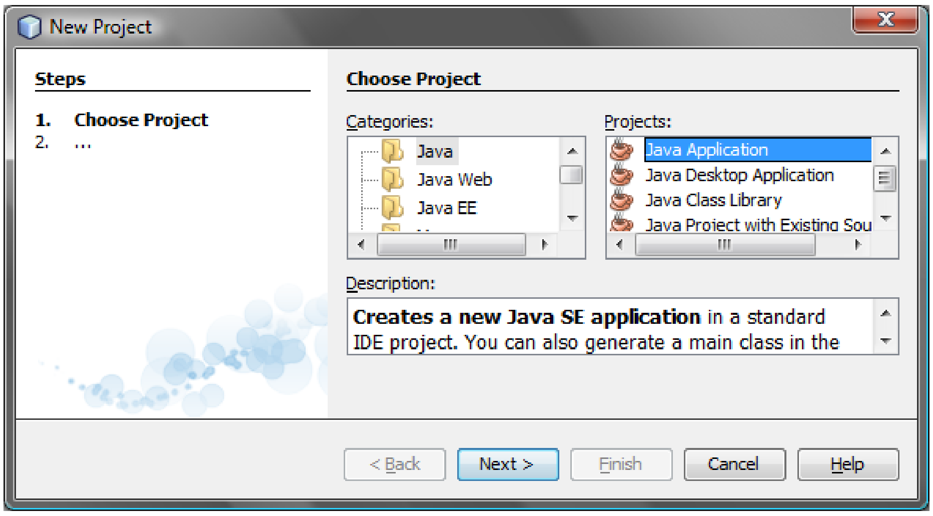
5 Creating and coding a simple GUI

The first part is going to show how to create a GUI version of the *MyWeight* application. There will be fields to enter a value and display the result, and two buttons - one to do the conversion and one to exit. The second part of this section is an exercise to create a GUI version of *MyHeight*.

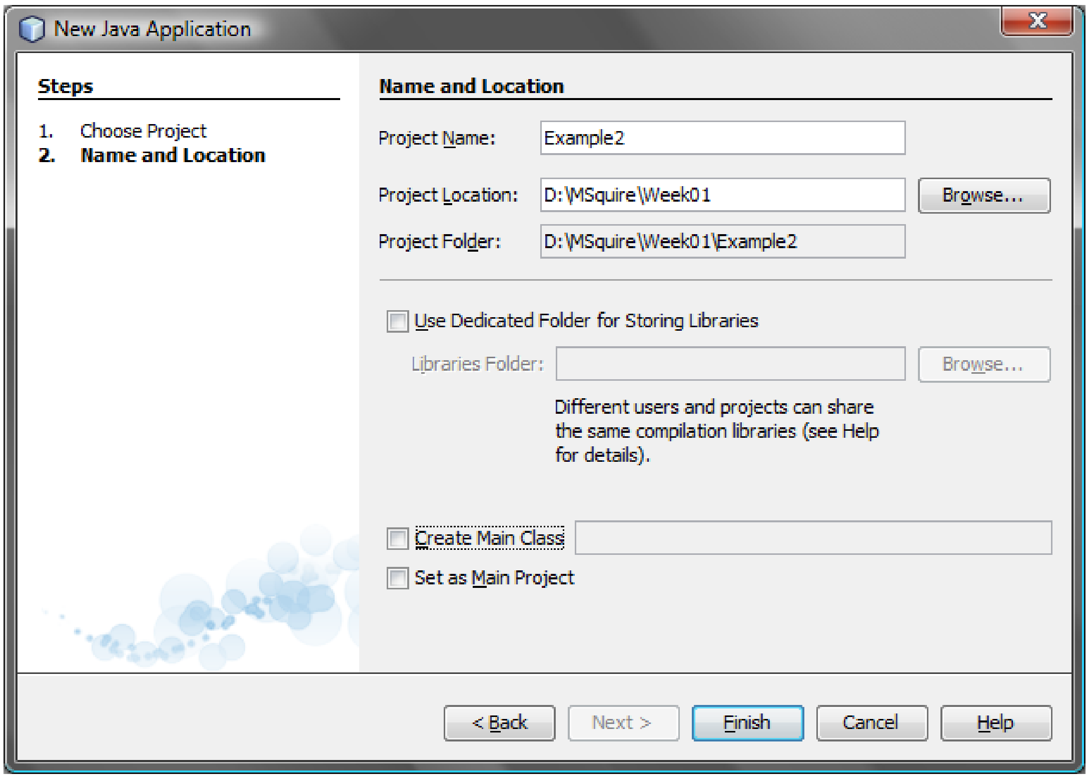
5.1 How to create a GUI application

- Create a new project for a Java Application:



