بسم الله الرحمن الرحيم

رحلة يوم مع لغة البرمجة جافا

تأليف: معتز عبدالعظيم الطاهر code.sd

إصدار ذي القعدة 1433 هجرية الموافق أكتوبر 2012 ميلادية

مقدمة

بسم الله الرحمن الرحيم والصلاة والسلام على أشرف الأنبياء والمرسلين، نبينا محمد وعلى آله وصحبه أجمعين. أما بعد. الهدف من هذا الكُتيب تعريف المبرمج في فترة وجيزة وكمدخل سريع للغة البرمجة جافا بإستخدام أداة التطوير NetBeans. وبهذا يكون هذا الكُتيب موجه ف قط لمن لديه خبرة في لغة برمجة أخرى حتى لو كانت قليلة. كذلك يُمكن الإستفادة من هذا الكتاب كم قدمة لتعلم برمجة الموبايل بإستخدام جافا، مثل نظام أندرويد أو جافا موبايل. فإن من يعرف هذه اللغة تصبح له برمجة الموبايل أسهل.

لغة جافا

لغة جافا هي لغة متعددة الأغراض ومتعددة المنصات تصلح لعدد كبير من التطبي قات. ومترجم جافا ي قوم بإنتاج ملفات في شكل Byte code وهو يختلف عن الملفات التفيذية التي تنتج عن لغات البرمجة الأخرى مثل سي وباسكال. وتحتاج البرامج المكتوبة بلغة جافا إلى منصة في أنظمة التشغيل المختلفة لتتمكن برامجها من العمل في هذه الأنظمة. وهذه المنصة تُسمة آلة جافا الإفتراضية Java Virtual Machine **أو إختصاراً بـ** JVM أو Java Run-time.
تتوفر هذه الآلة في عدد كبير من أنظمة التشغيل، و قبل تشغيل برنامج جافا لابد من التأكد من وجودها. وكل نظام تشغيل يحتاج لآلة إفتراضية خاصة به. مثلاً نظام وندوز 32 بت يحتاج لآلة إفتراضية مخصصة لوندوز 32 بت، ووندوز 64 بت يحتاج لآلة إفتراضية 64 بت. وهذا مثال لإسم ملف لتثبيت آلة جافا الإفتراضية لنظام وندوز 64 بت:

```
jdk-6u16-windows-x64.exe
```

وهو يُمثل نسخة جافا 1.6 أو مايُسمى جافا 6
وهذه اسم حزمة تحتوي على الآلة الإفتراضية لجافا 7 لنظام أوبونتو 32 بت:

```
openjdk-7-jre
```

عند إنتاج برامج جافا يُمكن تشغيلها في أي نظام تشغيل مباشرة عند وجود الآلة الإفتراضية المناسبة، ولا يحتاج البرنامج لإعادة ترجمة حتى يعمل في أنظمة غير النظام الذي تم تطوير البرنامج فيه. مثلاً يُمكن تطوير برنامج جافا في بيئة لينكس ثم تشغيل البرنامج مباشرة في وندوز أو ماكنتوش. وتختلف عنها لغة سي وأوبجكت باسكال في أنها تحتاج لإعادة ترجمة البرامج مرة أخرى في كل نظام تشغيل على حدة قبل تشغيل تلك البرامج. لكن برامج لغة سي وأوبجكت باسكال لاتحتاج لآلة إفتراضية في أنظمة التشغيل بل تتعامل مع نظام التشغيل مباشرة.

أداة التطوير NetBeans

وهي من أفضل أدوات التطوير للغة جافا، و قد تمت كتابتها بإستخدام لغة جافا نفسها بواسطة شركة أوراكل صاحبة تلك اللغة.

يُمكن استخدام هذه الأداة لتطوير برامج بلغات برمجة أخرى غير الجافا مثل PHP و سي++.

توجد أداة تطوير أخرى مشهورة و هي Eclipse وهي أخف وأسرع من أداة التطوير NetBeans إلا أني لم استخدمها من قبل.

المؤلف: معتز عبدالعظيم

أعمل مطور برامج واستخدم لغة أوبجكت باسكال كلغة أساسية، لكن منذ أكثر من سنة بدأت تعلم جافا وكتبت بها عدد من البرامج. وكان إختياري لها كلغة إضافية هي:

1. أنه يوجد عدد كبير من المبرمجين يستخدمون جافا، بل أن معظمهم درسها في الجامعة. لذلك يُمكن أن تكون لغة مشتركة بين عدد كبير من المبرمجين.
2. توجد مكتبات كثيرة ومجانية تدعم مجال الإتصالات مكتوبة بلغة جافا، وهو المجال الذي أميل للعمل فيه.
3. أنها مجانية ومتوفرة لها أدوات تطوير متكاملة ذات إمكانات عالية في عدد من المنصات. ماعلى المبرمج إلا إختيار المنصة المناسبة له
4. تدعم البرمجة الكائنية بصورة قوية

ترخيص الكتاب

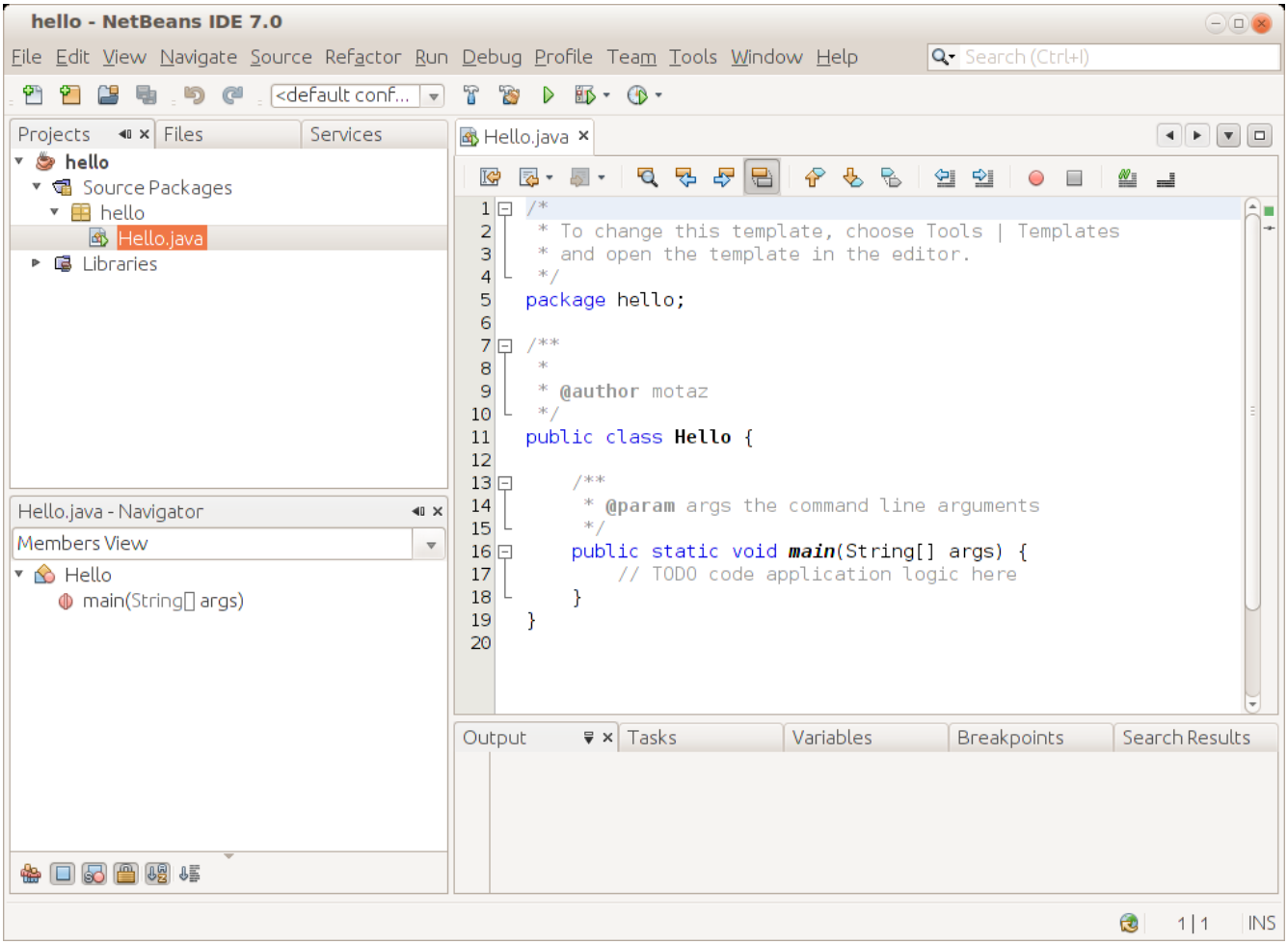
المحتويات

البرنامج الأول.

بعد تثبيت آلة جافا الإفتراضية وأداة التطوير NetBeans نـ قوم بإختيار New/Project ثم Java/Java Application. ثم نـ قوم بتسمية البرنامج *hello* ليظهر لنا الكود التالي:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package hello;

/*
 *
 * @author motaz
 */
public class Hello {

    /*
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

فإذا لم يظهر الكود نـ قوم بفتح الملف *hello.java* بواسطة شاشة المشروع التي تظهر يسار شاشة NetBeans كما في الشكل التالي:



بعد ذلك نـ قوم بكتابة السطر التالي داخل الإجراء main

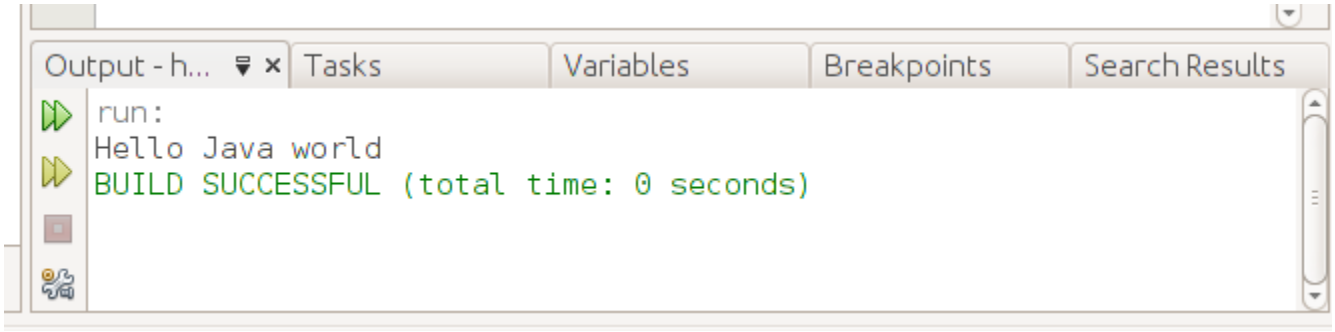
```
        System.out.print("Hello Java world\n");
```

ليصبح الكود كالتالي:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package hello;

/**
 *
 * @author motaz
 */
public class Hello {

    /**
     * @param args the command line arguments
     */
```

```
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.print("Hello Java world\n");

    }
}
```

يتم تشغيل البرنامج عن طريق المفتاح F6 ليظهر لنا المُخرجات في أسفل شاشة NetBeans



لتشغيل البرنامج الناتج خارج أداة التطوير، نـ قوم أولاً ببناء الملف التنفيذي بواسطة Build وذلك بالضغط على المفاتيح Shift + F11 . بعدها نبحث عن الدليل الذي يحتوي على برامج NetBeans ويكون اسمه في الغالب NetBeansProjects ثم داخل الدليل hello نجد دليل اسمه dist يحتوي على الملف التنفيذي. في هذه الحالة يكون اسمه hello.jar

يُمكن تنفيذ هذا البرنامج في سطر الأوامر في نظام التشغيل بواسطة كتابة الأمر التالي:

```
java -jar hello.jar
```

يُمكن نـ قل هذا الملف التنفيذي من نوع Byte code إلى أي نظام تشغيل آخر يحتوي على آلة جافا الإفتراضية ثم تنفيذه بهذه الطريقة. ونُلاحظ أن حجم الملف التنفيذي صغير نسبياً (حوالي كيلو ونصف) وذلك لأننا لم نستخدم مكتبات إضافية.

بعد ذلك نـ قوم بتغيير الكود إلى التالي:

```
        int num = 9;
        System.out.print(num + " * 2 = " + num * 2 + "\n");
```

وهذه طريقة لتعريف متغير صحيح أسميناه num وأسندنا له قيمة إبتدائية 9

وفي السطر الذي يليه قُمنا بكتابة قيمة المتغير، ثم كتابة قيمته مضروبة في الرقم 2. وفي نهاية الإجراء أضفنا الرمز \n والذي يُمثل رمز السطر الجديد في شاشة نظام التشغيل.

لطباعة التاريخ والساعة الحاليين نكتب هذه الأسطر:

```
        Date today = new Date();
        System.out.print("Today is: " + today.toString() + "\n");
```

ولابد من إضافة المكتبة المحتوية على الفئة Date في بداية البرنامج:

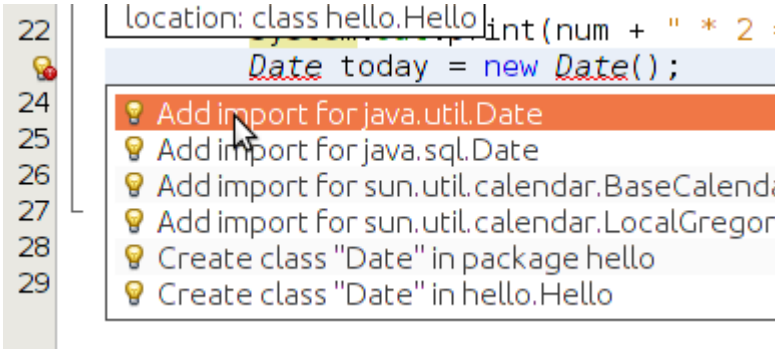
```
import java.util.Date;
```

فيصبح شكل كود البرنامج الكلي هو:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package hello;

import java.util.Date;


public class Hello {

    /*
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int num = 9;

        System.out.print(num + " * 2 = " + num * 2 + "\n");
        Date today = new Date();
        System.out.print("Today is: " + today.toString() + "\n");


    }
}
```

ملحوظة:

يُمكن إضافة إسم المكتبة تلقائياً عند ظهور العلامة الصفراء شمال السطر الموجودة فيه الفئة Class التي تحتاج لتلك المكتبة كما تظهر في هذه الصورة:



ثم اختيار *Add import for java.util.Date*

وهذه ميزة مهمة في أداة التطوير NetBeans تغني عن حفظ أسماء المكتبات المختلفة.

برنامج واجهة رسومية

من أهم الأشياء في أدوات التطوير هو دعمها للواجهات الرسومية أو مايُسمى بالـ Widgets. كل نظام تشغيل يحتوي على مكتبة أو أكثر تُمثل واجهة رسومية، مثلاً يوجد في نظام لينكس واجهات GTK و QT و في نظام وندوز توجد مكتبة وندوز الرسومية، وفي نظام ماكنتوش توجد مكتبات Carbon و Cocoa. أما جافا فلها مكتباتها الخاصة والتي تعمل في كل هذه الأنظمة ومنها واجهة Swing.

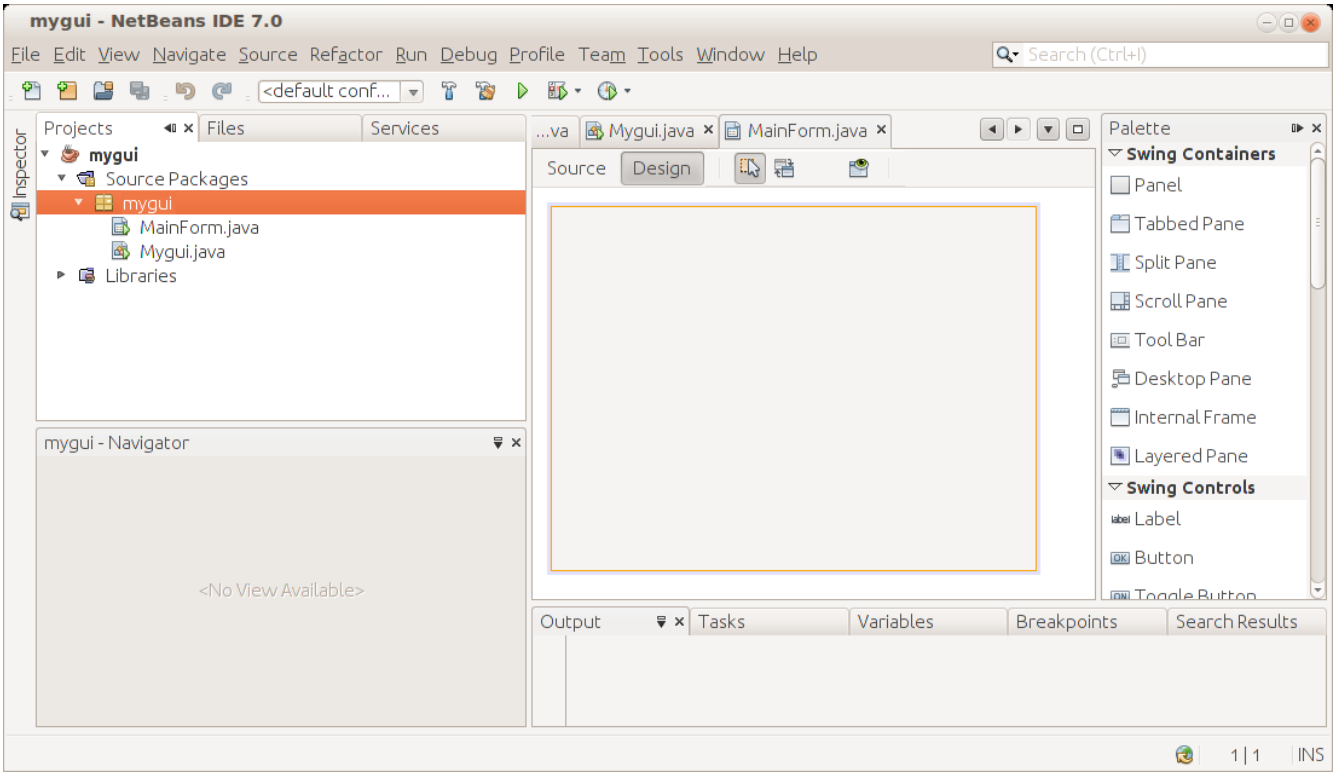
لكتابة أول برنامج ذو واجهة رسومية في جافا بإستخدام NetBeans نختار File/New Project

ثم نختار Java/Java Application

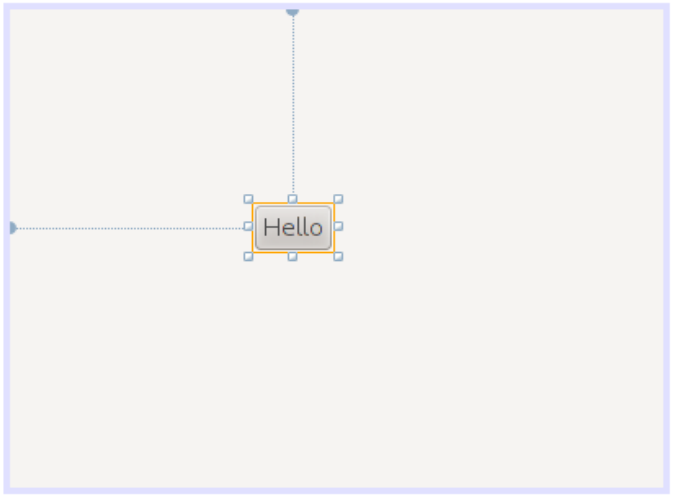
ون قوم بتسميته مثلاً mygui.

في شاشة Projects نختار الحزمة mygui ثم بالزر اليمين للماوس نختار New/JFrame Form

نسمى هذا الفورم MainForm فيتم إضافته للمشروع ويظهر بالشكل التالي:



يظهر الفورم الرئيسي المسمى MainForm.java في وسط الشاشة. وفي اليمين نلاحظ وجود عدد من **المكونات في صفحة الـ** Palette. ن قوم بإدراج زر Button في وسط الفورم الرئيسي، ثم ن قوم بتغيير عنوانه إلى Hello، وذلك إما بالضغط على زر F2 ثم تغيير العنوان، أو بالن قر على الزر اليمين في الماوس في هذا الزر ثم نختار Properties ثم Text



نرجع مرة أخرى للخصائص لنضيف حدث عند الضغط على الزر. هذه المرة نختار Events ثم في الخيار actionPerformed نختار الحدث jButton1ActionPerformed بعدها يظهر هذا الكود في شاشة الـ Source:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

أو يُمكن إظهار هذا الكود بواسطة الـ قر المزدوج على الزر double click

فنـ قوم بتكابة كود لإظهار عبارة (السلام عليكم) عند الضغط على هذا الزر. فيصبح الكود الحدث كالتالي:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String msg = "السلام عليكم";
        JOptionPane.showMessageDialog(null, msg);
    }
```

نلاحظ أننا قُمنا بتعريفا المتغير *msg* من النوع الـ قطعي *String* ثم قُمنا بإسناد قيمة إبتدائية له: *"السلام عليكم"*

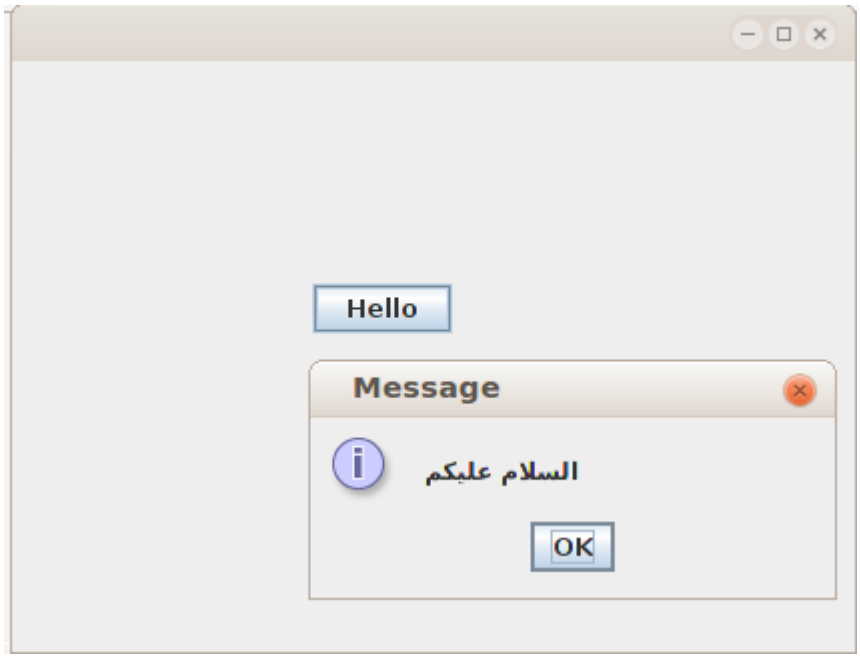
بعد ذلك نرجع للحزمة الرئيسية Mygui.java ثم نكتب الكود التالي في الإجراء main:

```
    public static void main(String[] args) {
        MainForm form = new MainForm();
        form.setVisible(true);
    }
```

في السطر الأول نُعرّف الكائن form من النوع MainForm الذي قُمنا بتصميمه، ثم نـ قوم بإنشاء نسخة من هذا النوع وتهيئته بواسطة new MainForm

وفي السطر الثاني قمنا بإظهار الفورم في الشاشة.

عند تنفيذ البرنامج يظهر بالشكل التالي عند الضغط على الزر:



نرجع مرة أخرى للفورم في شاشة Design ونـ قوم بإدراج المكون TextField لندخل فيه إسم المستخدم، ثم مكون من نوع Label نكتب فيه كلمة (الإسم) ثم مكون آخر من نوع Label نـ قوم بتغير إسمه إلى jlName وذلك في فورم الخصائص في صفحة Code في قيمة Variable Name

ثم نُدرج زر نكتب فيه كلمة (ترحيب) كما في الشكل التالي:

في الحدث ActionPerfomed لهذا الزر الجديد نكتب الكود التالي لكتابة إسم المستخدم في المكون jLabel2

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

   jlName.setText("مرحباً بك " + jTextField1.getText());
}
```

نلاحظ أن الإجراء getText يُستخدم لـ قراءة محتويات الحـ قل النصي Text Field والإجراء setText يـ قوم بوضع قيمة في عنوان المكون Label.

