Programación e Interfaz de usuario en Java – Programming Java Graphical User Interface

Senior Lecturer Outi Grotenfelt

Helsinki Metropolia University of Applied Sciences

Contents

1. Idea

- How to learn the basic idea of graphical user interface and it's programming

Helsinki Metropolia University of Applied Sciences

1. Idea: Graphical User Interface

- Build with GUI-components
  - Buttons
  - Text fields
  - Check boxes
  - Containers
  - ...
- GUI program is event driven
  - Events might be
  - Pushing a button
  - Mouse click
  - Mouse drag

1. Idea: Graphical User Interface

- An event driven program is listening
  - Events react when something happens on the screen
  - When event occurs listeners make the needed functionality in methods

1. Idea: Swing

- Package **`javax.swing`** contains a GUI library
  - components are programmed
  - Based on the old AWT-GUI-library

- Specially in the beginning it is important to use on the side API-documentation

2. Model-View-Controller-model

- Graphical User Interface and application logic should be kept away from each other
  - Makes the program better modular
  - For example GUI is possible to change without working anything with the application logic

- MVC (model-view-controller) is a planning model working with the previous principle

2. MVC-model

- Model implements application logic
- View implements GUI
- Controller controls and updates model and GUI
  - Checks event handling further

2. MVC-model and Java

- Every class belongs to one of the three components

- Most simple program contains 3 classes:
  - One class for application logic (Model)
  - One class for GUI (View)
  - One class for controlling (Controller)

```
public class MyModel {
    // instance variables to save the data
    public MyModel(){
      // methods to initialize values for variables
    }

    // methods for changing or seeking data
}
```

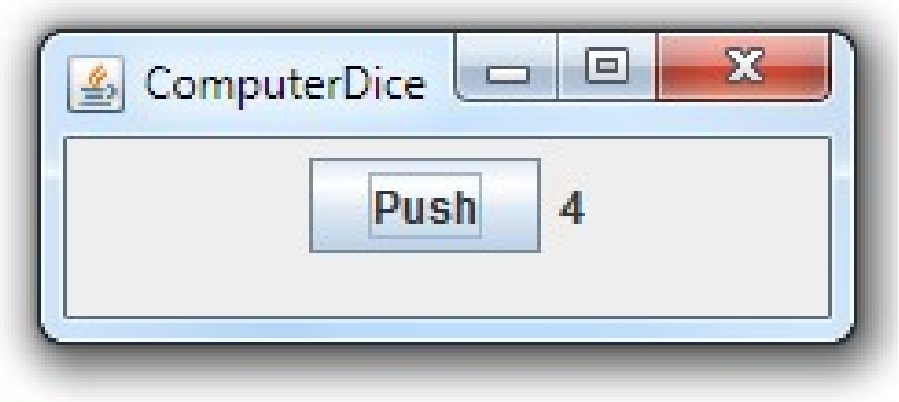
```
public class MyView {
    private MyController oneController;
    // swing-components as instance variables

    public MyView(){
       // building or initiateing the interface
    }
    public void registerMyController(MyController oneController){
        this.oneController = oneController;
    }
    // methods r
    // and handl
}
```

```
public class MyController {
    private MyModel mod;
    private MyView vie;

    public MyController(MyModel mod, MyView vie){
        this.mod = mod;
        this.vie = vie;
    }

    public static void main(String[] args){
        MyModel mod = new MyModel();
        MyView vie = new MyView();
        MyController cont = new MyController(mod,vie);
        vie.registerMyController(cont);
    }
    // methods needed for handling the events or
    // updating the model
}
```

Helsinki Metropolia University of Applied Sciences

3. The first program

- Program a GUI dice according to MVC-model:
  - Model
  - GUI
  - Controller
  - Listener



```
public class Dice {
  public int givePip() {
    return 1 + (int) (Math.random() * 6);
  }
}
```

```
import javax.swing.*;
import java.awt.event.*;

public clas
    private
    private
    private
    private

    public
        ini
    }

    public
        thi
    }
```

```
    private void initComponents() {
        setTitle("ComputerDice");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        myContentPanel = new JPanel();
        myCastButton = new JButton("Push");
        myResult = new JLabel("0");
        myContentPanel.add(myCastButton);
        myContentPanel.add(myResult);
        setContentPane(myContentPanel);
        myCastButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                cont.myCast();
            }
        });

        setSize(180, 80);
        setVisible(true);
    }

    public void putResult(int points) {
        myResult.setText(String.valueOf(points));
    }
}
```

Helsinki Metropolia University of Applied Sciences

```
public class DiceController {
    private Dice model;
  private DiceGUI myView;

  public DiceController(Dice model, DiceGUI myView) {
    this.model = model;
    this.myView = myView;
  }
  public static void main(String[] args) {
    Dice myDice = new Dice();
    DiceGUI myView = new DiceGUI();
    DiceController cont = new DiceController(myDice, myView);
    myView.registerController(cont);
  }
  public void myCast() {
    myView.putResult(model.givePip());
  }
}
```