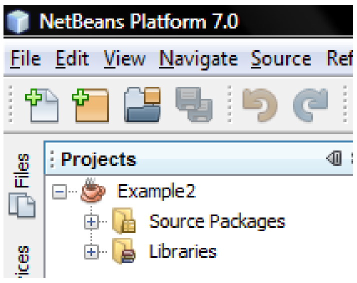
Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the

- Click **Next >**
- Fill in the form as shown below.
- Click **Finish**

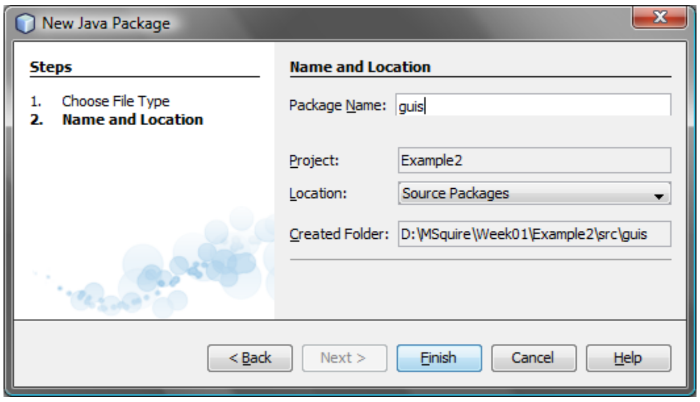


Project Name: Example2
Project Location: D:\MSquire\Week01
Project Folder: D:\MSquire\Week01\Example2
☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
☐ Create Main Class
☐ Set as Main Project

The project will be empty:



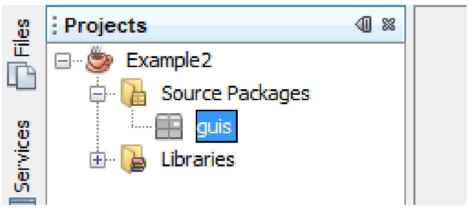
- Right-click the **Source Packages** node
- Select **New**
- Select **Java Package**
- Enter the package name **guis** (all lower case)
- Click **Finish**



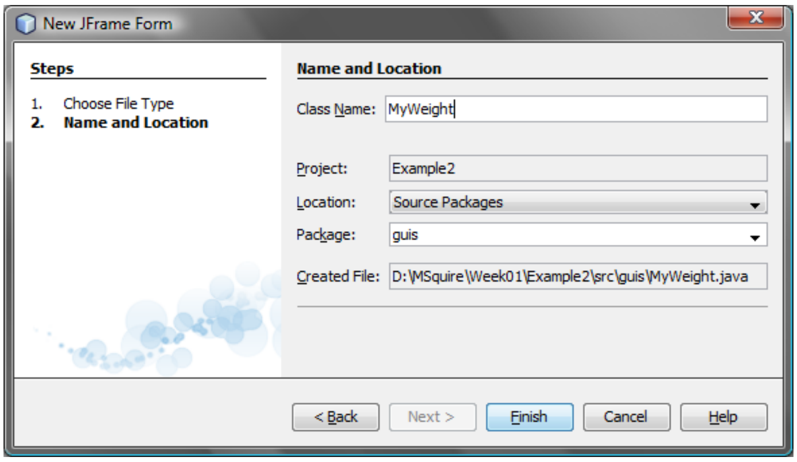
Package Name: guis
Project: Example2
Location: Source Packages
Created Folder: D:\MSquire\Week01\Example2\src\guis

The new package will have been created:

