



مفاهيم متقدمة في البرمجة بلغة الجافا

كتاب مفاهيم متقدمة في البرمجة بلغة الجافا

يتكون الكتاب من اربعة فصول تشرح بعض المفاهيم المتقدمة في البرمجة بلغة الجافا، في الفصل الاول من الكتاب نشرح كيفية انشاء واجهات المستخدم الرسومية، في الفصل الثاني نوضح فيه برمجة تطبيقات Applet واهميتها، في الفصل الثالث نتحدث عن برمجة قواعد البيانات باستخدام لغة البرمجة جافا، في الفصل الرابع نتحدث عن بعض مفاهيم برمجة الشبكات

بسم الله الرحمن الرحيم



مفاهيم متقدمة في البرمجة بلغة الجافا

محمد محمود إبراهيم موسى

جامعة الزعيم الأزهري

كلية علوم الحاسوب وتقنية المعلومات

الفهرس:

4: مقدمة
5 الفصل الاول: واجهات المستخدم الرسومية (GUI)
5 الرسم والتلوين في جافا:
9 رسم الاشكال الهندسية:
14 واجهات المستخدم الرسومية:
36Applets تطبيقات: الفصل الثاني
43 الفصل الثالث: برمجة قواعد البيانات باستخدام جافا
44 تسجيل قاعدة البيانات في مصدر قواعد البيانات المفتوحة ODBC:
53 الفصل الرابع: برمجة الشبكات باستخدام جافا
53 الشبكات:
54:Socket Connection Stream والعميل باستخدام برمجة الخادم
60:Connectionless Datagram Socket والعميل باستخدام برمجة الخادم
65:المراجع

اهداء:

إلى اساتذتي الافاضل بكلية علوم الحاسوب وتقانة المعلومات جامعة الزعيم الازهري الي كل من علمني حرفا إلى طلاب كلية علوم الحاسوب وتقانة المعلومات جامعة الزعيم الازهري اهدي لكم هذا العمل المتواضع.

مقدمة:

الحمد لله الواحد المعبود، عم بحكمته الوجود، وشملت رحمته كل موجود، أحمده سبحانه وأشكره وهو بكل لسان محمود، وأشهد أن لا إله إلا الله وحده لا شريك له الغفور الودود، وعد من أطاعه بالعزة والخلود، وتوعد من عصاه بالنار ذات الوقود، وأشهد أن نبينا محمداً عبد الله ورسوله، صاحب المقام المحمود، واللواء المعقود، والحوض المورود، صلى الله عليه وعلى آله وأصحابه، الركع السجود، والتابعين ومن تبعهم من المؤمنين الشهود، وسلم تسليماً كثيراً إلى اليوم الموعود.

قدمنا في الكتاب الاول اساسيات البرمجة بلغة الجافا عدد من المفاهيم المتعلقة بالبرمجة بلغة الجافا بالإضافة الي المفاهيم الاساسية في البرمجة كائنية المنحى Object Oriented Programming في هذا الكتاب سنتعرف على بعض المفاهيم المتقدمة في البرمجة بلغة الجافا، يتكون الكتاب من اربعة فصول، الفصل الاول من هذا الكتاب نعرض بعض المفاهيم المتعلقة بالرسم و برمجة واجهات المستخدم الرسومية باستخدام الجافا ، في الفصل الثاني نتحدث بصورة مبسطة عن برمجة تطبيقات الانترنت (Applet)، الفصل الثالث فيه نوضح برمجة تطبيقات قواعد البيانات باستخدام لغة الجافا، الفصل الرابع نعرض فيه بعض المفاهيم المتعلقة ببرمجة الشبكات باستخدام لغة البرمجة جافا.

الفصل الاول: واجهات المستخدم الرسومية (GUI)

كل البرامج التي قمنا بكتابتها وتنفيذها في الكتاب الأول أساسيات البرمجة بلغة الجافا يتم التفاعل معها عن طريق شاشة سطر الأوامر Command، لكن هذا غير عمل لحد ما فتطبيقات الحاسوب التي نشاهدها ونستخدمها في حياتنا تظهر بصورة أكثر وضوحاً وأجمل شكلاً وأسهل استخداماً وأقرب لفهم المستخدم غير المختص بالحاسوب، فنجد الأشكال والرسوم والألوان المختلفة التي تجعل البرنامج جاذب للمستخدم، كما نلاحظ الأدوات المختلفة مثل القوائم والأزرار التي تجعل عملية تفاعل المستخدم مع البرنامج سهلة. يتكون هذا الفصل من قسمين، القسم الأول الرسومات في الجافا Java Graphics في هذا القسم نتعرف على الأدوات التي توفرها لغة جافا للرسم والتلوين على شاشة البرنامج. القسم الثاني واجهات المستخدم الرسومية (GUI) Graphical User Interface في هذا القسم سنتحدث عن مجموعة من أدوات لغة جافا الخاصة بتصميم واجهات المستخدم أو شاشات البرنامج، والتي تساعد المستخدم على التفاعل مع البرنامج بصورة أسهل وأبسط ولا تتطلب أي معرفة بالبرمجة ولغاتها.

الرسم والتلوين في جافا:

توفر لغة البرمجة جافا مجموعة كبيرة من أدوات الرسم والتلوين موجودة في المكتبة او الحزمة java.awt وقبل أن نتطرق إليها يجب أولاً أن نتعرف على الفئة JFrame والموجود في الحزمة javax.swing، تظهر أي فئة ترث من الفئة JFrame تلقائياً على الشاشة على شكل نافذة مستطيلة عند التنفيذ، ولهذه الفئة خصائص ودوال كثيرة للتحكم في حجم وموقع ظهور وتنسيق هذه الشاشة. نتعرض في هذا القسم لبعض خصائص الفئة JFrame، تكمن أهمية هذه الفئة بالنسبة للرسم في أنها تمثل التي سيتم عليها الرسم أو التلوين، كما ان لهذه الفئة إمكانية نداء دالة الرسم paint تلقائياً عند انشاء كائن من فئة ترث خصائص الفئة JFrame. يوضح البرنامج التالي كيفية رسم نافذة على الشاشة تحتوي على العبارة Java Graphics.

```
import java.awt.*;
import javax.swing.*;
public class FirstGraphics extends JFrame
{
    public FirstGraphics ()
    {
        super ("My First Window");
        setSize(200 , 100);
        setVisible (true);
    }
    public void paint (Graphics g)
    {
```

```

    g.drawString("Java Graphic" , 50 , 50);
}
public static void main(String args[])
{
    FirstGraphics fg = new FirstGraphics();
}
}

```

الفئة Graphics:

عند تعريف كائن من الفئة **FirstGraphics**، يتم رسم نافذة صغيرة على الشاشة ويتم الحصول على مواصفات هذه النافذة من المشيد constructor، فيمكن كتابة عنوان لهذه النافذة باستخدام مشيد الفئة الاب عن طريق استدعاء **super()** وتمرير عنوان النافذة كوسيط، يمكن كذلك تحديد أبعاد النافذة باستخدام الدالة **setSize** والتي تستقبل طول وعرض النافذة، والوحدة في القياس المستخدمة هي **Pixel**، تستخدم الدالة **setVisible** لتحديد ما إذا كنا نرغب في إظهار النافذة التي تم إعدادها أم لا، فالوسيط **true** يعني إظهارها فوراً والوسيط **false** يعني عدم إظهارها حالياً وربما تم إعدادها للظهور لاحقاً خلال التنفيذ، الدالة **paint** هي دالة يتم نداؤها تلقائياً بعد تنفيذ المشيد مباشرة، وتحتوي على جميع التعليمات الخاصة بالرسم والتلوين، وذلك باستخدام الكائن من الفئة **Graphics** والذي يحتوي على جميع الدوال الخاصة بالرسم والتلوين، نجد في هذا المثال الدالة **drawString** والتي تستخدم لرسم نص على النافذة، وهو في هذا المثال **Java Graphics**، ويحتل الرقمان التاليان اللذان تستقبلهما الدالة الأبعاد التي يبدأ عندها رسم النص.



شكل 1-1 مخرجات البرنامج

الفئة Color:

تحتوي الحزمة **java.awt** إضافة إلى الفئة **Graphics** على الفئة **color** والتي توفر إمكانية تحديد وتغيير ألوان الأشكال والخلفيات، ويمكن إعداد لون معين لاستخدامه في الرسم بإحدى طريقتين:
 1. باستخدام الألوان التي الموجود في الفئة **color** أصلاً على شكل ثوابت.


```
Color c = color.RED;
```

ومن الألوان الموجودة أيضاً ' GREEN, ' PINK, ' GRAY, ' YELLOW, ' BLACK, ' WHITE و BLUE وغيرها.

2. باستخدام نظام الألوان (Red-Green-Blue) RGB، والذي ينص على أن كل لون هو ناتج مزج نسبة معينة من اللون الأحمر والأخضر والأزرق، تتراوح هذه النسبة بين الصفر و225 لكل لون، فمثلاً يتكون اللون الأحمر من 255 من اللون الأحمر و0 من اللونين الأخضر والأزرق، ولذلك فإن مكونات اللون الأحمر هي (255.0.0) وبنفس الطريقة ينتج اللون الأبيض من مزج الألوان الثلاثة بنسبة 225 ليكون رمز اللون الأبيض هو (225.255.255). ونجد أن رمز اللون البرتقالي هو (255,200,0) لاستخدامنا الألوان الرئيسية بهذه النسب للحصول على اللون البرتقالي وهكذا. وبالتالي يمكننا الحصول على عدد كبير من الألوان بمختلف الدرجات باستخدام نظام RGB وتستخدم الكثير من برامج الرسم بالحاسوب هذا النظام.

```
Color c = new Color(128 , 20 , 64) ;
```

الفئة Font:

تساعدنا الفئة Font على اختيار نوع وحجم وشكل خط الرسم على الشاشة.

```
Font font = new Font("Times New Roman" , Font.PLAIN , 20) ;
```

لإنشاء Font، نحتاج لثلاثة معطيات، أولها اسم الخط المستخدم، ويمكن أن يكون أي خط من الخطوط المعروفة في نظام التشغيل الذي المستخدم، ثانياً شكل الخط، فيجب تحديد ما إذا كان الخط عادي plain أو مائل italic أو عريض bold، وذلك باستخدام الثوابت الموجودة بالفئة font، وأخيراً يجب تحديد حجم خط الكتابة. يوضح المثال التالي استخدام الألوان والخطوط المختلفة لرسم عبارات على الشاشة.

```
import java.awt.*;
import javax.swing.*;
public class FontsAndColors extends JFrame
{
    public FontsAndColors ()
    {
        super("Fonts and Color");
        setSize(300 , 400);
        setVisible(true);
    }
    public void paint(Graphics g)
```

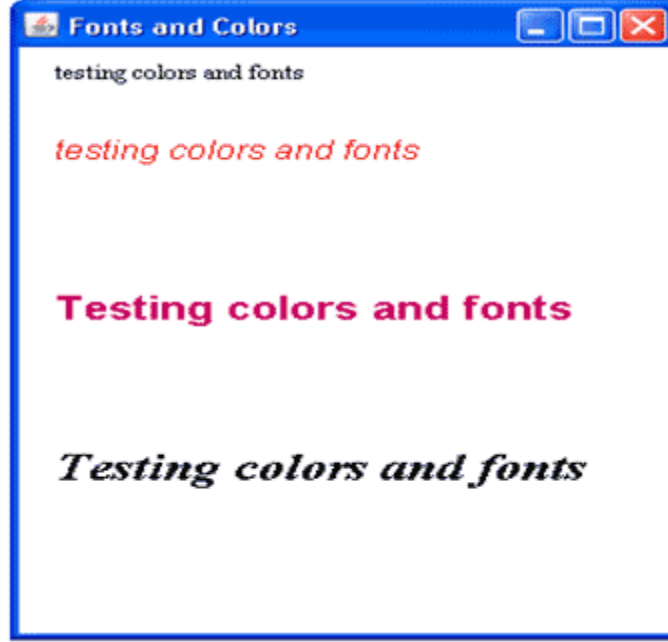


```

{
    Color color = Color.BLACK;
    Font font = new Font("serif" , Font.PLAIN , 12);
    g.setColor(color);
    g.setFont(font);
    g.drawString("TEST COLOR AND FONTS" , 20 , 50);
    g.setColor(Color.RED);
    g.setFont(new Font("monospaced" , Font.ITALIC,
16));
    g.drawString("testing colors and fonts" , 20 ,
100);
    g.setColor(new Color(200, 0, 100));
    g.setFont(new Font("sans serif" , Font.BOLD, 20));
    g.drawString("Testing colors and fonts" , 20 ,
200);
    g.setColor(color);
    g.setFont(new Font("times new roman" , Font.BOLD +
Font.ITALIC, 24));
    g.drawString("Testing colors and fonts" , 20 ,
300);
}
public static void main(String args[])
{
    new FontsAndColors();
}
}

```

تستخدم الدالة `setColor` الموجودة في الفئة `Graphics` لتحديد اللون الذي سيتم استخدامه للرسم على النافذة وتستقبل اللون المطلوب كوسيط، كذلك تستخدم الدالة `setFont` لتحديد خصائص الخط الذي ستظهر به الكتابة. نلاحظ أنه يمكن للكتابة أن تكون **Bold** و *Italic* معاً، وذلك باستخدام علامة الجمع.



شكل 1-2 مخرجات البرنامج

رسم الاشكال الهندسية:

الوسائط	الوظيفة	اسم الدالة
x1: الاحداثي السيني لنقطة بداية الخط. y1: الاحداثي الصادي لنقطة نهاية الخط. x2: الاحداثي السيني لنقطة نهاية الخط. y2: الاحداثي الصادي لنقطة نهاية الخط.	رسم خط مستقيم	Drawline (x1 , y1 , x2 , y2) .
X: الاحداثي السيني لنقطة العلوية اليسرى للمستطيل.	رسم مستطيل. رسم مستطيل ممتلئ.	DrawRect (x , y , width , height) . FillRect (x , y , width , height) .

<p>Y: الاحداثي الصادي للمنطقة العلوية اليسرى للمستطيل. Width: عرض المستطيل. Height: ارتفاع المستطيل. B: متغير من النوع المنطقي Boolean يأخذ القيمة true ليظهر المستطيل مرتفع او False ليظهر المستطيل منخفض.</p>	<p>رسم مستطيل ثلاثي الابعاد. رسم مستطيل ممتلئ ثلاثي الابعاد.</p>	<p>Draw3DRect(x , y , width , height , b). Fill3DRect(x , y , width , height , b).</p>
<p>X: الاحداثي السيني للمنطقة العلوية اليسرى للمستطيل الذي يمس الشكل البيضاوي. Y: الاحداثي الصادي للمنطقة العلوية اليسرى للمستطيل الذي يمس الشكل البيضاوي. Width: عرض الشكل البيضاوي.</p>	<p>رسم شكل بيضاوي. رسم شكل بيضاوي ممتلئ.</p>	<p>DrawOval(x , y , width , height). FillOval(x , y , width , height).</p>

Height: ارتفاع الشكل البيضاوي.		
X: الاحداثي السيني لمركز الدائرة. Y: الاحداثي الصادي لمركز الدائرة. Width: عرض الدائرة. Height: ارتفاع الدائرة. Start: الزاوية التي يبدأ منها رسم القوس (0 - 360). Angle: قيمة الزاوية واتجاه القوس (-360 الي 360).	رسم قوس من دائرة. رسم قطاع دائري ممتلئ.	DrawArc(x , y , width , height , start , angle). FillArc(x , y , width , height , start , angle).

المثال التالي يستخدم بعض من هذه الدوال لرسم الاشكال الهندسية.

