COMPS203F
Topic 01: Graphical User Interface
(v2.4)

Kelvin Lee

Dialog Boxes

- Add import statement
  `import javax.swing.*;`
- Message box:
  `JOptionPane.showMessageDialog(null, "Hello");`
- Input box: returns a string (returns `null` if Cancel)

```
String inputNumber =
JOptionPane.showInputDialog(null, "Input number:");
```

Dialog Boxes

- Confirm box (returns an integer):

```
int option =
JOptionPane.showConfirmDialog(null, "Are you sure?");
```

the integer equals to

- `JOptionPane.YES_OPTION`,
- `JOptionPane.NO_OPTION`, or
- `JOptionPane.CANCEL_OPTION`

Use `if`-statement to check which option is chosen

```
if (option == JOptionPane.YES_OPTION) {
  ...
```


Simple Frame


```
// A Simple Frame
import javax.swing.*;
import java.awt.*;

public class SimpleFrame extends JFrame {
  private JButton button = new JButton("a button");

  public SimpleFrame() {
    // a container can contain other GUI objects
    Container contentPane = getContentPane();  // (1)
    contentPane.add(button);                   // (2)
    setSize(300, 100);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
  }
} // ** new JDK version allows "directly adding" to JFrame **
  // ** will omit (1) and omit "contentPane." in (2)       **
```


Test Simple Frame

```
// Test Simple Frame

public class TestSimpleFrame {
  public static void main(String[] args) {

    // create a frame with a title
    SimpleFrame myFrame = new SimpleFrame();

    // set the frame visible (old verion : myFrame.show();)
    myFrame.setVisible(true);
  }
}
```


Simple Frame 2

```
// Simple Frame version 2: with action handling
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleFrame2 extends JFrame
    implements ActionListener {
  private JButton button = new JButton("a button");

  public SimpleFrame2() {
    add(button);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    button.addActionListener(this);
    pack();
  }
```

Simple Frame 2

```
  // event handling
  public void actionPerformed(ActionEvent event) {
    button.setText("button pressed");
  }
}
```

Test Simple Frame 2

```
// Test Simple Frame version 2

public class TestSimpleFrame2 {
  public static void main(String[] args) {

    // create a frame with a title
    SimpleFrame2 myFrame = new SimpleFrame2();

    // set the frame visible
    myFrame.setVisible(true);
  }
}
```


Simple Frame 3

```
// Simple Frame version 3: with action handling
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleFrame3 extends JFrame
    implements ActionListener {
  private JButton button = new JButton("clicks: 0");

  public SimpleFrame3() {
    add(button);
    button.addActionListener(this);
    button.setToolTipText("Click to increase count by 1");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
  }
```


Simple Frame 3

```
  // event handling
  public void actionPerformed(ActionEvent event) {
    String buttonText    = button.getText();
    String newButtonText = buttonText.substring(0,8) +
      (Integer.parseInt(buttonText.substring(8)) + 1);
    button.setText(newButtonText);
  }
}
```

Simple Frame 4

```
// Simple Frame version 4
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleFrame4 extends JFrame
implements ActionListener {
  private JButton button = new JButton("set");
  private JLabel label = new JLabel("a label");
  private JTextField textField = new JTextField(10);

  public SimpleFrame4() {
    setLayout(new FlowLayout());
    add(textField);
    add(label);
    add(button);
    button.addActionListener(this);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
  }
```


Simple Frame 4

```
  // event handling
  public void actionPerformed(ActionEvent event) {
    String newLabel = textField.getText();
    label.setText(newLabel);
    pack();
  }
}
```

Exercise: Dialog boxes

- Write a class `TestDialogs` to display the dialogs on slides 2 and 3
- Output the following to the console when suitable buttons are pressed:

  Input box (assume "1234" is input):

  ```
  The input number is:1234
  ```

  Confirm box (assume user click "No"):

  ```
  The answer is: No
  ```

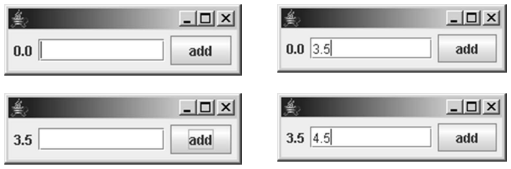

Exercise: Simple Button

- Write a class `SimpleButton` to display a button with a "0".
- If the user clicks it, increase it by 1.