- Right-click the **guis** node

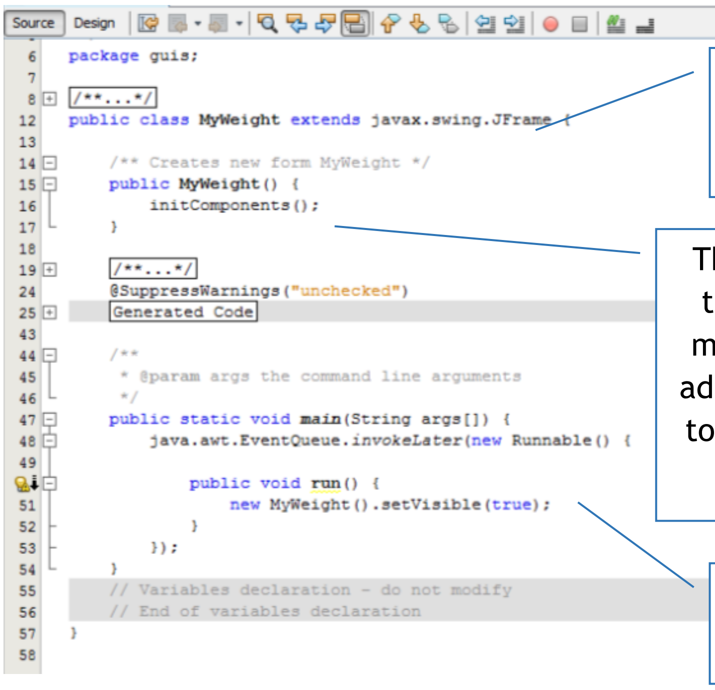


- Select **New**
- Select **JFrame Form**
- Enter the class name **MyWeight**
- Click **Finish**
- The designer is opened automatically - click the **Source** button now to view the source.



Class Name: MyWeight
Project: Example2
Location: Source Packages
Package: guis
Created File: D:\MSquire\Week01\Example2\src\guis\MyWeight.java

This is the generated source code (some comments have been collapsed so that they are not too obtrusive).



```
package guis;

/**...*/
public class MyWeight extends javax.swing.JFrame {

    /** Creates new form MyWeight */
    public MyWeight() {
        initComponents();
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new MyWeight().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    // End of variables declaration
}
```

The class is an extension of (inherits) the JFrame class that provides the basic GUI functionality.

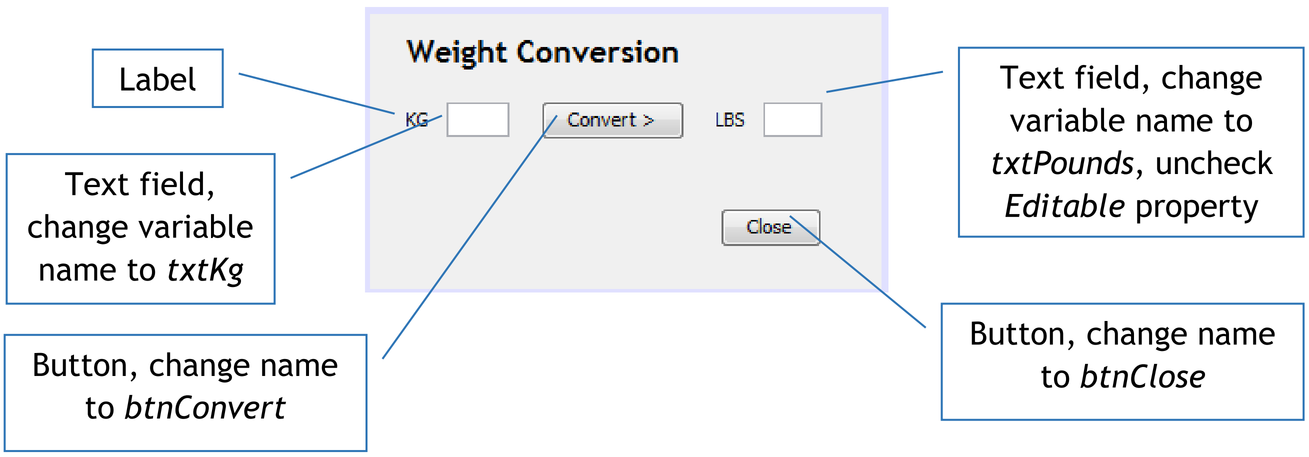
This is the constructor, which sets up the GUI. It calls the initComponents method to set up the window. We can add code to the constructor, but do not touch the initComponents methods nor the declaration of GUI objects.

This is the method that runs the application.

- Click the **Design** button.
- In the palette to the right find Label and click it.
- Click towards the top left of the form to insert a Label. *Note that the designer will try to arrange components by snapping them to the window borders, and/or other components.*
- Right-click the **Label** you have added.
- Select **Variable name** and change it to **lblTitle**
- Right-click the **Label** again.
- Select **Edit text**
- Change the text to **Weight Conversion**
- Right-click again.
- Select **Properties**
- Click the ... button alongside the **Font** property and select a suitable font for the title.
- Click **Close**.

- Now add further components as shown below.
- You can re-size the window if necessary by dragging borders.