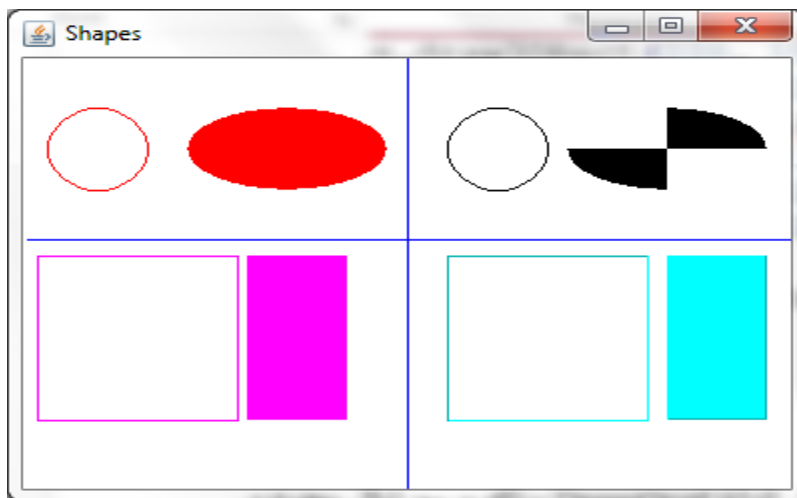
```
import java.awt.*;
import javax.swing.*;
public class Shapes extends JFrame
{
    public Shapes ()
    {
        super ("Shapes" );
        setSize (400,300);
        setVisible (true);
    }
    public void paint (Graphics g)
    {
        g.setColor (Color.BLUE);
        g.drawLine (10, 140, 400, 140);
        g.drawLine (200, 30, 200, 300);
        g.setColor (Color.RED);
        g.drawOval (20, 60, 50, 50); // circle
    }
}
```

```

g.fillOval(90, 60, 100, 50);
g.setColor(Color.MAGENTA);
g.drawRect(15, 150, 100, 100); // square
g.fillRect(120, 150, 50, 100);
g.setColor(Color.CYAN);
g.draw3DRect(220, 150, 100, 100, false);
g.fill3DRect(330, 150, 50, 100, true);
g.setColor(Color.BLACK);
g.drawArc(220, 60, 50, 50, 0, 360); // circle
g.fillArc(280, 60, 100, 50, 0, 90);
g.fillArc(280, 60, 100, 50, 270, -90);
}
public static void main(String args[ ])
{
    new Shapes ();
}
}

```

نلاحظ في هذا المثال أننا رسمنا دائرة بطريقتين: عن طريق الدالة DrawOval وذلك برسم شكل بيضاوي عرضه يساوي ارتفاعه، وعن طريق الدالة DrawArc ورسم قطاع زاويته المركزية 360. مستخدمين الدالة DrawArc ، قمنا برسم قطاع دائرة يبدأ من الزاوية صفر، ويفتح بزاوية 90 في الاتجاه الموجب، وقطاع دائرة آخر يبدأ من الزاوية 270 ويفتح بزاوية 90 في الاتجاه السالب، أي يبدأ القطاع عند الزاوية 270 وينتهي عند الزاوية 180.



شكل 1-3 مخرجات البرنامج

يوضح لنا هذا المثال كيفية استخدام دوال الفئة Graphics لرسم وتلوين الأشكال الهندسية المنتظمة. هناك دوال أخرى تستخدم لرسم الأشكال الأكثر تعقيداً مثل المضلعات المختلفة والنجوم وأي خطوط متصلة لتكون شكلاً مفتوحاً أو مغلقاً. تستخدم الدالة drawPolygon لرسم الأشكال المغلقة كثيرة الأضلاع وتستقبل مصفوفة بالإحداثيات السينية لجميع النقاط المكونة للشكل، ومصفوفة ثنائية بالإحداثيات الصادية لجميع النقاط المكونة للشكل، ثم عدد هذه النقاط. تقوم الدالة بتوصيل النقاط مع بعضها البعض بخطوط مستقيمة حسب ترتيب النقاط، ثم وصل النقطة الأخيرة مع الأولى للحصول على الشكل المغلق، تستخدم الدالة fillPolygon لملء الشكل المغلق باللون المعين، هناك أيضاً الدالة drawPolyline، ويشبه الـ polyline الـ polygon إلا أن الـ polyline هو شكل مفتوح أي أنه مجموعة من النقاط المتصلة ببعضها عن طريق خطوط مستقيمة وتستقبل الدالة كذلك مصفوفة الإحداثيات السينية للنقاط ومصفوفة الإحداثيات الصادية للنقاط وعددها، ولا تقوم بإغلاق الشكل في النهاية، يوضح المثال التالي استخدام الدالة drawPolygon و drawPolyline.

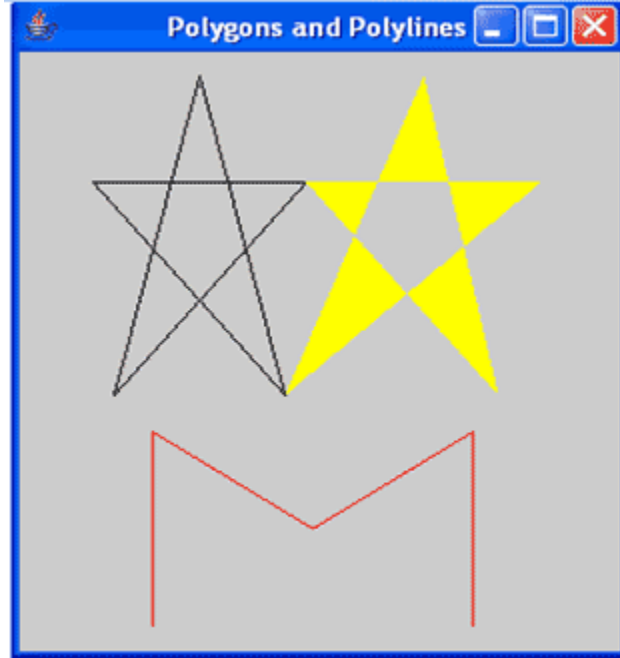
```
import java.awt.*;
import javax.swing.*;

public class PolygonsAndPolylines extends JFrame
{
    public PolygonsAndPolylines ()
    {
        super ("Polygons and Polylines");
        setSize (300, 200);
        setVisible (true);
    }
    public void paint (Graphics g)
    {
        int x1 [] = {50, 125, 100, 75, 150};
        int y1 [] = {70, 200, 35, 200, 70};
        g.setColor (Color.BLACK);
        g.drawPolygon (x1, y1, 5); // star
        int x2 [] = {150, 240, 205, 140, 260};
        int y2 [] = {90, 200, 35, 200, 90};
        g.setColor (Color.YELLOW);
        g.fillPolygon (x2, y2, 6); // filled star
        int x3 [] = {50, 50, 125, 200, 200};
        int y3 [] = {400, 300, 350, 300, 400};
        g.setColor (Color.RED);
        g.drawPolyline (x3, y3, 7); // M
    }
}
```

```

}
public static void main(String args[ ])
{
    new PolygonsAndPolylines ( ) ;
}
}

```



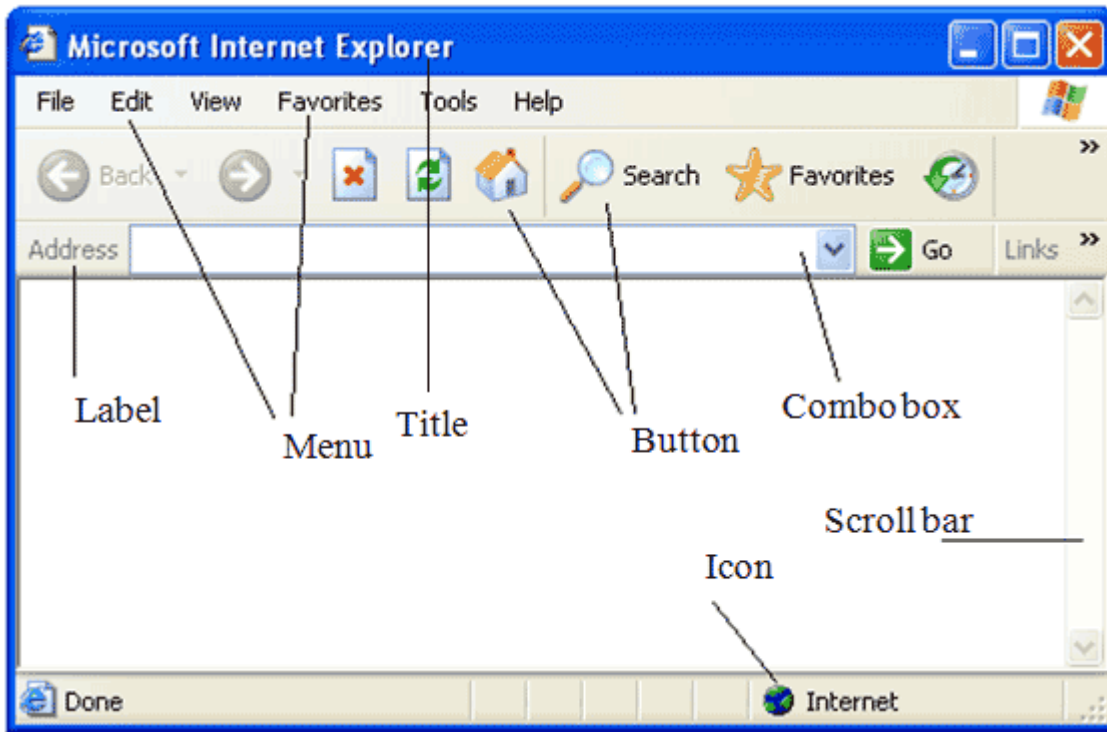
شكل 4-1 الخرج من البرنامج

لاحظ الفرق بين الـ polygon والـ polyline، حيث لا يتم إغلاق الشكل عند رسم polyline عن طريق وصل النقطة الأخيرة مع الأولى كما في polygon. باستخدام دوال الرسم مع العدد الكبير للألوان التي توفرها جافا، يمكن أن تصمم مختلف الشاشات واللوحات، تستخدم دوال الفئة Graphics التي تطرقنا إليها لرسم الأشكال البسيطة والتصاميم المختصرة، وتوجد دوال أخرى تستخدم لرسم وتصميم الشاشات المعقدة مثل استخدام التظليل واختلاف سمك فرشاة الرسم وتصميم وحدات صغيرة يتم تكرارها للتلوين بدلا عن استخدام لون واحد بسيط. جميع هذه المميزات موجودة في الفئة Graphics2D أو الرسم ببعدين tow dimensional، وتحتوي على كل الأدوات اللازمة للرسم.

واجهات المستخدم الرسومية:

تعتبر واجهة المستخدم الرسومية من أهم الأدوات والأكثر استخداماً في لغات البرمجة لجميع التطبيقات، فهي تمكن المبرمج من تصميم واجهة رسومية للبرنامج تسهل وتبسط من تعامل المستخدم مع البرنامج. تحتوي

أغلب برامج الحاسوب التي نراها اليوم على واجهات رسومية، وهي أي شاشة أو نافذة نراها عند تشغيل برنامج معين، وقد يحتوي البرنامج الواحد على عدد من الواجهات أو الشاشات، ويمكن أن تحتوي الواجهة على معلومات عن البرنامج أو يمكن أن تحتوي على منطقة لإدخال البيانات، ويتم تصميم البرنامج بحيث تظهر كل خطواته ومعالجته على الواجهة ليتمكن المستخدم من متابعة البرنامج بدراسة تامة ودون ملل. فمتصفح الإنترنت يحتوي على واجهة تسمح للمستخدم بإدخال العنوان المطلوب، وتحتوي على أزرار متعددة الاستخدامات، والمنطقة التي تظهر فيها الصفحة هي منطقة المخرجات ويمكن للمبرمج متابعة تحميل الصفحة بمراقبة شريط التحميل أسفل الصفحة.



شكل 1-5 واجهة متصفح الانترنت

توفر لغة جافا مجموعة كبيرة من الفئات والدوال لتصميم الواجهات الرسومية لتفاعل البرامج مع المستخدم وهي موجودة في الحزمة `java.awt` والحزمة `javax.swing`. في القسم السابق تعرفنا على الفئة `JFrame` وكيف أنها تظهر على الشاشة وتسمح للمبرمج بتصميم الأشكال عليها. تستخدم الفئة `JFrame` كأساس لتصميم واجهات المستخدم الرسومية وتضاف جميع مكونات الواجهة إليها، نتعرف خلال هذا القسم على أهم هذه المكونات واستخداماتها وكيف يتعامل معها المستخدم.

تعتبر الفئة Container الموجودة في الحزمة java.awt من أهم الفئات التي تستخدم إلى جانب الفئة JFrame، وتكمن أهميتها في أنه يتم ربطها بال-JFrame ثم تتم إضافة جميع مكونات النافذة (components) إليها.

في المثال التالي نوضح كيفية إعداد شاشة تحتوي على عبارات وصور باستخدام الفئة JLabel والتي تستخدم لكتابة نص أو إظهار صور على نافذة البرنامج.

```
import java.awt.*;
import javax.swing.*;
public class LabelTest extends JFrame
{
    public LabelTest ()
    {
        super ("Label Test");
        Container container = getContentPane ();
        container.setLayout (new FlowLayout ());
        JLabel label1 = new JLabel ("This is a label");
        container.add (label1);
        Icon image = new ImageIcon ("Image.gif");
        JLabel label2 = new JLabel (image);
        container.add (label2);
        JLabel label3 = new JLabel ();
        label3.setText ("This is another label");
        container.add (label3);
        setSize (300, 150);
        setVisible (true);
    }
    public static void main (String args [ ])
    {
        new LabelTest ();
    }
}
```

يبدأ المشيد بتحديد عنوان النافذة، ثم تعريف الوعاء container الذي ستضاف إليه المكونات فيما بعد وربطها بال-JFrame باستخدام الدالة getContentPane. يتم تحديد طريقة تنظيم المكونات على النافذة بواسطة الدالة setLayout، وستتعرف على الطرق المختلفة لتنظيم الشاشة في نهاية الفصل. بعد ذلك يتم تعريف المكونات المناسبة التي نرغب في إضافتها إلى النافذة. وفي هذا المثال تم تعريف JLabel لطباعة نص على الشاشة، حيث نقوم بتمرير هذا النص إلى المشيد الخاص بالفئة JLabel ثم يضاف إلى الوعاء container باستخدام

الدالة add. ولعرض صورة على الشاشة، نقوم بتجهيزها أولاً باستخدام reference من الفئة Icon وكائن من الفئة ImageIcon، مع توضيح اسم الملف الذي يحتوي على الصورة المطلوبة لحظة إنشاء الكائن، وهو في هذا المثال الملف image.gif والذي نفترض وجوده في نفس المسار الذي يوجد فيه هذا البرنامج. بعدها يتم إنشاء JLabel باستخدام الكائن image الذي يحتوي على الصورة، ثم إضافة الـ JLabel إلى الوعاء container. وأخيراً يتم تحديد حجم الـ JFrame باستخدام الدالة setSize وعرضه عن طريق تمرير القيمة true إلى الدالة setVisible.



شكل 6-1 مخرجات البرنامج

نلاحظ من المثال أن هناك عدد من الطرق يمكن بواسطتها يتم إنشاء labels عن طريق استخدام المشيدات المختلفة. فهناك مشيد يستقبل نصاً ليظهر على الشاشة، وآخر يستقبل ايقونة لعرضها بواسطة JLabel، وثالث يستقبل نصاً وصورة معاً. وهناك خيار لتعريف JLabel باستخدام الدالة setText و setIcon اللتين تستقبلان نصاً وايقونة على التوالي.