3. How this functions

- In `main`-method we initialize model, GUI & controller
- Knowledge of the controller is delivered to the model (`registerController()`)
- GUI is build in method `initComponents()`
  - Building is ready when `setVisible(true)`-method is called – it makes GUI to become visible. No other building commands are aloud after that.
- Button `CastButton` is given a anonymous inner class to listen the event.
- `ActionPerformed()` is called when the button is pushed

3. How this functions

- This method gives the control to the controller, that asks dice pip and returns it to GUI
  - Event listener functions through interface `ActionListener` that gives the actual code for method `actionPerformed()`
- Main window is interited from `JFrame`-class
  - By calling `setDefaultCloseOperation(EXIT_ON_CLOSE);` the execution is ended when the window is closed
- Contents of the window are collected to `JPane`-holder with `add`-method
- The method call `setContentPane(myContentPanel)` changes the filled `JFrame` holder

4. Event handling

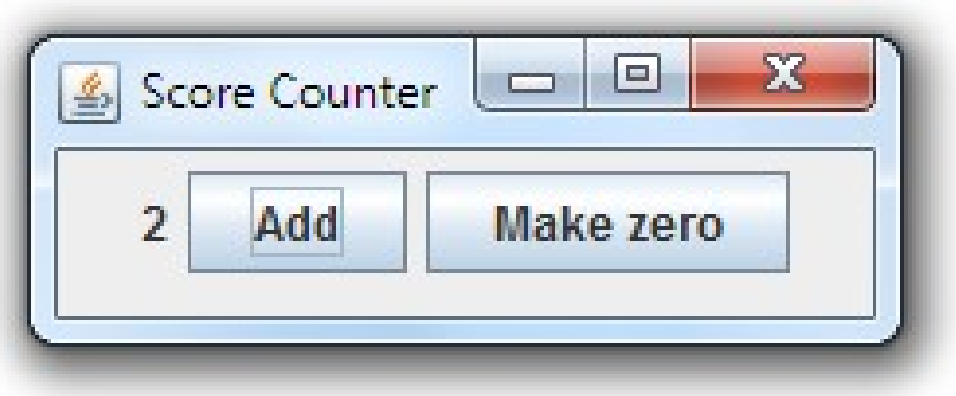
- GUI program is event guided
- Each event comes from some event source
- Events make a event object
  - To a button `ActionEvent`
- Event listener class controls what/how the application reacts
  - With `ActionListener` interface: method `actionPerformed`
  - Listener class is registered to be thet components listener
  - Listener class is often coded as an anonymous inner class calling controller not depending on Swing-class

4. Event handling, button listening

- Create Model-, Controller- & View-classes
  - View-class contains functionality to build the interface
- Each button are given an anonymous inner class introduced as `ActionEvent` handler
  - Write method `ActionPerformed();` functionality needed to happer after pushing the button is programmed here
  - In practice you call for the controller method dealing with listener
  - That method does not contain any `Swing`-components
- Each listener is registerd to a button/component
  - `addActionListener()`-method from class `JButton` is called

4. Event handling, example

- Two button listening can be used through two anonymous inner classes



4. Event handling, example, Model

```
public class ScoreCounter {
  private int score;

  public void increase() {
    score++;
  }

  public void init() {
    score = 0;
  }

  public int getScore() {
    return score;
  }
}
```

```
public class ScoreCounterController {
    private ScoreCounter scoreCounter; // malli
  private ScoreCounterGUI scoreCounterGUI; // näkymä

  public ScoreCounterController(ScoreCounter m, ScoreCounterGUI w)
{
    scoreCounter = m;
    scoreCounterGUI = w;
  }

  public static void main(String[] args) {

    ScoreCounter scoreCounter= new ScoreCounter ();
    ScoreCounterGUI scoreCounterGUI = new ScoreCounterGUI();
    ScoreCounterController cont =
      new ScoreCounterController(scoreCounter, scoreCounterGUI);
    scoreCounterGUI.registerController(cont);
    scoreCounterGUI.updateScore(scoreCounter.getScore());
  }
```

```
    public void bumbUp() {
      scoreCounter.increase();
      scoreCounterGUI.updateScore(scoreCounter.getScore());
    }

    public void zero() {
      scoreCounter.init();
      scoreCounterGUI.updateScore(scoreCounter.getScore());
    }
}
```

Helsinki Metropolia University of Applied Sciences

```
import javax.swing.*;
import java.awt.event.*;

public class ScoreCounterGUI extends JFrame{
    private

  private JI
  private JP

  public Sco
    initComp
  }
```

```
  private void initComponents() {
    setTitle("Score Counter");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JPanel myContentPane = new JPanel();
    myLabel = new JLabel();
    addButton = new JButton("Add");
    zeroButton = new JButton("Make zero");
    myContentPane.add(myLabel);
    myContentPane.add(addButton);
    myContentPane.add(zeroButton);
    setContentPane(myContentPane);

    addButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        count.bumbUp();
      }
    });
    zeroB
      pub
        c
      }
    });

    pack(
    setVi
  }
```

```
  public void updateScore(int score) {
    myLabel.setText(String.valueOf(score));
    pack();
  }

  public void registerController(ScoreCounterController
count) {
    this.count = count;
  }
}
```