Label



Label

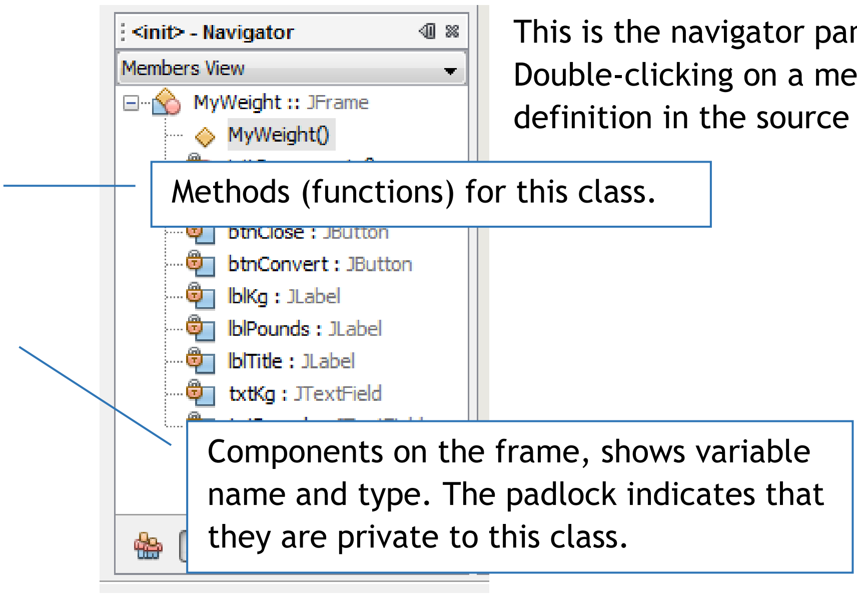
Text field, change variable name to *txtKg*

Button, change name to *btnConvert*

Text field, change variable name to *txtPounds*, uncheck *Editable* property

Button, change name to *btnClose*

- Click the **Source** button to return to the source code.
- *If you wish to see the code that adds the components, expand the 'generated code'. Note that you are not able to edit any of this. The declarations of the variables can be found at the bottom of the class. Again you can view this but not edit it.*

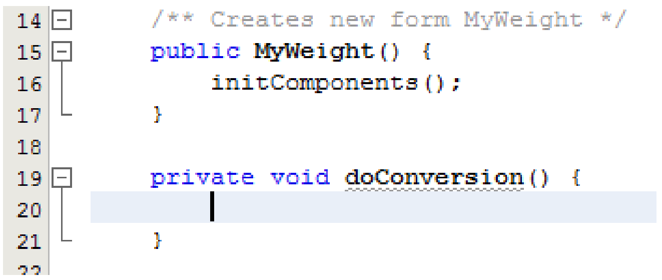


Methods (functions) for this class.

Components on the frame, shows variable name and type. The padlock indicates that they are private to this class.

This is the navigator pane showing the class members. Double-clicking on a member will take you to its definition in the source code.

- Below the **MyWeight** constructor method, add a new method as shown ⇨

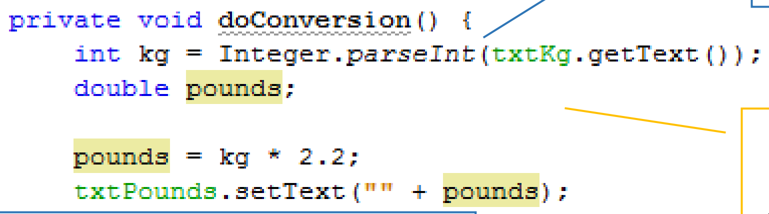


```
    /** Creates new form MyWeight */
    public MyWeight() {
        initComponents();
    }

    private void doConversion() {
        
    }
```

- In the method, add this code:

Integer.parseInt takes a String and converts it to an integer (as long as it is a valid integer!)



```
private void doConversion() {
    int kg = Integer.parseInt(txtKg.getText());
    double pounds;

    pounds = kg * 2.2;
    txtPounds.setText("" + pounds);
```

getText is the method used to retrieve text from a GUI component.

setText puts a String into a GUI component. Note the shorthand way of converting the double variable pounds to a String

- Click the **Design** button
- Right-click the **Convert >** button on your GUI design
- Select **Events > Action > actionPerformed**
- The event handler is added to the code, just add code like this:

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

- Save the project (**File > Save all** or

- Right-click the **MyWeight.java** node in the structure pane.
- Select **Run file**.