نستخدم الـ labels عموماً لكتابة العناوين والتعليمات على نافذة البرنامج، وقد يكون إحدى الوسائل المستخدمة لعرض مخرجات البرامج. يستخدم حقل النص text filed لاستقبال المدخلات من المستخدم، فهي تسمح له باستخدام لوحة المفاتيح وكتابة النصوص المختلفة بداخلها. توفر الفئة JTextField إمكانية تخاطب المستخدم مع البرنامج، حيث يسمح له بإدخال القيم المختلفة ليتعامل معها البرنامج. وقبل أن نشرح برنامجاً يتعامل مع الفئة JTextField، نريد أن نوضح طريقة تفاعل البرنامج مع الأحداث الصادرة من المستخدم، والتي تعرف بمعالجة الأحداث Event Handling. الفكرة في معالج الأحداث هي أن يقوم المستخدم بعمل حدث معين في البرنامج مثل ضغط زر أو طباعة حرف أو الضغط على المفتاح Enter أو تحريك الفأرة أو أي حدث آخر، فيقوم البرنامج بتنفيذ فعل محدد استجابة لأفعال المستخدم. ولعملية معالجة الأحداث ثلاثة مكونات:

1. **مصدر الحدث event source**: ويمكن أن يكون أي مكون في نافذة البرنامج أو مفتاح في لوحة المفاتيح أو الفأرة.
2. **كائن الحدث event object**: كل حدث معين في البرنامج ينشأ أو يتولد عنه كائن محدد، يبحث هذا الكائن عن المنطقة التي ستعالج الحدث في البرنامج وتقوم بالاستجابة المناسبة للحدث.
3. **معالجة الحدث event handler**: وهو جزء من البرنامج دالة عادة يكون في حالة استماع دائم لنوع محدد من الأحداث يعالجها ويقوم بالاستجابة المحدد، ولذلك يعرف أيضا بالمستمع Listener. هناك عدة أنواع من الأحداث التي ترتبط بمكون بعينه، وخلال الأمثلة التالية نتعرف على هذه الأحداث إلى جانب المكونات components المرتبطة بها. أول هذه الأحداث هو الحدث Action، واحدي أسباب تولده هو عند الضغط على مفتاح enter عندما يكون مؤشر الكتابة موجود بداخل text field. والمثال التالي يشرح ذلك.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TextFieldTest extends JFrame
{
    Handler handler = new Handler ();
    JTextField text;
    public TextFieldTest ()
    {
        super ("TextField Test");
        Container container = getContentPane ();
        container.setLayout ( new FlowLayout ());
        container.add (new JLabel ("Enter your name and
press enter"));
        text = new JTextField (20);
        text.addActionListener ( handler );
        container.add ( text );
        setSize (250, 100);
        setVisible (true) ;
    }
    public static void main (String args [ ] )
    {
        new TextFieldTest ();
    }
}

```

```

class Handler implements ActionListener
{
    public void actionPerformed (ActionEvent event )
    {
        JOptionPane.showMessageDialog(null, "Hello " +
event.getActionCommand());
    }
}

```

يتم تعريف textfield عن طريق إنشاء كائن من الفئة JTextFiled الموجودة في الحزمة javax.swing بالطول 20 ثم إضافته إلى الوعاء container باستخدام الدالة add ويكون ذلك بعد أن يتم تحديد معالجة الحدث event handler وهو كائن من الفئة الذي يحتوي على الدالة التي ستنفذ تلقائياً عند حدوث الحدث. وفي حالة حدوث event من نوع(ActionEvent) يتم تنفيذ الدالة actionPerformed تلقائياً عند تولد هذا الحدث بشرط أن يتم ربط معالج الحدث مسبقاً مع المكون component، أي تحديد أن هذا المستمع ينتظر تولد الأحداث من هذا المكون component، وذلك باستخدام الدالة addActionListener العبارة:

```

text.addActionListener ( handler );

```

تعني أن الكائن المسمى handler مشتق من التي تحتوي على الدالة actionPerformed التي تنتظر تولد أحداث من النوع(ActionEvent) من الـ JTextFiled المسمى text. وليكون الكائن قادراً على معالجة أحداث من النوع(ActionEvent) يجب أن يكون implements للواجهة ActionListener ويجب أن يحتوي على تعريف الدالة actionPerformed الموجودة في الواجهة ActionListener والموجودة بدورها في الحزمة java.awt.event بنفس الطريقة الموضحة. وبعد هذا، فإن مستمع الحدث الخاص بالكائن يكون في حاله استماع دائم لأي حدث من نوع(ActionEvent) ينشأ من الـ text filed عن طريق الضغط على المفتاح enter بداخله، ومتى ما حدث ذلك، يتم تنفيذ الدالة actionPerformed تلقائياً.

كلما تم الضغط على مفتاح enter في وجود المؤشر داخل حقل النص text يتم تنفيذ الدالة actionPerformed الموجودة في الفئة Handler، لأنه تم تعيينها كمستمع للأحداث من النوع(ActionEvent) الصادرة من الكائن text. عند حدوث الحدث، يتولد كائن من النوع(ActionEvent) وينتقل إلى الدالة actionPerformed التي يتم تنفيذ خطواتها. أي أننا نكتب بداخل الدالة actionPerformed التعليمات التي نرغب في تنفيذها عند الضغط على المفتاح enter من داخل حقل النص text. وفي هذا البرنامج نستخدم الدالة showMessageDialog التابعة لفئة JOptionPane والموجود في الحزمة javax.swing، وهي تقوم

بإظهار رسالة على الشاشة تستقبلها كـ String، هذا النص مكون من العبارة Hello بالإضافة الي الاسم المكتوب بداخل حقل النص text والذي حصلنا عليه من الكائن event باستخدام الدالة `getActionCommand`، والتي تستخدم عموماً للحصول على النص المكتوب بداخل المكون component الذي حدث منه الحدث. نلاحظ وجود المستمع Handler في فئة مستقلة عن بقية البرنامج، وهو ليس لازماً كما سنرى في الأمثلة القادمة.



شكل 7-1 مخرجات البرنامج

الازرار Buttons:

من مكونات البرامج الأكثر أهمية واستخداماً في البرنامج الأزرار buttons، فهي تستخدم في البرامج لحدث المستخدم لبداية عملية معينة. مثلاً في متصفح الإنترنت، نجد الزر Back للعودة إلى الصفحة السابقة والزر Stop لوقف التحميل، والزر Refresh لإعادة التحميل. وفي برنامج الآلة الحاسبة نجد زر يمثل كل رقم وكل عملية حسابية. إذن الزر هو مشغل لعملية معينة يضغط عليه المستخدم عندما يرغب في تنفيذ العملية. تستخدم الفئة JButton لإضافة أزرار إلى الشاشة للقيام بعمليات محددة. يتم إنشاؤها ووضع اسم عليها باستخدام المشيد ثم إضافتها إلى الواجهة. عند الضغط على زر معين، يتولد حدث من نوع `ActionEvent`، ولذلك لا بد من وجود مستمع ينتظر حدوث الحدث للرد عليه. ولذلك يتم كتابة العبارات المطلوب تنفيذها عند الضغط على الزر بداخل الدالة `actionPerformed`. يوضح المثال التالي عدداً من الأزرار تقوم بوظائف مختلفة.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ButtonTest extends JFrame implements
ActionListener
{
    JPasswordField password;
    JButton ok, clear, exit;
    public ButtonTest ()
    {
        super ("Button Test");
        Container c = getContentPane ();
        c.setLayout (new FlowLayout ());
        c.add (new JLabel ("Enter Password"));
        password = new JPasswordField (20);
        c.add (password);
        ok = new JButton ("OK");
        ok.addActionListener (this);
        c.add (ok);
        clear = new JButton ("Clear");
        clear.addActionListener (this);
        c.add (clear);
        exit = new JButton ("Exit");
        exit.addActionListener (this);
        c.add (exit);
        setSize (350, 100);
        setVisible (true);
    }
    public void actionPerformed (ActionEvent event )
    {
        if (event.getSource () == exit)
            System.exit (0);
        else
            if (event.getSource () == clear)
                password.setText (" ");
            else
                { // ok
                    if (password.getText ().equals ("Sudan"))
                        JOptionPane.showMessageDialog (this,
"Password is correct");
                }
    }
}

```



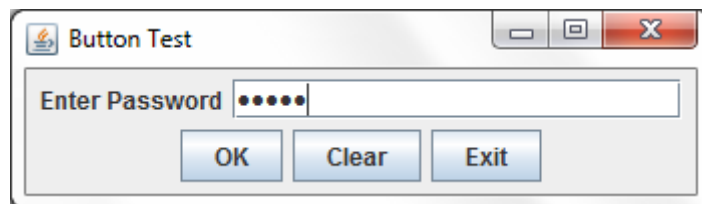
```

else
    JOptionPane.showMessageDialog(this, "Sorry..
wrong password");
}
}
public static void main(String args[])
{
    new ButtonTest ();
}
}

```

عند الضغط على الزر exit، يتم إغلاق النافذة وإنهاء تنفيذ البرنامج باستخدام الدالة System.exit(). وعند الضغط على الزر clear يتم إخلاء محتويات حقل النص text باستخدام الدالة setText() التي تظهر النص الذي يتم تمريره إليها في حقل النص. عند الضغط على الزر ok يتم فحص كلمة السر التي أدخلها المستخدم، إذا كانت الكلمة هي sudan تظهر الرسالة password is correct، وإذ كانت غير ذلك تظهر الرسالة Sorry. Wrong password.

لاحظ أن JPasswordField هو فئة تشبه الفئة JTextField. إلا أن الحروف التي يطبعها بداخله المستخدم لا تظهر على الشاشة، بل تكون مخفية على شكل نجوم، وهو أمر مرغوب عند إدخال كلمات السر والبيانات الخاصة. تم ربط الأزرار الثلاثة مع مستمع واحد، أي أن نفس الدالة actionPerformed ستنفذ عند الضغط على أي من الأزرار، ولذلك يجب التفريق بين الأزرار لمعرفة أيها تم الضغط عليها ليتم تنفيذ رد الفعل المناسب. تستخدم الدالة getSource التابعة للفئة(ActionEvent للحصول على اسم الكائن الذي حدث فيه الحدث، أو اسم الزر الذي تم الضغط عليه. في المثال أعلاه تم استخدام الدالة getSource في عبارات if للتعرف على الزر الذي تم الضغط عليه. لاحظ أن الفئة المستمعه في هذا المثال هو نفسها فئة البرنامج الرئيسي، ولذلك تم تعريفه على أنها تستخدم الواجهة ActionListener. وعند ربط الأزرار مع المستمع تم استخدام المؤشر this، لأن الكائن من هذه الفئة هو نفسه المستمع الذي يحتوي على الدالة actionPerformed.





شكل 8-1 الخرج من البرنامج

تعتبر الأزرار buttons وحقول النص textfields من أهم المكونات الرئيسية في الواجهات الرسمية لأنها تمثل الأساس في عملية الإدخال والمعالجة والإخراج، ويمكن أن تستخدم إلى جانب المكونات الأخرى. توفر جافا مجموعة كبيرة من المكونات التي سنتطرق للمكونات الأكثر استخداماً خلال الأمثلة التالية. أحد هذه المكونات هو أزرار الاختيار radiobutton، وهي مجموعة من الخيارات يقوم المستخدم بإدخال أحدها بالنقر على الخيار بالفأرة، ولا يمكن اختيار أكثر من خيار واحد. ويمكن أن نجد هذا النوع من الأزرار مثلاً عند اختيار درجة صعوبة اللعبة فيكون الاختيار بين مبتدئ ومتوسط ومتقدم، أو عند ملء استمارة معينة ببيانات شخصية يتم اختيار النوع إما ذكر أو أنثى وهكذا. توفر جافا الفئة JRadioButton لتستخدم مع الخيارات التي لا يسمح فيها باختيار أكثر من خيار في نفس الوقت. يتم تعريف عدد من الكائنات من الفئة JRadioButton بعدد الخيارات الموجودة، ويتم تعريف كائن من الفئة ButtonGroup وإضافة جميع أزرار الاختيار إليها لتوضح أن هذه الأزرار المضافة تابعة لنفس المجموعة ولا يمكن اختيار أكثر من خيار واحد من بينها. عندما يتم اختيار radio button معين ضمن مجموعة يتم إنشاء حدث من النوع ItemEvent، ولذلك فإن الكائن الذي يستمع لهذا النوع من الأحداث يجب أن يستخدم الواجهة ItemListener، تحتوي هذه الواجهة على دالة واحدة فقط هي itemStateChanged، وبالتالي يجب تعريفها ويتم وضع العبارات المطلوب تنفيذها عندما يختار المستخدم خياراً معيناً ضمن الخيارات الموجودة بداخلها. المثال التالي يشرح ذلك.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class RadioButtonTest extends JFrame implements
ItemListener
{
    JLabel text;
    JRadioButton plain, bold, italic, capital, small;
    ButtonGroup styleGroup , caseGroup;
    public RadioButtonTest ()
    {
        super ("Radio Buttons" );
        Container c= getContentPane ();
        c.setLayout (new FlowLayout ());
        styleGroup = new ButtonGroup ();
        plain = new JRadioButton ("Plain", true);
        plain. addItemListener (this);
        styleGroup.add (plain);
        c.add (plain);
        bold = new JRadioButton ("Bold", false);
        bold.addItemListener (this);
        styleGroup.add (bold);
        italic = new JRadioButton ("italic", false);
        italic.addItemListener (this);
        styleGroup.add (italic);
        c.add (italic);
        caseGroup = new ButtonGroup ();
        capital = new JRadioButton ("Capital", true);
        capital.addItemListener (this);
        caseGroup.add (capital);
        c.add (capital);
        small = new JRadioButton ("small", false);
        small.addItemListener (this);
        caseGroup.add (small);
        c.add (small);
        text = new JLabel ("Radio Buttons");
        c.add (text);
        setSize (180, 140);
        setVisible (true);
    }
}