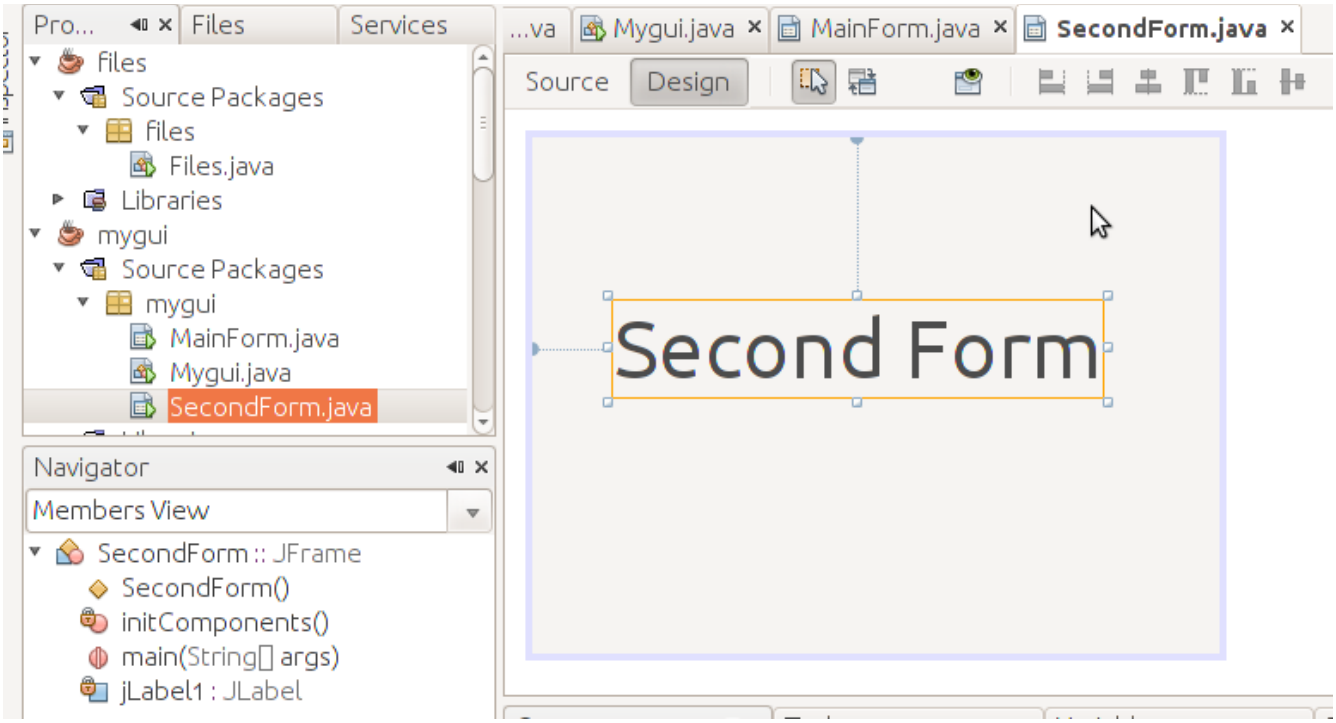
ملحوظة:

هذه الطريـ قة أفضل من إختيار Java /Java Desktop Application وذلـك بسبب أن هذا الخيار أختفى في النُسخ الجديدة من أداة التطوير NetBeans مثلاً النسخة ر قم 7.2 ويتعذر عرض البرامج التي تم فيها استخدام هذه الطريـ قة لإنشائها. لذلـك لابد أن نتذكر أن نـختار دائماً *New/Java/Java Application*

الفورم الثاني

لإضافة وإظهار فورم ثاني في نفس البرنامج، نتبع الخطوات في المثال التالي:

نـ قوم بإضافة JFrame Form ونسميه SecondFrom ونضع فيه Label نكتب فيه عبارة "Second Form" ونزريد حجم الخط في هذا العنوان بواسطة Properties/Font.



في خصائص هذا الفورم الجديد نـ قوم بتغيير الخاصية *defaultCloseOperation* إلى *Dispose* بدلاً من EXIT_ON_CLOSE لأننا إذا تركناها في الخيار الأخير يتم إغلاق البرنامج عندما نغلق الفورم الثاني. وجرت العادة في أن يتم إغلاق أي برنامج عند إغلاق شاشته الرئيسية. إغلاق الشاشات الفرعية يفترض به أن يـ قودنا إلى الشاشات الرئيسية.

نضيف زر في الفورم الرئيسي MainForm ونكتب الكود التالي في الحدث ActionPerformed في هذا الزر الجديد لإظهار الفورم الثاني، أو يمكن كتابة هذا الكود في زر الترحيب.

```
SecondForm second = new SecondForm();
second.setVisible(true);
```

يُمكن إرسال كائن أو متغير للفورم الجديد. مثلاً نريد كتابة رسالة الترحيب في الفورم الثاني.
لعمل ذلك نحتاج لتغير إجراء التهيئة constructor في الفورم الثاني والذي اسمه SecondForm ، نضيف إليه مدخلات:

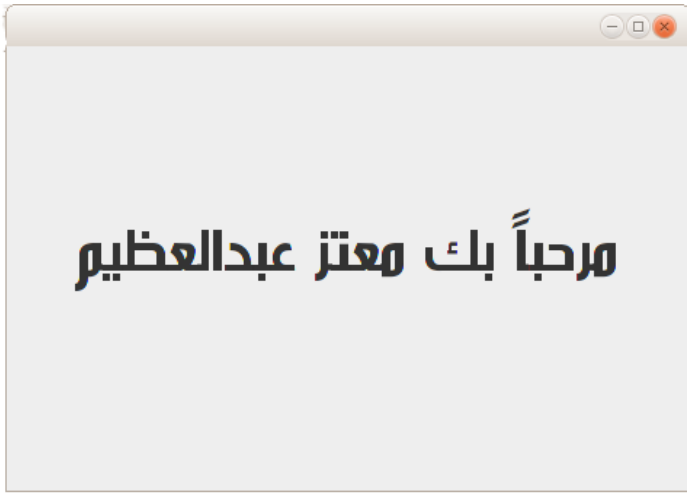
```
    public SecondForm(String atext) {
        initComponents();
        jLabel1.setText(atext);
    }
```

ثم نظهر هذه المدخلات – والتي هي عبارة عن رسالة الترحيب – في العنوان jLabel1

وعند تهيئة الفورم الثاني من الفورم الرئيسي نـ قوم بتعديل إجراء التهيئة إلى الكود التالي، وهذا الكود كتبناه في إجراء زر الترحيب:

```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        jlName.setText("مرحباً بك " + jTextField1.getText());

        SecondForm second = new SecondForm(jlName.getText());
        second.setVisible(true);
    }
```

عند التنفيذ يظهر هذا الشكل:



كتابة نص في ملف

في المثال التالي نُريد الكتابة في ملف نصي بإستخدام برنامج بدون واجهة رسومية (console application) وذلك بإختيار Java/Java Application.

هذه المرة نُريد كتابة إجراء جديد نعطيه إسم الملف المُراد إنشاءه والكتابة فيه والنص الذي نُريد كتابته في هذا الملف.

قمنا بتسمية المشروع *files*، وكتبنا الإجراء الجديد أسفل الإجراء *main* الموجود مسبقاً. وأسمينا الإجراء الجديد *writeToTextFile* وعرفناه بهذه الطريقة:

```
private static boolean writeToTextFile(String aFileName, String text)
  {

  }
```

نلاحظ أننا قُمنا بتعريف مُدخلين لهذا الإجراء وهما aFileName وهو من النوع النصي ليست قبل إسم الملف المراد كتابته، والآخر text من النوع النصي أيضاً والذي يُمثل محتويات الملف المُراد كتابتها.

ثم نقوم بكتابة الكود التالي داخل هذا الإجراء:

```
private static boolean writeToTextFile(String aFileName, String text)
  {
   try{
     FileOutputStream fstream = new FileOutputStream(aFileName);

     DataOutputStream textWriter = new DataOutputStream(fstream);

     textWriter.writeBytes(text);
     textWriter.close();
     fstream.close();
     return (true); // success


    }
      catch (Exception e)
    {
      System.err.println("Error: " + e.getMessage());
      return (false); // fail
    }

 }
```

نلاحظ أولاً أننا استخدمنا مايُعرف بمعالجة الإستثناءات exception handling وذلك لأن كتابة أو قراءة ملف يُمكن أن يحدث عنها خطأ في و قت التنفيذ، مثلاً يمكن أن لا تكون هنالك صلاحية للمستخدم لكتابة ملف جديد في دليل معين، أو أن الملف المُراد قراءته غير موجود، وغيرها من الإحتمالات. لذلك وجب علينا إحاطة هذه الإجراءات والأجزاء من الكود

التي يتو قع فيها حصول خطأ و قت التنفيذ بهذه العبارة:

```
    try{
      // Put the code you want to protect here

       return (true);

     }
       catch (Exception e)
     {
       System.err.println("Error: " + e.getMessage());
       return (false); // fail
     }
```

فإذا حدث أي خطأ بين الـ قوسين الموجودين بعد كلمة try يـ قوم البرنامج بالإنت قال مباشرة إلى الجزء الموجود بين الـ قوسين بعد عبارة catch والخطأ الذي حصل ترجع معلوماته في الكائن e من النوع Exception.

نلاحظ أيضاً أننا قُمنا بإرجاع الـ قيمة true في حال أن الكتابة في الملف تمت بدون حدوث خطأ. أما في حالة حدوث الخطأ أرجعنا الـ قيمة false وذلك ليعرف من يُنادي هذا الإجراء أن العملية نجحت أم لا.

بالنسبة لتعريف الملف وتعريف طريـ قة الكتابة عليه قُمنا بكتابة هذين السطرين:

```
      FileOutputStream fstream = new FileOutputStream(aFileName);

      DataOutputStream textWriter = new DataOutputStream(fstream);
```

في العبارة الأولى قُمنا بتعريف الكائن fstream من نوع الفئة FileOutputStream وهو كائن للكتابة في ملف. و قد أعطيناه إسم الملف في المدخلات. وفي العبارة الثانية قُمنا بتعريف الكائن textWriter من النوع DataOutputStream وذلك للكتابة على الملف، ومُدخلاته هو الكائن fstream.