Your application should run. Try typing in some values with known results (e.g. 10) to check that the correct results are output.

5.2 A simple GUI exercise

Now follow the structure for 5.1 again, but this time make the application do a height conversion, as in the previous *MyHeight* example.

When you have finished close the project.

6 Classes and GUIs

This section will show how to create a simple class with a GUI to interact with it. Remember that in this module we are looking at **Object-Oriented Software Engineering**, and starting to develop an increasing degree of professionalism. This exercise is fairly straightforward and the use of a class may seem too complex here, but the key idea is to start simple, with one class and a simple GUI.

The example to be used is for a simple bank account.

6.1 New project *BankExample1*

Create a new project called BankExample1.

Hint:

Use File > New project, then select Java Application.

Enter the project name and remember to uncheck the boxes (Create Main Class and Set as Main Project).

Create a new package called bankentities.

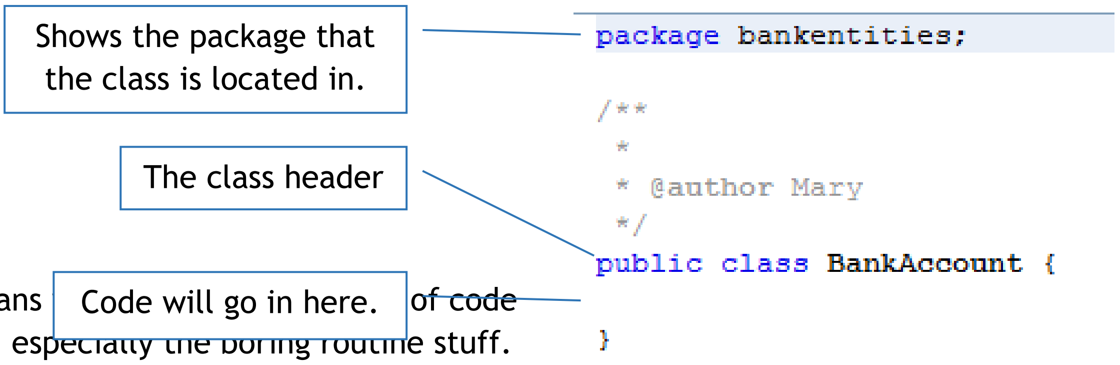
Hint:

Right-click the project in the structure pane and select New > Java Package.

Ensure you type the package name all in lower case.

6.2 The *BankAccount* class

- Right-click the **bankentities** package node in the structure pane.
- Select **New > Java Class**
- Enter **BankAccount** as the class name and click **Finish**.



```
package bankentities;

/**
 *
 * @author Mary
 */
public class BankAccount {

}
```

NetBeans [illegible] of code for us, especially the boring routine stuff. We will see how that's done later on, initially it will pay to type in the code for this class. We haven't done much on classes yet, so don't worry too much about what it's all about (but if you want, please do ask during the tutorials and we will explain).

6.2 Attributes

The first step is to declare **instance variables** to hold the data for the object's attributes.

Add this code in the class:

```
public class BankAccount {
    private double balance;
    private double overdraft;
    private String holder;
}
```

6.3 Constructor Method

The constructor is used on creating an object and initialises the instance variables. Add this code:

```
public class BankAccount {
    private int balance;
    private int overdraft;
    private String holder;

    public BankAccount(String holder) {
        this.holder = holder;
        this.overdraft = 500;
        this.balance = 100;
    }
}
```

Allow the account holder's name to be set by a parameter.

Set balance and overdraft to default initial values (holder must deposit 100.00 to open an account)

6.4 Method to deposit money

This method will accept a parameter containing the amount to deposit and will add that to the balance. Add this code:

```
public class BankAccount {
    private int balance;
    private int overdraft;
    private String holder;

    public BankAccount(String holder) {
        this.holder = holder;
        this.overdraft = 500;
        this.balance = 100;
    }

    public void depositMoney(int amount) {
        balance += amount;
    }
}
```

This is how you tell Java that no value will be returned.

This is an *assignment operator* (see Java Syntax Summary). It is equivalent to:
`balance = balance + amount;`

6.5 Method to withdraw money

Not quite as straightforward as the deposit method, this takes a parameter containing the amount to withdraw and checks that there are sufficient funds. If not, the method returns *false*, otherwise it deducts the amount from the balance and returns *true*.