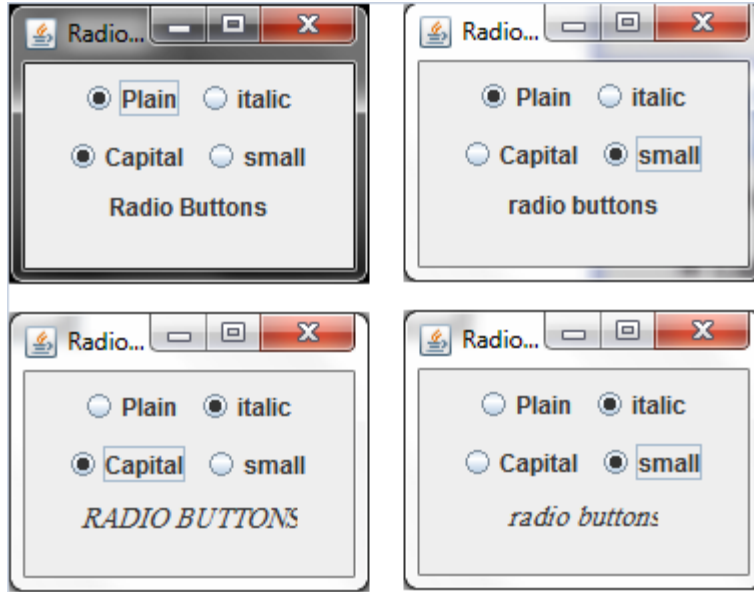
```

```

public void itemStateChanged(ItemEvent event)
{
    if(event.getSource() == plain)
        text.setFont(new Font("serif", Font.PLAIN,
14));
    if (event.getSource() == bold)
        text.setFont(new Font("serif", Font.BOLD, 14));
    if (event.getSource() == italic)
        text.setFont(new Font("serif", Font.ITALIC,
14));
    if (event.getSource() == capital)
        text.setText("RADIO BUTTONS");
    if (event.getSource() == small)
        text.setText("radio buttons");
}
public static void main(String args[])
{
    new RadioButtonTest();
}
}

```

يتم جمع أزرار الاختيار bold و plain و italic ضمن مجموعة واحدة لكي يتم اختيار خيار واحد على الأكثر من بينها، حيث يتم تعديل خط الكتابة بناء على هذا الاختيار. أما الأزرار small و capital فتستخدم لتحويل حالة الحروف إلى كبيرة أو صغيرة حسب اختيار المستخدم. وإذا تم إضافتها لنفس المجموعة السابقة لن نستطيع أن نختار حالة الحروف إلى جانب خط الكتابة، لأننا لن نستطيع اختيار أكثر من خيار واحد ضمن المجموعة، يتم تعريف زر الاختيار بتحديد النص الذي نرغب بظهوره إلى جانب الزر، كما يتم تحديد الخيار الابتدائي للمجموعة، فالزر الذي يحمل القيمة true يمثل الخيار الابتدائي، ويكون الزر الذي يحمل القيمة false غير مظلل ابتدائياً. عند اختيار زر معين، يتولد حدث من النوع ItemEvent وبالتالي يتم نداء الدالة itemStateChanged تلقائياً، ويتم التفريق بين الأزرار المختلفة باستخدام الدالة getSource.



شكل 9-1 الخرج من البرنامج

توفر جافا الفئة JCheckBox لاختيار اكثر من خيار في نفس الوقت، فهي تشبه الفئة JRadioButton في الشكل والاستخدام، إلا أنه عند استخدام checkbox يمكن أن نختار عدداً من الخيارات في الوقت الواحد، وبالتالي لا نحتاج لتقسيمها إلى مجموعات، يتولد عند الضغط على JCheckBox حدث من النوع ItemEvent أيضاً.

القوائم JList و JComboBox:

من الأدوات الهامة لتصميم الشاشات القوائم، وتوفر جافا نوعين من القوائم هي JList و JComboBox ، ونسبة للشبة الكبير بينهما نتطرق هنا إلى إحداهما وهي JComboBox . ويوضح البرنامج التالي كيفية تعريف القائمة والتعامل معها:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ComboBoxTest extends JFrame implements
ItemListener
{
    JLabel codeLabel;
    String[] countries = {"Egypt", "Jordan", "Saudi
Arabia", "Sudan", "United Arab Emirates"};
    String[] codes = {"20", "962", "966", "249", "971"};
```

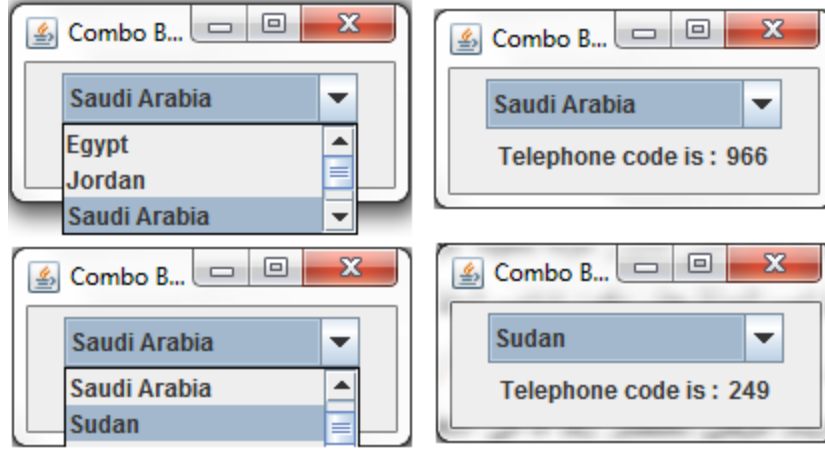
```

JComboBox countryCodes;
public ComboBoxTest ()
{
    super ("Combo Box Test");
    Container c = getContentPane ();
    c.setLayout (new FlowLayout ());
    countryCodes = new JComboBox (countries);
    countryCodes.setMaximumRowCount (3);
    countryCodes.addItemListener (this);
    c.add (countryCodes);
    c.add (new JLabel ("Telephone code is :"));
    codeLabel = new JLabel ();
    c.add (codeLabel);
    setSize (200, 100);
    setVisible (true);
}
public void itemStateChanged (ItemEvent e)
{
    int index = countryCodes.getSelectedIndex ();
    codeLabel.setText (codes [index]);
}
public static void main (String args [])
{
    new ComboBoxTest ();
}
}

```

يتم تعريف مصفوفة من النوع String تستخدم في إنشاء القائمة، وهي في هذا البرنامج المصفوفة countries حيث تضع أسماء خمس دول في قائمة، وتسمح باختيار دولة بالنقر على القائمة لفتحها. عندما تفتح القائمة يكون طولها مقدار ثلاثة أسطر، والتي تم تحديدها باستخدام الدالة setMaximumRowCount . عند اختيار دولة يتولد حدث من نوع ItemEvent ومن ثم يتم نداء الدالة itemStateChanged تلقائياً، ولمعرفة العنصر الذي تم اختياره من القائمة تستخدم الدالة getSelectedIndex والتي ترجع قيمة integer تمثل ترتيب العنصر الذي تم اختياره. في هذا المثال يترتب على اختيار دولة معينة عرض مفتاح الدولة، حيث تم وضع المفاتيح في المصفوفة codes بنفس ترتيب الدولة حتى يكون ترتيب المفتاح في مصفوفة المفاتيح هو نفسه ترتيب الدول في القائمة. مثلاً عند اختيار Sudan من القائمة تعيد الدالة getSelectedIndex القيمة 3،

وهو ترتيب Sudan في القائمة، ويتم عرض العنصر رقم 3 في المصفوفة codes باستخدام codeLable وهو 249 والذي يمثل مفتاح السودان.



شكل 10-1 مخرجات البرنامج

يمكن الفرق بين القائمة من نوع JComboBox والقائمة من نوع JList في أن JComboBox يتم النقر عليها وفتحها لاختيار عنصر ثم تغلق مجدداً، بينما القائمة من النوع JList هي قائمة مفتوحة دائماً، وتظهر عناصرها على الشاشة. كذلك تسمح القائمة من نوع JList باختيار عدة عناصر في المرة الواحدة إذا تم إعدادها لذلك، بينما لا يمكن اختيار أكثر من عنصر من عناصر القائمة من نوع JComboBox.

توفر جافا مجموعة كبيرة من الأدوات المستخدمة لتنسيق الشاشات ودعم عمليات الإدخال والإخراج في البرنامج. وتقوم جميعها على مبدأ تعريف وإعداد الأداة لاستخدامها، ثم انتظار المستخدم ليتعامل معها، ثم الاستجابة لأفعاله عن طريق معرفة نوع الحدث الناتج عن التعامل مع الأداة المعينة. ومن هذه الأدوات مساحات النصوص JTextArea والتي تسمح بطباعة عدد من السطور على الشاشة. هناك أيضاً القائمة الرئيسية JMenu لإنشاء القوائم الرئيسية للشاشات. يمكن كذلك التعامل مع الجداول باستخدام JTable إضافة إلى الكثير من الأدوات المختلفة والتي لا يتسع المقام لذكرها.

كثيراً ما تتفاعل البرامج والشاشات مع لوحة المفاتيح أي أن هناك أحداث نرغب في التحكم بها بواسطة لوحة المفاتيح. فمثلاً نلاحظ في كثير من البرامج العبارة: (للمتابعة اضغط على أي مفتاح). في لغة جافا تستخدم الواجهة KeyListener لمتابعة مدخلات لوحة المفاتيح، وتحتوي على ثلاث دوال هي:

1. KeyPressed: يتم نداؤها تلقائياً عند الضغط على أي مفتاح في لوحة المفاتيح مثل: 2، ALT، Enter، A، * وغيرها.

2. keyTyped: ويتم نداؤها تلقائياً عند الضغط على أي مفتاح يظهر على الشاشة مثل: S، 2، *، A وغيرها.

3. keyReleased: ويتم نداؤها تلقائياً عند إنهاء الضغط على المفتاح أو تركه بعد الضغط عليه.
يتم تعريف هذه الدوال حتى وإن لم نرغب في استخدامها جميعاً، ويجب أن يتم تسجيل العنصر المعين بواسطة الدالة addKeyListener لتتفاعل الدوال مع أحداث لوحة المفاتيح التي تحدث من ذلك العنصر.
يوضح المثال التالي كيفية التعامل مع هذه الدوال.

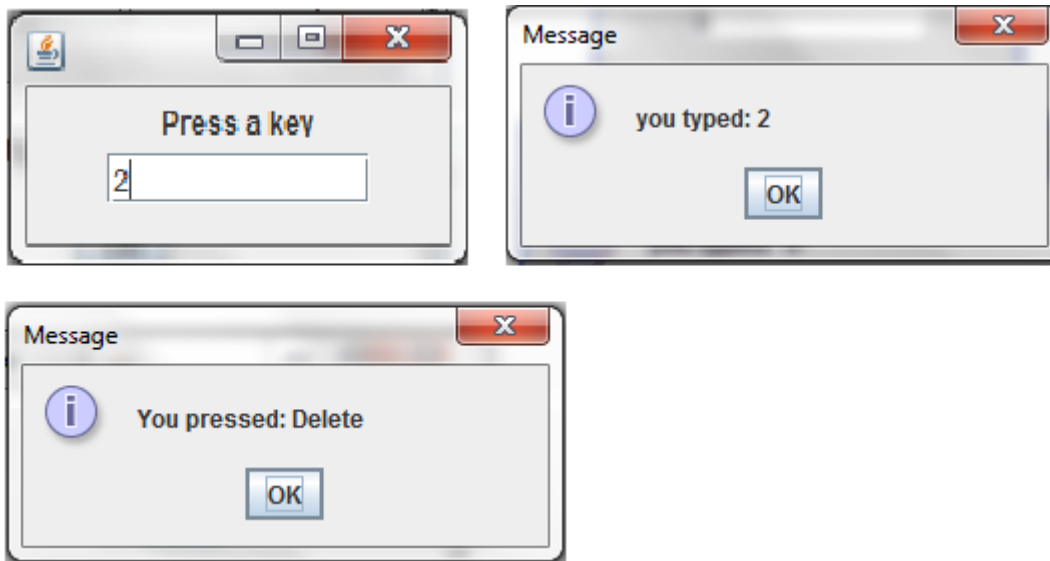
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class KeyboardApplication extends JFrame
implements KeyListener
{
    JTextField text;
    public KeyboardApplication ()
    {
        Container c= getContentPane ();
        c.setLayout (new FlowLayout ());
        c.add (new JLabel ("Press a key"));
        text = new JTextField (10);
        text.addKeyListener (this);
        c.add (text);
        setSize (200, 100);
        setVisible (true);
    }
    public void keyTyped (KeyEvent e)
    {
        JOptionPane.showMessageDialog (this, "you typed: "
+ e.getKeyChar ());
    }
    public void keyPressed (KeyEvent e)
    {
        JOptionPane.showMessageDialog (this, "You pressed: "
+ e.getKeyText (e.getKeyCode ()));
    }
}
```

```

public void keyReleased(KeyEvent e)
{
    JOptionPane.showMessageDialog(this, "You
released: " + e.getKeyText(e.getKeyCode()));
}
public static void main(String args[])
{
    new KeyboardApplication();
}
}

```

لدى الفئة `KeyEvent` عدة دوال تستخدم للتفاعل مع الأحداث الصادرة من لوحة المفاتيح. تستخدم الدالة `getKeyChar` للحصول على الحرف الذي تم الضغط على مفتاحه، وذلك عندما يكون المفتاح خاصاً بحرف يظهر على الشاشة مثل 9 و w وغيرها. وعند الضغط على المفاتيح الأخرى مثل Shift أو F1 أو غيرها، يتم استخدام الدالة `getKeyCode` للحصول على الرمز العددي الممثل لهذا المفتاح، ثم تحويل هذا الرمز إلى نص باستخدام الدالة `getKeyText` والتي تستقبل هذا الرمز، ومن ثم طباعته على الشاشة.



شكل 11-1 مخرجات البرنامج

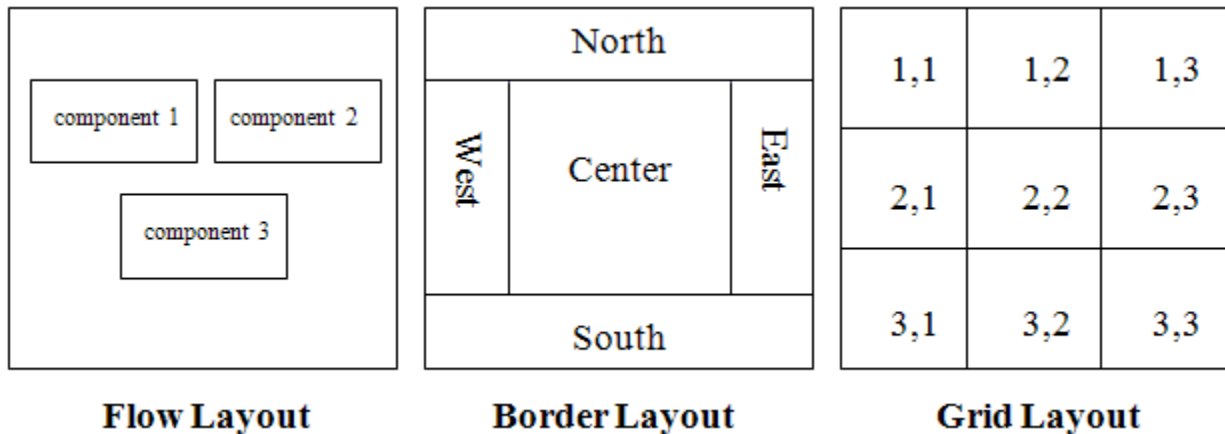
يوضح هذا البرنامج كيف أن لغة جافا تدعم برمجة التطبيقات التي تتفاعل مع مدخلات لوحة المفاتيح. توفر جافا كذلك طرقاً لتفاعل البرامج مع الأحداث التي تصدر عن الفأرة `mouse` مثل الضغط على زر الفأرة

(النقر) أو تركه، ودخول الفأرة إلى منطقة معينة أو خروجها منه. وكلها عبارة عن دوال تابعة للواجهة MouseListener والواجهة MouseMotionListener يمكن استخدامها للتفاعل مع حركات الفأرة.

الوعاء Panel:

هناك أيضا مفهوم ال- Panel وهو عبارة عن وعاء يمكن أن يحتوي على عدد من المكونات بداخله ومن ثم إضافته للوعاء container الرئيسي لنافذة البرنامج، ويتم إنشاء panel باستخدام الفئة JPanel وتكمن أهميتها في ترتيب مكونات النافذة، واستخدام طرق مختلفة للترتيب وتنسيق المكونات داخل النافذة. وقبل أن نوضح استخدام الفئة JPanel في مثال، نريد أن نقدم بعض الطرق المستخدمة لتصميم وتنظيم المكونات داخل النافذة Layout. هناك عدد كبير من الطرق التي يمكن بها تنظيم وعرض مكونات الشاشة و سنتناول هنا أشهر ثلاث طرق وهي:

- **Flow Layout**: وهو يقوم بوضع المكونات داخل الشاشة بالترتيب الذي تمت إضافتهم به، حتى تمتلئ النافذة عرضياً، وحينها يقوم بمتابعة وضع المكونات في السطر التالي وهكذا.
- **Border Layout**: ويقسم الشاشة إلى خمسة أجزاء وهي: الوسط الذي يخصص له أكبر مساحة، إضافة إلى الشرق والغرب والشمال والجنوب، ويجب تحديد القسم الذي نرغب في إضافة العنصر إليه في عبارة الإضافة add.
- **Grid Layout**: وهو يقسم النافذة إلى شبكة من الخانات متساوية الحجم يحدد عددها (طولها وعرضها) المبرمج. يتم إضافة أول عنصر إلى الخانة أعلى شمال النافذة، وتتواصل إضافة العناصر من الشمال إلى اليمين بالصف الأول حتى يمتلئ، ليتم الانتقال إلى الصف الثاني وهكذا.



شكل 12-1 أشهر طرق تنظيم نافذة البرنامج

في جميع الأمثلة السابقة تم استخدام طريقة التنظيم Flow Layout لتنظيم مكونات النافذة للبرنامج، حيث تظهر بنفس الترتيب الذي تمت إضافته به، ويأخذ كل عنصر حجمه العادي. خلال الأمثلة التالية نستخدم طريقة Border Layout و Grid Layout للتحكم في الشكل النهائي لنافذة البرنامج إلى جانب استخدام الواجهة JPanel للتعامل مع عدة Layouts في نفس النافذة.

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutTest extends JFrame
{
    JButton b[];
    public GridLayoutTest ()
    {
        super ("Testing grid layout");
        Container c = getContentPane ();
        c.setLayout (new GridLayout (4,4));
        b = new JButton[16];
        for (int i = 0; i <16; i ++)
        {
            b[i] = new JButton (String.valueOf (i));
            c.add (b[i]);
        }
        setSize (300, 300);
        setVisible (true);
    }
    public static void main (String args [])
    {
        new GridLayoutTest ();
    }
}
```

في هذا المثال تم استخدام حلقة for لإنشاء الأزرار وإضافتها، حيث إن النص الذي يظهر فوق كل زر هو عبارة عن ترتيب ذلك الزر في المصفوفة. لاحظ أن هذه الأزرار ليس لها وظيفة معينة وإنما تم تصميم الشاشة بغرض توضيح شكل نافذة البرنامج عند استخدام طريقة التصميم Grid Layout ومقارنتها بطريقة Flow Layout.