بعد ذلك قُمنا بكتابة النص المُرسل داخل الملف بإستخدام الكائن textWriter بالطريـ قة التالية:

```
      textWriter.writeBytes(text);
```

ولنداء هذا الإجراء يجب إستدعاءه من الإجراء الرئيسي main بالطريـ قة التالي:

```
writeToTextFile("myfile.txt", "my text");
```

ويُمكن تحديد المسار أو الدليل الذي نُريد كتابة الملف عليه كما فعلنا في المثال التالي لنداء هذا الإجراء. و قد قُمنا

بإضافة التاريخ والو قت الذي تمت فيه كتابة الملف:

```
    public static void main(String[] args) {
        // TODO code application logic here
        Date now = new Date();
        boolean result;
        result = writeToTextFile("/home/motaz/java.txt",
                "This file has been written\n using Java\n" + now.toString());
        if (result)
            System.out.print("File has been written successfully\n");
        else
            System.out.print("Error has occured while writing in the file\n");

    }
```

كذلك ف قد قُمنا بتعريف المتغير *result* من النوع المنط قي boolean والذي يحتمل ف قط ال قيم true/false وذلك لإرجاع نتيجة العملية هل نجحت أم لا.

و قد قُمنا بفحص قيمة المتغير *result* لعرض رسالة تُفيد بأن العلمية نجحت، أو فشلت في حالة أن قيمته *false*. وعبارة

```
if (result)
```

معناها أن قيمة *result* إذا كانت تحمل ال قيمة *true* قم بتنفيذ العبارة التالية، أما إذا لم تكن تحمل تلك ال قيمة ف قم بتنفيذ الإجراء بعد الكلمة *else*

لتنفيذ هذا البرنامج نحتاج لإضافة المكتبات التالية، والتي تساعد أداة التطوير في إضافتها تل قائياً:

```
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Date;
```

قُمنا بعد ذلك بكتابة إجراء آخر ل قراءة محتويات ملف نصي، أسميناه *readTextFile* وكتبناه أسفل إجراء كتابة الملف:

```
    private static boolean readTextFile(String aFileName)
    {
     try{
       FileInputStream fstream = new FileInputStream(aFileName);

       DataInputStream textReader = new DataInputStream(fstream);
       System.out.print("Reading " + aFileName + "\n-------------\n");

       byte ch;
       while (textReader.available() != 0) {
```

```
        ch = textReader.readByte();
        System.out.write(ch);
      }

      textReader.close();
      fstream.close();
      return (true); // success


     }
       catch (Exception e)
     {
       System.err.println("Error in readTextFile: " + e.getMessage());
       return (false); // fail
     }

   }
```

نلاحظ أننا استخدمنا Input بدلاً من Output في كائنات قراءة الملف. كذلك فقد استخدمنا حلقة while لإستمرار القراءة من الملف حرف حرف إلى أن لا يتبقى ما يُقرأ في الملف. أي تصبح قيمة الدالة *available* صفراً في الكائن *textReader*

```
textReader.available()
```

وعبارة:

```
while (textReader.available() != 0)
```

تعني أن يقوم بتنفيذ الحلقة مادامت قيمة الدالة لاتساوي صفراً.

قُمنا كذلك بتعريف واستخدام المتغير ch من النوع byte وذلك لقراءة بايت واحد من الملف ثم كتابته في الشاشة، والبايت يُمثل رمز واحد character أو حرف من الملف النصي.

يُمكن تحويل كود القراءة في هذا الإجراء لأن يقوم البرنامج بـ قراءة محتويات الملف سطراً سطراً بدلاً من قراءة حرف واحد فقط في المرة الواحدة:

```
   private static boolean readTextFile(String aFileName)
   {
    try{
      FileInputStream fstream = new FileInputStream(aFileName);

      DataInputStream textReader = new DataInputStream(fstream);
      System.out.print("Reading " + aFileName + "\n-------------\n");

      BufferedReader lineReader =
        new BufferedReader(new InputStreamReader(textReader));

      String line;
```

```
      while ((line = lineReader.readLine()) != null)
        System.out.println (line);

      fstream.close();
      return (true); // success


    }
      catch (Exception e)
    {
      System.err.println("Error in readTextFile: " + e.getMessage());
      return (false); // fail
    }

  }
```

قُمنا بتعريف كائن جديد اسمه *lineReader* لـ قراءة سطر في مرة واحدة من نوع الفئة *BufferedReader*. لكن مُدخلاته هي من نوع *InputStreamReader*. لذلك أثناء تهيئته قُمنا بتهيئة متغير من هذا النوع في نفس العبارة:

```
      BufferedReader lineReader =
        new BufferedReader(new InputStreamReader(textReader));
```

وكان يُمكن أن نكتبها في عبارتين لتكون أكثر وضوحاً لمبرمج جافا الجديد:

```
      InputStreamReader isr = new InputStreamReader(textReader);
      BufferedReader lineReader = new BufferedReader(isr);
```

هذا المرة احتفظنا بمؤشر للكائن في إسم المتغير *isr*، لكن بما أننا لا نحتاج له إلا أثناء التهيئة للكائن *lineReader* فـ قمنا بتجاهل إسناد المتغير *isr* في المرة الأولى.

قُمنا بنداء الإجراء الجديد من داخل main. ليصبح الإجراء كاملاً هو:

```
  public static void main(String[] args) {
      // TODO code application logic here
      Date now = new Date();
      boolean result;
      result = writeToTextFile("/home/motaz/java.txt",
              "This file has been written\n using Java\n" + now.toString());
      if (result)
          System.out.print("File has been written successfully\n");
      else
          System.out.print("Error has occured while writing in the file\n");

      readTextFile("/home/motaz/java.txt");
  }
```

تعريف الكائنات والذاكرة

من الأمثلة السابقة نُلاحظ أننا استخدمنا البرمجة الكائنية في قراءة وكتابة الملفات والتاريخ. ونُلاحظ أن تعريف الكائن وتهيئته يمكن أن تكون في عبارة واحدة، مثلاً لتعريف التاريخ ثم تهيئته بالوقت الحالي استخدمنا:

```
Date today = new Date();
```

وكان يُمكن فصل التعريف للكائن الجديد من تهيئته بالطريقة التالية:

```
Date today;
today = new Date();
```

هذه المرة في العبارة الأولى قُمنا بتعريف الكائن today من نوع الفئة Date. لكن إلى الآن لا يُمكننا استخدام الكائن today فلم يتم حجز موقع له في الذاكرة.

أما في العبارة الثانية فقد قُمنا بحجز موقع له في الذاكرة بإستخدام الكلمة new ثم تهيئة اللكائن بإستخدام الإجراء

```
Date();
```

والذي بدوره يقوم بقراءة التاريخ والوقت الحالي لإسناده للكائن الجديد today. وهذا الإجراء يُسمى في البرمجة الكائنية constructor.

في هذا المثال Date هي عبارة عن فئة لكائن أو تُسمى class في البرمجة الكائنية. و المتغير today يُسمى كائن object أو instance ويُمكن تعريف أكثر من كائن instance من نفس الفئة لإستخدامها. وتعريف كائن جديد من فئة ما وتهيئتها تُسمى instantiation في البرمجة الكائنية.

بعد الفراغ من استخدام الكائن نقوم بتحريره من الذاكرة وذلك بإستخدام الدالة التالية:

```
today = null;
```

وهي تعني جعل متغير الكائن today لايؤشر إلى شيء في الذاكرة. لكن لم نقم بإستخدام تلك الطريقة في أمثلتنا السابقة، وذلك لأن لغة جافا تتميز بما يُعرف بال garbage collector وهي آلية لحذف الكائنات الغير مستخدمة من الذاكرة تلقائياً عندما ينتهي الإجراء المعرفة في نطاقه. أما لغات البرمجة الأخرى مثل سي وأوبجكت باسكال فعند استخدامها لابد من تحرير الكائنات يدوياً في معظم الحالات.

يمكن كذلك تهيئة كائن جديد بواسطة إسناد مؤشر كائن قديم له، في هذه الحالة يكون كلا المتغيرين يؤشران لنفس الكائن في الذاكرة:

```
Date today;
Date today2;
today = new Date();
today2 = today;
today = null;
System.out.print("Today is: " + today2.toString() + "\n");
```

نلاحظ أننا لم نـ قم بتهيئة المتغير today2 لكن بدلاً من ذلك جعلناه يؤشر لنفس الكائن today الذي تمت تهيئته من قبل.

بعد ذلك قُمنا بتحرير المتغير today، إلا أن ذلك لم يؤثر على الكائن، حيث أن الكائن مايزال مرتبط بالمتغير today2. ولا ت قوم آلية garage collector بتحرير الكائن من الذاكرة إلا عندما تصبح عدد المتغيرات التي تؤشر له صفراً. فإذا قُمنا بتحرير المتغير today2 أيضاً تحدث مشكلة عند تنفيذ السطر الأخير، وذلك لأن الكائن تم تحريره من الذاكرة ومحاولة الوصول إليه بال قراءة أو الكتابة ينتج عنها خطأ.

ولمعرفة ماهو الخطأ الذي ينتج قُمنا بإحاطة الكود بعبارة try catch كما في المثال التالي:

```
try {
  Date today;
  Date today2;
  today = new Date();
  today2 = today;
  today = null;
  today2 = null;
  System.out.print("Today is: " + today2.toString() + "\n");
} catch (Exception e) {
    System.out.print("Error: " + e.toString() + "\n");
}
```

والخطأ الذي تحصلنا عليه هو:

```
java.lang.NullPointerException
```

ملاحظة:

في لغة جافا أُصطلح على تسمية الفئات classes بطريقة أن يكون الحرف الأول كبير captital مثل Date, String, حتى الفئات التي يقوم بتعريفها المستخدم. أما الكائنات objects/instances فتبدأ بحرف صغير وذلك للتفريق بين الفئة والكائن، مثل today, today2, myName.

برنامج إختيار الملف

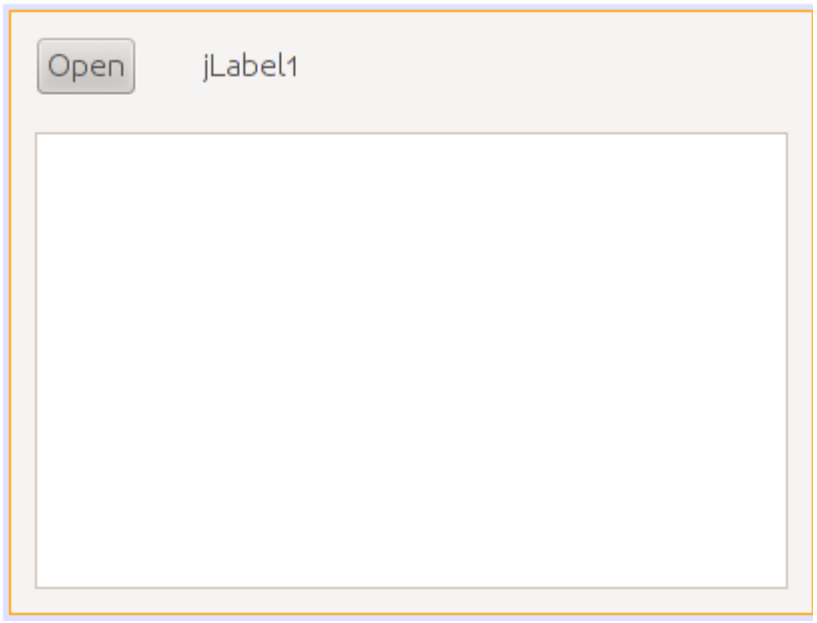
هذه المرة نُريد عمل برنامج ذو واجهة رسومية يسمح لنا بإختيار الملف بالماوس، ثم تُعرض محتوياته في صندوق نصي.