Exercise: Simple Calculator

- Write a class `SimpleCalculator` with
  - a textfield for input, a label for result, a button to add
- Add other buttons to subtract, multiply and divide. To check which button is press, use:

  ```
  if (event.getSource() == addButton)
  ```



Layout Managers

- `BorderLayout` : default, 5 positions only, can be omitted (see p.27, Unit 9 of MT201)


```
// define buttons ...
setLayout(new BorderLayout());
  // ** old JDK uses: getContentPane().setLayout()
add(buttonEast, BorderLayout.EAST);
add(buttonSouth, BorderLayout.SOUTH);
add(buttonWest, BorderLayout.WEST);
add(buttonNorth, BorderLayout.NORTH);
add(buttonCenter, BorderLayout.CENTER);
```


Layout Managers

- `FlowLayout` : components from left to right (see p.26, Unit 9 of MT201)


```
// define buttons ...
setLayout(new FlowLayout());

add(button1);
add(button2);
add(button3);
```


Layout Managers

- `GridLayout` : like a equal-size-cell table (see p. 31, Unit 9 of MT201)


```
// define buttons ...
setLayout(new GridLayout(row,col))

add(button1);
add(button2);
add(button3);
add(button4);
add(button5);
add(button6);
```

Layout Managers Exercise

- Exercise
  - Create the GUI of Q1 of Self-test 9.2 in MT201 Unit 9 using `BorderLayout` Manager
  - Do Q2 of Self-test 9.2

Text Area

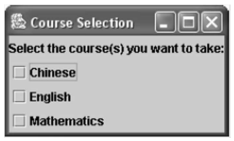
- `TextArea` (similar to `TextField`): p.34
  - multi-line, can have scroll bars

```
import javax.swing.*;
public class FrameWithTextArea2 extends JFrame {
  private JTextArea textArea;
  public FrameWithTextArea2() {
    String content = "Hello World!\n";
    for (int i=0; i < 5; i++) content += content;
    textArea = new JTextArea(content, 5, 40);
    JScrollPane scrollPane = new JScrollPane(textArea);
    add(scrollPane);
    // ...
  }
}
```


Check boxes

- Checkboxes (more than one choice can be selected): p.40

```
public class CourseSelectionFrame extends JFrame {
  private JCheckBox chinese = new JCheckBox("Chinese");
  private JCheckBox english = new JCheckBox("English");
  private JCheckBox math = new JCheckBox("Mathematics");
  public CourseSelectionFrame() {
    setLayout(new GridLayout(4, 1));
    add(chinese);
    add(english);
    add(math);
    // ...
  }
}
```



Check boxes

- Exercise: write a class `ComputerPrice` and its testing class
  - use `isSelected()` to check if a choice is selected (e.g. `if (english.isSelected()) ...`)
  - use `getText()` to get the text of a choice

Radio buttons

- Radio buttons (only one of the choices can be selected): p.42

```
public class RegionSelectionFrame extends JFrame {
  private JLabel label = new JLabel(
                         "Select the region:");
  private JRadioButton hkButton =
          new JRadioButton("Hong Kong Island", true);
  private JRadioButton klnButton =
          new JRadioButton("Kowloon");
  private JRadioButton ntButton =
          new JRadioButton("New Territories");
  private ButtonGroup optionGroup = new ButtonGroup();
```

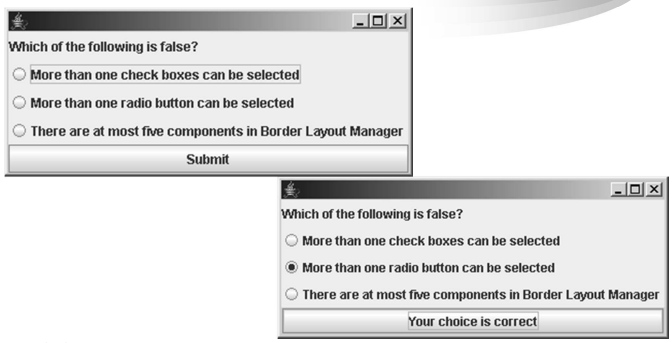

Radio buttons

- Radio buttons (only one of the choices can be selected): p.42

```
public RegionSelectionFrame() {
  setLayout(new GridLayout(4, 1));
  add(label);
  add(hkButton);
  add(klnButton);
  add(ntButton);
  optionGroup.add(hkButton);
  optionGroup.add(klnButton);
  optionGroup.add(ntButton);
  // ...
}
```

