*****************************************************************
   //  يمثل الكلاس التالي مستمع الحدث لزري التحكم كليهما
   //*****************************************************************
   private class ButtonListener implements ActionListener
   {
      //-----------------------------------------------------------------
      //  توقف الدالة التالية الاختيار الحالي (إن وجد) في أي حالة
      //  إذا كان زر التشغيل مضغوطاً، فتبدأ بتشغيله من جديد
      //-----------------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         if (current != null)
            current.stop();

         if (event.getSource() == playButton)
            if (current != null)
               current.play();
      }
   }
}
```

يتم تعريف قائمة الأغاني المعروضة في صندوق الاختيار في شكل مصفوفة من السلاسل النصية، وسيظهر المدخل الأول في المصفوفة في صندوق الخيارات المنسدلة بشكل افتراضي وغالباً يستخدم لإرشاد المستخدم. يجب علينا أن نفهم أن بقية البرنامج لم يتم فيه محاولة استخدام هذا الخيار كأغنية صحيحة.

يتم عرض زري التشغيل *play* والإيقاف *stop* باستخدام معنون نصي وأيقونة صورية. وقد تم عمل حروف اختصار لهما بحيث يمكننا التحكم بالـ *Jukebox* جزئياً باستخدام لوحة المفاتيح.

يقوم صندوق الخيارات المنسدلة بتوليد حدث فعلي عندما يقوم المستخدم باختيار أحد العناصر فيه. ويستخدم البرنامج *Jukebox* كلاس مستمع حدث واحد لصندوق الخيارات المنسدلة، ومستمع آخر لكلا الزرين. مع ذلك، يمكننا دمج هذين المستمعين، واستخدام الكود في تحديد ما هو المكون الذي قام بتوليد الحدث.

يتم تنفيذ الدالة *actionPerformed* الخاصة بالكلاس *ComboListener* عندما يتم اختيار أحد خيارات صندوق الخيارات المنسدلة. وفي حالة وجود مقطع صوتي في حالة تشغيل، فإنه يتوقف. ومن ثم يتم تحديث المقطع الصوتي الحالي بحيث يعكس الاختيار الجديد. لاحظ أنه لا يتم تشغيل المقطع الصوتي مباشرة عند هذه النقطة. حيث يقتضي تصميم هذا البرنامج أن يقوم المستخدم بضغط زر التشغيل *play* لسماع الاختيار الجديد.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

أما الدالة *actionPerformed* الخاصة بالكلاس *ButtonListener* فيتم تنفيذها عندما يتم ضغط أي من الزرين. إذا كان هناك مقطع صوتي حالياً في حالة تشغيل، فسيتم توقيفه. إذا قمنا بضغط الزر *stop*، فإن المهمة تنتهي. أما إذا تم ضغط زر التشغيل *play*، فسيتم تشغيل المقطع الصوتي الحالي من جديد من بدايته.

٣.٩ مقاطع التمرير SCROLL PANES

مفهوم أساسي Key Concept

يعتبر مقطع التمرير *scroll pane* مفيداً لاستعراض الكائنات الكبيرة أو الكميات الكبيرة من البيانات.

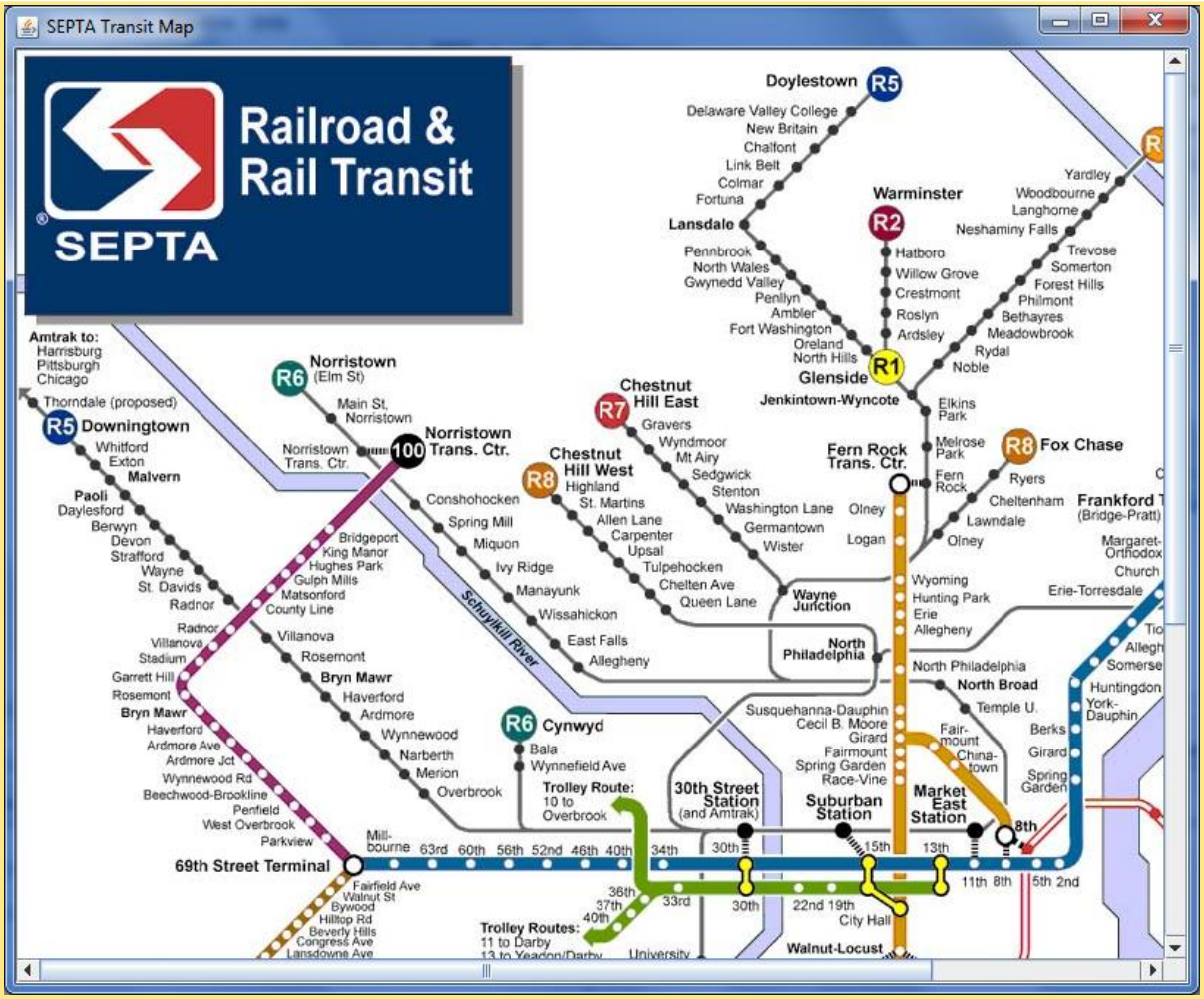
نحتاج في بعض الأحيان أن نتعامل مع صورة أو معلومات حجمها كبير جداً بحيث لا تتلائم مع مساحة معقولة. ويعتبر مقطع التمرير *scroll pane* مفيداً في مثل هذه الحالات. إن شريط التمرير هو عبارة عن حاوي *container* يقدم عرضاً محدوداً للمكون، ويزود بأشرطة تمرير عمودية أو أفقية *vertical or horizontal scroll bars* لتغيير منطقة العرض. وفي أي لحظة، يمكن فقط رؤية جزء من المكون الكلي، ولكن أشرطة التمرير تسمح للمستخدم باستعراض أي جزء من المكون. وتعتبر أشرطة التمرير *scroll bars* مفيدة عندما تكون مساحة الجوئي محدودة أو عندما يكون المكون المراد استعراضه كبيراً أو أن حجمه قابل للتغيير بشكل ديناميكي.

يوضح البرنامج في المثال ٦.٩ إطاراً يحتوي على مقطع تمرير واحد. يستخدم مقطع التمرير هذا لاستعراض صورة لخريطة طرق كبيرة الحجم جداً لولاية فيلادلفيا والمناطق المجاورة لها. تم وضع الصورة ضمن معنون، وتم إضافة هذا المعنون إلى مقطع التمرير باستخدام الباني الخاص بالكلاس *JScrollPane*.

مثال رقم ٦.٩

