COMP122 Week 11

Graphical User Interfaces

UNIVERSITY OF LIVERPOOL

Dr. Patrick Totzke
totzke@liverpool.ac.uk

https://liverpool.instructure.com/courses/59716

Feedback please!

You should have received a link to a EvaSys module survey.
(Also via Canvas→COMP122→Course Surveys)

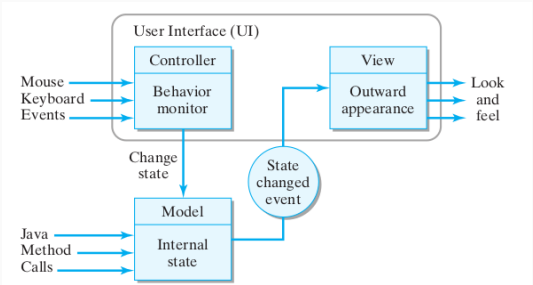
Help me by filling this in, thanks!

Graphical User Interfaces

The MVC pattern

Model–View–Controller is a design pattern commonly used for user interfaces that divides an application into three interconnected parts.

- internal representations of information (Model)
- the way information is presented (View)
- user input (Controller).



Further Reading

- Gamma et al. – Design Patterns
- Morelli, Chapter 13

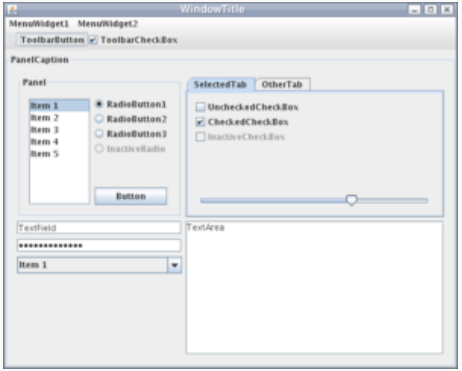
Views

Graphical Components

- graphical elements that visualize information are an idea of the 60s, commercialized in the late 70s and popularized in the late 80s.
- Common metaphor: **W**indows-**I**cons-**M**enus-**P**ointer
- modern GUI libraries are often called toolkits and come with lots of widgets (also: components)
- Examples of widget classes are buttons, sliders, menus; but also containers to structure the layout of the GUI.

Components and Containers

The view component of a GUI is typically build from combining "widgets" (buttons, text fields, menus, frames, etc.) in a hierarchical manner: some widgets are containers, so contain others.

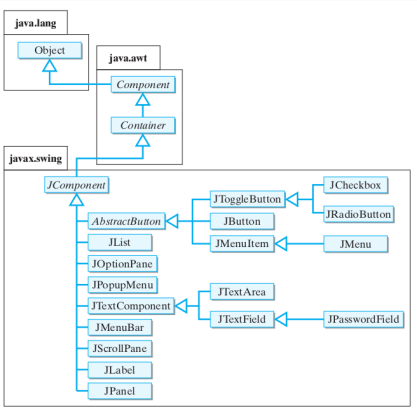


- A Frame (topmost widget) contains a Panel contains a GridLayout contains (..) a RadioButton.
- Containers can contain multiple child-widgets, including other containers.
- in an OO-world, all widgets are (instances of) some widget class, which extends a more general Container class.

Java GUI Toolkits

A Note on Java GUI toolkits

Java's GUI framework is largely based in the "Swing" library. The original GUI framework is called the "Abstract Windowing Toolkit" (AWT).



- These two libraries aren't entirely separate from each other, as Swing components ultimately inherit from the core AWT components.
- Swing components typically have names that begin with the letter "J" to distinguish them from older AWT components.
- For example, AWT has a "Frame" class while Swing has a "JFrame" class. Similarly, AWT has a "Button" class while Swing has a "JButton" class.

Creating a JFrame

Most all GUI applications are based on a main window frame which contains other objects. To create a `JFrame`, we import the class from the `javax.swing` library.

We create a `JFrame` and can pass a title of the frame to the constructor:

```
1  JFrame frame = new JFrame("My JFrame");
```

We could instead set the title by calling the appropriate method:

```
1  frame.setTitle("My JFrame");
```

What happens when the JFrame is closed?

The `setDefaultCloseOperation` method is used to tell the JFrame what to do when the user attempts to close the window. There are four options that are available for the JFrame class. These options are

DO_NOTHING_ON_CLOSE	
HIDE_ON_CLOSE	(the default)
DISPOSE_ON_CLOSE	(the best option unless applied to the main frame)
EXIT_ON_CLOSE	(implemented directly in JFrame)

The "EXIT" option should be only used on a main application window since it will close the application. It is recommended that "DISPOSE" be used on other windows.

Sizing and showing/hiding the JFrame

We can use the `setBounds` method to set the initial position and size of the JFrame.

```
1  frame.setBounds(0, 0, 500, 200);
```

The first two parameters set the location of the top left hand corner position, and the second two set the width and height in pixels.