شكل 13-1 مخرجات البرنامج

```

import java.awt.*;
import javax.swing.*;
public class BorderLayoutTest extends JFrame
{
    JLabel title, nameLabel, passwordLabel;
    JTextField name, password;
    JButton signUp, login, clear, exit;
    JPanel textPanel, buttonPanel;
    public BorderLayoutTest ()
    {
        super ("Testing Border Layout");
        Container c = getContentPane ();
        c.setLayout (new BorderLayout ());
        title = new JLabel ("Login page");
        c.add(title, BorderLayout.NORTH);
        nameLabel = new JLabel ("Name");
        passwordLabel = new JLabel ("Password");
        name = new JTextField(20);
        password = new JTextField(20);
        textPanel = new JPanel ();
        textPanel.setLayout (new FlowLayout ());
        textPanel.add(nameLabel);
        textPanel.add(name);
    }
}

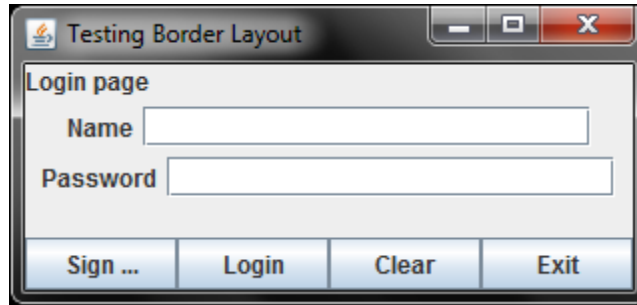
```

```

textPanel.add(passwordLabel);
textPanel.add(password);
c.add(textPanel, BorderLayout.CENTER);
signUp = new JButton("Sign Up");
login = new JButton("Login");
clear = new JButton("Clear");
exit = new JButton("Exit");
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1, 4));
buttonPanel.add(signUp);
buttonPanel.add(login);
buttonPanel.add(clear);
buttonPanel.add(exit);
c.add(buttonPanel, BorderLayout.SOUTH);
setSize(320, 150);
setVisible(true);
}
public static void main(String args[])
{
    new BorderLayoutTest();
}
}

```

في هذا البرنامج تم تصميم نافذة البرنامج باستخدام Border Layout، وتم أولاً إضافة JLabel إلى المنطقة الشمالية بالشاشة، وكما نعلم فإن هذا النوع له خمس مناطق فقط، وبالتالي يسمح بإضافة خمسة عناصر فقط. لكن البرنامج يحتوي على مكونات كثيرة، فمثلاً هناك أربعة أزرار نرغب في إضافتها جميعاً بجنوب النافذة، ولذلك نضيفهم جميعاً إلى JPanel منظم بالطريقة المناسبة (في هذا المثال Layout Grid)، وفي النهاية يضاف الـ JPanel جنوب النافذة. ونلاحظ أن طريقة تصميم الـ JPanel تكون مستقلة تماماً عن الطريقة المستخدمة في النافذة الأساسية. بنفس الطريقة نعرف JPanel ونصممه بطريقة Flow Layout ونضيف الـ JLabels والـ TextFields والتي نرغب بإضافتها في وسط النافذة، وأخيراً نضيف الـ JPanel إلى وسط النافذة. نلاحظ أنه يتم تحديد أقسام النافذة الخمسة التي يوفرها Layout Border باستخدام ثوابت موجودة في class BorderLayout وهي NORTH، SOUTH، EAST، WEST، CENTER.



شكل 14-1 مخرجات البرنامج

في الأمثلة السابقة استعنا بالوعاء JPanel لاستخدام طرق تصميم مختلفة عن التي تم استخدامها مع نافذة البرنامج الأساسية. يمكن أن تستخدم الـ JPanels أيضاً لفصل تنسيق بعض العناصر عن بقية عناصر الشاشة، ومن أمثلة هذا التنسيق لون الخلفية ومواصفات الخط المستخدم للكتابة والمسافات بين العناصر داخل المنطقة المحددة.

توفر جافا طرقاً أخرى لتصميم شاشات البرامج، ومنها GridBagLayout و CardLayout و BorderLayout ولكل منها شكله العام وطريقة استخدامه. عندما لا نحدد layout معين للنافذة، يتم افتراضه BorderLayout وهو طريقة التصميم الافتراضية لأي JFrame، بينما طريقة التصميم الافتراضية لأي JApplet هي BorderLayout يمكن في بعض البرامج تحديد نوع التصميم null عند استخدام الدالة setLayout ، وعندها لن يتم تقسيم نافذة البرنامج بأي طريقة، وإنما يمكن تحديد أماكن إضافة مكونات النافذة باستخدام الدالة setBounds لكل العناصر، حيث تستقبل الإحداثي السيني والصادي لنقطة بداية رسم الركن الشمالي الأعلى من العنصر، وطول وعرض العنصر. ويعتبر استخدام هذه الطريقة صعباً ومعقداً ولا يتم اللجوء إليه إلا للضرورة، لأن تصميم نافذة أي برنامج لا تخرج عادة عن الطرق التي أعدتها وهيأتها لغة جافا مسبقاً للتسهيل على المبرمج، فيتم التعامل مع الدوال والثوابت بأقل وقت وأقل جهد.

الفصل الثاني: تطبيقات Applets

تزامن تصميم لغة الجافا في بداية التسعينات مع ظهور وانتشار الإنترنت وتطبيقاته، لذلك كانت من اولويات مصممو لغة الجافا برمجة صفحات الانترنت، ومحاولة الاستفادة بأكبر قدر ممكن من خصائص هذه اللغة الجديدة في تطبيقات الإنترنت. مما ميزها عن باقي اللغات عالية المستوى.

تسمى جميع البرامج التي تمت كتابتها خلال الفصول الماضية بتطبيقات جافا java applications، واهم ما يميزها هو بداية تنفيذها عند الدالة الرئيسية main. توجد طريقة أخرى لكتابة برامج جافا لكي تكون ملائمة للتحميل والتشغيل على صفحات الإنترنت، وتسمى البرمجيات applets. الـ applet هو برنامج جافا مكتوب بصورة معينة تجعله قابلاً للتضمين داخل صفحات HTML (Hyper Text Markup Language) وهي إحدى أشهر وأسهل اللغات المستخدمة لبرمجة صفحات الإنترنت. عند فتح صفحة HTML يتم تحميل الـ applet على الصفحة وتشغيلها، مما يمكن الإنترنت من الاستفادة من جميع مزايا لغة جافا ويسهل تداول وانتشار برامج جافا بين عدد كبير من المستخدمين.

لا تختلف البرمجة بواسطة البرمجيات applets عما تحدثنا عنه حتى الآن، فهي تتمتع بنفس خواص لغة جافا ويمكن أن تحتوي على معظم ما توفره اللغة من تقنيات. يأتي الفرق بين البرمجيات والبرامج التقليدية في طريقة التنفيذ والصفات الإضافية التي تتمتع بها لكونها معدة للتحميل والتشغيل على صفحات الإنترنت. لكي يطلق على برنامج معين applet، يجب أن ترث الفئة التي يتم تعريفها خصائص الفئة JApplet، وهذه الفئة موجودة في الحزمة javax.swing. هذه الفئة تحتوي على أربع دوال تؤدي وظائف معينة ويمكن إعادة تعريفها بداخل البرنامج. فيما يلي أسماء هذه الدوال واستخداماتها.

الدالة ()init:

هي أول دالة يتم تنفيذها تلقائياً في البرنامج، حيث لا يحتوي الـ applet على مشيد أو دالة رئيسية main. تستخدم لتعريف الكائنات التي يحتاجها البرنامج، ووضع القيم الابتدائية وإعداد الشاشة وغيرها.

الدالة ()start:

يبدأ تنفيذها تلقائياً بعد الدالة ()init، وتحتوي على التعليمات اللازمة لتشغيل البرنامج الذي تم إعداده وتجهيزه في الدالة ()init، كما يتكرر نداؤها أيضاً إذا انتقل المستخدم من صفحة الإنترنت التي تحتوي على الـ applet إلى صفحة أخرى عن طريق رابط link لفترة ثم عاد إليها مرة أخرى.

الدالة () :paint

ولها نفس الاستخدام الذي ذكرناه في الفصل السابق، حيث تستخدم للوصول إلى جميع دول الفئة Graphics الموجودة في الحزمة java.awt لإعداد الرسوم والألوان على الشاشة. يتم نداء هذه الدالة تلقائياً بعد الدالة .start ()

الدالة () :stop

عندما يضغط المستخدم على رابط link في الصفحة، يتم تنفيذ ما بداخل هذه الدالة إذا كان هناك ما نريد إيقاف تشغيله لحين عودة المستخدم إلى صفحة ال-applet مرة أخرى. مثلاً أن نوقف تشغيل ملف صوت عند الانتقال إلى صفحة أخرى ونواصل تشغيله عند عودة المستخدم إلى الصفحة عن طريق الدالة () .start

الدالة () :destroy

ونكتب بداخلها ما نريد تنفيذه عند إغلاق الصفحة، أي عند توقف عمل ال-applet نهائياً، مثل إغلاق الملفات التي قام بفتحها البرنامج وغيره.

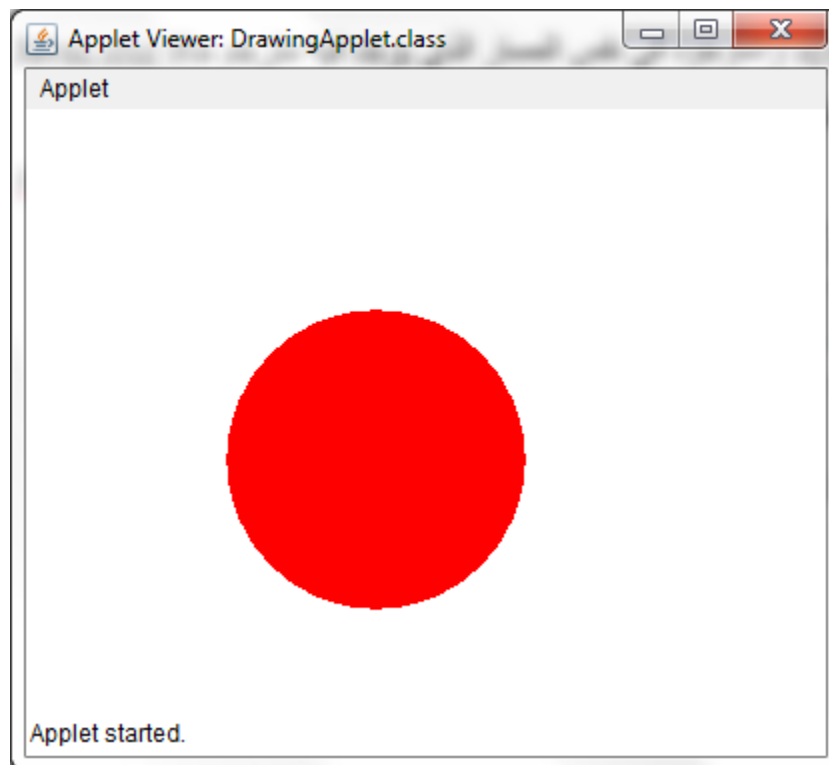
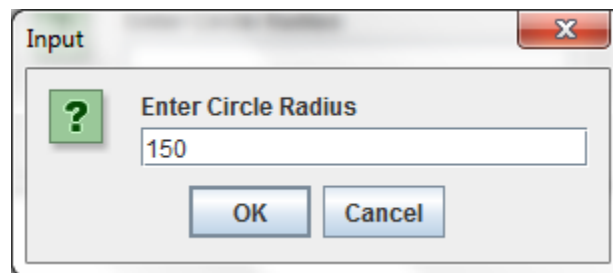
تحتوي الفئة applet على جميع خصائص ودوال الفئة JFrame، وذلك لأن الغرض من ال-applet هو ظهور التطبيقات على صفحة الإنترنت، لذلك فهي مصممة أن تكون JFrame، وهذا أكسبها الكثير من الخصائص والدوال، فيما يلي مثال لـ applet وشرح لكيفية ترجمتها وتضمينها في صفحة انترنت بسيطة:

```
import javax.swing.*;
import java.awt.*;
public class DrawingApplet extends JApplet
{
    int radius = 100;
    public void start ()
    {
        String s = JOptionPane.showInputDialog(this, "Enter
Circle Radius");
        radius = Integer.parseInt(s);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.RED);
        g.fillArc(100, 100, radius, radius, 0, 360);
    }
}
```

يبدأ تنفيذ البرنامج ببناء الدالة start تلقائياً ليقوم المستخدم بإدخال نصف قطر الدائرة، ومن ثم تنفيذ الدالة paint لرسم هذه الدائرة على نافذة البرنامج. بعد ترجمة البرنامج للحصول على الملف DrawingApplet.class يمكن تنفيذ البرنامج لرؤية المخرجات بإحدى الطريقتين التاليتين:

1. باستخدام البرنامج applet viewer والموجود في نفس المسار الذي يوجد فيه مترجم جافا Javac.exe ومفسر جافا Java.exe ، وذلك بكتابة العبارة:

```
appletviewer DrawingApplet
```



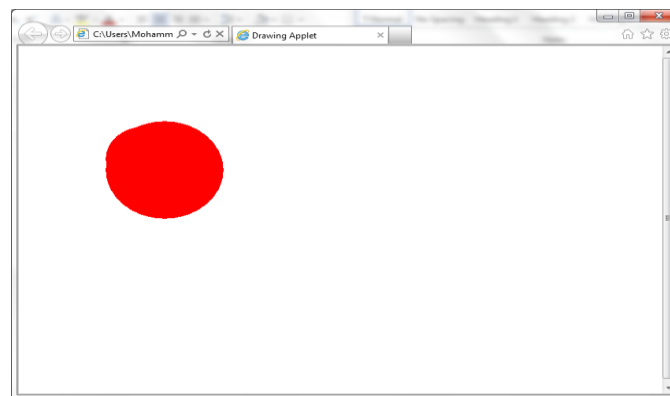
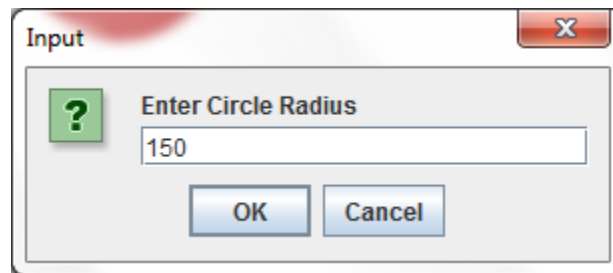
شكل 1-2 مخرجات البرنامج

ويستخدم البرنامج applet viewer لتجريب واختبار الـ applet قبل تضمينها على صفحة الإنترنت.