```
//********************************************************************
//  TransitMap.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام مقطع التمرير
//********************************************************************
import java.awt.*;
import javax.swing.*;
public class TransitMap
{
   //-----------------------------------------------------------------
   //  تقدم الدالة الرئيسية إطاراً يحوي مقطع تمرير يستخدم لعرض خريطة
   //  كبيرة لنظام النقل في فيلادلفيا
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      // سبتا اختصار لـ سلطة النقل في جنوب شرق بنسلفينيا
      JFrame frame = new JFrame ("SEPTA Transit Map");

      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      ImageIcon image = new ImageIcon ("septa.jpg");
      JLabel imageLabel = new JLabel (image);

      JScrollPane sp = new JScrollPane (imageLabel);
      sp.setPreferredSize (new Dimension (450, 400));

      frame.getContentPane().add (sp);
      frame.pack();
      frame.setVisible(true);
   }
}
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com



يمكن أن يمتلك مقطع التمرير شريط تمرير عمودي في الجهة اليمنى من الحاوي وكذلك شريط تمرير أفقي في أسفل الحاوي. ومع هذين الشريطين، يمكن للمبرمج أن يحدد فيما إذا كانت تستخدم دائماً، أو لا تستخدم أبداً، أو تستخدم عند الحاجة لاستعراض المكون كاملاً. والوضع الافتراضي، هو أن شريطي التمرير العمودي والأفقي يستخدمان عند الحاجة. ويعتمد البرنامج *TransitMap* على هذا الوضع الافتراضي، ويظهر شريطا التمرير، كلاهما، لأن الصورة كبيرة من حيث الارتفاع والعرض.

ولتحريك شريط التمرير، يمكن للمستخدم أن يضغط على العقدة *knob* الموجودة في شريط التمرير ويسحبها، وهي التي ستحدد الموضع الحالي (في ذلك البعد: فوق / تحت أو يمين / شمال). وكخيار بديل، يمكن للمستخدم أن يضغط الشريط الموجود على يمين وشمال العقدة، أو على الأسهم الموجودة في كل طرف من أطراف شريط التمرير، وذلك من أجل التحكم بالموقع. ويمكن للمبرمج أن يحدد كمية التغير التي تحصل لمنطقة العرض عند حدوث أي من هذه الأحداث.

لاحظ أنه لا توجد حاجة لمستمعات حدث كي نستخدمها مع مقطع التمرير في هذه الطريقة. فمقطع التمرير يستجيب تلقائياً للأوضاع التي تتكيف عليها أشرطة التمرير التابعة له.

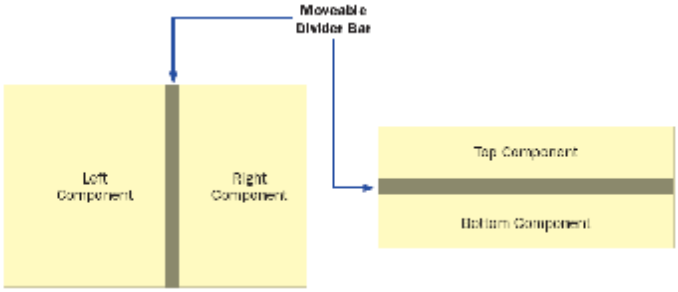
٤.٩ مقاطع الفصل SPLIT PANES

إن مقطع الفصل *split pane* هو عبارة عن حاوي يعرض مكونين مفصولين بواسطة شريط فاصل قابلة للحركة. واعتماداً على طريقة تهيئة مقطع الفصل، يتم عرض المكونين أما جنباً إلى جنب أو واحداً فوق الآخر، كما هو واضح في الشكل ١.٩. ويمكننا في لغة جافا إنشاء مقطع فصل من خلال الكلاس المسمى *JSplitPane*.