4. Event handling, other possibilities

- In different classes
- In named inner classes
  - An own class for each button/component
  - A common inner class for all buttons/components
  - Event source with some function command
  - A button can get a String with `setActionCommand()`-method
  - A String can be asked from a handling object with `getActionCommand()`-method

4. Event handling, other possibilities

```
...
    ActionListener al1 = new IncreaseListener();
    ActionListener al2 = new ZeroListener();

    addButton.addActionListener(al1);
    zeroButton.addActionListener(al2);
...
```

```
class IncreaseListener implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    cont.bumbUp();
  }
}

class ZeroListener implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    cont.zero();
  }
}
```

4. Event handling, other possibilities

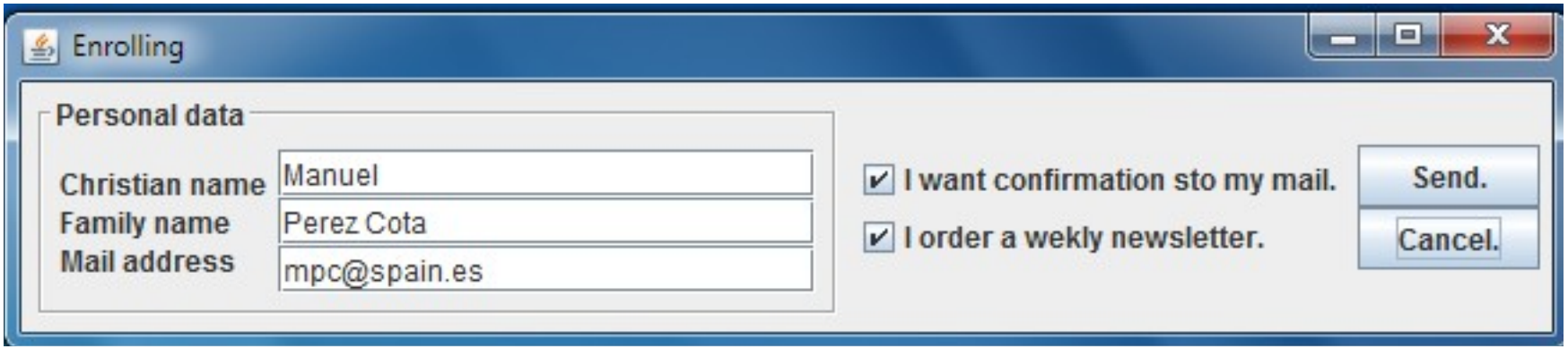
```
...
    ActionListener myListener = new myButtonListener();

    addButton.setActionCommand(”addNum");
    addButton.addActionListener(myListener);
    zeroButton.setActionCommand(”makeZero");
    zeroButton.addActionListener(myListener);
...
```

```
class myButtonListener implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().compareTo(“addNum")==0){
       cont.bumbUp();
    }
    else if (e.getActionCommand().compareTo(“makeZero")==0){
       cont.zero();
    }
}
```

5. GUI Components, Containers

- Containers bind together GUI components
- Containers can be placen inside each other (several levels)

5. GUI Components, Containers




```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
```

5. GUI Components, Containers

```
public class EnrolGUI extends JFrame {

  private JPanel mainPanel, personalPanel,
    explainPanel, fieldPanel, extraPanel,
    buttonPanel;
  private JTextField christianNameField, familyNameField,
    mailAddressField;
  private JLabel definition1, definition2, definition3;
  private JCheckBox choise1, choise2;
  private JButton button1, button2;
  private Border myBorder;
  private TitledBorder personalBorder;
```

5. GUI Components, Containers

```
public EnrolGUI() {
    setTitle("Enrolling");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    mainPanel = new JPanel();
    personalPanel = new JPanel(new FlowLayout());
    explainPanel = new JPanel(new GridLayout(3, 2));
    fieldPanel = new JPanel(new GridLayout(3, 2));
    extraPanel = new JPanel(new GridLayout(2, 1));
    buttonPanel = new JPanel(new GridLayout(2, 1));

    christianNameField = new JTextField(12);
    familyNameField = new JTextField(12);
    mailAddressField = new JTextField(20);
    fieldPanel.add(christianNameField);
    fieldPanel.add(familyNameField);
    fieldPanel.add(mailAddressField);
```