2. عن طريق تضمين الـ applet داخل صفحة إنترنت، وفيما يلي نوضح أبسط برنامج مكتوب بلغة HTML لإظهار applet على صفحة انترنت:

```
<html>
  <head>
    <title> Drawing Applet </title>
  </head>
  <body>
    <applet code="DrawingApplet.class" width="500"
height="500">
    </applet>
  </body>
</html>
```

المعلومات التي نحتاجها عن الـ applet ليتم كتابتها في برنامج HTML هو اسم الملف بالامتداد class، والذي يحتوي على الـ applet بعد الترجمة. كذلك نحتاج إلى طول وعرض النافذة التي نرغب في تخصيصها للـ applet على الصفحة. تحفظ العبارات الموضحة اعلاه في ملف بالامتداد html أو htm، ويتم فتحه بواسطة متصفح انترنت.



شكل 2-2 مخرجات البرنامج على متصفح الانترنت

أحياناً قد يحتاج الـ applet المضمن داخل صفحة انترنت إلى بعض المعلومات من تلك الصفحة، مثلاً قد تكون الـ applet مستخدمة في صفحة تغير من مظهرها باستمرار، فترغب الـ applet في معرفة الحجم الحالي للصفحة ونوع خط الكتابة المستخدم أو أي معلومة أخرى. توفر الـ applet طريقة لتبادل الوسائط parameters بينها وبين برنامج HTML حيث تقوم عبارات HTML معينة بتحديد اسم وقيمة كل وسيط، وتحصل الـ applet على قيم هذه الوسائط باستخدام الدالة `getParameter ()`، كما هو موضح في المثال التالي:

```
import java.awt.*;
import javax.swing.*;
public class ParameterTest extends JApplet
{
    int edge = 50;
    int max = 1;
    public void init ()
    {
        edge = Integer.parseInt (getParameter ("edge"));
        max = Integer.parseInt (getParameter ("maximum"));
    }
    public void paint (Graphics g)
    {
        for (int i = 1; i <= max; i++)
            g.drawRect (i * 50 , i * 50 , edge, edge);
    }
}
```

هذا البرنامج هو مثال لـ applet تستقبل قيمةً لوسائط معينة من بصفحة HTML، ثم تستخدمها الـ applet في تنفيذ عباراتها. الدالة `getParameter ()` تسترجع قيمة الوسيط المحدد بين قوسيه حسب ما هو موجود في صفحة HTML. لنفرض أن جزء برنامج HTML المتضمن للـ applet مكتوب كما يلي:

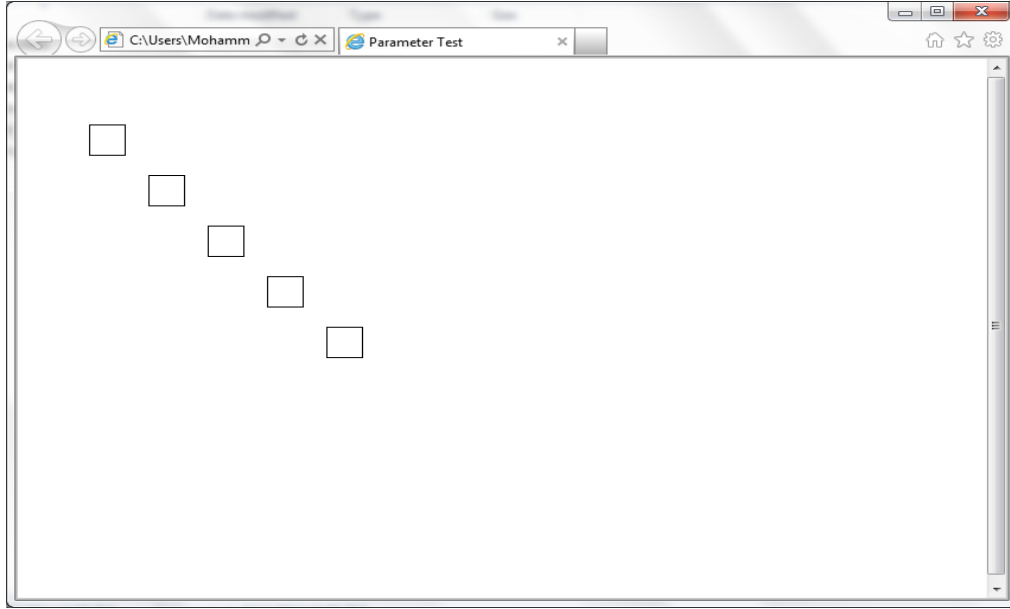
```
<html>
  <head>
    <title> Parameter Test </title>
  </head>
  <body>
    <applet code="ParameterTest.class" width="500"
height="500">
```

```

        <param name="edge" value="30"/>
        <param name="maximum" value="5"/>
    </applet>
</body>
</html>

```

عند تنفيذ هذا البرنامج يتم تمرير قيم الوسائط edge و maximum للـ applet ، وتستخدم القيم في تحديد عدد وأبعاد المربعات الناتجة، كما موضح في مخرجات البرنامج أدناه.



شكل 2-3 مخرجات البرنامج

تستخدم الـ java applets كثيراً في تطبيقات الإنترنت خصوصاً في مجال الترفيه، مثل الألعاب وتطبيقات الصور والأصوات والرسوم المتحركة. فالـ java applets تتميز بجميع مميزات لغة جافا، مما يجعلها غنية بالعبارات الملائمة لكثير من التطبيقات، كما توفر درجة عالية من الأمن لضمان سرية وسلامة جهاز المستخدم. والسبب في زيادة درجة الأمن التي توفرها الـ applet أكثر من غيرها هو أنه يتم تحميلها من جهاز مقدم خدمة الإنترنت إلى أجهزة المستخدمين بخلاف غيرها من التطبيقات التي عادة ما تعمل على جهاز معين أو عدد محدود من الأجهزة. يمكن تلخيص مميزات الأمن والسرية التي توفرها الـ applets في النقاط التالية:

1. لا يسمح للـ applet بالدخول إلى نظام ملفات الجهاز التي تم تحميلها عليه لا قراءة ولا كتابة.
2. لا يسمح للـ applet بتشغيل برنامج آخر على جهاز المستخدم.
3. لا يسمح للـ applet بتوصيل جهاز المستخدم مع أي جهاز آخر.

تمثل النقاط السابقة الفرق بين برامج الـ applet وبرامج الـ application. يمكن أن نحول أي application إلى applet مادام أنه يخضع لقوانين أمن الـ applets السابقة، أما الـ applets فيمكن أن تتحول إلى application بدون شروط.

الفصل الثالث: برمجة قواعد البيانات باستخدام جافا

توفر لغة البرمجة جافا عدة طرق للتخزين المعلومات على الملفات الا ان تخزين المعلومات على الملفات لا يوفر إمكانية الاستعلام عن البيانات واسترجاعها بشكل فعال، ان أنظمة قواعد البيانات توفر لنا إمكانية تخزين وتنظيم البيانات بطريقة تسمح لنا بالاستعلام عن البيانات واسترجاعها بفعالية، ومن اشهر أنواع قواعد البيانات قواعد البيانات العلائقية Relational Database System وفي هذه النوع من قواعد البيانات تستخدم لغة الاستعلامات الهيكلية Structure Query Language (SQL) للاستعلام عن البيانات واسترجاعها، وللإستفادة من قواعد البيانات بصورة مثالية لا بد من ربطها بتطبيقات سطح المكتب او تطبيقات الانترنت، توفر معظم لغات البرمجة إمكانية الاتصال بقواعد البيانات واجراء عمليات الاستعلام عليها باستخدام عبارات SQL، من اشهر نظم قواعد البيانات العلائقية ما يلي: Microsoft Access، Microsoft SQL Server، Oracle، Informix وغيرها. وسنتعرف الان على كيفية كتابة برنامج بلغة الجافا يتصل بقواعد البيانات من النوع Microsoft Access، هذا باعتبار ان لديك معرفة بلغة الاستعلام SQL ومفاهيم نظم قواعد البيانات.

قاعدة البيانات:

سنقوم الان بإنشاء قاعدة بيانات من النوع Microsoft Access واسمها University، تتكون قاعدة البيانات University من اثنين جدول هما جدول الكليات College وجدول الطلاب Students والشكل التالي يوضح تفاصيل الجداول:

	Field Name	Data Type
🔑	Coll_ID	Number
	Coll Name	Short Text

شكل 1-3 هيكل الجدول College

والشكل التالي يوضح بيانات الجدول College:

	Coll_ID	Coll_Name	Click to Add
+	1	Computer Scie	
+	2	Medicine	
+	3	Engineering	
+	4	Economy	
*	0		

شكل 2-3 بيانات الجدول College

	Field Name	Data Type
🔑	St_ID	Number
	St_Name	Short Text
	St_Address	Short Text
	Coll_ID	Number

شكل 3-3 هيكل الجدول Students

St_ID	St_Name	St_Address	Coll_ID	Click to Add
1	Mohammed Mahmoud	Gadarif	1	
2	Hani Abbass	Kadogli	2	
3	Maaz Mustafa	Umdorman	3	
4	Omer Osman	Obaid	4	
5	Hisham Zain	Khartoum	2	
6	Alsadig Khalifa	Obaid	1	
7	Musab Mukhtar	Khartoum Bahri	3	
*	0		0	

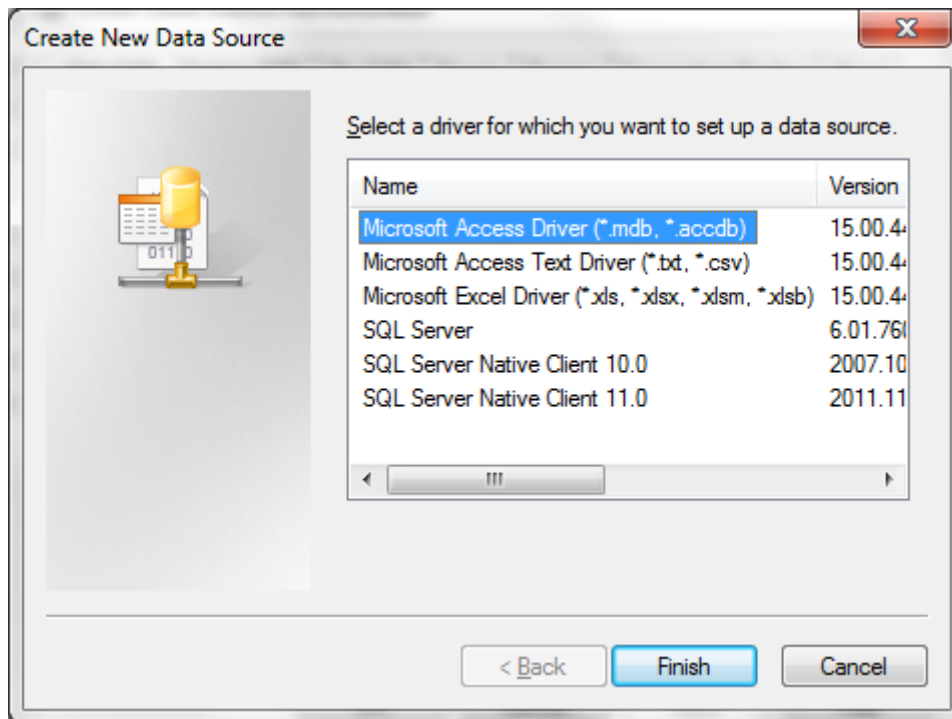
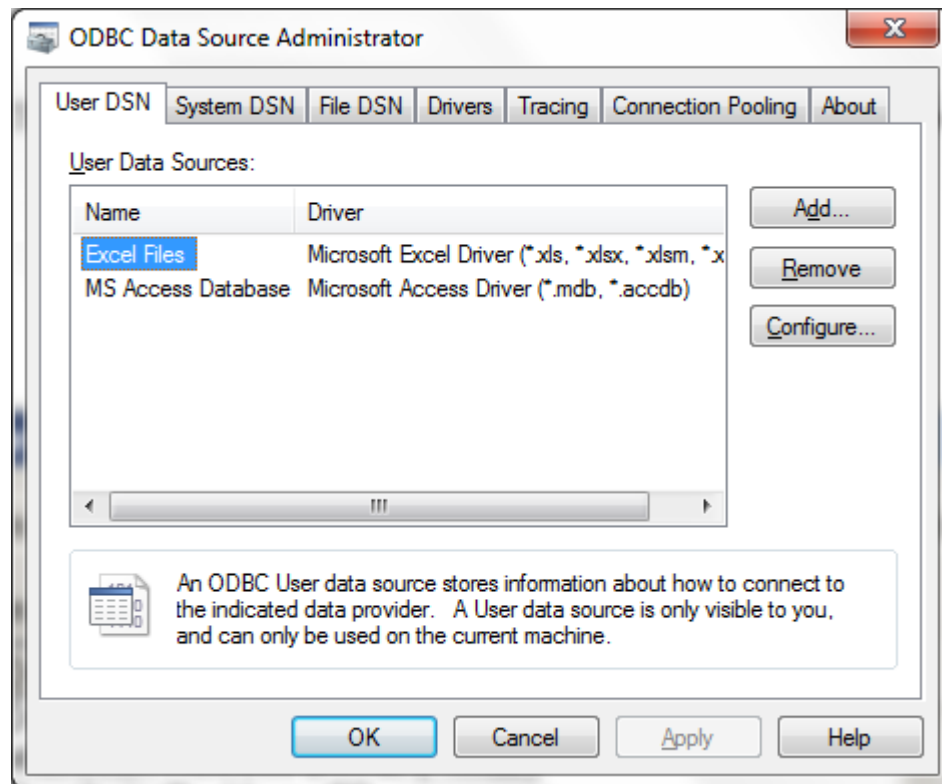
شكل 4-3 بيانات الجدول Students

تسجيل قاعدة البيانات في مصدر قواعد البيانات المفتوحة ODBC:

قبل ربط قاعدة البيانات ببرنامج جافا يجب ان يتم تسجيلها كمصدر بيانات في مصدر قواعد البيانات المفتوحة Open Database Connection، الان سنعرف كيف يتم تعريف قاعدة البيانات كمصدر بيانات ODBC وذلك باتباع الخطوات التالية:

إذا كنت تستخدم نظام التشغيل Windows اذهب الي لوحة التحكم Control Panel، ثم الي الخيار Administrative Tools انقر نقر مزدوج على الخيار (ODBC) Data Source سيظهر لك صندوق الحوار الموضح في الشكل 3-5، من قائمة User DSN انقر على الزر Add ليظهر لك صندوق انشاء مصدر بيانات جديد كما موضح في الثاني.

وبما انا نتعامل مع قاعدة بيانات من النوع Microsoft Access سوف نستخدم Microsoft Access Driver، ثم اضغط على الزر Finish.