مفهوم أساسي Key Concept

يعرض مقطع الفصل *split pane* مكونين جنباً إلى جنب أو واحداً فوق الآخر.

يتم ضبط اتجاه مقطع الفصل باستخدام قيم ثابتة تابعة للكلاس *JSplitPane*، ويمكن ضبط الاتجاه أما أثناء إنشاء الحاوي أو لاحقاً بشكل صريح. فالثابت *HORIZONTAL_SPLIT* يحدد أن المكونين سيعرضان جنباً إلى جنب. وعلى العكس من ذلك، فإن الثابت *VERTICAL_SPLIT* يحدد أن المكونين سيعرضان بحيث يكون أحدهما فوق الآخر.



الشكل رقم ١.٩ تشكيلات مقطع الفصل.

يحدد موقع شريط الفصل *divider bar* كمية المساحة المرئية المخصصة لكل مكون في مقطع الفصل *split pane*. ويمكن سحب شريط الفصل عبر الحاوي باستخدام الماوس. وأثناء تحرك شريط الفصل، فإن المساحة المرئية تزداد لمكون وتقل للآخر. وتتوزع المساحة الكلية لكلا المكونين فقط إذا تغير الحجم الكلي لمقطع الفصل.

يخضع *JSplitPane* للحجم الأدنى للمكونات التي يعرضها. ولهذا فإن شريط الفصل قد لا يسمح بإنقاص حجم أحد الأقسام خارج حد معين. وللتعامل مع هذه النقطة، يمكننا أن نغير الأحجام الصغرى للمكونات المعروضة.

يمكن تهيئة شريط الفصل الخاص بالكائن *JSplitPane* بحيث يمكن توسيعه في اتجاه أو في آخر، وذلك بضغطة ماوس واحدة. والوضع الافتراضي هو أن شريط الفصل لا يملك هذه الخاصية ولا يمكن تحريكه إلا عن طريق سحبه. وإذا ما تم تفعيل هذه الخاصية، فسيظهر شريط الفصل وعليه سهمين صغيرين في اتجاهين متضادين. وعملية الضغط على أحد هذين السهمين يسبب في تحريك شريط الفصل بالكلية في ذلك الاتجاه ويعمل بذلك على تكبير المساحة لأحد المكونات. ويتم تفعيل هذه الخاصية باستخدام الدالة *setOneTouchExpandable* والتي تأخذ وسيطاً بولياناً. يمكن أيضاً أن نقوم وبشكل صريح بضبط حجم شريط الفصل وموقعه الابتدائي .

هناك خاصية أخرى يمكن ضبطها لـ *JSplitPane* وهي فيما إذا كان سيتم باستمرار تكييف وإعادة رسم المكونات كلما تحرك شريط الفصل. إذا لم يتم تفعيل هذه الخاصية فسيتم استدعاء مدراء التخطيط للمكونات بمجرد توقف تحريك شريط الفصل. وهذه الخاصية في الوضع الافتراضي تكون غير مفعلة، ويمكن تفعيلها عند إنشاء كائن *JSplitPane* أو باستخدام الدالة *setContinuousLayout*.

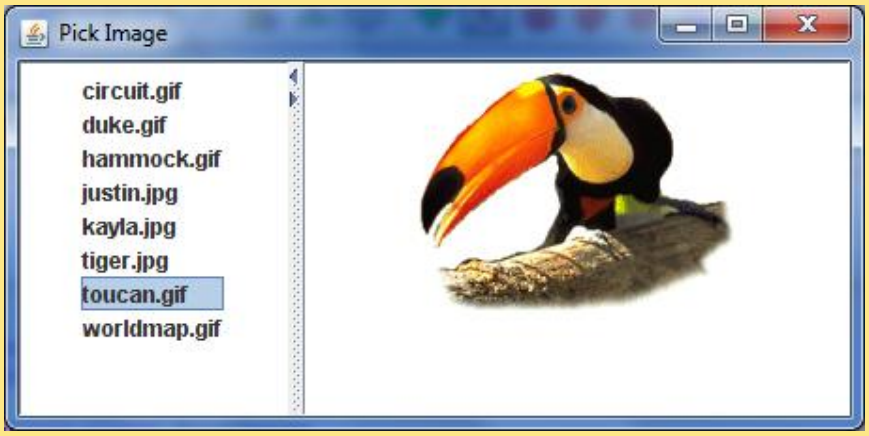
يمكن عمل تداخل لمقاطع الفصل عند طريق وضع مقطع فصل في أحد أو في كلا الجانبين لمقطع فصل آخر. مثلاً، يمكننا تقسيم الحاوي إلى ثلاثة أقسام عن طريق وضع مقطع فصل في المكون العلوي لمقطع فصل آخر. حينها سيكون لدينا شريطي فصل، أحدهما يفصل المساحة الكلية إلى قسمين رئيسيين، والآخر يقسم أحد هذه الأقسام إلى قسمين آخرين. وتعتمد كمية المنطقة المرئية لكل قسم على مواقع أشرطة الفصل.

يبين البرنامج الموضح في المثال ٧.٩ قائمة لأسماء ملفات صور خاصة بالمستخدم. وعندما يتم اختيار أحد أسماء الملف يتم عرض الصورة المناسبة في الجانب الأيمن من مقطع الفصل.

تم إنشاء مقطع الفصل في الدالة *main* وتم إضافته إلى الإطار الذي سيتم عرضه. وتم ضبط اتجاه مقطع الفصل باستخدام الثابت *HORIZONTAL_SPLIT*. بحيث يتم عرض اللوحة التي تحتوي قائمة الأسماء والمعنون الذي يحتوي الصورة جنباً إلى جنب. واستدعاء الدالة *setOneTouchExpandable* يسبب في عرض الأسهم على شريط الفصل الخاص بمقطع الفصل، مما يسمح للمستخدم بتوسيع المقاطع في اتجاه أو في آخر وذلك بضغطة واحدة على الماوس.