Add this code below the **depositMoney** method and above the final **}** of the class:

```
public boolean withdrawMoney(int amount) {
    if ((balance + overdraft) < amount)
        return false;
    else {
        balance -= amount;
        return true;
    }
 }
```

This is how you tell Java that a value will be returned, and also what type it will be.

Another *assignment operator*.

This is how a value is returned from the method

6.6 Some utility methods

We are going to allow only the overdraft amount to be changed, and the balance and holder will be read only (except for the deposit/withdraw methods for the balance).

In Java we use a method starting with 'set' to change an attribute's value, and 'get' to retrieve a value. Methods are required for this because all our instance variables containing these values have been declared **private** (more on this later).

Add these methods below the **withdrawMoney** method and above the final **}** of the class:

```
public int getBalance() {
    return balance;
}

public String getHolder() {
    return holder;
}

public int getOverdraft() {
    return overdraft;
}

public void setOverdraft(int overdraft) {
    this.overdraft = overdraft;
}
```

That is it for the class, so save all files before we go on to the GUI.

6.7 Create the GUI

We need to separate our **entity** classes (those that represent data) from our **boundary** classes (those that provide an interface between user and data). Why?

- Right-click the **BankExample1** project node in the structure pane.
- Select **New > Java Package**
- Enter the name as **bankgui** and click **Finish**.

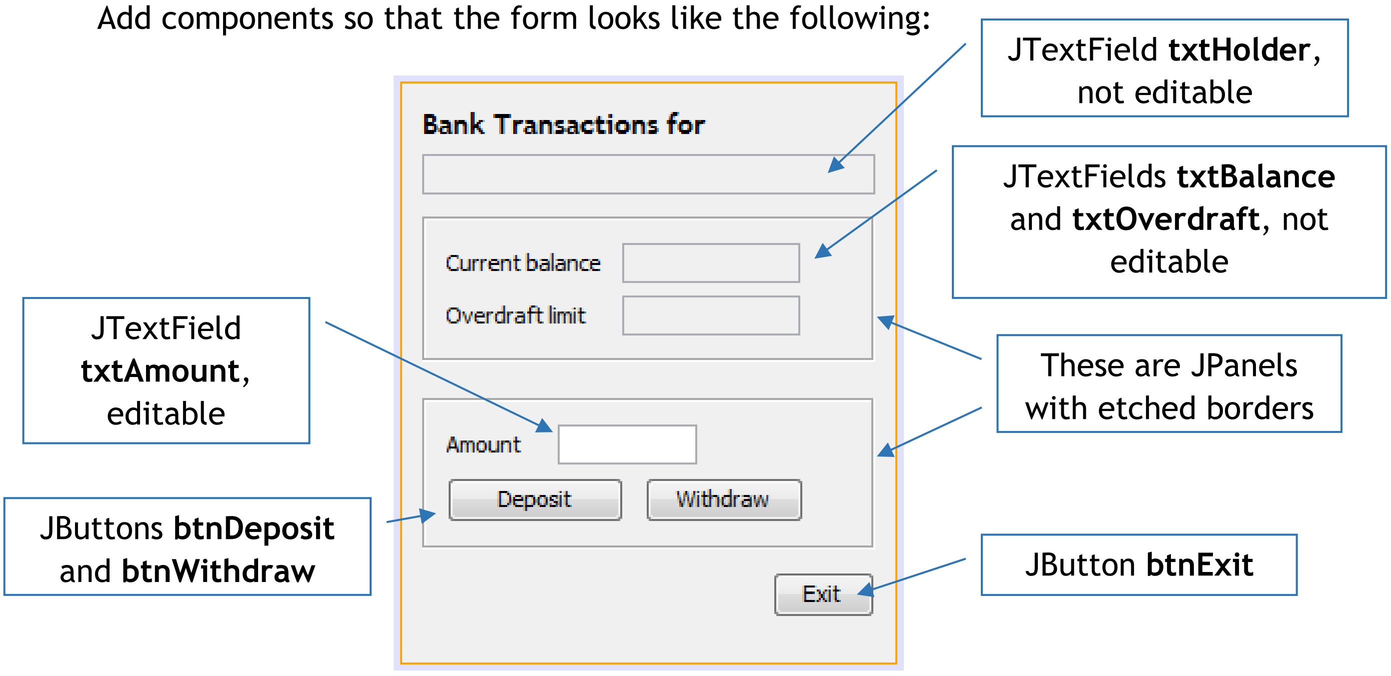
Now add a new **JFrameForm** to this package and call it **Banking**.