شكل 3-5 صندوق حوار ODBC

سيظهر لنا الان صندوق الحوار الخاص بمصدر قواعد البيانات المفتوحة Microsoft Access، حيث نقوم بتحديد التالي:

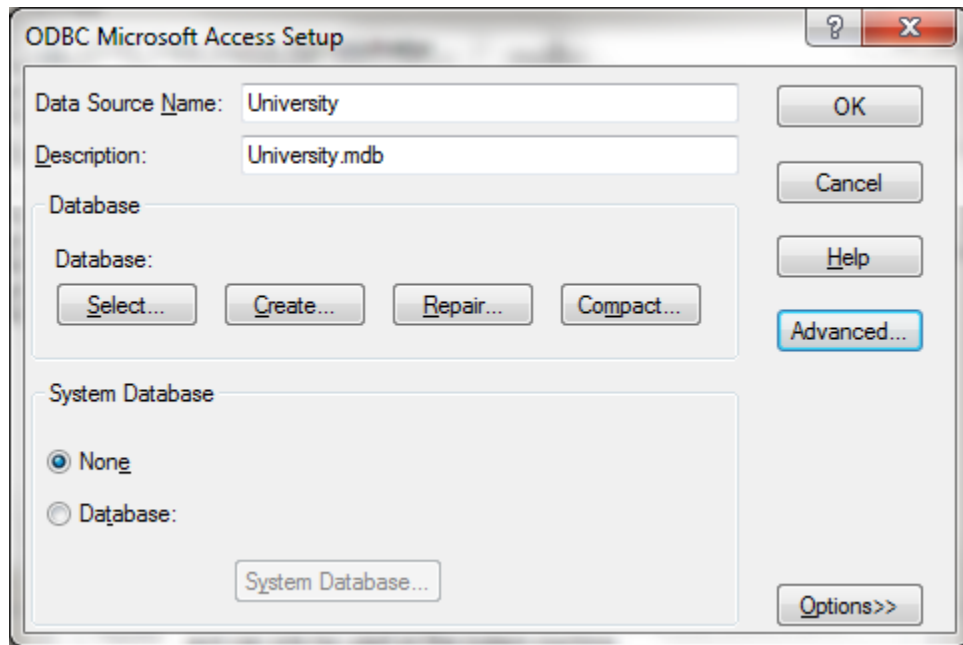
ادخل اسم قاعدة البيانات الذي سيستخدم من قبل كائن جافا الخاص بالاتصال بقواعد البيانات JDBC في الحقل Data Source Name، كما يمكنك إضافة وصف لقاعدة البيانات في الحقل Description.

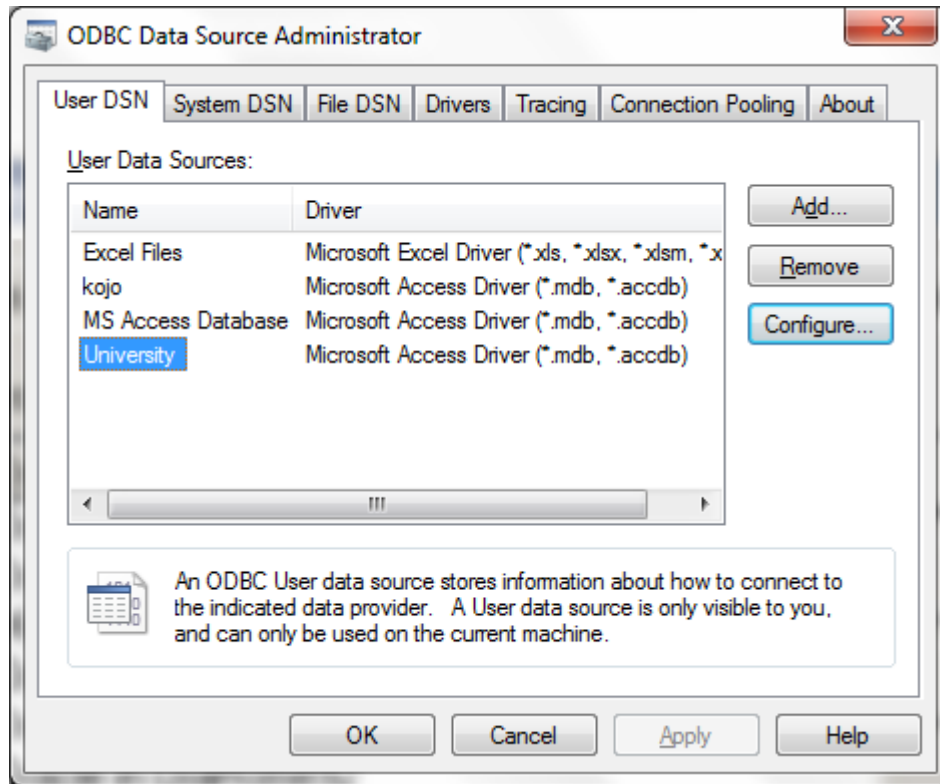
1. انقر على الزر Select، ثم قم بالبحث عن قاعدة البيانات (في هذا المثال University.mdb)، واضغط على الزر OK.

2. اضغط على الزر Advance لإظهار قائمة الخيارات المتقدمة، ادخل في حقل اسم الدخول aau، وفي حقل كلمة المرور aau، واضغط على الزر OK.

3. انقر على الزر OK للخروج من صندوق الحوار ODBC Microsoft Access Setup، ثم انقر على الزر OK مرة أخرى للخروج من صندوق الحوار ODBC Data Source Administrator.

يمكنك الآن كتابة برنامج بلغة الجافا للاتصال بقاعدة البيانات University.





شكل 3-6 انشاء مصدر قاعدة البيانات University

في المثال التالي سنقوم بكتابة برنامج بسيط لعملية الاستعلام من قاعدة البيانات University.mdb، حيث يوضح البرنامج كيفية الاتصال بقاعدة البيانات والاستعلام عن البيانات.

```
import java.sql.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class TableDisplay extends JFrame
{
    private Connection connection;
    private JTable table;
    public TableDisplay ()
    {
        String url = "jdbc:odbc:Driver={Microsoft Access
Driver (*.mdb)};DBQ=University.mdb;";
        String username = "anonymous";
        String password = "guest";
```

```

    try
    {
        Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
        connection = DriverManager.getConnection( url,
username, password );
    }
    catch(ClassNotFoundException cnfex )
    {
        System.err.println("Failed to load JDBC/ODBC
driver.");
        cnfex.printStackTrace();
        System.exit( 1 ); // terminate program
    }
    catch ( SQLException sqlex )
    {
        System.err.println( "Unable to connect" );
        sqlex.printStackTrace();
    }
    getTable();
    setSize( 450, 150 );
    show();
}
private void getTable()
{
    Statement statement;
    ResultSet resultSet;
    try
    {
        String query = "SELECT * FROM Students";
        statement = connection.createStatement();
        resultSet = statement.executeQuery( query );
        displayResultSet( resultSet );
        statement.close();
    }
    catch ( SQLException sqlex )
    {
        sqlex.printStackTrace();
    }
}
}

```

```

    private void displayResultSet(ResultSet rs) throws
SQLException
    {
        boolean moreRecords = rs.next();
        if ( ! moreRecords )
        {
            JOptionPane.showMessageDialog(
this, "ResultSet contained no records" );
            setTitle( "No records to display" );
            return;
        }
        setTitle( "Authors table from Books" );
        Vector columnHeads = new Vector();
        Vector rows = new Vector();
        try
        {
            ResultSetMetaData rsmd = rs.getMetaData();
            for ( int i = 1; i <= rsmd.getColumnCount(); ++i
)
                columnHeads.addElement( rsmd.getColumnName(
i ) );
            do
            {
                rows.addElement( getNextRow( rs, rsmd ) );
            } while ( rs.next() );
            table = new JTable( rows, columnHeads );
            JScrollPane scroller = new JScrollPane( table );
            getContentPane().add(
                scroller, BorderLayout.CENTER );
            validate();
        }
        catch ( SQLException sqlex )
        {
            sqlex.printStackTrace();
        }
    }

    private Vector getNextRow( ResultSet
rs,ResultSetMetaData rsmd ) throws SQLException
    {
        Vector currentRow = new Vector();

```

```

        for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
            switch( rsmd.getColumnType( i ) )
            {
                case Types.VARCHAR:
                    currentRow.addElement( rs.getString( i )
);
                    break;
                case Types.INTEGER:
                    currentRow.addElement( new Long(
rs.getLong( i ) ) );
                    break;
                default:
                    System.out.println( "Type was: " +
rsmd.getColumnTypeName( i ) );
            }
        return currentRow;
    }
    public void shutdown()
    {
        try
        {
            connection.close();
        }
        catch ( SQLException sqlex )
        {
            System.err.println( "Unable to disconnect" );
            sqlex.printStackTrace();
        }
    }
    public static void main( String args[] )
    {
        final TableDisplay app = new TableDisplay();
        app.addWindowListener( new WindowAdapter()
        {
            public void windowClosing( WindowEvent e )
            {
                app.shutdown();
                System.exit( 0 );
            }
        }
    );
}

```

```

}
}

```

قمنا أولاً باستدعاء الحزمة `java.sql` والتي تحتوي على كل الفئات المتعلقة بإدارة قواعد البيانات العلائقية في لغة الجافا، تستخدم الفئة `Connection` بين برنامج الجافا وقاعدة البيانات كما انها توفر إمكانية تنفيذ جمل SQL لمعالجة قاعدة البيانات والعمليات التي يتم تنفيذها عليها، في المشيد نقوم بإنشاء الاتصال بقاعدة البيانات وتنفيذ الاستعلام ويتم عرض النتائج باستدعاء الدالة `getTable()`.

```

String url = "jdbc:odbc:University";
String username = "anonymous";
String password = "guest";

```

ليتمكن المشيد من الاتصال بقاعدة البيانات، موضحة في التعليمات أعلاه، وهي: موقع قاعدة البيانات المراد الاتصال بها من خلال تحديد العنوان URL والذي يحدد البروتوكولات المستخدمة في عملية الاتصال بقاعدة البيانات وهي البروتوكول الرئيسي `jdbc` والبروتوكول الفرعي `odbc` بالإضافة الي اسم مصدر قاعدة البيانات الذي قمنا بإنشائه سابقاً.

توفر لغة البرمجة جافا مشغل للاتصال باي نوع من أنواع قواعد بيانات باستخدام تقنية `odbc` اسمه `jdbc:odbc:jdbcodbcDriver` يجب تحميل هذا المشغل قبل الاتصال بقاعدة البيانات باستخدام الدالة `forName()`.

تستخدم الدالة `getConnection()` الموجودة في الفئة `DriverManager` للاتصال بقاعدة البيانات وتأخذ ثلاثة وسائط عنوان قاعدة البيانات URL واسم المستخدم وكلمة المرور.

الدالة `getTable()` تقوم بالاستعلام عن البيانات ومن ثم استدعاء الدالة `displayResultSet()` لإنشاء جدول وعرض نتائج الاستعلام من خلاله.

```
Statement statement;
```

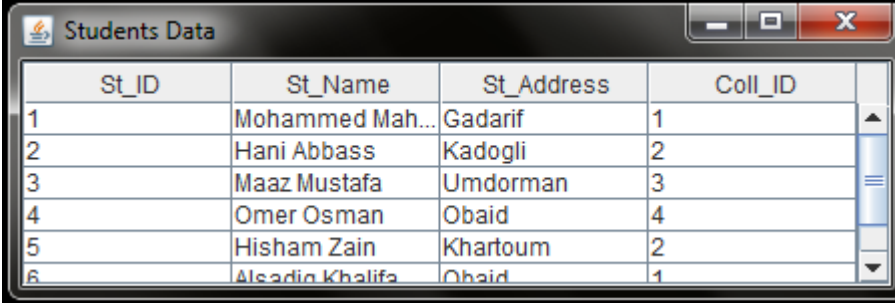
في هذا السطر قمنا بتعريف مرجع لعبارات SQL من النوع `Statement` وهو موجود في الحزمة `java.sql` بواسطة هذا الكائن يتم نقل عبارة SQL الي قاعدة البيانات لتنفيذها.

```
ResultSet resultSet;
```

في هذا الامر قمنا بتعريف الكائن `resultSet` والذي يقوم بإرجاع ناتج تنفيذ عبارات SQL.

ResultSetMetaData rsMetaData

يستخدم هذا الكائن للحصول على المعلومات التفصيلية عن الجدول الموجود في الكائن resultSet مثل أسماء حقول الجدول ونوع بياناتها تسمى هذه المعلومات بالبيانات التفصيلية Meta Data.



St_ID	St_Name	St_Address	Coll_ID
1	Mohammed Mah...	Gadarif	1
2	Hani Abbass	Kadogli	2
3	Maaz Mustafa	Umdorman	3
4	Omer Osman	Obaid	4
5	Hisham Zain	Khartoum	2
6	Alsadig Khalifa	Obaid	1

شكل 3-7 مخرجات البرنامج

الشبكات:

تعتبر شبكات الحاسوب ذات أهمية بالغة في تطبيقات الحاسوب، فهي الطريقة الأساسية لتشارك البيانات والمصادر والأجهزة المختلفة، وهي وسيلة الاتصال الرئيسية بين أجهزة النظام المختلفة. ويمكن أن نرى أثر الشبكات واضحاً في مختلف أنواع الأنظمة. فالإنترنت هي مثال لأكبر وأشهر شبكة تتعامل معها من أي مكان. وبالشبكات أصبح من الممكن وجود آلة واحدة للطباعة يتشارك فيها عدد كبير من الأجهزة. وتطبيقات الشبكات لا حصر له، ففي مستشفى مثلاً، يمكن للطبيب المتخصص أن يطالع تشخيص الطبيب العمومي لمتابعة حالة المريض، ويستطيع موظف الاستقبال أن يحدد مواعيد المقابلات حسب رغبة الطبيب، وأن يحدد ما إذا كان الدواء الذي يطلبه المريض متوفراً بالصيدلية أو لا، ويمكن للموردين الاطلاع على بيانات المخازن والمعامل والصيدلية لطلب المواد الضرورية في الوقت المناسب. تقوم أجهزة الحاسوب بتسهيل وتبسيط جميع هذه العمليات عن طريق الشبكات، ويمكن أن نتصور مقدار الجهد والوقت والمال الذي سيتم إنفاقه لإجراء هذه العمليات بدون وجود الشبكات. لكل هذه الأسباب، أولت أنظمة التشغيل ولغات البرمجة اهتماماً كبيراً بالشبكات وتطبيقاتها.

لكي يتسنى لنا تنفيذ برنامج عبر شبكة، يجب أن يتكون من برنامجين على الأقل: برنامج الخادم Server، وهو الذي يقدم الخدمة أو يمتلك المعلومة التي يطلبها العميل، وبرنامج العميل Client، وهو الذي يطلب الخدمة التي يوفرها الخادم. توفر لغة جافا حزمة برمجية لبرمجة تطبيقات الشبكات وهي java.net. وفي هذا القسم نقدم شرحاً وافياً لبرمجة كل من تطبيقات الخادم والعميل باستخدام بروتوكولي TCP و UDP. ولكن قبل ذلك لا بد لنا من التعرف على معاني بعض المصطلحات المستخدمة في برمجة تطبيقات الشبكات.

:Socket

تسمح بتخاطب العميل والخادم، وتبسط للمبرمج تفاصيل البرمجة الخاصة بالشبكة، فتجعل عملية التخاطب مع الشبكة أشبه بالتخاطب مع الملفات.

:Port

هو رقم يحدده الخادم كي يستطيع العميل معرفة مكان موقع برنامج الخادم بجهاز الحاسوب.

:Datagram

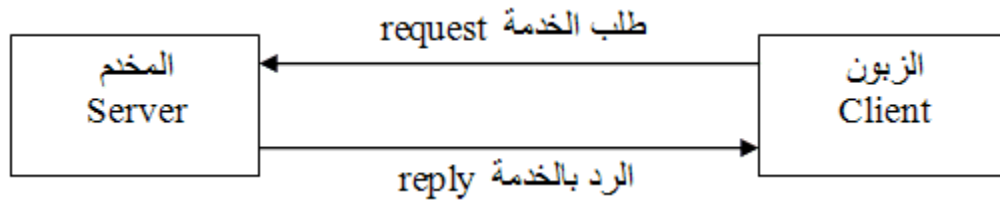
وحدات ذات أحجام معينة تحمل البيانات التي يرغب المرسل بإرسالها إضافة إلى عنوان المستقبل. يعتمد عدد الوحدات على حجم البيانات المرسلة.