مثال رقم ٧.٩

```
//********************************************************************
//  PickImage.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا المثال استخدام مقطع الفصل والقائمة
//********************************************************************

import java.awt.*;
import javax.swing.*;

public class PickImage
{
   //-----------------------------------------------------------------
   //  الدالة الرئيسية: تنشئ وتعرض إطاراً يحوي مقطع فصل
   //  يختار المستخدم اسم الصورة من القائمة لعرضها
   //-----------------------------------------------------------------

   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Pick Image");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JLabel imageLabel = new JLabel();
      JPanel imagePanel = new JPanel();
      imagePanel.add (imageLabel);
      imagePanel.setBackground (Color.white);

      ListPanel imageList = new ListPanel (imageLabel);

      JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                     imageList, imagePanel);

      sp.setOneTouchExpandable (true);

      frame.getContentPane().add (sp);
      frame.pack();
      frame.setVisible(true);
   }
}
```



يقوم الكلاس *ListPanel* الموضح في المثال ٨.٩ بتعريف اللوحة التي تحتوي على قائمة أسماء الملفات. وقد قمنا باستخدام مكون القائمة *list* والمعرف بواسطة الكلاس *JList* لعرض قائمة أسماء الملفات. ويتم تهيئة محتويات القائمة كمصفوفة من الكائنات النصية يتم تمريرها إلى الباني الخاص بالكلاس *JList*.

وبشكل عام، تكون جميع خيارات المكون *JList* مرئية. وعندما يقوم المستخدم باختيار عنصر باستخدام الماوس فسيتم تظليله، وإذا ما تم اختيار عنصر آخر فسيتم تلقائياً إلغاء تظليل العنصر السابق.

يمكن تحديد مكونات الـ *JList* باستخدام مصفوفة كائنات يتم تمريرها إلى الباني. ويتم استخدام دوال الكلاس *JList* لإدارة القائمة بطرق مختلفة بما في ذلك عملية استرجاع العنصر المحدد حالياً.

لاحظ التشابهات والفروقات بين صناديق الاختيار المنسدلة *combo boxes* (والذي تم مناقشتها في القسم ٢.٩) وكائن القائمة *JList*. فكلاهما يسمح للمستخدم باختيار عنصر من مجموعة خيارات. لكن الخيارات التابعة للقائمة تكون دائماً معروضة ويكون الخيار الحالي مظللاً، بينما في صندوق الخيارات المنسدلة يتم عرض الخيارات فقط عندما يقوم المستخدم بضغط القائمة باستخدام الماوس، ويكون العنصر الوحيد الذي يتم عرضه طوال الوقت في صندوق الخيارات المنسدلة هو العنصر المختار حالياً.

مثال رقم ٨.٩

```
//********************************************************************
//  ListPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//   PickImage يمثل هذا الكلاس قائمة الصور لبرنامج
//********************************************************************

import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class ListPanel extends JPanel
{
   private JLabel label;
   private JList list;
```

```
   //-----------------------------------------------------------------
    //  البـاني: يـقـوم بـتـحـمـيـل أسـمـاء الـصـور إلـى الـقـائـمـة
    //-----------------------------------------------------------------
    public ListPanel (JLabel imageLabel)
    {
       label = imageLabel;

       String[] fileNames = { "circuit.gif",
                              "duke.gif",
                              "hammock.gif",
                              "justin.jpg",
                              "kayla.jpg",
                              "tiger.jpg",
                              "toucan.gif",
                              "worldmap.gif" };

       list = new JList (fileNames);
       list.addListSelectionListener (new ListListener());
       list.setSelectionMode (ListSelectionModel.SINGLE_SELECTION);

       add (list);
       setBackground (Color.white);
    }

    //*****************************************************************
    //  الـكـلاس الـتـالـي يـمـثـل الـمـسـتـمـع لـصـور الـقـائـمـة
    //*****************************************************************
    private class ListListener implements ListSelectionListener
    {
       public void valueChanged (ListSelectionEvent event)
       {
          if (list.isSelectionEmpty())
             label.setIcon (null);
          else
          {
             String fileName = (String)list.getSelectedValue();
             ImageIcon image = new ImageIcon (fileName);
             label.setIcon (image);
          }
       }
    }
}
```

يقوم كائن الـ *JList* بتوليد حدث اختيار قائمة *ListSelectionEvent*. وذلك كلما تم تغيير الخيار الحالي. وتحتوي الواجهة *ListSelectionListener* على دالة واحدة تسمى *valueChanged*. في هذا البرنامج يعمل الكلاس الخاص المسمى *ListListener* على تعريف المستمع لقائمة أسماء الملفات.

في الدالة *valueChanged* الخاصة بالمستمع يتم استدعاء الدالة *isSelectionEmpty* التابعة لكائن الـ *JList*، وذلك لتحديد إذا ما كان هناك قيمة مختارة حالياً أم لا. إذا لم يكن ذلك، يتم ضبط أيقونة المعنون على القيمة *null*. أما إذا كان هناك قيمة مختارة فيتم جلب اسم الملف باستخدام الدالة *getSelectedValue*. ثم بعدها يتم إنشاء الأيقونة الصورية المناسبة وعرضها على المعنون.

يمكن ضبط كائن الـ *JList* بحيث يمكن اختيار عناصر متعددة في وقت واحد، حيث يمكن أن يأخذ نمط اختيار القائمة *ListSelectionMode* واحداً من ثلاثة خيارات، كما هو موضح في الشكل ٢.٩.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

يتم تعريف نمط اختيار القائمة باستخدام الكائن *ListSelectionMode*. والوضع الافتراضي هو أن القائمة تسمح باختيار خيارات متعددة. ويمكننا بشكل صريح ضبط نمط اختيار القائمة وذلك عن طريق استدعاء الدالة *setSelectionMode* ثم استخدام الثوابت المعرفة ضمن الكلاس *ListSelectionModel*.