5. GUI Components, Containers

```
explainPanel.add(new JLabel("Christian name"));
explainPanel.add(new JLabel("Family name"));
explainPanel.add(new JLabel("Mail address"));

personalPanel.add(explainPanel);
personalPanel.add(fieldPanel);
myBorder = BorderFactory.createEtchedBorder();
personalBorder = BorderFactory.createTitledBorder(
  personalBorder, "Personal data");
personalPanel.setBorder(personalBorder);

choise1 = new JCheckBox("I want confirmation" +
  " sto my mail.");
choise2 = new JCheckBox("I order a wekly" +
  " newsletter.");
extraPanel.add(choise1);
extraPanel.add(choise2);
```

```
    button1 = new JButton("Send.");
    button2 = new JButton("Cancel.");
    buttonPanel.add(button1);
    buttonPanel.add(button2);

    mainPanel.add(personalPanel);
    mainPanel.add(extraPanel);
    mainPanel.add(buttonPanel);

    setContentPane(mainPanel);
    pack();
    setVisible(true);

  }

  public static void main(String[] args) {
    new EnrolGUI();
  }
}
```

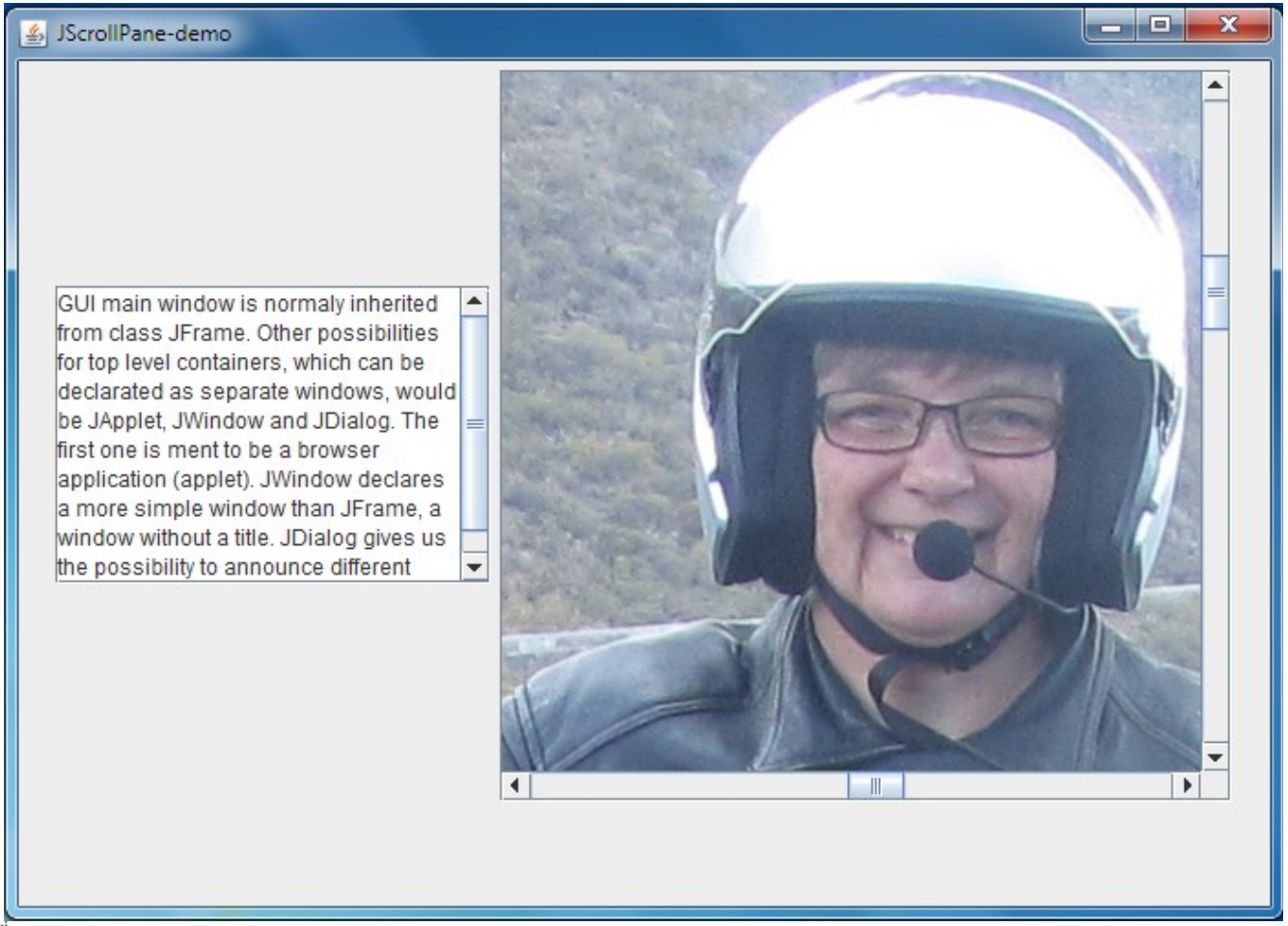
5. GUI Components, Containers

- **`JFrame`** is normaly the top level container
  - Other possibilities: Japplet, Jdialog, Jwindow