Radio buttons

- Exercise: write a Class `MultipleChoice`



Pull-down menu

- Menu bar, menus, menu items: p.45

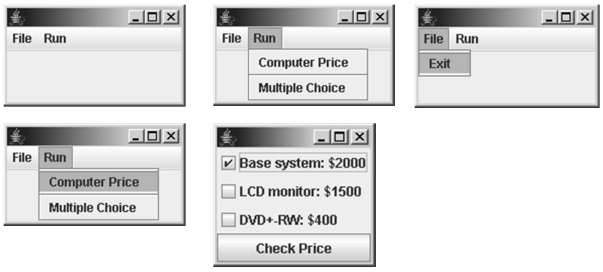
```
private JMenuBar menuBar = new JMenuBar();
private JMenu quizMenu = new JMenu("Quiz");
private JMenuItem quizTestingMenuItem =
        new JMenuItem("Quiz-testing");
private JMenuItem quiz01MenuItem =
        new JMenuItem("Quiz 01");

private JMenu settingMenu = new JMenu("Setting");
private JMenuItem fontSizeMenuItem =
        new JMenuItem("Font size");
// ...
```


Pull-down menu

```
setJMenuBar(menuBar);
menuBar.add(quizMenu);
quizMenu.add(quizTestingMenuItem);
quizMenu.add(quiz01MenuItem);
menuBar.add(settingMenu);
settingMenu.add(fontSizeMenuItem);
// ...
quizTestingMenuItem.addActionListener(this);
quiz01MenuItem.addActionListener(this);
fontSizeMenuItem.addActionListener(this);
// ...
public void actionPerformed(ActionEvent event) {
  if (event.getSource()== quiz01MenuItem) { ... }
}
```


Pull-down menu

- Exercise: write a class `PullDownMenu`



Pull-down menu

```
priceMenuItem.addActionListener(this);

public void actionPerformed(ActionEvent event) {
  Object source = event.getSource();
  if (source == priceMenuItem) {
    // lines copied from TestComputerPrice.java
    ComputerPrice pcPrice = new ComputerPrice();
    pcPrice.setVisible(true);
  }
```


JPanel: Complex GUI

- JPanel is another container
- can have its own layout manager

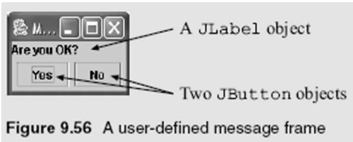


Figure 9.56 A user-defined message frame

JPanel: Complex GUI

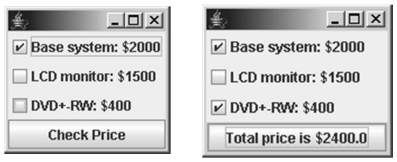
```
public class MessageFrame extends JFrame {
  private JLabel messageLabel = new JLabel();
  private JButton okButton = new JButton("Yes");
  private JButton cancelButton = new JButton("No");
  private JPanel buttonPanel = new JPanel();

  public MessageFrame(String message) {
    buttonPanel.add(okButton);
    buttonPanel.add(cancelButton);
    add(messageLabel, BorderLayout.CENTER);
    add(buttonPanel, BorderLayout.SOUTH);
    messageLabel.setText(message);
    // ...
  }
}
```


JPanel: Complex GUI

- Exercise: Q2 and Q3 on P.51 of Unit 9

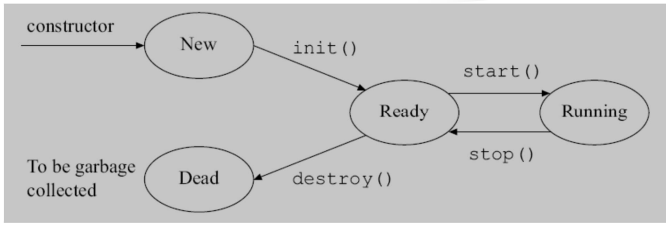
Text Area and Check boxes Ex.

- Checkboxes
  - Exercise: write a class `ComputerPrice2` and its testing class using a list of checkboxes



Applets

- Small programs run on Web browsers
- Have its own life cycle:



Applets

```
import javax.swing.*;
public class DemoApplet extends JApplet {
  public DemoApplet() {
    getContentPane().add(new JLabel("Hello World"));
    System.out.println("DemoApplet.DemoApplet()");
  }
  public void init() {
    System.out.println("DemoApplet.init()");
  }
  public void start() {
    System.out.println("DemoApplet.start()");
  }
  public void destroy() {
    System.out.println("DemoApplet.destroy()");
  }
  public void stop() {
    System.out.println("DemoApplet.stop()");
  }
}
```