نمط اختيار القائمة *List Selection Mode*	الوصف *Description*
الاختيار المفرد *Single Selection*	يمكن فقط اختيار عنصر واحد في كل مرة.
الاختيار المتصل المفرد *Single Interval Selection*	يمكن اختيار عدة عناصر متصلة في الوقت نفسه.
الاختيار المتصل المتعدد *Multiple Interval Selection*	يمكن اختيار أي تركيبة من العناصر.

الشكل رقم ٢.٩ أنماط اختيار القائمة.

وقد قمنا في برنامج *PickImage* بضبط نمط اختيار القائمة على الاختيار المفرد *single selection*، وذلك لأنه سيتم عرض صورة واحدة في كل مرة. مع ذلك، لو سمحنا بالاختيارات المتعددة في برنامجنا هذا، فإن الدالة *getSelectedValue* سترجع العنصر الأول من العناصر المختارة وسيكون هو العنصر الذي سيتم عرض صورته.

هناك دالة مشابهة تسمى *getSelectedValues* تعمل على إرجاع مصفوفة من الكائنات تمثل العناصر التي تم اختيارها عند سماحنا بالخيارات المتعددة.

بدلاً من مصفوفة الكائنات النصية، يمكن أن يستقبل الباني الخاص بالكلاس *JList* مصفوفة كائنات *ImageIcon*، وفي هذه الحالة يمكن عرض الصورة على القائمة نفسها.

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

الفصل العاشر

١.١٠ الاستدعاء الذاتي في الرسومات RECURSION IN GRAPHICS

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

١.١٠ الاستدعاء الذاتي في الرسومات RECURSION IN GRAPHICS

إن لمفهوم الاستدعاء الذاتي *recursion* استخدامات عديدة في الصور *images* والرسومات *graphics*. وفي القسم التالي سنستكشف بعض أمثلة الاستدعاء الذاتي المعتمدة على الصور والرسومات.

الصور الرخامية Tiled Pictures

قم بعناية باختبار عرض الآبلت المسمى *TiledPictures* الموضح في المثال ١.١٠. يوجد فعلياً ثلاث صور متمايزة ضمن معرض صور الوحوش. وقد تم تقسيم المساحة الكلية إلى أربعة أرباع متساوية. في الربع الأيمن السفلي تم عرض صورة للعالم (توجد دائرة تشير إلى منطقة جبال الهملايا). أما الربع الأيسر السفلي فيحتوي على صورة لقمة جبل افرست. وفي الربع الأيمن العلوي صورة لماعز جبلي.

أما الجزء الشيق من الصورة فيوجد في الربع الأيسر العلوي. فهذا الربع يحتوي على نسخة للرسمة كاملة، بما في ذلك هذا الجزء نفسه. وفي هذا الجزء الصغير يمكنك رؤية الثلاث الصور البسيطة في أرباعها الثلاثة. ومرة أخرى، نرى أنه في الزاوية الشمالية العليا، تم تكرير الصورة (متضمنة نفسها). يستمر هذا التكرار لمستويات عديدة. وهذا شبيه بالتأثير الذي يتولد عند النظر إلى مرآة مناظرة لمرآة أخرى.

ويتم توليد هذا التأثير المرئي بسهولة كبيرة باستخدام الاستدعاء الذاتي. تعمل الدالة *init* التابعة للآبلت على التحميل الأولي للصور الثلاث. تقوم بعد ذلك الدالة *paint* باستدعاء الدالة *drawPictures*، والتي تستقبل وسيطاً يعرف حجم المنطقة التي تعرض فيها الصور. وتقوم برسم ثلاث صور باستخدام الدالة *drawImage*، وذلك باستخدام الوسائط التي تهيئ الصورة على الحجم والموضع الصحيحين. بعد ذلك يتم استدعاء الدالة *drawPictures* بتقنية الاستدعاء الذاتي لرسم الربع المتواجد في الزاوية الشمالية العليا.

في كل استدعاء، إذا كانت منطقة الرسم كبيرة بما فيه الكفاية، يتم استدعاء الدالة *drawPictures* مرة أخرى، باستخدام منطقة رسم أصغر. وفي الأخير، تصبح منطقة الرسم صغيرة جداً بحيث لا يمكن إتمام عملية الاستدعاء الذاتي. لاحظ أن الدالة *drawPictures* تفترض إحداثي نقطة الأصل *(0, 0)* كموقع نسبي للصور الجديدة، بعض النظر عن حجمها.

وتحدد الحالة الأساسية *base case* في هذا المثال الحجم الأدنى لمنطقة الرسم. ولأن الحجم يقل في كل مرة، يتم الوصول أخيراً إلى الحالة الأساسية ومن ثم يتوقف النداء الذاتي. وهذا هو السبب وراء كون الزاوية الشمالية العليا فارغة من الجزء الأصغر من الصور الكلية.