- **`JPanel`** is the basic container on lower level
  - Components added with `add()`-method

- Other containers
  - `JScrollPane`
  - `JTabbedPane`
  - `JSplitPane`

5. GUI Components, Containers, Example



5. GUI Components, Containers, Example

```
import javax.swing.*;
import java.awt.*;

public class JScrollPaneGUI extends JFrame {

  private JPanel inputPanel;
  private JScrollPane textPanel,  picPanel;
  private JTextArea textArea;
  private ImageIcon myPic;

  public JScrollPaneGUI() {

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setTitle("JScrollPane-demo");
    inputPanel = new JPanel();

    String teksti = "GUI main window is normaly inherited from class
JFrame. Other possibilities for top level containers, which can be
declarated as separate windows, would be JApplet, JWindow and
JDialog. The first one is ment to be a browser application (applet).
JWindow declares a more simple window than JFrame, a window without
a title. JDialog gives us the possibility to announce different
messages to the user.";
```

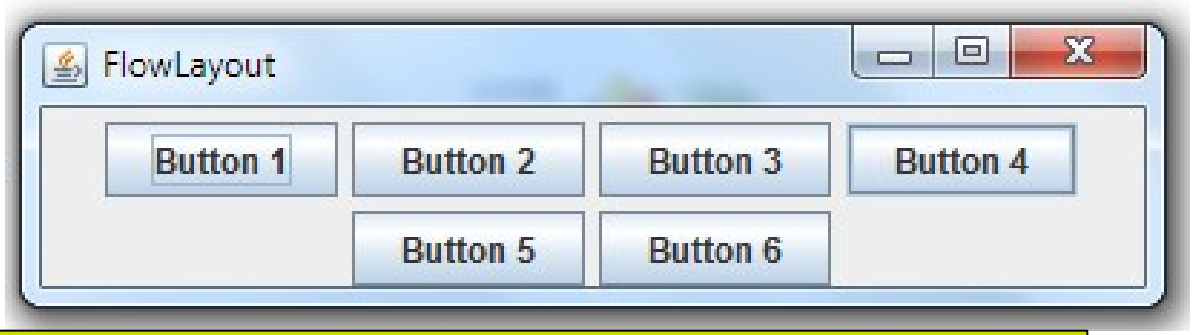
5. GUI Components, Containers, Example

```
textArea = new JTextArea(teksti, 10, 20);
    textArea.setLineWrap(true);
    textArea.setEditable(false);
    textArea.setWrapStyleWord(true);
    textPanel = new JScrollPane(textArea);

    // aseta alla oleva polku osoittamaan kuvatiedostoon
    myPic = new ImageIcon("C:\\05Kuvat\\201202\\IMG_0257.jpg");
    picPanel = new JScrollPane(new JLabel(myPic));
    picPanel.setPreferredSize(new Dimension(400, 400));
    inputPanel.add(textPanel);
    inputPanel.add(picPanel);
    setContentPane(inputPanel);
    setSize(700, 500);
    setVisible(true);
  }

  public static void main(String[] args) {
    new JScrollPaneGUI();
  }
}
```

5. GUI Components, Layouts

- a layout decides the positions for different components
- Three different examples:
  - `FlowLayout`
    - From left to right, up to down
  - `GridLayout`
    - A table
  - `BorderLayout`
    - Central area & north, west, south & east
- If anything else is not introduced, `FlowLayout` will be used

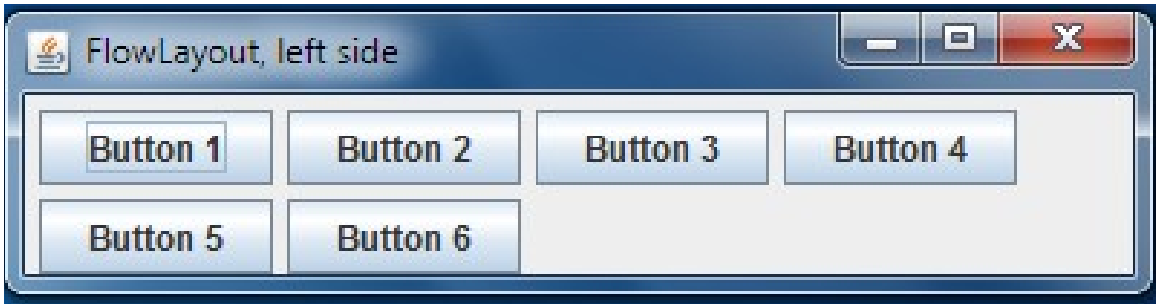
5. GUI Components, Layouts, example




```
import javax.swing.*;
import java.awt.*;

public class MyLayouts extends JFrame {
    private JButton buttonTable[] = new JButton[6];
    private JPanel myPanel;

    public MyLayouts(){
        setTitle("FlowLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        for (int i=0; i<buttonTable.
            buttonTable[i] = new JBu
        myPanel = new JPanel(new Flo
        for (int i=0; i<buttonTable.
            myPanel.add(buttonTable[
```

```
setContentPane(myPanel);
        setSize(400, 100);
        setVisible(true);
    }
    public static void main(String[] args) {
    new MyLayouts();
  }
}
```