لعمل هذا البرنامج نفتح مشروع جديد بواسطة Java/Java Application. نسمي هذا المشروع openfile

نُضيف JFrame Form نسميه MainForm ونضع فيه المكونات التالية:

Button, Label, Text Area

كما في الشكل التالي:



بعد ذلك نكتب هذا الكود في الإجراء main في ملف البرنامج الرئيسي Openfile.java لإظهار الفورم فور تشغيل البرنامج:

```
    public static void main(String[] args) {

        MainForm form = new MainForm();
        form.setVisible(true);
    }
```

نـ قوم بنسخ الإجراء readTextFile من البرنامج السابق إلى كود البرنامج الحالي، ونعدله قليلاً، نضيف له مدخل جديد من نوع JTextArea وذلك لكتابة محتويات الملف في هذا المربع النصي بدلاً من شاشة سطر الأوامر Console. وهذا هو الإجراء المعدل:

```
    private static boolean readTextFile(String aFileName, JTextArea textArea)
    {
     try {
       FileInputStream fstream = new FileInputStream(aFileName);
```

```
      DataInputStream textReader = new DataInputStream(fstream);

      InputStreamReader isr = new InputStreamReader(textReader);
      BufferedReader lineReader = new BufferedReader(isr);

      String line;
      textArea.setText("");

      while ((line = lineReader.readLine()) != null)
        textArea.append(line + "\n");

      fstream.close();
      return (true); // success


     }
       catch (Exception e)
     {
       textArea.append("Error in readTextFile: " + e.getMessage() + "\n");

       return (false); // fail
     }
```

وفي الحدث ActionPerformed في الزر نكتب الكود التالي:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

     final JFileChooser fc = new JFileChooser();
     int result = fc.showOpenDialog(null);
     if (result == JFileChooser.APPROVE_OPTION) {
            jLabel1.setText(fc.getSelectedFile().toString());
            readTextFile(fc.getSelectedFile().toString(), jTextArea1);
      }
    }
```

و قد قُمنا بتعريف كائن إختيار الملف في السطر التالي:

```
final JFileChooser fc = new JFileChooser();
```

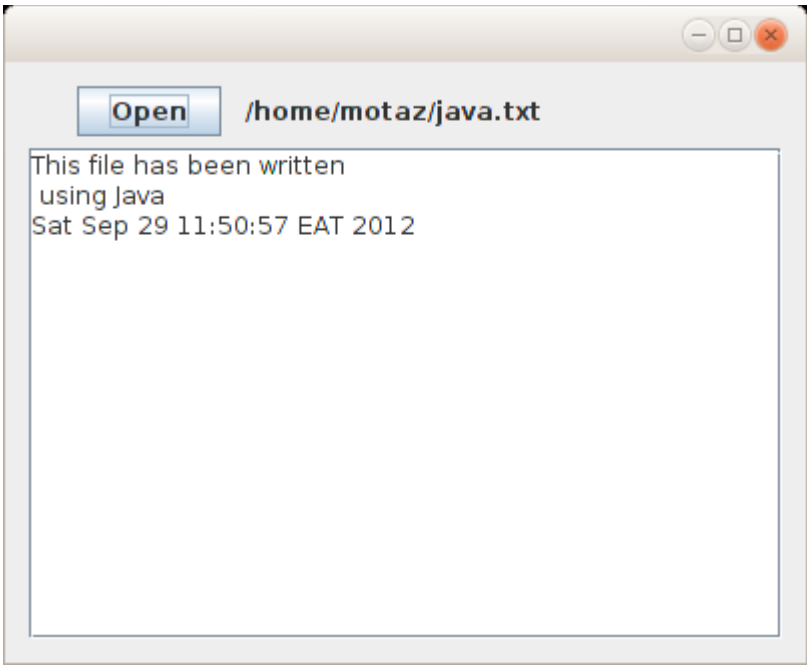
ثم قمنا بإظهاره ليختار المستخدم الملف في السطر التالي. ويِ قوم بإرجاع النتيجة: هل قام المستخدم بإختيار ملف أم ضغط إلغاء:

```
     int result = fc.showOpenDialog(null);
```

فإذا قام بإختيار ملف نـ قوم بكتابة اسمه في العنوان *jLabel1* ثم نظهر محتوياته داخل مربع النص:

```
     if (result == JFileChooser.APPROVE_OPTION) {

            jLabel1.setText(fc.getSelectedFile().toString());
            readTextFile(fc.getSelectedFile().toString(), jTextArea1);
      }
```

وعند تشغيل البرنامج يظهر لنا بهذا الشكل بعد إختيار الملف:



كتابة فئة كائن جديد New Class

لغة جافا تعتمد ف قط البرمجة الكائنية في هيكلها، و قد مر علينا في الأمثلة السابقة استخدام عدد من الكائنات، سواءً كانت لـ قراءة التاريخ أو للتعامل مع الملفات أو الكائنات الرسومية مثل Label والـ Text Area والفورم JFrameForm. لكن حتى تصبح البرمجة الكائنية أوضح لابد من إنشاء فئات classes جديدة لتعريف كائنات منها.

في هذا المثال سوف نـ قوم بإضافة فئة class جديدة نُدخل لها جملة نصية لإرجاع الكلمة الأولى والأخيرة من الجملة.

قمنا بفتح برنامج جديد من نوع Java Application، و أسميناه newclass،

بعد ذلك أضفنا MainForm من نوع JFrameForm

ثم قُمنا بإدراج Text Field و Button و Text Area بهذا الشكل في الفورم الرئيسي:

ولا ننسى تعريف الفورم وتهيئته لإظهاره مع تشغيل البرنامج في الإجراء الرئيسي في الملف Newclass.java:

```
    public static void main(String[] args) {

        MainForm form = new MainForm();
        form.setVisible(true);
    }
```

بعد ذلك قمنا بإضافة class جديدة وذلك بإختيار Source Packages/new class بالزر اليمين ثم إختيار New/Java Class من الـ قائمة. ثم نسمي الفئة الجديدة Sentence فيظهر لنا هذا الكود:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package newclass;

/*
 *
 * @author motaz
 */
public class Sentence {

}
```

وفي داخل كود الفئة –بين الـ قوسين المعكوفين {} – قمنا بإضافة متغير مـ قطعي اسميناه mySentence لنحفظ فيه الجملة التي يتم إرسالها لتكون محتفظة بـ قيمة الجملة طوال فترة حياة الكائن.

ثم أضفنا الإجراء الذي يُستخدم في تهيئة الكائن، ولابد أن يكون اسمه مطابق لإسم الفئة:

```
    String mySentence;

    public Sentence (String atext){
       super();
       mySentence = atext;
    }
```

نلاحظ أنه في هذا الإجراء تم إسناد قيمة المُدخل atext إلى المتغير mySentence المُعرف على نطاق الكائن. حيث أن المتغير atext نطا قه ف قط الإجراء Sentence وعند الإنتهاء من نداء هذا الإجراء يصبح غير معروف. لذلك إحتفظنا بالجملة المُدخلة في متغير في نطاق أعلى لتكون حياته أطول، حيث يُمكن إستخدامه مادام الكائن لم يتم حذفه من الذاكرة.

بعد ذلك قُمنا بإضافة إجراء جديد في نفس فئة الكائن اسمه getFirst وهو يـ قوم بإرجاع الكلمة الأولى من الجملة:

```
public String getFirst(){
    String first;
    int firstSpaceIndex;
    firstSpaceIndex = mySentence.indexOf(" ");

    if (firstSpaceIndex == -1)
        first = mySentence;
    else
        first = mySentence.substring(0, firstSpaceIndex);

    return (first);
}
```

نلاحظ اننا استخدمنا الإجراء *indexOf* في المتغير أو الكائن الـ قطعي mySentence و قُمنا إرسال مـ قطع يحتوي على مسافة. وهذا الإجراء أو الدالة مفترض بـه في هذه الحالة أن يـ قوم بإرجاع مو قع أول مسافة في الجملة، وبهذه الطريـ قة نعرف الكلمة الأولى، حيث أنها تـ قع بين الحرف الأول وأول مسافة.

أما إذا لم تكن هُنالك مسافة موجودة في الجملة فتكون نتيجة الدالة *indexOf* يساوي -1 وهذا يعني أن الجملة تتكون من كلمة واحدة فـ قط في هذه الحالة نـ قوم بإرجاع الجملة كاملة (الجملة = كلمة واحدة).

وإذا وُجدت المسافة فعندها نـ قوم بنسخ مـ قطع من الجملة بإستخدام الدالة substring والتي نُعطيها بداية ونهاية الـ قطع المُراد نسخـه. ونتيجة النسخ ترجع في المتغير أو الكائن الـ قطعي *first*

الدالة أو الإجراء الآخر الذي قُمنا بإضافته في الفئة Sentence هو *getLast* وهو يـ قوم بإرجاع آخر كلمة في الجملة:

```
public String getLast(){
    String last;
    int lastSpaceIndex;
    lastSpaceIndex = mySentence.lastIndexOf(" ");

    if (lastSpaceIndex == -1)
        last = mySentence;
    else
        last = mySentence.substring(lastSpaceIndex + 1, mySentence.length());

    return (last);

}
```

وهو مشابـه للدالة الأخرى، ويختلف في أنـه يـ قوم بالنسخة من آخر مسافة موجودة في الجملة (*lastIndexOf*) إلى نهاية الجملة mySentence.length

والكود الكامل لهذه الفئة هو:

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package newclass;

/*
 *
 * @author motaz
 */
public class Sentence {

    String mySentence;

    public Sentence (String atext){

       super();
       mySentence = atext;
    }

    public String getFirst(){

        String first;
        int firstSpaceIndex;
        firstSpaceIndex = mySentence.indexOf(" ");

        if (firstSpaceIndex == -1)
            first = mySentence;
        else
            first = mySentence.substring(0, firstSpaceIndex);

        return (first);
    }

    public String getLast(){

        String last;
        int lastSpaceIndex;
        lastSpaceIndex = mySentence.lastIndexOf(" ");

        if (lastSpaceIndex == -1)
            last = mySentence;
        else
            last = mySentence.substring(lastSpaceIndex + 1, mySentence.length());

        return (last);
    }
}
```

في كود الفورم الرئيسي للبرنامج MainForm.java قمنا بتعريف وتهيئة ثم استخدام هذا الكائن، واستقبلنا الجملة في مربع النص Text Field. وهذا هو الكود الذي يتم تنفيذه عند الضغط على الزر:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        Sentence mySent = null;
        mySent = new Sentence(jTextField1.getText());

        jTextArea1.append("First: " + mySent.getFirst() + "\n");
        jTextArea1.append("Last: " + mySent.getLast() + "\n");
    }
```

قمنا بإنشاء كائن جديد وتهيئته في هذا السطر:

```
        mySent = new Sentence(jTextField1.getText());
```

والجُملة المُدخلة أثناء التهيئة تحصلنا عليها من مربع النص jTextField1 بواسطة الإجراء *getText* الموجود في هذا الكائن.