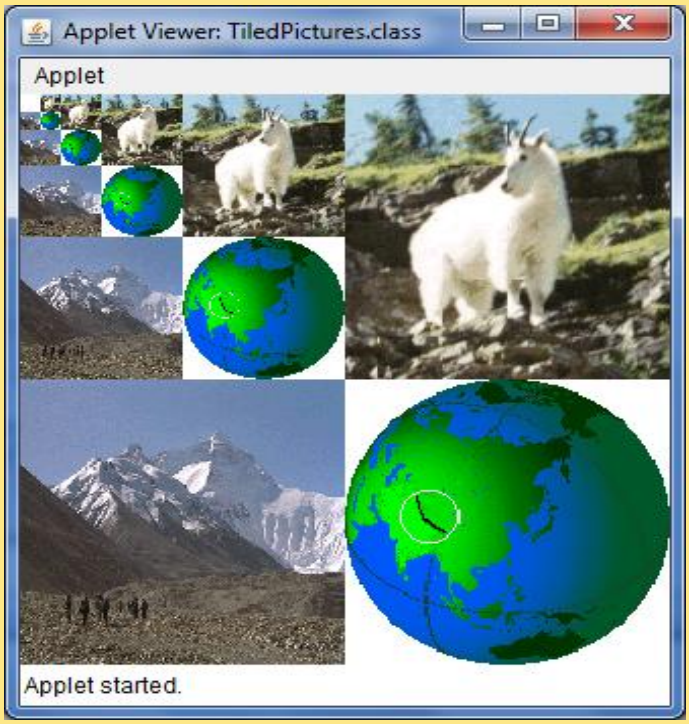
مثال رقم ١.١٠

```
//********************************************************************
// TiledPictures.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يشرح هذا الآبلت استخدام النداء الذاتي
//********************************************************************

import java.awt.*;
import javax.swing.JApplet;

public class TiledPictures extends JApplet
{
   private final int APPLET_WIDTH = 320;
   private final int APPLET_HEIGHT = 320;
   private final int MIN = 20;  // smallest picture size

   private Image world, everest, goat;


   //-----------------------------------------------------------------
   //  هنا يتم تحميل الصور
   //-----------------------------------------------------------------
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
    public void init()
    {
       world = getImage (getDocumentBase(), "world.gif");
       everest = getImage (getDocumentBase(), "everest.gif");
       goat = getImage (getDocumentBase(), "goat.gif");

       setSize (APPLET_WIDTH, APPLET_HEIGHT);
    }

    //-----------------------------------------------------------------
    //  دالة ترسم الصور الثلاث، ثم تستدعي نفسها ذاتياً
    //-----------------------------------------------------------------
    public void drawPictures (int size, Graphics page)
    {
       page.drawImage (everest, 0, size/2, size/2, size/2, this);
       page.drawImage (goat, size/2, 0, size/2, size/2, this);
       page.drawImage (world, size/2, size/2, size/2, size/2, this);

       if (size > MIN)
          drawPictures (size/2, page);
    }

    //-----------------------------------------------------------------
    //   drawPictures هنا يتم الاستدعاء الإبتدائي للدالة
    //-----------------------------------------------------------------
    public void paint (Graphics page)
    {
       drawPictures (APPLET_WIDTH, page);
    }
}
```

العرض Display



إن الشكل الكسوري *fractal* هو شكل مثلثي يمكن تكوينه باستخدام نفس النمط وتكريره بمقاييس واتجاهات مختلفة. وطبيعة الشكل الكسوري تنطبق على تعريف الاستدعاء الذاتي. وقد تزايد الاهتمام بالأشكال الكسورية بشكل كبير في السنوات الأخيرة، خصوصاً من العالم الرياضي البولندي *Benoit Mandelbrot* المولود في عام ١٩٢٤.

مفهوم أساسي Key Concept

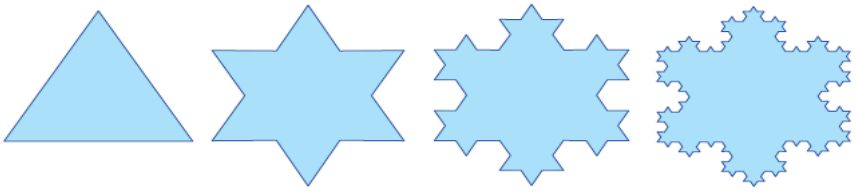
الشكل الكسوري *fractal* هو شكل مثلثي يعرف بشكل طبيعي بطريقة الاستدعاء الذاتي.

فقد أوضح هذا العالم أن الأشكال الكسورية تظهر في عدة أماكن في الرياضيات والطبيعة. وقد سهلت الحواسيب من عملية توليد الأشكال الكسورية ودراستها. وخلال الربع الأخير من القرن الماضي، اعتبرت الصورة البراقة الممتعة التي أمكن توليدها بواسطة الأشكال الكسورية شكلاً فنياً وأيضاً مجالاً رياضياً.

ومن الأمثلة الخاصة على الأشكال الكسورية ما يسمى بنبتة كخ الثلجية *Koch snowflake*، والتي سميت بذلك نسبة إلى العالم السويدي *Helge von Koch*. وهذا الشكل يبدأ بمثلث متساوي الأضلاع، والذي يعتبر شكل كخ الكسوري من الرتبة ١. ويتم بناء أشكال كخ كسورية ذات رتب أعلى عن طريق التعديل المتكرر لكل المقاطع الخطية للشكل.

فلإنشاء الرتبة الأعلى التالية لشكل كخ الكسوري، يتم تعديل كل مقطع خطي في الشكل عن طريق استبدال ثلثه الأوسط بنتوء حاد مكون من مقطعين خطيين، كلاهما يملكان نفس الطول للجزء الذي تم استبداله. وبالنسبة للشكل الكلي، نجد أن النتوء في كل مقطع خطي دائماً يشير إلى الخارج. ويوضح الشكل ١.١٠ رتب متعددة لأشكال كخ الكسورية. ونلاحظ أنه كلما زادت الرتبة، كلما اقترب الشكل الكسوري من شكل النبتة الثلجية *snowflake*.

ويرسم الآبلت الموضح في المثال ٢.١٠ نبتة كخ الثلجية لرتب عديدة مختلفة. وتعمل الأزرار الموجودة في أعلى الآبلت على تمكين المستخدم من زيادة وتنقيص رتبة الشكل الكسوري. وفي كل مرة يتم فيها الضغط على الزر، يتم إعادة رسم الصورة الكسورية. ويعمل الآبلت كمستمع للأزرار.