5. GUI Components, Layouts, example

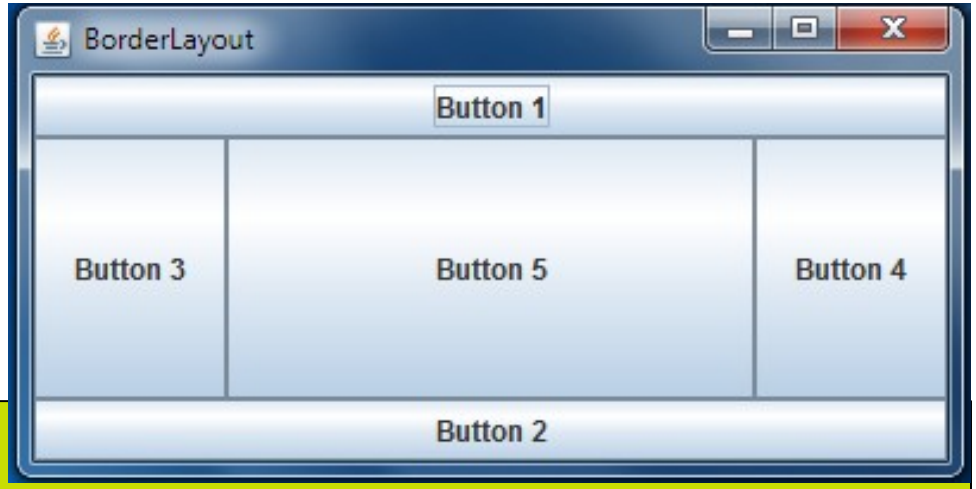



```
import javax.swing.*;
import java.awt.*;

public class MyLayouts extends JFrame {
    private JButton buttonTable[] = new JButton[6];
    private JPanel myPanel;

    public MyLayouts(){
        setTitle("FlowLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        for (int i=0; i<buttonTable.length; i++)
            buttonTable[i] = new JButton("Button "+(i+1));
        myPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        for (int i=0; i<buttonTable.length; i++)
            myPanel.add(buttonTable[i]);
```

5. GUI Components, Layouts, example


```
import javax.swing.*;
import java.awt.*;

public class MyLayouts extends JFrame {
    private JButton buttonTable[] = new JButton[12];
    private JPanel myPanel;

    public MyLayouts(){
        setTitle(”GridLayout");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        for (int i=0; i<buttonTable.length; i++)
            buttonTable[i] = new JButton("Button "+(i+1));
        myPanel = new JPanel(new GridLayout(4,3));
        for (int i=0; i<buttonTable.length; i++)
            myPanel.add(buttonTable[i]);
```

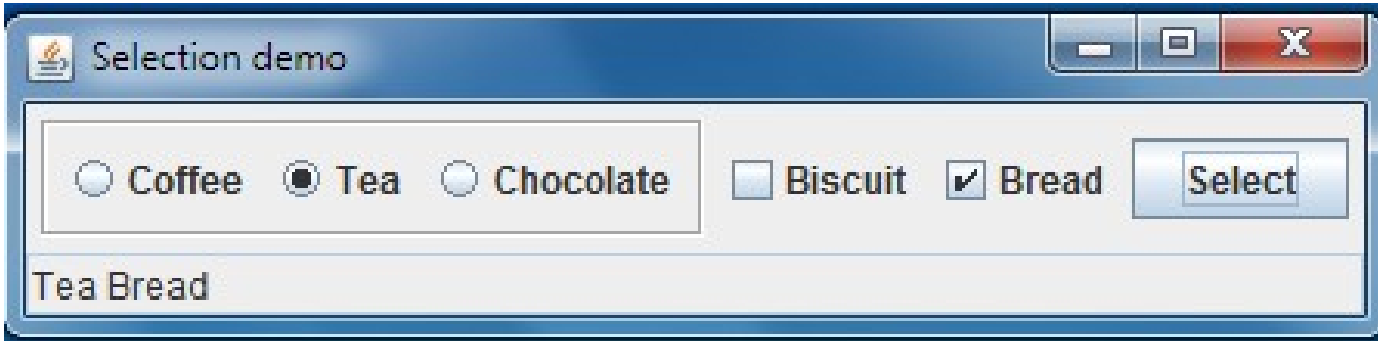
5. GUI Components, Layouts, example




```
public MyLayouts(){
 setTitle("BorderLayout");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    for (int i=0; i<buttonTable.length; i++)
        buttonTable[i] = new JButton("Button "+(i+1));
    myPanel = new JPanel(new BorderLayout());

    myPanel.add(buttonTable[0], BorderLayout.NORTH);
    myPanel.add(buttonTable[1], BorderLayout.SOUTH);
    myPanel.add(buttonTable[2], BorderLayout.WEST);
    myPanel.add(buttonTable[3], BorderLayout.EAST);
    myPanel.add(buttonTable[4], BorderLayout.CENTER);
```