Exercise: Calculator Applet

- Change the class `SimpleCalculator` to `CalculatorApplet` and replace `JFrame` by `JApplet` and compile
- Comment out the line which causes error messages
- Copy a java.policy file on OLE to your JRE folder:

  C:\Program Files\Java\jre1.xxx\lib\security or

  C:\Program Files (x86)\Java\jre1.xxx\lib\security

where the 1.xxx "jre1.xxx" is the JRE version (see output of "appletviewer -J-version")

- Create a file `CalculatorApplet.html` and run with `appletviewer CalculatorApplet.html`

```
<html>
  <head><title> Simple Calculator Applet </title></head>
  <body>
    <applet code="CalculatorApplet.class" width=250
   height=50></applet>
  </body>
</html>
```

- To run in browser such as Firefox, add "file:///" to exception site list (and copy java.policy to suitable folder if JRE version is different from appletviewer's version)

User interface Design

- Reference: COMPS201 Unit 5
- can be found on OLE

Characteristics of good user interfaces

- Enable us to check whether a user interface is good or not
  - Speed of learning
  - Speed of use
  - Speed of recall
  - Error prevention
  - Attractiveness
  - Consistency
  - Feedback
  - Support of multiple skill levels
  - Error recovery (undo facility)
  - User guideline and on-line help

Speed of learning

- The time required to learn the use of a software application
  - A good user interface for a software application should enable users to master it in as short a period as possible
- To minimize the speed of learning, the following measures can be used
  - Use of metaphors and intuitive command names
  - Consistency
  - Component-based interfaces

Use of metaphors and intuitive command names

- A metaphor is an abstraction of a real life object or concept used in user interface design.
  - As users get used to these real world objects or concepts, they can operate the software application right away.
- Examples
  - Scrolling with iOS (iPhone, iPad)
  - Button panel of Microsoft Media Player
  - Angry birds

Consistency

- It is preferable for software applications to realize their equivalent operations in a similar way, so that the ways of operating the software applications are consistent.
- Examples
  - Cut-copy-paste operation
  - Web browsers and web-based applications

Component-based interfaces

- Users can figure out the ways of using the GUI components, if multiple software applications share the same set of GUI components.
- Examples
  - The Minimize, Maximize and Close button at the top-right corner of all software applications
    - The Look and Feel feature of Java enables you to build the user interface with native GUI components for the platforms.

Speed of use

- The time and the efforts required by the users to initiate and execute an operation is known as the *speed of use*.
- The time and effort required for the users to perform an operation should be minimal.
- Therefore, a software application that requires lengthy commands is undesirable.

Speed of recall

- Once the users learn how to use an interface, they should be able to recall it in a short period of time.
- In order to minimize the time required for the users to recall a less frequently used application, the user interface should be based on metaphors and intuitive command names.

Error prevention

- If the software application does not perform in the desired way, it is considered to be an error.
- A good user interface should minimize the error rate and the scope for committing errors when the users are using the user interface
- One possible way to prevent errors is to ask the users to confirm potentially destructive operations before the operation is performed.

Attractiveness

- A good user interface should be attractive and gain the user's attention.
- Examples
  - A graphic user interface is more attractive than text-based (or character-based) user interface.
  - Different versions of Microsoft Windows

Consistency

- The ways of operating the user interface should be consistent so that the users can generalize the knowledge from one aspect of the user interface to other aspects.
  - Improves speed of learning and speed of recall and reduces the error rate.
- Examples
  - Dialogs should be in similar style
  - Microsoft Office Suite
    - The user interfaces of various components were standardized

Feedback

- The user interface should keep users informed of the state of their processing requests.
  - A good user interface should provide suitable feedback to various user operations, particularly for lengthy operations.
- Examples
  - Copying large sized files can be a lengthy operation.
  - Loading a webpage

Support of multiple skill levels

- Software applications need to be used by users with different levels of experience.
- Therefore, a good user interface should support multiple levels of sophistication and be able to issue commands for users with different skill levels.
- Software application should support multiple ways of issuing the same operation
- Examples,
  - The use of dialog by novice users
  - The use of short-cuts by experts
  - Microsoft Office Suite enables users to customize the user interface in their own way.

Error recovery (undo facility)

- A good user interface should enable a user to undo an error with the user interface
- Examples
  - Word processer software applications provide an undo facility because users may change their mind while editing.
- Some operations cannot be undone,
  - Examples
    - Overwriting a file,
    - Formatting a drive, and
    - Sending an electronic mail
  - The software applications should prompt the users for confirmation before performing unrecoverable operations.