الشكل رقم ١.١٠ رتب متعددة لنبتة كخ الثلجية

مثال رقم ٢.١٠

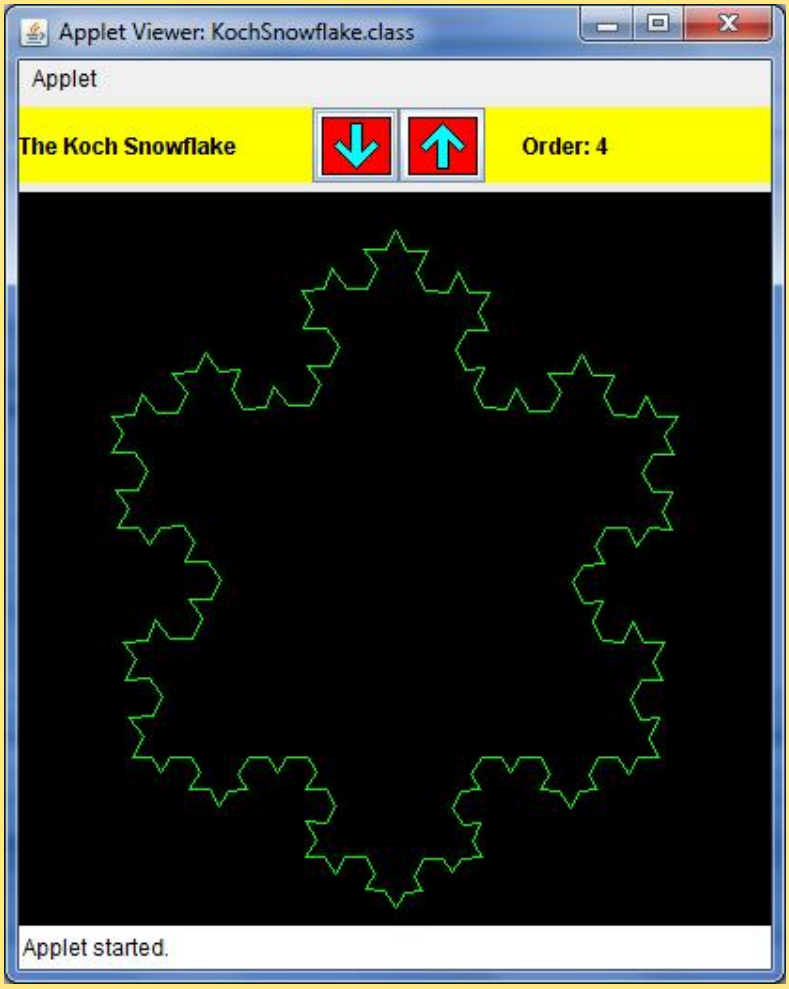
```
//*******************************************************************
//    KochSnowflake.java          المؤلف: Lewis/Loftus      المترجم : Almashraqi
//
//  يشرح هذا البرنامج استخدام الاستدعاء الذاتي في الرسومات
//*******************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KochSnowflake extends JApplet implements ActionListener
{
   private final int APPLET_WIDTH = 400;
   private final int APPLET_HEIGHT = 440;

   private final int MIN = 1, MAX = 9;
```

```
private JButton increase, decrease;
private JLabel titleLabel, orderLabel;
private KochPanel drawing;
private JPanel appletPanel, tools;

//-----------------------------------------------------------------
//  تقوم دالة بدء الآبلت بتهيئة مكوناته
//-----------------------------------------------------------------
public void init()
{
   tools = new JPanel ();
   tools.setLayout (new BoxLayout(tools, BoxLayout.X_AXIS));
   tools.setPreferredSize (new Dimension (APPLET_WIDTH, 40));
   tools.setBackground (Color.yellow);
   tools.setOpaque (true);

   titleLabel = new JLabel ("The Koch Snowflake");
   titleLabel.setForeground (Color.black);

   increase = new JButton (new ImageIcon ("increase.gif"));
   increase.setPressedIcon (new ImageIcon ("increasePressed.gif"));
   increase.setMargin (new Insets (0, 0, 0, 0));
   increase.addActionListener (this);

   decrease = new JButton (new ImageIcon ("decrease.gif"));
   decrease.setPressedIcon (new ImageIcon ("decreasePressed.gif"));
   decrease.setMargin (new Insets (0, 0, 0, 0));
   decrease.addActionListener (this);

   orderLabel = new JLabel ("Order: 1");
   orderLabel.setForeground (Color.black);

   tools.add (titleLabel);
   tools.add (Box.createHorizontalStrut (40));
   tools.add (decrease);
   tools.add (increase);
   tools.add (Box.createHorizontalStrut (20));
   tools.add (orderLabel);

   drawing = new KochPanel (1);

   appletPanel = new JPanel();
   appletPanel.add (tools);
   appletPanel.add (drawing);

   getContentPane().add (appletPanel);

   setSize (APPLET_WIDTH, APPLET_HEIGHT);
}

//-----------------------------------------------------------------
//  تحدد الدالة التالية أياً من الأزرار تم ضغطه، وتضبط الرتبة الجديدة
//  إذا كانت ضمن المدى المسموح
//-----------------------------------------------------------------
public void actionPerformed (ActionEvent event)
{
   int order = drawing.getOrder();
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
      if (event.getSource() == increase)
         order++;
      else
         order--;

      if (order >= MIN && order <= MAX)
      {
         orderLabel.setText ("Order: " + order);
         drawing.setOrder (order);
         repaint();
      }
   }
}
```

Display العرض



مثال رقم ٣.١٠

```
//*******************************************************************
//  KochPanel.java       المؤلف: Lewis/Loftus   المترجم : Almashraqi
//
//  يمثل هذا الكلاس سطح الرسم الذي سنرسم عليه نبتة سنو فليك
//*******************************************************************

import java.awt.*;
import javax.swing.JPanel;

public class KochPanel extends JPanel
{
   private final int PANEL_WIDTH = 400;
   private final int PANEL_HEIGHT = 400;

   private final double SQ = Math.sqrt(3.0) / 6;

   private final int TOPX = 200, TOPY = 20;
   private final int LEFTX = 60, LEFTY = 300;
   private final int RIGHTX = 340, RIGHTY = 300;

   private int current;  // current order

   //-----------------------------------------------------------------
   //  الباني: بتهيئة الترتيب الكسوري الإبتدائي باستخدام القيمة المحددة
   //-----------------------------------------------------------------
   public KochPanel (int currentOrder)
   {
      current = currentOrder;
      setBackground (Color.black);
      setPreferredSize (new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
   }

   //-----------------------------------------------------------------
   //  تقوم الدالة التالية برسم الشكل الكسوري باستخدام النداء الذاتي
   //  الحالة الأساسية هي الرتبة ١ ويتم فيها رسم خط مستقيم بسيط
   //  فيما عدا ذلك يتم حساب ثلاث نقاط وسيطة، ومنها يتم رسم كل مقطع خطي
   //  بشكل كسوري
   //-----------------------------------------------------------------
   public void drawFractal (int order, int x1, int y1, int x5, int y5,
                            Graphics page)
   {
      int deltaX, deltaY, x2, y2, x3, y3, x4, y4;

      if (order == 1)
         page.drawLine (x1, y1, x5, y5);
      else
      {
         deltaX = x5 - x1;  // distance between end points
         deltaY = y5 - y1;

         x2 = x1 + deltaX / 3;  // one third
         y2 = y1 + deltaY / 3;

         x3 = (int) ((x1+x5)/2 + SQ * (y1-y5));  // tip of projection
         y3 = (int) ((y1+y5)/2 + SQ * (x5-x1));
```


Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

```
            x4 = x1 + deltaX * 2/3;  // two thirds
            y4 = y1 + deltaY * 2/3;

            drawFractal (order-1, x1, y1, x2, y2, page);
            drawFractal (order-1, x2, y2, x3, y3, page);
            drawFractal (order-1, x3, y3, x4, y4, page);
            drawFractal (order-1, x4, y4, x5, y5, page);
         }
      }

      //-----------------------------------------------------------------
      //   drawFractal تنجز الدالة التالية الاستدعاءات الإبتدائية للدالة
      //-----------------------------------------------------------------
      public void paintComponent (Graphics page)
      {
         super.paintComponent (page);

         page.setColor (Color.green);

         drawFractal (current, TOPX, TOPY, LEFTX, LEFTY, page);
         drawFractal (current, LEFTX, LEFTY, RIGHTX, RIGHTY, page);
         drawFractal (current, RIGHTX, RIGHTY, TOPX, TOPY, page);
      }

      //-----------------------------------------------------------------
      //  تقوم هذه الدالة بضبط ترتيب الشكل الكسوري على القيمة المحددة
      //-----------------------------------------------------------------
      public void setOrder (int order)
      {
         current = order;
      }

      //-----------------------------------------------------------------
      //  ترجع هذه الدالة الرتبة الحالية
      //-----------------------------------------------------------------
      public int getOrder ()
      {
         return current;
      }
}
```

يتم رسم الصورة الكسورية *fractal image* على اللوحة المعرفة من خلال الكلاس الموضح في المثال ٣.١٠. تقوم الدالة *paint* بعمل النداءات الابتدائية لدالة الاستدعاء الذاتي المسماة *drawFractal*. وتمثل الاستدعاءات الثلاث في الدالة *paint* الأوجه الثلاثة الأصلية للمثلث متساوي الأضلاع والتي تمثل شكل كخ كسوري من الرتبة ١.

يمثل المتغير *current* رتبة الشكل الكسوري الذي سيتم رسمه. ويعمل أي استدعاء ذاتي للدالة *drawFractal* على إنقاص الرتبة بمقدار ١. وتتحقق الحالة الأساسية *base case* للنداء الذاتي عندما تصبح رتبة الشكل الكسوري ١، والتي تنتج مقطع خطي بسيط بين الإحداثيات المحددة بالوسائط.

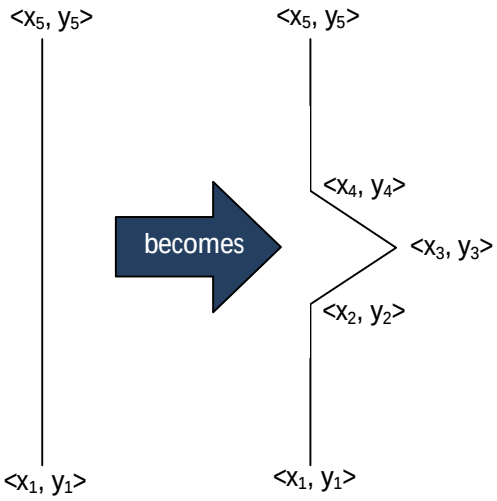
إذا ما كانت رتبة الشكل الكسوري أعلى من ١. فسيتم حساب ثلاث نقاط إضافية. وبالترابط مع هذه الوسائط، تشكل هذه النقاط المقاطع الخطية الأربعة للشكل الكسوري المعدل. ويبين الشكل ٢.١٠ هذا الانتقال.

واعتماداً على موضع النقطتين الطرفيتين للمقطع الخطي الأصلي، يتم حساب النقطة البعيدة عن المسار بينهما بمقدار الثلث وكذا النقطة البعيدة عن هذا المسار بمقدار الثلثين. إن حساب النقطة *<x3, y3>*، والتي تمثل رأس

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com

النتوء، يعتبر أكثر تعقيداً ويستخدم ثابت تبسيط يتضمن العديد من العلاقات المثلثية المتعددة. إن الحسابات المستخدمة لتحديد النقاط الثلاث الجديدة ليس لها أي علاقة بتقنية النداء الذاتي المستخدم في رسم الشكل الكسوري، ولذلك لن نناقش تفاصيل هذه الحسابات هنا.

ومن المزايا الشيقة في نبتة كخ الثلجية هو أن لها محيط غير منته ولكن مساحتها منتهية. فكلما زادت رتبة الشكل الكسوري، فإن المحيط يزداد بشكل أسي، بحد رياضي هو ما لا نهاية. مع ذلك، فإن أي مستطيل كبير بما فيه الكفاية للإحاطة بالشكل الكسوري من الرتبة الثانية لنبتة كخ الثلجية يكون أيضاً كافياً لاحتواء كل الأشكال الكسورية ذات الرتب الأعلى. تكون مساحة الشكل دائماً مقيدة، لكن المحيط يزداد طوله في التنامي إلى ما لا نهاية.



الشكل رقم ٢.١٠ انتقال كل مقطع خطي في نبتة كخ الثلجية

تم بحمد الله وتوفيقه ..

Abdulfattah Almashraqi Email: fatta7mail@yahoo.com