Notes on Java Programming
User Interfaces
IST JAV 2012

Java graphical user interfaces rely on the following concepts :

1) A graphical user interface description

A graphical user interface relies on three kind of components :

- Containers that contain either other containers or atomic components
  - The main containers are the Frame and the Panel.
- Atomic component that shows something to the user
  - For instance a Button is an atomic component
- A layout manager that places components (atomic or container) within a container.
  - The BorderLayout uses east, north, west... placement directives, the FlowLayout place components on after the other. Other Layout managers are available in standard.

For historical reasons Java maintains two graphical related packages. The oldest java.awt contains graphical elements, layout managers and event management mecanism. The more recent javax.swing package contains efficient graphical elements and some new layout managers, but the eventing system is still the original java.awt.event package.
An example of graphical user interface:

```
package me;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JTextArea;
import java.awt.BorderLayout;

public class Test {
  public static void main(String [] arg) {
    JFrame j = new JFrame("Hello");

    JPanel north = new JPanel();
    JButton jbut = new JButton("Hello");
    north.add(jbut);

    JPanel center = new JPanel();
    JTextArea jt = new JTextArea("Hello");
    center.add(jt);

    j.add(north, BorderLayout.NORTH);
    j.add(center, BorderLayout.CENTER);

    j.setSize(100,100);
    j.setVisible(true);
  }
}
```

A graphical user interface is a tree of containers and atomic elements. The previous structure is structure like this. As the header expresses it is composed of :
2 container classes: JFrame and JPanel
2 atomic elements: JButton, JTextArea
We never mix graphical elements from two libraries. Here they are all prefixed by the javax.swing package name.
2 layout managers : BorderLayout and a FlowLayout. These two layout manager are not explicitly expressed because they are automatically associated to a container. So when one creates a JFrame it is automatically associated to a BorderLayout manager object, and when one creates a JPanel it is automatically associated to a FlowLayout manager object. I show now a possible description of the tree structure. Each line as the following grammar. <attrName> :: <ClassName> [prop1, prop2...] where attrName represents the attribute name local to the method, ClassName the name of the class, a list of property values.

```
j :: javax.swing.JFrame [visible, (100,100), borderLayout]
     north :: javax.swing.JPanel [north, flowLayout]
          jbut :: javax.swing.JButton [“Hello”]
     center :: javax.swing.JPanel [center, flowLayout]
          jt :: javax.swing.JTextArea [“Hello”]
```

Such a graphical user interface works, but does not react to user actions. Let's present the eventing mechanism.

2) An eventing mechanism

An asynchronous eventing mechanism must be designed for managing random user events. Many eventing mechanism are based on a publish/subscribe infrastructure. In our case, each graphical component may publish event type and some object may subscribe to these kind of event.

For the beginning, we know that each atomic graphical elements generates a standard event called the ActionEvent. When clicked a button generates that event, when hit <return> key in TextArea, or in a ComboList... As soon as we know that a specific component generates a event of class FooEvent, we know that a class that wishes to manage this event must comply with a FooListener behavior. The FooListener behavior is captured with a Java interface. The system will call methods from this interface as a reaction to user generated events.

The following lines are added to the previous user interface to react to two kind of generated events. The Action and the Focus events that any atomic component may generate. In the code only specific components will have their events propagated as soon as they are subscribed.

```
package me;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JTextArea;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.FocuseListener;
public class Test implements java.awt.event.ActionListener,
                             java.awt.event.FocusListener {
  public static void main(String [] arg) {
    Test t =new Test(); /* We need a specific object to handle events */
    JFrame j = new JFrame("Hello");
    JPanel north = new JPanel();
    JButton jbut = new JButton("Hello");
    jbut.addActionListener(t); /* Listener subscription to component event */
    jbut.addFocusListener(t);

    north.add(jbut);
    JPanel center = new JPanel();
    JTextArea jt = new JTextArea("Hello");
    center.add(jt);
    j.add(north, BorderLayout.NORTH);
    j.add(center, BorderLayout.CENTER);
    j.setSize(100,100);
    j.setVisible(true);
  }
  //Reactive methods, brought by interfaces.
  //The event name class corresponds to the kind of event
  public void actionPerformed(ActionEvent e) {
    System.out.println("Something happened");
  }
  public void focusLost(FocusEvent e) {
    System.out.println("Something focus");
  }
  public void focusGained(FocusEvent e) {
    System.out.println("Something gain focus");
  }
}
```