Hint:

Right-click the package in the structure pane

Select New > JFrame Form

Add components so that the form looks like the following:


JTextField **txtHolder**, not editable
JTextFields **txtBalance** and **txtOverdraft**, not editable
These are JPanels with etched borders
JTextField **txtAmount**, editable
JButtons **btnDeposit** and **btnWithdraw**
JButton **btnExit**
Bank Transactions for
Current balance
Overdraft limit
Amount
Deposit
Withdraw
Exit

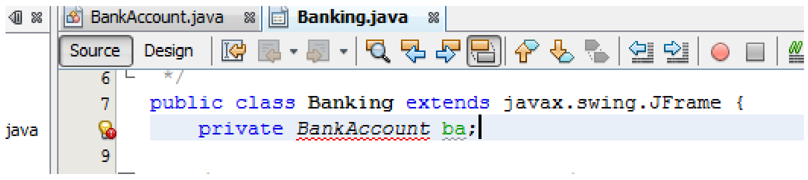
Save all files, then click **Source** to return to the source view.

6.8 Creating an object for the GUI

In the previous example, we just created the variables to hold the data within the relevant method. There was only one action required there and it was very simple.

The new GUI can deposit and withdraw money, so there are two actions. Each of these needs to be able to access the same bank account. So this time the variable to be used will be a **BankAccount** object, and because it will need to be accessed in various places in the GUI, the best place to declare it is as an attribute of the **Banking** class. This will mean the same object can be used in any method within the Banking class.

When you add the declaration in a moment, you will get a wiggly red line under BankAccount, like this:



```
    */
public class Banking extends javax.swing.JFrame {
    private BankAccount ba;
```

If you hover the cursor over **BankAccount** the message will be:

Cannot find symbol
  symbol: BankAccount
  location: class bankgui.Banking

It does not recognise the word *BankAccount* as it cannot find any definition of it within the class *Banking* within the package *bankgui*, which is perfectly correct! The class *BankAccount* is actually defined in a package called *bankentities*.

To overcome this error, all we have to do is tell it where the class can be found, which we do with an *import* statement.

So now you can set up the variable in the class as follows (note - some auto-generated comments have been removed for clarity):

```
package bankgui;

import bankentities.BankAccount;

public class Banking extends javax.swing.JFrame {
    private BankAccount ba;
```

The variable we have just declared contains the value *null* at the moment. The object itself will not be created until we explicitly do so.

In this example we will create the object when the form itself is created - in the constructor - so add this code:

```
public class Banking extends javax.swing.JFrame {
  private BankAccount ba;

  /** Creates new form Banking */
  public Banking() {
    initComponents();
    ba = new BankAccount("Jane Smith");
    txtBalance.setText("" + ba.getBalance());
    txtOverdraft.setText("" + ba.getOverdraft());
    txtHolder.setText(ba.getHolder());
  }
```

This creates a BankAccount object, puts the name we have supplied into the *holder* attribute, and initialises the other attributes as in the BankAccount constructor.

These statements get the current values from the bank account object and put them into the relevant fields on the GUI.

6.9 Methods for events

Below the form constructor, add this method to take the String value from the *amount* text field, convert it to an integer, deposit the amount in the *BankAccount* object, then refresh the text field that shows the balance and clear the amount text field.

```
private void doDeposit() {
    int amount = Integer.parseInt(txtAmount.getText());
    ba.depositMoney(amount);
    txtBalance.setText("" + ba.getBalance());
    txtAmount.setText("");
}
```

Add a similar method to do a withdrawal. *(For the time being, don't worry about the fact that you are not checking whether there are sufficient funds).*

Save all files at this point.

6.10 Adding the actual event handlers

- Click the **Design** button
- Right-click the **Deposit** button.
- Select **Events > Action > actionPerformed**
- The event handler is added to the code, just add code like this:

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

- Do the same for the **Withdraw** button.
- Add an event handler for the **Exit** button, with the code **System.exit(0);**
- Save all files again.

