GUIs in Java – an overview & comparison to VB.net

RORY KELLY

15 APRIL 2016

About Me

1st year student – University of Southampton

Previous South Downs College student

Programming experience

- VB.net
- Java
- C#
- Python
- JavaScript
- Bash
- Web development - HTML/CSS/PHP
- etc.

VB.net and Visual Studio

VB.net

- Object-oriented, event-driven language
- Windows – some cross-platform implementation
- No technical standard
- Good first language

Visual Studio

- Main IDE for VB.net
- Also supports C++, C#, etc.
- Plug-in support only recently implemented (beyond Pro version)
- Lot of code generation (GUI – seen later!)

Java and Eclipse

Java

- Object-oriented language
- Not natively event-driven
- Multi-platform – JVM (WORA)
- Java Technical Standard
- Typical language taught in universities

Eclipse

- One of many IDEs for Java
- Plug-in support
- Open-source, non-profit

VB to Java quick reference table

Visual Basic .NET	Java
Sub-procedures and functions	Methods*
Inherits	extends
Imports	import
Dim *x* As *Class*	*Type x* = new *Type(…);*†
Dim *x* As *Primitive* (e.g. Integer)	*Primitive x* = value*;*†
Control	Component
Form	JFrame/JDialog
Namespaces	Packages
Ending statements – implicit with lines	; used at end of statement
Blocks of code	Braces define the block of code {}

Quick explanations

* To tell if a method in Java is a sub-procedure or a function, look at it's **return type**.

- `void` means the method is a sub-procedure – returns a void, i.e. nothing!
- Any other type (e.g. `int`, `boolean`, **`String`**, etc.) means the method is a function

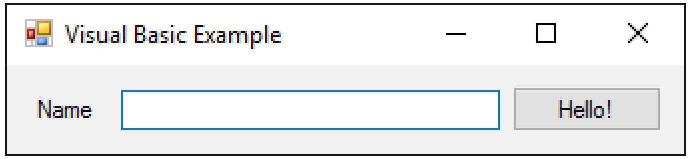
† Primitives are base types - lowest level of variables. All classes are built on these.

- `byte`/`short`/`int`/`long` – 8/16/32/64-bit signed two's complement integer, respectively
- `float`/`double` – 32/64-bit IEEE 754 floating point
- `boolean` – `true` or `false`
- `char` – single character

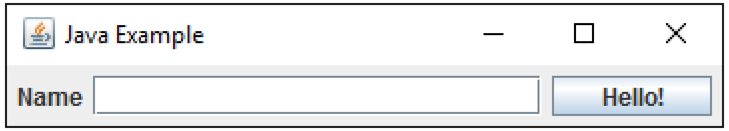
`Classes` are all other objects – built out of primitives, other classes, and methods

- Many different classes exist, native and in libraries. Your own classes can be made as well
- Primitives also have classes – wrapper classes (**`Integer`**, **`Character`**, **`Boolean`**, etc.)

GUI Example - quick comparison



- Native GUI implementation
- Three controls used – `Button`, `Label`, and `TextBox`
- Placed using precise co-ordinates and dimensions
- Only one line of code required to show dialog
- Most generation code is hidden from the programmer



- Java Swing and AWT (Abstract Window Toolkit)
- NOT native to Java – imports required
- Five “components” used – **`JFrame`**, **`JLabel`**, **`JTextField`**, **`JButton`**, and **`JOptionPane`**
- Placed and sized (generally automatically) with a **`LayoutManager`**
- Generation and listener code is all defined manually by the programmer

Source Code

Swing

- Java’s main GUI library
- Contains classes and methods for components, listeners, etc.
- Operates on the “Event Dispatch Thread” – thread safety
- Compared to VB.net, it’s a lot more work – no native drag-and-drop designer
  - Plug-ins do exist to add this functionality to the IDE
- Doing the code generation of VB yourselves - extremely versatile and customizable
- Must be imported in order to use (`import javax.swing.*`)

Components and their Properties

- A graphical object that can be interacted with by the user
- Instantiated like a variable
- Properties set using methods (e.g. `btnNew.setPreferredSize(x, y)`)
- Have to be added to a **`JPanel`** or **`JFrame`** to be visible
- **`JPanels`** also have to be attached to the **`JFrame`**, directly or through other panels

Layout Managers

- Ways that define the positions and sizes of Components within a `JFrame`/`JPanel`/`JDialog`

- Several available layout managers to choose from
  - `FlowLayout`
  - `BoxLayout`
  - `BorderLayout`
  - `GridLayout`
  - `null`
  - etc.