5. GUI components, selections

- Components: `JCheckBox` and `JRadioButton`
- A group of buttons must be grouped with `ButtonGroup`



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
public class SelectionGUI extends JFrame {

    private Selecti
    private JButtor
    private JTextFi
    private Button(
    private JRadioE
    private JCheckE
    private JPanel
```

```
public SelectionGUI() {

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Selection demo");
        Border kehys = BorderFactory.createEtchedBorder();

        vp1 = new JRadioButton("Coffee");
        vp2 = new JRadioButton("Tea");
        vp3 = new JRadioButton("Chocolate");

        selectionPanel = new JPanel();
        selectionGroup = new ButtonGroup();
        selectionGroup.add(vp1);
        selectionGroup.add(vp2);
        selectionGroup.add(vp3);
        vp1.setSelected(true);

        vp1.setActionCommand("Coffee");
        vp2.setActionCommand("Tea");
        vp3.setActionCommand("Chocolate");

        box1 = new JCheckBox("Biscuit");
        box2 = new JCheckBox("Bread");
```

5.

```
inputPanel = new JPanel();
        selectionPanel.add(vp1);
        selectionPanel.add(vp2);
        selectionPanel.add(vp3);
        selectionPanel.setBorder(kehys);
        inputPanel.add(selectionPanel);
        inputPanel.add(box1);
        inputPanel.add(box2);

        myButton = new JButton("Select");
        inputPanel.add(myButton);

        // BorderLayout-sijoittelija mahdollistaa tuloskentän
        // sijoittamisen alareunaan.
        contentPanel = new JPanel(new BorderLayout());
        contentPanel.add(inputPanel, BorderLayout.CENTER);
        resultField = new JTextField();
        resultField.setEditable(false);
        contentPanel.add(resultField, BorderLayout.SOUTH);
        setContentPane(contentPanel);
        myButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                cont.select();
            }
        });

        pack();
        setVisible(true);
    }
```

```
 public void setResult(String choise) {
        resultField.setText(choise);
    }

    public String giveSelection() {
        return selectionGroup.getSelection().getActionCommand();
    }

    public boolean biscuitSelected() {
        return box1.isSelected();
    }

    public boolean breadSelected() {
        return box2.isSelected();
    }

    public void registerController(SelectionController cont) {
        this.cont = cont;
    }
}
```

```
public class SelectionController {
    private SelectionGUI myView;

    public SelectionController(SelectionGUI myView) {
        this.myView = myView;
    }

    public static void main(String[] args) {
        SelectionGUI myView = new SelectionGUI();
        SelectionController myCont = new
SelectionController(myView);
        myView.registerController(myCont);
    }

    public void select() {
        String mySel = myView.giveSelection();
        if (myView.biscuitSelected()) {
            mySel = mySel.concat(" Biscuit");
        }
        if (myView.breadSelected()) {
            mySel = mySel.concat(" Bread");
        }
        myView.setResult(mySel);
    }
}
```

5. GUI components, lists and menus

- JList, JComboBox

```
...
  private JComboBox weekDayList;
  private JButton mtButton;
  private String[] myDays= {”Monday", ”Tuesday",
    ” Wednesday", ”Thursday", ”Friday", ”Saturday",
    ”Sunday"
  };
...
    Border myBorder=
BorderFactory.createEtchedBorder();