User guideline and on-line help

- Software applications should enable the users to start using them right away, and user guidance and on-line help are important characteristics of a good user interface.
- Examples,
  - Tooltips
  - Help feature
  - Office Assistant feature

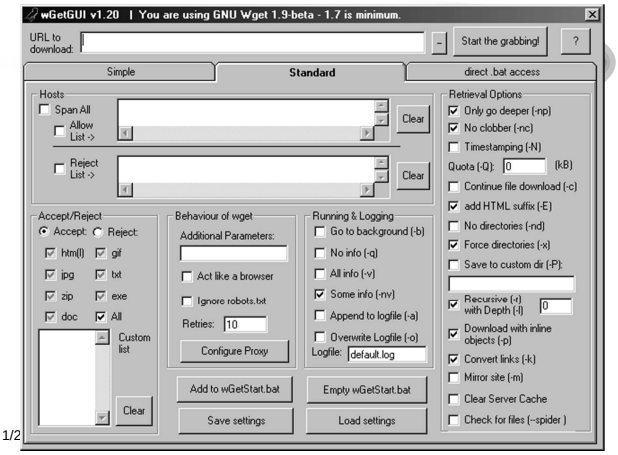
Key principles of user interface design

- Six principles of user interface design proposed by Constantine and Lockwood (1999)
  - The structure principle
  - The simplicity principle
  - The visibility principle
  - The feedback principle
  - The tolerance principle
  - The reuse principle

The structure principle

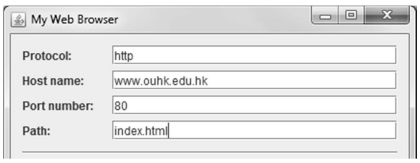
- *The design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.*
- It is necessary to present information in a structured way, so that it is easier for people to scan and understand.
- Example
  - wGetGUI - a lot of controls on its user interface without any intuitive structure
- Solution
  - allow users to customize the interface of the application.
  - breaking the GUI components or information into distinct sections according to their nature and interpretations.

wGetGUI



The simplicity principle

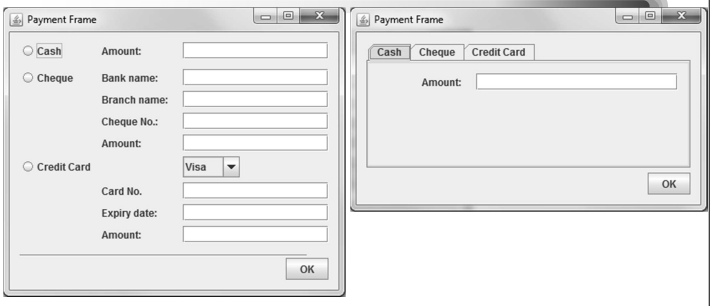
- *The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.*
- *Less is more*, as long as the user interface matches the users' goal
  - The model behind the user interface should be as simple as possible.
  - If a user interface model has fewer concepts, the user takes less time to master the concepts, provided that the user interface provides the required functionality.
- User interfaces should use the users' own language so that the users can use the interface easily, without any ambiguity



The visibility principle

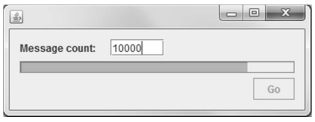
- *The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.*
- Software developers can address the visibility principle by examining each GUI component on their user interfaces and ask whether they are necessary or redundant.
- For GUI components that provide options, it is necessary to verify whether some options are never applicable and can be removed.

Which one is better?



The feedback principle

- *The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.*
- Software developers need to make sure that any applications which require some time to complete give feedback to the user.
  - The use of progress bar
- The message to be shown should be clear and unambiguous
  - Determine the proper ways of notifying the user of the state of each operation during execution and upon completion.
  - Investigate whether the notification is presented in a clear, concise and unambiguous way to the user.



The tolerance principle

- *The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.*
- Examples,
  - Deleting a file by Windows compared with deleting a file by DOS
    - more flexible and tolerant, and the cost of mistakes is minimized
  - Undo/redo feature
  - The Google search engine
    - rectifying inaccurate user search entries
- Software developers should ensure that the user interface of a software application tolerates the mistakes made by the users, and minimizes the cost of the mistakes made

The reuse principle

- *The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.*
- Use a standard set of GUI components, preferably those commonly used by all software applications on the platform
  - the users can use operate the user interfaces in the usual way, and hence shorten the time it takes to learn the application.
- Extend the classes for the standard GUI components by inheritance, such as overriding their behaviours and extending their capabilities