:(Transmission Control Protocol) TCP

من أشهر بروتوكولات تخاطب الشبكات، وفيها يتم فتح قناة اتصال بين الجهازين ثم إرسال جميع البيانات المراد إرسالها على شكل stream، ثم إغلاق قناة الاتصال. يتميز هذا البروتوكول بالاعتمادية reliability.

:(Datagram Protocol User) UDP

هو بروتوكول آخر للشبكات. وفيه يتم إرسال البيانات عن طريق datagrams مستقلة عن بعضها دون فتح قناة اتصال ثابتة، يتميز هذا البروتوكول بالسرعة.



شكل 1-4 عملية الاتصال بين الخادم والعميل

:Socket Connection Stream باستخدام العميل والخادم والعميل

يستخدم هذا النوع من الاتصال بروتوكول TCP، حيث يتم فتح قناة اتصال بين برنامج الخادم وبرنامج العميل، وقد يكون البرنامجان على نفس جهاز الحاسوب أو على جهازين مختلفين.

برنامج الخادم:

```
import java.io.*;
import java.net.*;
public class Server
{
    public static void main(String args[])
    {
        try
        {
            ServerSocket serversocket = new
ServerSocket(7000);
            Socket socket = ServerSocket.accept();
            DataInputStream fromClient = new
DataInputStream(socket.getInputStream());
            DataOutputStream toClient = new DataOutputStream
(socket.getOutputStream());
            while(true)
            {
```

```

        double number = fromClient.readDouble();
        System.out.println("Client sent : " +
number);
        double squareRoot = Math.sqrt(number);
        toClient.writeDouble(squareRoot);
        toClient.flush();
        System.out.println("The square root = " +
squareRoot);
    }
}
catch(IOException e)
{
    e.printStackTrace();
}
}
}

```

هذا برنامج بسيط للخادم، يتم أولاً إنشاء كائن من الفئة `ServerSocket` للإعلان عن وجود خادم وتحديد رقم البوابة (port) الذي سيتخاطب عبره البرنامج مع العملاء. بعدها ينتظر الخادم قدوم طلب الخدمة من العميل، وعند حدوث ذلك يتم إنشاء كائن من الفئة `Socket`، وهو الذي يحمل معلومات قناة الاتصال مع العميل. ومن خلاله يمكن إنشاء `DataInputStream` لاستلام البيانات من العميل باستخدام الدالة `getInputStream()`. يختلف باقي البرنامج باختلاف الخدمة التي يقدمها الخادم للعميل. وفي هذا البرنامج يستقبل خادم من العميل عدد حقيقي ويحسب الجزر التربيعي للعدد، ويعيد الجزر التربيعي إلى العميل، في مثال لخدمة بسيطة يقدمها المخدم للزبائن. تستمر هذه العملية بواسطة الحلقة `while (true)` إلى أن ينهي برنامج الخادم أو العميل باستخدام `Ctrl+C`. لاحظ أن هذا الخادم يستطيع خدمة زبون واحد فقط في المرة الواحدة، وسنعرف لاحقاً كيف يمكن للخادم التخاطب مع أكثر من العميل.

برنامج العميل:

البرنامج التالي هو برنامج العميل، وفيه يقوم العميل بإجراء اتصال ببرنامج الخادم الذي من المفروض أن يتم تشغيله أولاً لكي يستجيب عند محاولة العميل الاتصال به.

```

import java.io.*;
import java.net.*;
public class Client
{
    public static void main(String args[])
    {

```

```

try
{
    Socket socket = new Socket("localhost", 7000);
    DataInputStream fromServer = new
DataInputStream(socket.getInputStream());
    DataOutputStream toServer = new
DataOutputStream(socket.getOutputStream());
    while(true)
    {
        System.out.print("Enter a number: ");
        double number = readDouble();
        toServer.flush();
        double squareRoot = fromServer.readDouble();
        System.out.println("Square Root = " +
squareRoot);
    }
}
catch(IOException e)
{
    e.printStackTrace();
}
}
public static double readDouble()
{
    DataInputStream in = new
DataInputStream(System.in);
    try
    {
        double d = Double.parseDouble(in.readLine());
        return d;
    }
    catch(Exception e)
    {
        System.out.println(e);
        return 0;
    }
}
}
}

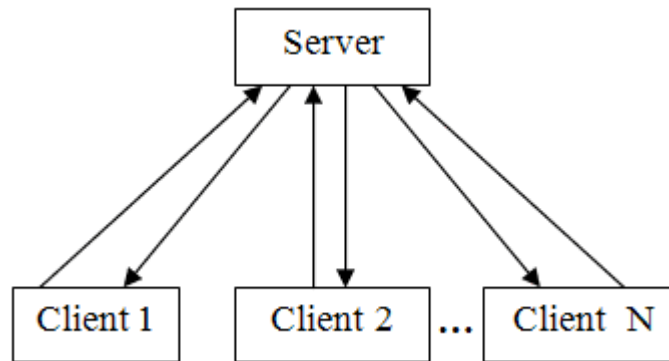
```

يبدأ برنامج العميل بإنشاء الكائن socket للاتصال ببرنامج الخادم الموجود على الجهاز localhost، والتي تشير إلى نفس البرنامج الذي يحتوي على برنامج العميل وفي حال لم يكن في نفس الجهاز، يتم استبدال العبارة

localhost برقم الجهاز الذي يحتوي على برنامج الخادم، مثلاً " 160.251.44.2 ". يتم كذلك توضيح رقم البوابة (port) المخصص للتخاطب بين البرنامجين، وهو في هذا المثال 7000. يتم إعداد DataInputStream و DataOutputStream لاستلام وإرسال البيانات من وإلى الخادم بنفس الطريقة التي تم إنشاؤها في برنامج الخادم. يطلب برنامج العميل من المستخدم إدخال عدد يتم قراءته من الشاشة سطر الاوامر باستخدام الدالة `readDouble()` المعرفة داخل الفئة `System.in`، والتي تستخدم لقراءة سطر من شاشة الإدخال ثم تحوله إلى النوع `double`. يتم إرسال العدد إلى برنامج الخادم والذي كما نعلم يقوم بحساب الجذر التربيعي للعدد، ومن ثم استقبال الجذر التربيعي وطباعته على شاشة برنامج العميل. يشرح هذا المثال المكونات الرئيسية لبرنامج الخادم والعميل اللازمة لعملية الاتصال بين البرنامجين، وبعد معرفة الفئات والدوال الأساسية، يمكن تطبيق أي نظام جافا على شبكة وإضافة مكونات GUI لتحسين مظهر البرنامج.

التخاطب مع أكثر من عميل:

سنقوم الآن بتعديل على برنامج الخادم ليكون قادراً على التخاطب مع عدد من العملاء في نفس الوقت، كما نتوقع من أغلب الخوادم، حيث يكون هناك عدد من العملاء الذين يحاولون الاتصال بخادم واحد لطلب الخدمة. يتم تعديل الخادم بحيث يقوم باستقبال طلب الخدمة من عميل محدد، ويقوم بإنشاء `thread` تتولى عملية التخاطب مع هذا العميل حتى تكتمل عملية الاتصال، وبالتالي يستطيع الخادم التخاطب مع عدد كبير من الزبائن.



شكل 2-4 التخاطب بين الخادم وعدد من العملاء

```

import java.io.*;
import java.net.*;
public class Server
  
```

```

{
    public static void main(String args[])
    {
        try
        {
            int clientID = 1;
            ServerSocket serverSocket = new
ServerSocket(7000, 10);
            while(true)
            {
                Socket socket = serverSocket.accept();
                System.out.println("Connection established
For client: " + clientID);
                InetAddress clientAddress =
socket.getInetAddress();
                System.out.println("Client IP address: " +
clientAddress.getHostAddress());
                ClientThread clientThread = new
ClientThread(socket, clientID);
                clientThread.start();
                clientID++;
            }
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
    }
}
class ClientThread extends Thread
{
    Socket socket;
    int clientID;
    public ClientThread(Socket s, int id)
    {
        socket = s;
        clientID = id;
    }
    public void run()
    {

```

```

try
{
    DataInputStream fromClient = new
DataInputStream(socket.getInputStream ());
    DataOutputStream toClient = new
DataOutputStream(socket.getOutputStream ());
    while(true)
    {
        double number = fromClient.readDouble ();
        System.out.println("Client " + clientID + "
sent: " + number);
        double squareRoot = Math.sqrt(number);
        toClient.writeDouble(squareRoot);
        toClient.flush();
        System.out.println("The square root = " +
squareRoot);
    }
}
catch(IOException e)
{
    e.printStackTrace();
}
}
}

```

عند إنشاء كائن من `ServerSocket`، يمكن أن نحدد أقصى عدد من العملاء الذين يمكنهم التخاطب مع الخادم في اللحظة الواحدة، في هذا البرنامج 10. ونختار هذا العدد بناء على نوع التطبيق وحجم المصادر التي سيقوم الخادم بحجزها لكل طالب خدمة، فالهدف هو ألا يزيدحم الخادم فوق طاقته وألا تحدث مشاكل بسبب عدد العملاء الكبير. توفر الفئة `InetAddress` معلومات عن العملاء مثل عنوان الجهاز في الشبكة IP Address عن طريق الدالة `getHostAddress` أو اسم الجهاز باستخدام الدالة `getHostName`. يتم تخصيص وتشغيل `thread` لكل طالب خدمة بعد مده بواسطة `socket` التابع لهذا العميل ورقمه الذي يتابع به الخادم العملاء `clientID`. يتابع ال- `thread` عمل الخادم طوال فترة اتصال العميل. لا يكون هناك أي تعديل على برنامج العميل، ويتم التفاعل بين البرنامجين تماما كما في المثال السابق.

برمجة الخادم والعميل باستخدام **Connectionless Datagram Socket**

يستخدم هذا النوع بروتوكول UDP، وسمي بـ connectionless لعدم وجود قناة اتصال ثابتة بين الخادم والعميل كما في TCP، بل يتم إرسال البيانات على شكل وحدة datagram ذات حجم معين. ولذلك فإن كل وحدة يجب أن تحتوي على عنوان الجهاز الذي سيتم إرسالها إليه. فيما يلي برنامج ل خادم بسيط و عميل و شرح لعملية إرسال واستقبال الوحدات بينهما.

```
import java.io.*;
import java.net.*;
public class Server
{
    private DatagramSocket socket;
    public Server()
    {
        try
        {
            socket = new DatagramSocket( 5000 );
        }
        catch( SocketException socketException )
        {
            socketException.printStackTrace();
            System.exit( 1 );
        }
    }
    private void waitForPackets()
    {
        while ( true )
        {
            try
            {
                byte data[] = new byte[ 100 ];
                DatagramPacket receivePacket = new
                DatagramPacket( data, data.length );
                socket.receive( receivePacket );
                System.out.println( "\nPacket received:" +
                "\nFrom host: " +
                receivePacket.getAddress() +
                "\nHost port: " + receivePacket.getPort()
                +
```

```

        "\nLength: " + receivePacket.getLength() +
        "\nContaining:\n\t" + new String(
receivePacket.getData(),
        0, receivePacket.getLength()));
        sendPacketToClient( receivePacket );
    }
    catch( IOException ioException )
    {
        System.out.println( ioException.toString() +
"\n" );
        ioException.printStackTrace();
    }
}
}
private void sendPacketToClient( DatagramPacket
receivePacket ) throws IOException
{
    System.out.println( "\n\nEcho data to client..."
);
    DatagramPacket sendPacket = new DatagramPacket(
receivePacket.getData(),
receivePacket.getLength(),
receivePacket.getAddress(),
receivePacket.getPort() );
    socket.send( sendPacket ); // send packet
    System.out.println( "Packet sent\n" );
}
public static void main(String args[])
{
    Server application = new Server();
    application.waitForPackets();
}
}

```

تستخدم الفئة DatagramSocket لتحديد البوابة port للتخاطب بين الخادم والعميل. DatagramPacket هو فئة يكون الكائن منه عبارة عن الوحدة datagram المطلوب إرسالها عبر الشبكة. ولإرسال datagram، يجب أن تحتوي بداخلها على بعض المعلومات وهي:

1. عنوان الجهاز الذي نرغب في إرسال الوحدة إليه.
2. رقم البوابة port المخصص للتخاطب.

3. البيانات المطلوب إرسالها.

4. طول البيانات المرسلة.

عند إرسال وحدة معينة إلى الشبكة، يتم تضمين هذه المعلومات بداخل الوحدة عن طريق المشيد الخاص بـ DatagramPacket عند إنشاء الوحدة. وعند استقبال وحدة من الشبكة، يتم استخلاص المعلومات التالية منها:

1. عنوان الجهاز الذي تم إرسالها منه.

2. رقم البوابة port المخصص للتخاطب.

3. البيانات التي أرسلها المرسل.

4. طول البيانات الواردة.

فيما يلي برنامج الزبون client الذي يتخاطب مع البرنامج السابق باستخدام datagrams.

```
import java.io.*;
import java.net.*;
import javax.swing.*;
public class Client
{
    private DatagramSocket socket;
    private BufferedReader br;
    public Client ()
    {
        try
        {
            socket = new DatagramSocket ();
            br = new BufferedReader (new
InputStreamReader (System.in) );
        }
        catch ( SocketException socketException )
        {
            socketException.printStackTrace ();
            System.exit ( 1 );
        }
    }
    private void waitForPackets ()
    {
        while ( true )
        {
            try
            {
```

```

        String message = br.readLine();
        byte data[] = message.getBytes();
        DatagramPacket sendPacket = new
DatagramPacket( data,
                data.length,
InetAddress.getLocalHost(), 5000 );
        socket.send( sendPacket ); // send packet
        System.out.println( "Packet sent\n" );
    }
    catch ( IOException ioException )
    {
        System.out.println(
ioException.toString() + "\n" );
        ioException.printStackTrace();
    }
    try
    {
        byte data[] = new byte[ 100 ];
        DatagramPacket receivePacket = new
DatagramPacket( data, data.length );
        socket.receive( receivePacket ); // wait for
packet
        System.out.println( "\nPacket received:"
+
                "\nFrom host: " +
receivePacket.getAddress() +
                "\nHost port: " +
receivePacket.getPort() +
                "\nLength: " +
receivePacket.getLength() +
                "\nContaining:\n\t" + new String(
receivePacket.getData(),
                0, receivePacket.getLength() ) );
    }
    catch( IOException exception )
    {
        System.out.println( exception.toString()
+ "\n" );
        exception.printStackTrace();
    }
}

```

```

}
public static void main( String args[] )
{
    Client application = new Client ();
    application.waitForPackets ();
}
}

```

تستخدم `BufferedReader` لقراءة بيانات العميل من الشاشة. تقوم الدالة `getLocalHost` التابعة للفئة `InetAddress` بتوفير عنوان جهاز الخادم المطلوب إرسال البيانات إليه، وهو في هذا المثال نفس الجهاز الذي يوجد به برنامج العميل. `localhost` تعبر عن هذا الجهاز، وإذا كان برنامج الخادم موجوداً في جهاز آخر وهو الحال في أغلب البرامج - يجب تعديل العنوان إلى عنوان الجهاز الذي يوجد به برنامج الخادم.

- البرمجة بلغة الجافا – جامعة السودان المفتوحة
- برمجة الحاسب – الإدارة العامة لتصميم وتطوير المناهج المملكة العربية السعودية
- Java How to Program 9th Edition