User Interface Programming

OOP/Java Primer
Step 2
Programming Java/Swing

- Basic items are classes (placed in packages).
- A class contains recipes (methods) for how the object works.
- One object should never manipulate another object’s data directly:
  - Communication through defined interfaces and methods.
- Program structure through class communication.
Methods

- Methods are defined within classes
- Can perform calculations and manipulate data
- Methods are in themselves "procedural definitions"

```java
public int add(int x, int y) {
    int temp = x + y;
    return temp;
}
```
Methods

- can call other methods
  - within the class:
    - int temp = addAll(integerList); \local call
    - int temp = this.addAll(integerList); \local call using this
  - or in other classes:
    - Position pos = cube.getPosition();

- this
  - reserved keyword, points to the current instance of the class
  - can be used to clarify code
Constructors

- Special methods that are used to initialize an instance of a class
  - has the name of the class and returns nothing
- Cannot be called explicitly
public class Point {
    public int x = 0;
    public int y = 0;
    //constructor
    public Point(int a, int b) {
        x = a;
        y = b;
    }
}

Note! The second usage is not really recommended, but should be seen as an illustration of the principles.
Variables

- public int num; \ integer that is public
- private int num; \ integer that is local
- static final int num; \ constant global value
- etc.

*Important! Variables are strongly typed. Conversion between two types can be done by explicit call to method, e.g.*

```
Integer.toString(integer); or through casting:
int temp = (int) dateString;
```
Swing

- OOP-toolkit for Java GUIs

- Very good example of the OO-principle

- All GUI-components are grouped in a hierarchical structure

- Simplifies the construction of interfaces
Swing hierarchy

JFrame and JDialog are special cases, and inherit from AWT Frame, and AWT Dialog
AWT

- AWT is a more primitive toolkit, which was developed before swing

- Used for Applets

- Swing is better in most cases

- Some awt-classes are still used heavily
Hierarchic structures

- A GUI is basically a hierarchic construction

- Frame (window) contains
- Panel contains
- Panels and Buttons…
- Etc.
JFrame

- A Basic GUI class: A window with decorations
  - Borders, titlebar, etc.

- Inherits from the AWT Frame Class

- Never work in the window itself!
  - Container c = getContentPane();
JPanel

- Content holder
  - one of the most important components in Swing

- Can’t be used as standalone
  - Needs a container, e.g.,
    - a JFrame
    - A JPanel
    - or other component that inherits from Container!
public class TestFrame extends JFrame {

    JPanel panel = new JPanel();

    public TestFrame() {
        super();  \Init superclass
        setSize(new Dimension(400,400));

        panel.setBackground(Color.red);
        this.getContentPane().add(panel);

        this.setVisible(true);
    }
}
Start of program

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        new TestFrame();
    }

}
Using inheritance

- JPanel – a versatile component in the UI
- Tempting to let a new object inherit from JPanel
- Sometimes OK, but consider also a "contains"-relation
Example:

class Person extends JPanel {
   private String namn;
   private String adress;
   private JButton okButton = new JButton("OK");

   public Person (String n) {
      this.namn = n;
      this.setBackground(Color.white);
      this.add(okButton);
   }
}

Special Dialogues

- Predefined dialogues
  - Perfect for simple applications
  - Inflexible structure (difficult to change)

- Simple to use
JOptionPane

- `showInputDialog()`
- `showMessageDialog()`
- `showConfirmDialog()`
- `showOptionDialog()`
JOptionPane

- Class Methods!
- Several alternatives
- Try it out!

  - JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.");
LayoutManagers

- FlowLayout
- GridLayout
- BorderLayout
- GridBagLayout
BorderLayout
Working with GUIs

The sketch:
The result

![Dialog box for selecting a ball type with options: liten, mellan, stor, anpassa]
Practical realisation

![Image of a window with options: liten, mellan, stor, anpassa]
Composition

- Window – JFrame, JPanel jp1
  - BorderLayout
- JPanel jp2 – South – standardbuttons
  - FlowLayout
- JPanel jo3 – East – radio buttons
  - BoxLayout (vertical)
- JPanel jp4 – North – Text
- JPanel jp5 – Center – Text
Events

- It is necessary to catch events generated by the user:
  - mouse clicks, keyboard actions, etc.

- Listeners are interfaces, that support interaction
  - Defines methods that will be called when something happens
public class Person extends JPanel implements ActionListener {
    private JButton okButton = new JButton("OK");

    public Person (String n) {
        name = n;
        setBackground(Color.white);
        add(okButton);
        okButton.addActionListener(this);
    }

}
Doing what?

```java
public void actionPerformed(ActionEvent e) {
    if ( e.getSource() == okButton ) {
        // if it is the right button, then
        // do some great stuff!

        :
        :
        :
    }
}
```
Problem!

- Interfaces require ALL methods to be implemented
- Even if we only need one!
- E.g. WindowsListener has a large number of methods to implement

- Lots of extra work
Simple Solution!

- For all events there are Adapter Classes
- XXXListener ➔ XXXAdapter

- All methods are predefined, and you override those that you need to define!
KeyBoard Events

- Are sent to the Component currently in focus
  - Marked

- The active window gets
  - KeyPressed
  - KeyReleased
  - KeyPushed
Drawing in Java!

- JComponent is the most abstract component in Swing
- Contains one important method
  - (+ many more, of course)
- Provides the basic behaviour for all (graphic) Swing components
  - Contains the graphic method
  - paintComponent(Graphics g);
    - ( in AWT it is paint(Graphics g); )
Graphics in Java

- Painting on a "surface"
- Two main steps
  - Definition of what is to be painted
  - Drawing the definitions on the surface

- This is an IMPORTANT property of drawing in Java!
  - Will look at it more in much more detail in GuiDev2
Graphics

- Graphics is a class that provides a toolbox for drawing
- Is NOT created by the programmer

- It is more or less an environment, that defines the "sheet" that will be painted on in paintComponent(Graphics G).
Smooth graphics

- It is Java that decides when to paint and not
  - Some updates are automagic (when windows move or resize, e.g.)

- We can ask Java for a repaint();
  - (AWT paint(); )

- We never call paintComponent(g); in our programs!
Practise exercise

- Create a small program.
- The program should draw a JFrame.
- The JFrame should have a JPanel with a BorderLayout
- In that JPanel you put:
  - A JPanel in the centre to draw on.
  - An appropriate number of buttons in the lower border (a JPanel with a FlowLayout, e.g.)
  - Any other controls you may like
Practise exercise, part 2

- Add ActionListeners to the button, so that when you click on a Button, something gets painted on the JPanel in the centre. Try to make it into a small random paint application.

- You should be able to exit the application with one button!
Get your tools for the mind too:

- The Java API (with Swing)
  - http://download.oracle.com/javase/1.5.0/docs/api/
  - The documentation of Java. Here you can find all you need to write any program possible. Sometimes the information is a bit hard to get, so you should also get
- A Java Swing tutorial or two
- Maybe a first Java tutorial could be useful?
Bookmark these!

- Even if you don’t want to go through these now

- They are good reference sources and sources of inspiration for solutions

- It is OK to use pieces of code you find here
  - With due references of course!!!
  - And unless there is some limitations on the site!