- Versatile, but can be fiddly

- Combine many to get a desirable GUI

Listeners

- Swing’s implementation of “events” and “event handlers”

- Uses anonymous classes in order to provide custom implementations

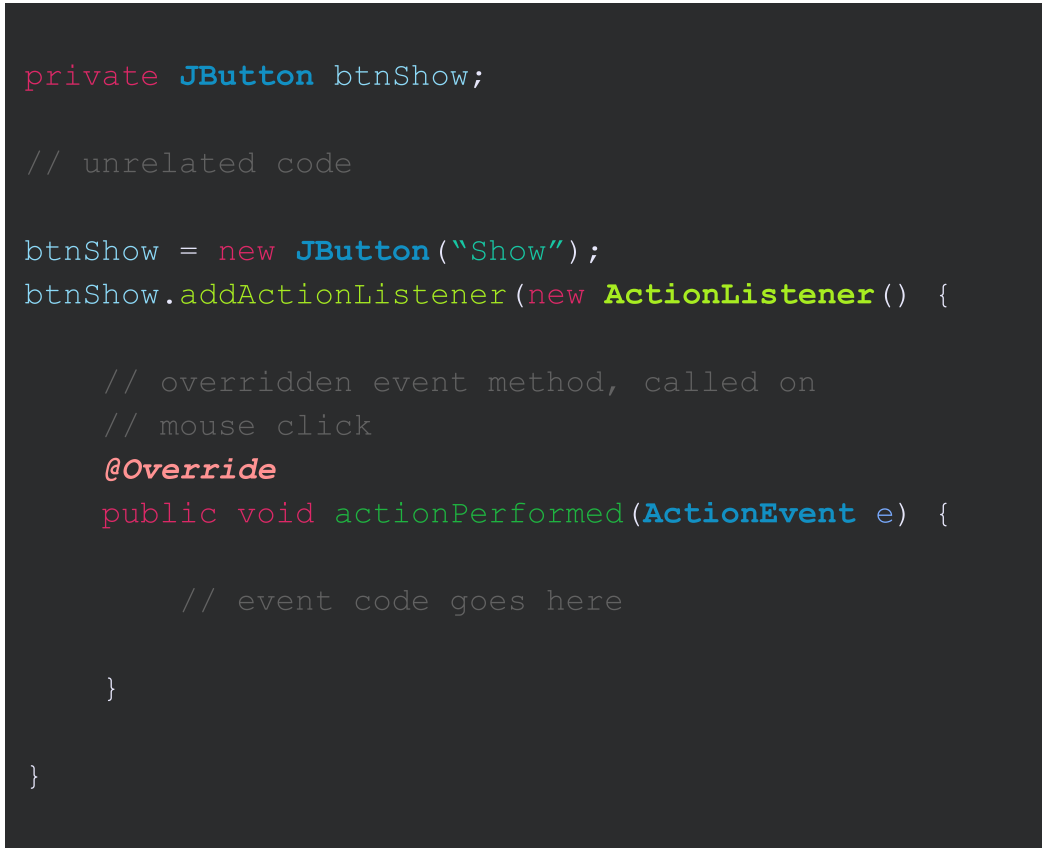
- Inner methods are overridden

- Different listeners for different controls and different events
  - `Mouse(Motion)Listener`
  - `ItemChangedListener`
  - `ActionListener`
  - etc.