قاعدة البيانات SQLite

قاعدة البيانات SQLite هي عبارة عن قاعدة بيانات في شكل مكتبة معتمدة على ذاتها *self-contained* للتعامل مع قاعدة SQLite. ويُمكن استخدام طريقة SQL للتعامل معها. ويمكن استخدامها في أنظمة التشغيل المختلفة بالإضافة إلى الموبايل، مثلاً في نظام أندرويد أو BlackBerry

يمكن الحصول على المكتبة الخاصة بها وبرنامج لإنشاء قواعد بيانات SQLite والتعامل مع بياناتها من هذا الرابط:

http://sqlite.org/download.html

لإستخدامها في نظام وندوز نبحث عن ملف يبدأ بالإسم sqlite-shell، أما في نظام لينكس يمكننا تثبيت تلك المكتبة وأدواتها بواسطة مثبت الحزم. فقط نبحث عن الحزمة sqlite3

بعد ذلك نقوم بالإنتقال إلى شاشة الطرفية terminal لتشغيل البرنامج وهو من نوع برامج سطر الأوامر، ثم نختار دليل معين لإنشاء قاعدة البيانات ثم نكتب هذا الأمر:

```
sqlite3 library.db
SQLite version 3.7.9 2011-11-01 00:52:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

بهذه الطريقة نكون قد أنشأنا قاعدة بيانات في ملف إسمه library.db

والآن مازلنا نستخدم هذه الأداة للتعامل مع قاعدة البيانات. ثم قُمنا بإضافة جدول جديد اسمه books بهذه الطريقة:

```
sqlite> create table books(BookId int, BookName varchar(100));
```

ثم أضفنا كتابين في هذا الجدول:

```
sqlite> insert into books values (1, "Introduction to Java 7");
sqlite> insert into books values (2, "One day trip with Java");
```

ثم عرضنا محتويات الجدول:

```
sqlite> select * from books;
1|Introduction to Java 7
2|One day trip with Java
sqlite>
```

الآن لدينا قاعدة بينات اسمها library.db وبها جدول اسمه books. يمكن الآن التعامل معها في برنامج جافا كما في المثال التالي.

برنامج لقراءة قاعدة بيانات SQLite

قبل بداية كتابة اي برنامج لـ قاعدة بيانات SQLite بواسطة جافا يجب أن نبحث عن مكتبة جافا الخاصة بها. وهي مكتبة إضافية غير موجودة في آلة جافا الإفتراضية. ويُمكن الحصول عليها من هذا المو قع:

http://www.xerial.org/trac/Xerial/wiki/SQLiteJDBC

وإسم المكتبة هو sqlite-jdbc ثم نختار ر قم النسخة المناسب. في هذه الأمثلة اخترت الملف:

```
sqlite-jdbc-3.7.2.jar
```

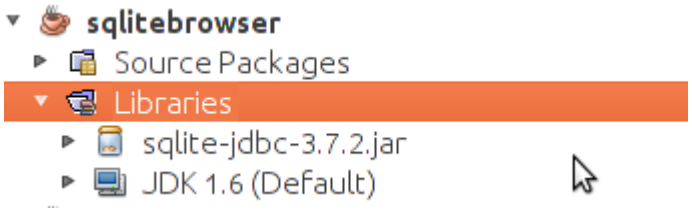
من هذا الرابط:

http://www.xerial.org/maven/repository/artifact/org/xerial/sqlite-jdbc/3.7.2/

وهذه المكتبة هي كُل مانحتاجه للتعامل مع قاعدة البيانات SQLite في برامج جافا، فهي لاتحتاج لمخدم لتثبيته حتى تعمل قاعدة البيانات كما قُلنا سابـ قاً.

قمنا بفتح مشروع جديد أسميناه sqlitebrowser لعرض قاعدة البيانات Library التي قُمنا بإنشائها سابـ قاً.

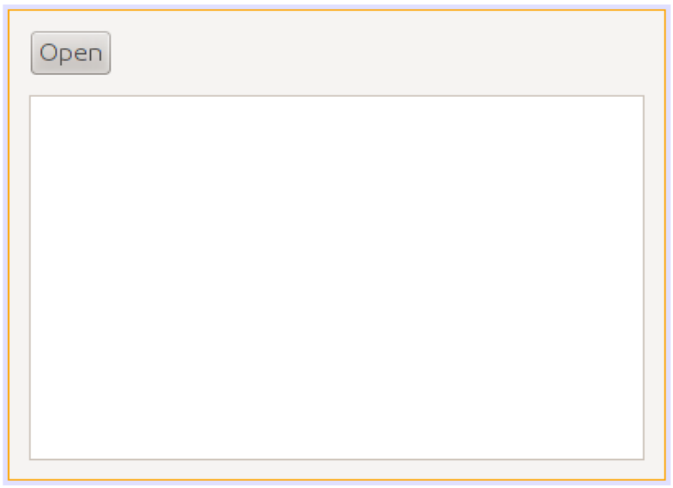
في شاشة المشروع يوجد فرع أسمه Libraries نـ قف عليه ثم نختار بالزر اليمين للماوس Add JAR/Folder ثم نختار الملف sqlite-jdbc-3.7.2.jar الذي قُمنا بتحميله من الإنترنت سابـ قاً.



أضفنا JFrame Form وأسميناه MainForm واستدعيناه من الملف الرئيسي Sqlitebrowser.java بالطرِ قة التالية:

```
    public static void main(String[] args) {
        MainForm form = new MainForm();
        form.setVisible(true);
    }
```

في الفورم أضفنا زر و مربع نص TextArea بالشكل التالي:



ثم أضفنا فئة كائن جديد New class أسميناه SqliteClient وهو الكائن الذي سوف يحتوي على إجراءات قراءة قاعدة بيانات *SQLite* والكتابة فيها.

قمنا بتعريف الكائن *dbConnection* من نوع *Connection* داخل كود فئة الكائن *SqliteClient* لتعريف مسار قاعدة البيانات والإتصال بها للإستخدام لاحقاً في باقي إجراءات الكائن SqliteClient.

ثم قمنا بكتابة الكود لإستقبال إسم قاعدة البيانات ثم الإتصال بها في الإجراء الرئيسي *constructor* لهذا الكائن:

```
public class SqliteClient {
        Connection dbConnection = null;

// Constructor
public SqliteClient (String aDatabaseName) {
        super();
        try {
          Class.forName("org.sqlite.JDBC");
          dbConnection = DriverManager.getConnection(
            "jdbc:sqlite:" + aDatabaseName);
        }
        catch (Exception e){
            System.out.println("Error while connecting: " + e.toString());
        }

    }
```

نلاحظ أننا قُمنا بحماية الكود بواسطة *try .. catch* وذلك لأنه من المتوقع أن تحدث مشكلة أثناء التشغيل، مثلاً أن تكون قاعدة البيانات المُدخلة غير موجودة، أو أن مكتبة SQLite غير موجودة.

الإجراء الأول (Class.forName) يقوم بتحميل مكتبة SQLite لنتمكن من نداء الإجراءات الخاصة بهذه القاعدة من تلك المكتبة التي قُمنا بتحميلها من الإنترنت، فإذا لم تكن موجودة سوف يحدث خطأ.

في السطر التالي قمنا بتهيئة الكائن dbConnection وإعطائه إسم الملف التي تم إرساله عند تهيئة الكائن SqliteClient

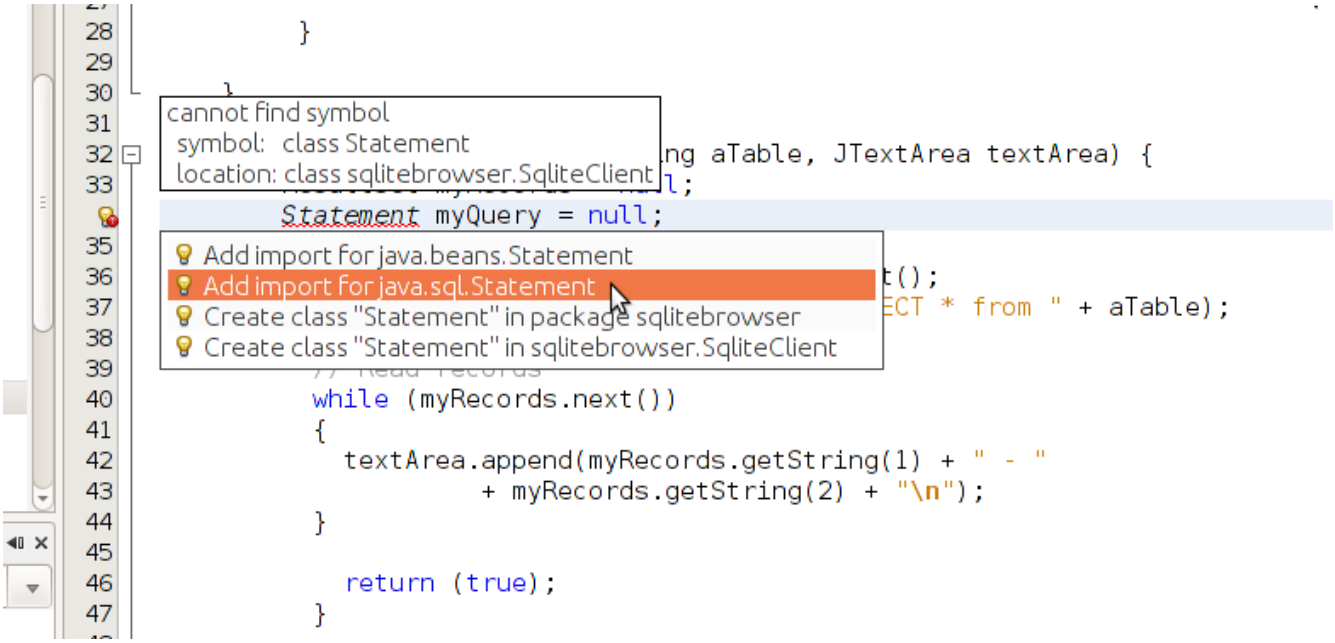
بعد ذلك قُمنا بإضافة الإجراء *showTable* إلى فئة الكائن SqliteClient لعرض محتويات الجدول المرسل لهذا الإجراء في مربع النص:

```
  public boolean showTable(String aTable, JTextArea textArea) {

      ResultSet myRecords = null;
      Statement myQuery = null;

      try {
        myQuery = dbConnection.createStatement();
        myRecords = myQuery.executeQuery("SELECT * from " + aTable);

        // Read records
        while (myRecords.next())
        {
          textArea.append(myRecords.getString(1) + " - "
                   + myRecords.getString(2) + "\n");
        }

       catch (Exception e){

          textArea.append("Error while reading table: " + e.toString() + "\n");
          return (false);
      }
  }
```

قمنا في هذا الإجراء بتعريف كائن من نوع *Statement* أسميناه *myQuery* يسمح لنا بكتابة query بلغة SQL على قاعدة البيانات.

وعند إضافة المكتبة المحتوية على الكائن *Statement* لابد من أن ننتبه لإختيار المكتبة java.sql.Statement ولا نختار الخيار الأول الذي يظهر عند الإضافة التا قائية java.beans.Statement التي تتسبب في أخطاء أثناء الترجمة.

الخيار الصحيح للمكتبة يظهر في الشكل التالي:



ثم قمنا بتهيئته على النحو التالي:

```
myQuery = dbConnection.createStatement();
```

ثم قمنا بنداء الإجراء executeQuery في الكائن myQuery وأعطيناه م قطع SQL والذي به أمر عرض محتويات الجدول. هذا الإجراء يُرجع كائن جديد من هو عبارة عن حزمة البيانات ResultSet. استقبلناه في الكائن *myRecords* والذي هو من نوع فئة الكائن ResultSet والذي قُمنا بتعريفه في بداية الإجراء دون تهيئته.

بعد هذه الإجراءات قُمنا بالمرور على كل السجلات في هذا الجدول وعرضنا بعض الحقول في مربع النص الذي تم إرساله كمُدخل للإجراء *showTable*:

```
// Read records
while (myRecords.next())
{
  textArea.append(myRecords.getString(1) + " - "
                + myRecords.getString(2) + "\n");
}
```

والإجراء *next* يقوم بتحريك مؤشر القراءة لبداية الجدول أو حزمة البيانات ثم الإنتقال في كل مرة إلى السجل الذي يليه ويرجع القيمة true. وعندما تنتهي السجلات أو لايكون هُناك سجلات من البداية ترجع القيمة false وعندها تتوقف الحلقة.

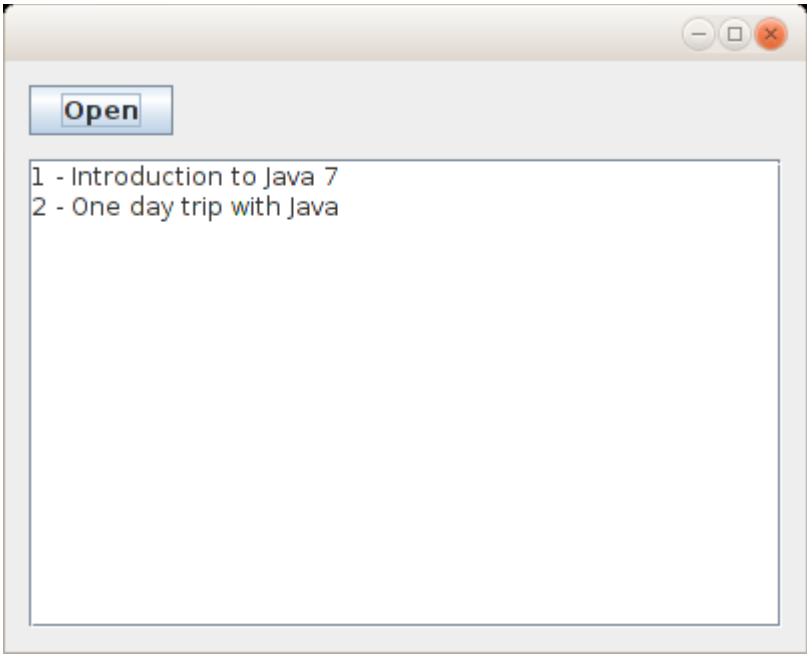
داخل الحلقة قرأنا الحقل الأول والثاني من الجدول المُرسل بواسطة الدالة *getString* وأعطيناها رقم الحقل Field/Column وهي تُرجع البيانات في شكل مقطع، ويُمكن استخدامها حتى مع الأنواع الأخرى مثل الأعداد الصحيحة مثلاً أو التاريخ، فكلها يُمكن تمثيلها في شكل مقاطع.

في الإجراء التابع للزر *Open* في الفورم الرئيسي MainForm قمنا بكتابة هذا الكود لعرض سجلات الجدول books الموجود في قاعدة البيانات library.db

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        SqliteClient sql = new SqliteClient("/home/motaz/library.db");

        jTextArea1.setText("");
        sql.showTable("books", jTextArea1);
    }
```

في السطر الأول قمنا بتعريف الكائن *sql* من نوع الفئة التي قمنا بإنشائها SqliteClient ثم تهيئتها بإرسال إسم ملف قاعدة البيانات. وهذا المثال لبرنامج في بيئة لينكس.

ثم في السطر الثاني قمنا بحذف محتويات مربع النص. ثم في السطر الثالث إستدعينا الإجراء showTable في هذا الكائن لعرض محتويات الجدول books وكانت النتيجة كالتالي:



لإضافة كتاب جديد في قاعدة البيانات في الجدول tables أولاً نـ قوم بإضافة إجراء جديد نسميه مثلاً *insertBook* في فئة الكائن SqliteClient بعد الإجراء showTable. وهذا هو الكود الذي كتبناه لإضافة كتاب جديد:

```
    public boolean insertBook(int bookID, String bookName) {

        try {
            PreparedStatement insertRecord = dbConnection.prepareStatement(
                "insert into books (BookID, BookName) values (?, ?)");
```

```
            insertRecord.setInt(1, bookID);
            insertRecord.setString(2, bookName);
            insertRecord.execute();
            return(true);
          }

          catch (Exception e){
            System.out.println("Error while reading table: " + e.toString());
            return (false);
        }
```

في العبارة الأولى لهذا الإجراء قمنا بتعريف الكائن *insertRecord* من نوع الفئة PreparedStatement وهي تُستخدم لتنفيذ إجراء على البيانات DML مثل إضافة سجل، حذف سجل أو تعديل. ونلاحظ أننا وضعنا علامة إستفهام في مكان الـ قيم التي نُريد إضافتها في الجزء values. وهذه تُسمى مدخلات parameters. سوف يتم تعبئتها لاحـ قاً.

في العبارة الثانية وضعنا ر قم الكتاب في المُدخل الأول بواسطة *setInt*، ثم في العبارة الثالثة وضعنا إسم الكتاب في المُدخل الثاني بواسطة *setString*، ثم قمنا بتنفيذ هذا الإجراء بواسطة execute والتي تـ قوم بإرسال طلب الإضافة هذا إلى مكتبة SQLite والتي بدورها تـ قوم بتنفيذه في ملف قاعدة البيانات library.db

بعد ذلـ نضيف فورم ثاني من نوع JFrame Form ونسميه AddForm نضع فيه المكونات jLabel و jTextField بالشكل التالي:

ولاننسى تحويل خاصية إغلاق الفورم defaultCloseOperation إلى Dispose بدلاً من Exit_On_Close حتى يتم إغلاق البرنامج عند إغلاق هذا الفورم الغير رئيسي.

وفي حدث الزر Insert نكتب فيه الكود التالي:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        SqliteClient query = new SqliteClient("/home/motaz/library.db");
```

```
        int bookID;
        bookID = Integer.parseInt(jTextField1.getText().trim());
        query.insertBook(bookID, jTextField2.getText());
        setVisible(false);
    }
```

في السطر الأول قمنا بتعريف الكائن query من نوع الفئة SqliteClient والتي تحتوي إجراء الإضافة الذي أضفناه مؤخرا.

في السطر الثاني قمنا بتعريف المتغير bookID من نوع العدد الصحيح.

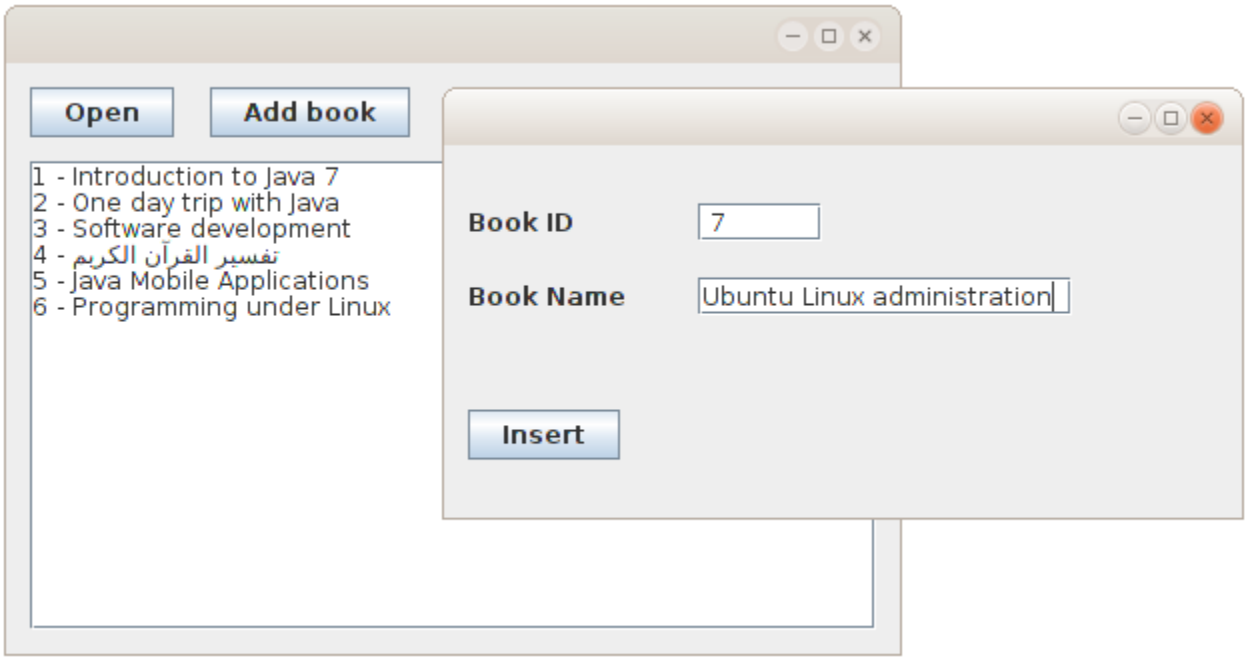
الـ قل jTextField1 يُرجع محتوياته بواسطة الإجراء getText في شكل مـ قطع String. ونحن نريده أن يستـ قبل ر قم الكتاب وهو من النوع الصحيح والمـ قطع يمكن أن يحتوي على عدد صحيح. فـ قمنا بتحويل المـ قطع إلى عدد صحيح بعد حذف أي مسافة غير مرغوب فيها –إن وجدت– بواسطة الدالة trim الموجودة في الكائن String، وذلك لأن العدد إذا كان يحتوي على حروف أو رموز أخرى أو مسافة فإن التحويل إلى ر قم بواسطة الإجراء parseInt سوف ينتج عنها خطأ. والدالة trim تـ قوم بإرجاع مـ قطع محذوفة فيه المسافة من بداية ونهاية النص، لكنها لاتؤثر على الكائن الذي تم تنفيذها فيه. مثلاً الكائن jTextField1 لايتم حذف المسافة منه. لتوضيح ذلك انظر المثال التالي:

```
        myText = aText.trim();
```

الكائن aText لايتأثر بالدالة trim أما الكائن myText فتيم تخزين مـ قطع فيه من المـ قطع aText بدون مسافة.

في السطر الرابع قمنا بإستدعاء الإجراء insertBook في الكائن query وأعطيناه ر قم الكتاب في المدخل الأول ثم اسم الكتاب في المدخل الثاني. ثم قمنا بإغلاق الفورم في السطر الأخير بواسطة setVisible وأعطيناها الـ قيمة false.

وهذا هو شكل البرنامج بعد تنفيذه:

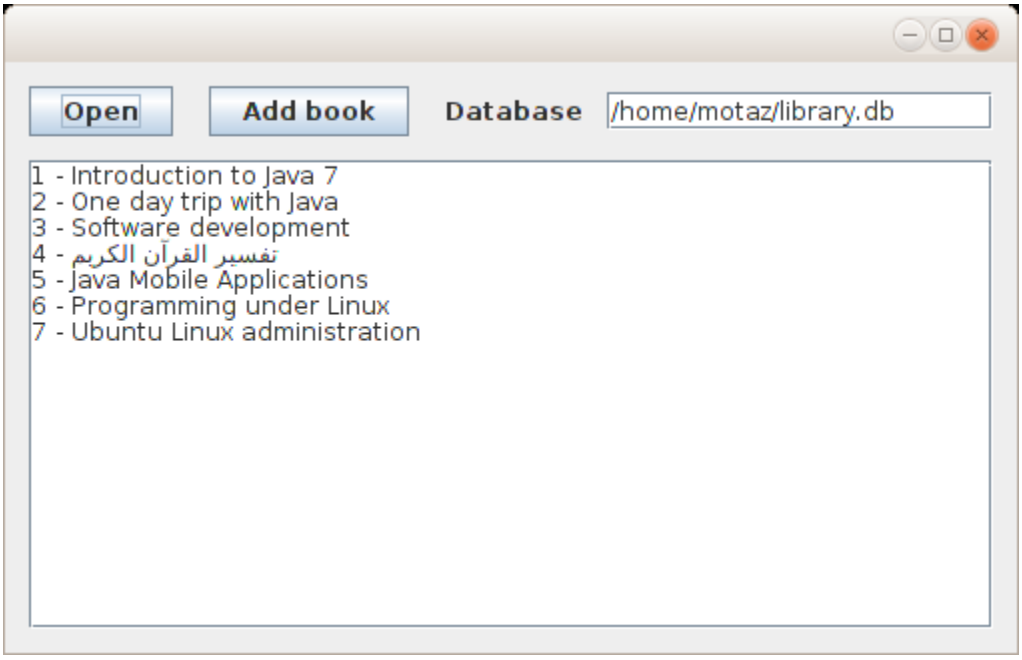


عند عمل build لهذا البرنامج بواسطة shift + F11 نلاحظ وجود دليل فرعي إسمه lib داخل الدليل dist وهو يحتوي على المكتبة التي استخدمناها والتي هي ليست جزء من آلة جافا الإفتراضية. وعند نـ قل البرنامج إلى أجهزة أخرى لابد من نـ قل الدليل lib مع البرنامج، وإلا تعذر تشغيل إجراءات قاعدة البيانات.

في هذا المثال نحتاج لنـ قل ثلاث ملفات:

- `sqlitebrowser.jar` وهو الملف التنفيذي للبرنامج في صيغة Byte code
- `sqlite-jdbc-3.7.2.jar` وهو ملف المكتبة داخل الدليل lib. ولابد أن نضعه في هذا الدليل داخل الدليل الذي نضع فيه البرنامج
- `library.db` وهو ملف قاعدة البيانات. وكان من الأفضل جعل مسار قاعدة البيانات خارج كود البرنامج، مثلاً

نضيف jTexField آخر ليكون البرنامج بالشكل التالي:



ونغير تهيئة كائن قاعدة البيانات بالشكل التالي:

```
        SqliteClient sql = new SqliteClient(jTextField1.getText());
```

بهذه الطريقة يكون البرنامج أكثر حرية في النقل portable ولايعتمد على ثوابت في نظام معين. وهي طريقة جيدة في تطوير البرامج تزيد من إمكانية إستخدامه، خصوصا عند إختيار لغة برمجة متعددة المنصات مثل جافا.

تكرار حدث بواسطة مؤقت

في هذا المثال نُريد كتابة التاريخ والساعة في الشاشة كل فترة معينة، مثلاً كل ثانية. ولعمل ذلك قمنا بفتح مشروع جديد سميناه timer ثم أضفنا MainForm من نوع JFrame Form، ثم أضفنا فئة جديدة new class أسميناها MyTimer فكان تعريفها بالشكل التالي:

```
public class MyTimer {
```

لكن قمنا بتغير هذا التعريف لنستخدم مايُعرف بالوراثة inheritance وذلك بدلاً من كتابة فئة كائن جديد من الصفر نستخدم كائن لديه خصائص مشابهة ثم نزيد فيها. وفئة هذا الكائن اسمها TimerTask. نقوم بوراثته بهذه الطريقة:

```
public class MyTimer extends TimerTask{
```

ثم نقوم بتعريف كائن myLabel بداخله حتى نقوم بعرض التاريخ والوقت فيه، ثم قمنا بكتابة إجراء التهيئة:

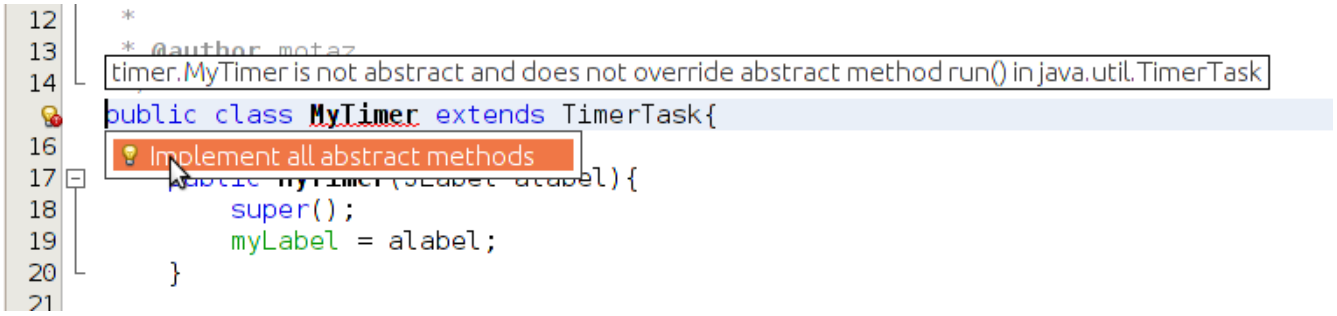
```
    JLabel myLabel;

    public MyTimer(JLabel alabel){
        super();
        myLabel = alabel;
    }
```

وفي هذا الإجراء نستقبل الكائن alabel ثم نقوم بحفظ نسخة منه في الكائن myLabel.

بعد ذلك نقوم بكتابة الإجراء الذي سوف يتم إستدعائه كل فترة وأسمه run بهذه الطريقة:

```
    @Override
    public void run() {
        Date today = new Date();
        myLabel.setText(today.toString());

    }
```

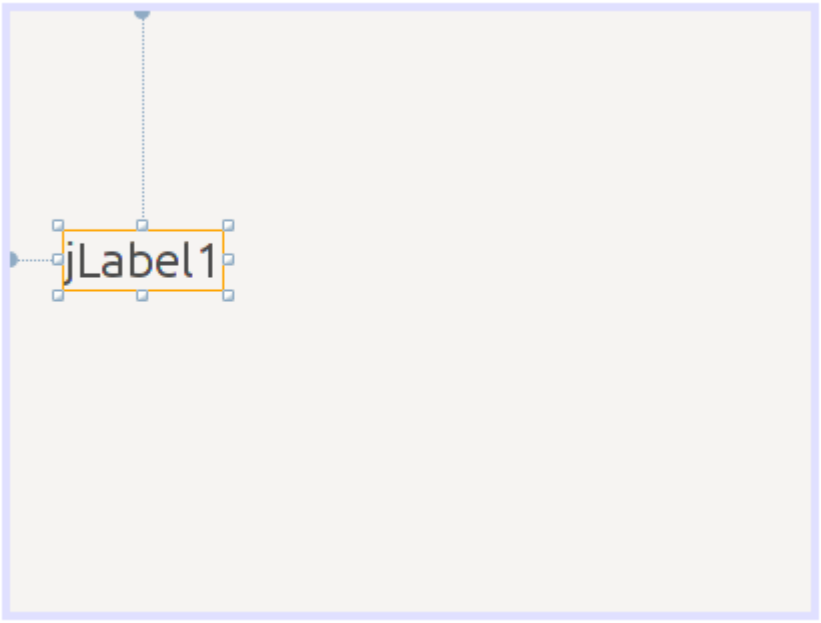
ويمكن إضافة تعريف هذا الإجراء تلقائياً بواسطة implement all abstract methods والتي تظهر في سطر تعريف الكائن MyTimer بالطريقة التالية:



وهذا هو كود الكائن كاملاً:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timer;

import java.util.Date;
import java.util.TimerTask;
import javax.swing.JLabel;

/*
 *
 * @author motaz
 */
public class MyTimer extends TimerTask{
    JLabel myLabel;
    public MyTimer(JLabel alabel){
        super();
        myLabel = alabel;
    }

    @Override
    public void run() {
        Date today = new Date();
        myLabel.setText(today.toString());

    }


}
```

نضع jLabel في الفورم الرئيسي MainForm ونزيد حجم الخط فيه ليكون بالشكل التالي:



في إجراء تهيئة هذا الفورم نعدل الكود إلى التالي:

```
    public MainForm() {
        initComponents();
        java.util.Timer generalTimer = null;

        MyTimer timerObj = new MyTimer(jLabel1);
        generalTimer = new java.util.Timer("time loop");
        generalTimer.schedule(timerObj, 2000, 1000);
    }
```

في هذا السطر:

```
        java.util.Timer generalTimer = null;
```

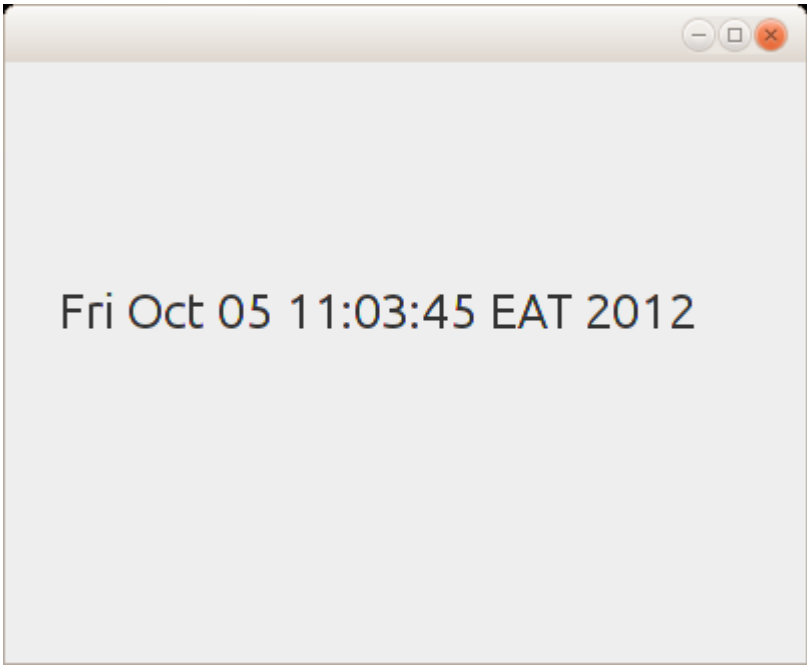
قمنا بتعريف الكائن generalTimer من النوع Timer. وهذا الكائن لديه خاصية تكرار الحدث بفترة زمنية محددة.

ثم قمنا بتعريف الكائن timerObj من الفئة MyTimer والتي قمنا بكتابتها لإظهار التاريخ والو قت.

ثم قمنا بتهيئة الكائن generalTimer. وفي السطر الأخير قمنا بتشغيل المُؤ قت shedule وأرسلنا له الكائن timerObj ليِ قوم بتنفيذ الإجراء run كل فترة معينة. والر قم الأول 2000 هو بداية التشغيل أول مرة، وهو بالملي ثانية، أي يِ قوم بالإنتظار ثانيتين ثم التشغيل أول مرة.

الر قم الثاني 1000 هو التكرار بالملي ثانية أيضاً. حيث يِ قوم بإظهار التاريخ والو قت كل ثانية.

نـ قوم بتشغيل البرنامج لنرى أن الثواني تتغير في الفورم الرئيسي:



وسوف يتم تشغيل هذا الإجراء تلقائياً إلى إغلاق البرنامج.

وفي الختام نتمنى أن تُنال الفائدة من هذا الكتاب.

معتز عبدالعظيم

code.sd