    weekDayList = new JComboBox(myDays);
    weekDayList.setBorder(myBorder);
    myContents = new JPanel();
    myContents.add(myDays);
...
```

6. Mouse events, background: JButton

- Event starts from `JButton` component
- Event object is constructed when an event occures
  - To a button `ActionEvent`
- An event listener class identifies functionality while an event occures
  - Event listener class is often an anonymous inner class
  - Class has `ActionListener`-interface
    - Method `actionPerformed`
  - Listener (mostly an anonymous inner class) is registered to a component listener

6. Mouse events

- A mouse event is created by a component, by which mouse reactions are listened
  - T.ex. `JFrame, JLabel`
- The object MouseEvent contains information on mouse moving..
  - T.ex. `getX(), getY(), getButton()`
- Mouse listener has to adopt a suitable interface:
  - `MouseListener`
  - `MouseMotionListener`
  - `MouseWheelListener`

6. Mouse events: `MouseListener` interface

- Five event handling method are in interface `MouseListener`
  - `public void mousePressed(MouseEvent e)`
  - `public void mouseReleased(MouseEvent e)`
  - `public void mouseClicked(MouseEvent e)`
  - `public void mouseEntered(MouseEvent e)`
  - `public void mouseExited(MouseEvent e)`

6. Mouse events: `MouseMotionListener` and `MouseWheelListener` interfaces

- Two event handling methods in interface `MouseMotionListener`
  - `public void mouseMoved(MouseEvent e)`
  - `public void mouseDragged(MouseEvent e)`

- One event handling method in interface `MouseWheelListener`
  - `public void mouseMoved(MouseEvent e)`

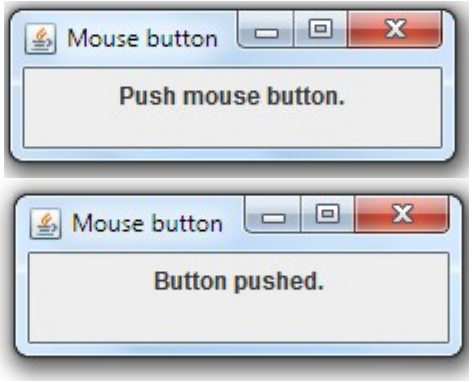
6. Mouse events: adopting classes

- Interfaces `MouseListener` and `MouseMotionListener` need several other classes to be programmed
- Instead of having the event listener programmed to this interface, it can be inhereted from an adopting class
  - Mouse adapter class is `MouseAdapter`
  - Adapter class contains empty methods for interface method scelets
- Suitable also as anonymous inner class event listeners

6. Mouse events: example

- In our example `JPanel` and inner panel are coped to mouse listener
  - In the listener the adapter class method `mouseClicked` is overrided
  - Other adapter class methods will remain as empty code
- Listener is programmed as an anynomous inner class
  - When anonymous inner class is created from a class we get a subclass
  - When anonymous inner class is created from interface (as `ActionListener`), we create a class with interface

6. Mouse events: example



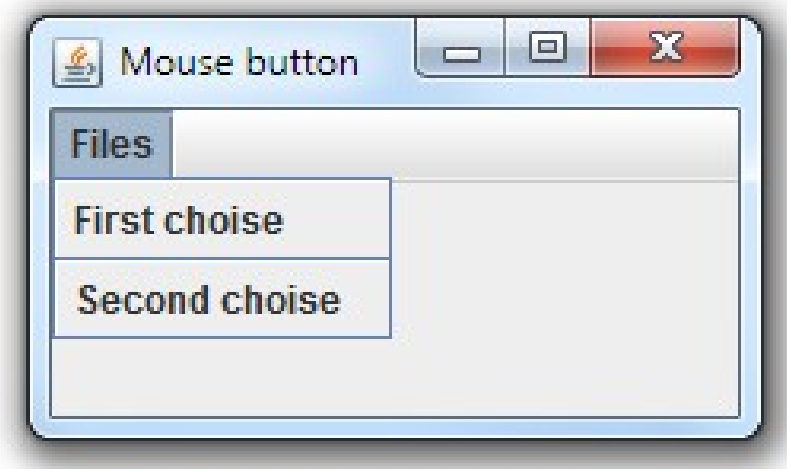

```
import javax.swing.*;
import java.awt.event.*;

public class MouseButton extends JFrame {

    private JPanel myContentPanel;
    private JLabel

    public MouseBu
        initCompone
    }
```

```
 public void initComponents() {
        setTitle("Mouse button");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        myContentPanel = new JPanel();
        intro = new JLabel("Push mouse button.");
        myContentPanel.add(intro);
        setContentPane(myContentPanel);
        myContentPanel.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                intro.setText("Button pushed.");
            }
        });
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        new MouseButton();
    }
}
```

7. Menus

- Menu creation makes navigations clearer in middle size and large applications
- Create a menu row:
  - `JMenuBar myMenuRow = new JMenuBar();`
- Create a menu:
  - `JMenu myMenu = new JMenu(”Files”);`
- Insert a menu line:
  - `myMenuRow.add(myMenu);`
- Insert choises:
  - `myChoiseA1 = new JMenuItem(”First choise”);`
  - `myChoiseA2 = new JMenuItem(”Second choise”);`

7. Menus



- Insert choises to menu:
  - `myMenu.add(myChoiseA1);`
  - `myMenu.add(myChoiseA2);`
- Insert a separator:
  - `myMenu.addSeparator();`
- Insert a function command to a choise:
  - `myChoiseA1.setActionCommand("A1");`
- Insert the choise a listener
  - `myChoiseA1.addActionListener(myListener);`
- Set menu line to this window (`JFrame`)
  - `setJMenuBar(myMenuRow);`

8. Look and Feel

```
try {
    UIManager.setLookAndFeel(
  UIManager.getCrossPlatformLookAndFeelClassName())
  ;
}
catch (Exception e) {
    System.out.println("Look and Feel - setting
  miss.")
}
```

- Look and Feel introduces user interface
  - Not depending on hardware (Java)
  - Depending on hardware (Windows)
- Exceptions needed to program: `ClassNotFoundException`, `IllegalAccessException`, `UnsupportedLookandFeelException` and `ClassCastException`