- Another import required in order to use these (`import java.awt.event.*`)

Anonymous Classes

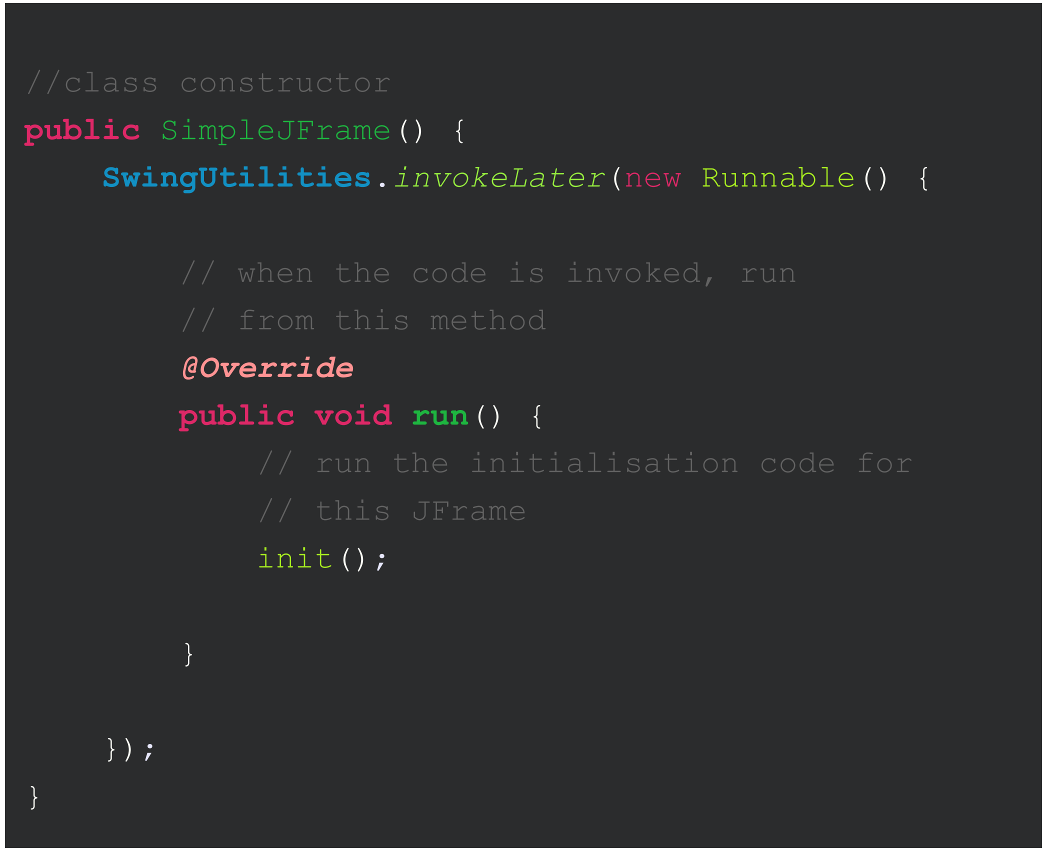
- Allows specific classes to be made without a whole new class to be created
- One big use of these include Swing listeners
- Most classes can be extended using anonymous classes
- They have access to any method/variable that is in scope to the enclosing class
- Implicit inheritance



```
private JButton btnShow;

// unrelated code

btnShow = new JButton(“Show”);
btnShow.addActionListener(new ActionListener() {

    // overridden event method, called on
    // mouse click
    @Override
    public void actionPerformed(ActionEvent e) {

        // event code goes here

    }

}
```

SwingUtilities.*invokeLater*(…);

- Swing is not thread-safe, particularly when updating GUIs
  - If multiple threads are concurrently working on components, there could be a clash
- In order to make Swing code safe, must use the method *`invokeLater`*`(…)` or *`invokeAndWait`*`(…)`
- Executes code within the Event Dispatch Thread
- Inner code is run like a thread – code must be within a Runnable class
  - Way of defining code to run within a “thread”
- Not 100% vital, but it is good practice, especially for multithreading apps



```
//class constructor
public SimpleJFrame() {
    SwingUtilities.invokeLater(new Runnable() {

        // when the code is invoked, run
        // from this method
        @Override
        public void run() {
            // run the initialisation code for
            // this JFrame
            init();

        }

    });
}
```