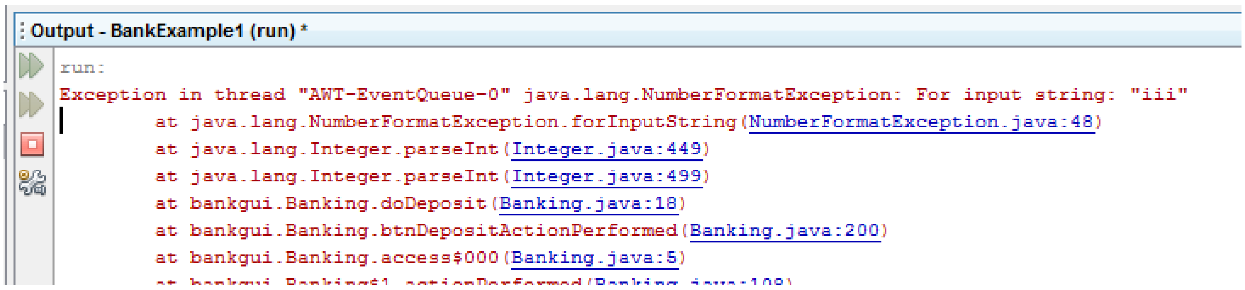
6.11 Try the program

- Right-click **Banking.java** in the structure pane.
- Select **Run file** and try out the application.

Try some withdrawals for which there are insufficient funds and see what happens.

Then try typing a non-numeric value into the amount text field and click **Deposit.**

Nothing appears to happen in the GUI, but you will see something like this in the NetBeans output window:



Output - BankExample1 (run) *

```
run:
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "iii"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:449)
        at java.lang.Integer.parseInt(Integer.java:499)
        at bankgui.Banking.doDeposit(Banking.java:18)
        at bankgui.Banking.btnDepositActionPerformed(Banking.java:200)
        at bankgui.Banking.access$000(Banking.java:5)
        at bankgui.Banking$1.actionPerformed(Banking.java:108)
```

This shows how a run-time error is reported. It is telling you that the Integer.parseInt function has detected non-numeric characters, and as a result has *thrown an exception*.

You can intercept this exception and give a more helpful error message. Modify the **doDeposit** method as follows:

```
private void doDeposit() {
    try {
        int amount = Integer.parseInt(txtAmount.getText());
        ba.depositMoney(amount);
        txtBalance.setText("" + ba.getBalance());
        txtAmount.setText("");
    }
    catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Please enter numbers only",
                "Error in amount", JOptionPane.ERROR_MESSAGE);
    }
}
```

Now try again with a non-numeric value - much better!

Update the **doWithdrawal** method in the same way.

6.12 Checking the withdrawal amount

The final task in this exercise is to show a message if the withdrawal amount is more than the funds available. The blocking of the withdrawal is already coded in the *withdraw* method in the *BankAccount* class, so all we have to do is check the result of the method - true if the transaction was OK, false otherwise.

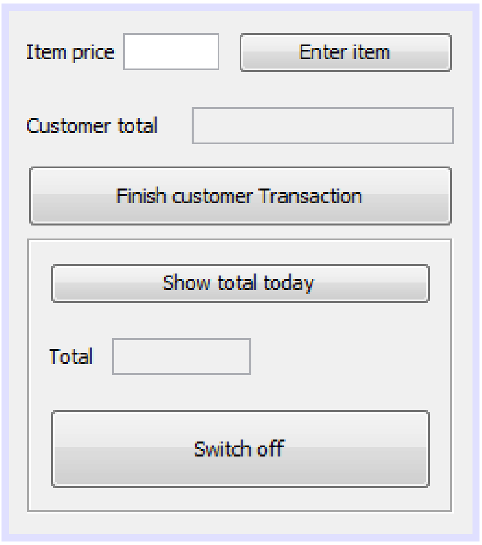
This is the new version:

```
private void doWithdrawal() {
  try {
    int amount = Integer.parseInt(txtAmount.getText());
    boolean fundsOK = ba.withdrawMoney(amount);
    if (fundsOK)
      txtBalance.setText("" + ba.getBalance());
    else
      JOptionPane.showMessageDialog(this,"Insufficient funds available\n"
                  + "withdrawal not actioned", "Insufficient funds",
                  JOptionPane.INFORMATION_MESSAGE);
      txtAmount.setText("");
  }
  catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Please enter numbers only",
            "Error in amount", JOptionPane.ERROR_MESSAGE);
  }
}
```

7 A final exercise (optional)

This exercise is a little more complicated than the previous one. It requires you to produce an application that models a cash till, where the cashier enters the prices of the goods (the price is added to the customer total and updated on the GUI), and then presses another button to complete, at which point the customer total is added to the till's daily total and the till display cleared ready for the next customer.

A suitable GUI layout is shown here ⇨.



- Add methods to provide the processing for the buttons, e.g.
  enterItem()
  finishCustomer()
  showDailyTotal()
- Add the event handlers for the buttons, using the methods above.