An alternative is to use `pack`, which sizes the frame to fit the components inside it.

```
1  frame.pack();
```

The `setVisible` method takes a Boolean and is used to show or hide the frame.

```
1  frame.setVisible(true);
```

Putting it all together

```
1  /** A simple example to illustrate the JFrame class. */
2  import javax.swing.JFrame;
3
4  public class EmptyGUI {
5
6     public static void main (String[] args) {
7
8        JFrame frame = new JFrame("My JFrame");
9        frame.setBounds(0, 0, 500, 200);
10       frame.setVisible(true);
11    }
12 }
```

The frame can be resized, moved, minimized, maximized, and closed (which quits the application). DEMO?

More Components

Of course we want to be able to add more components to this frame, so that we can start to do more useful/interesting things.

Other common components we can add include:

- `JLabel`
- `JButton`
- `JTextField` for one line of editable text
- `JTextArea` for multiple lines of editable text
- `JCheckbox`
- `JRadioButton` usually as part of a `ButtonGroup`
- `JComboBox`

All of these classes are subclasses of `JComponent`, hence share common attributes and methods.

Component Methods

Methods of `Component` include ones for controlling the position and size.

- `public void setLocation(Point p)`
- `public void setSize(Dimension d)`
- `public void setSize(int width, int height)`
- `public void SetBounds(int x, int y, int width, int height)`
- `public void setBounds(Rectangle r)`
- `public void setVisible(boolean b)`
- `public void setEnabled(boolean b)`

Here, `java.awt.Point` represents a point in 2-D space (with an x- and y-coordinate). `Dimension` is similar and represents a rectangular area with a width and height (but doesn't specify a location).

Adding Components to Frames – The JLabel class

JLabels are used largely to provide text in a GUI for information. Let's add one to `frame` as follows.

```
1  JLabel centeredLabel = new JLabel("Some text", SwingConstants.CENTER);
2  frame.add(centeredLabel);
```

- You can create a `JLabel` component by supplying text to its constructor.
- If desired, you can also provide justification for the text.
- Here, `javax.swing.SwingConstants` is an interface that defines constants `LEFT`, `CENTER`, and `RIGHT`.
- Finally use `JFrame`s method `add` to add the label to the frame. This is in fact inherited from `JContainer`.

A JLabel working example

```
1  import javax.swing.*;
2
3  public class LabelFrame extends JFrame  {
4
5     //  Constructor for the subclass, adds a label to the frame
6     public LabelFrame(String title)  {
7        super(title);
8        JLabel label = new JLabel("Here is some text.",
9                                  SwingConstants.CENTER);
10       add(label);
11       setBounds(0, 0, 250, 150);
12    }
13
14    public static void main(String[] args)  {
15       LabelFrame frame = new LabelFrame("Label Frame");
16       frame.setVisible(true);
17    }
18 }
```

Controller

Event-driven Programming

A typical GUI program spends most of its time waiting for something to happen and respond to.

Events are objects representing that something.

- They are emitted by some source; often a widget
- There are different types of events: button clicks, mouse movements, focus changes etc.
- Events can be ignored or interpreted by (one or more) Listeners

Side-note: Java Swing uses the same event model (and classes/interfaces etc.) as the older AWT library. The package `java.awt.event` defines for instance `ActionEvent`, `FocusEvent` and `MouseEvent`.

Event Listeners

The Swing-way of reacting to events is to define a class that implements the `EventListener` interface.

There are several such interfaces (one for every type of event), but they all declare are least one method that reacts to an event.

Example: `java.awt.event.ActionEvent` (for interpreting button-click events) declares

```
1  public void actionPerformed(ActionEvent event);
```

Example: Listeners for Focus change events would have to implement similar methods `focusGained` and `focusLost`.

Link Event Listeners to Sources

In order to associate some listener with possible GUI events, we have to *register* them with the source widgets that can emit these events.

Example: Say we have written a class that implements the `ActionEventListener` interface, and further, lets say we have a GUI that contains a `JButton button`. We can register a new instance of our listener with the button as follows.

```
1  l = new MyActionEventListener();
2  button.addActionListener(l);
```

Now, if a user clicks on the button,

1. the Button creates an `ActionEvent e` representing this event.
2. calls the handler `l.actionPerformed(e)` on the our EventListener `l`.

Demo: a Login frame

Let's write a "Login" frame that has a `JTextField` to input a user name string, and a `JPasswodField` to input a password string.

If the user clicks on a button, we want to print both on the terminal and exit.

DEMO.

- We can register one Listener to more than one UI Component.
- We can register more than one Listener to a UI Component.
- You could have your GUI Frame class implement various ActionListener interfaces (but don't do this, it's ugly and goes against the loose-coupling design principle!)

Your Questions?

Q44

Q: When I read a book file (Alice's Adventures in Wonderland), the apostrophe is stored as "?". What up?

A: This file is UTF-8 encoded but when reading (on Windows) you interpret bytes in a different encoding, meaning some non-ASCII chars are not interpreted well, which defaults to "?". Try the code below and use the cs50 IDE.

```
1 BufferedReader reader =
2     Files.newBufferedReader(file.toPath(),
3     java.nio.charset.StandardCharsets.UTF_8);
```

Q45

Q: How can I correctly round (Book.getPages)? For example if there are 4202 characters then it should return 6 but if you divide 4202/800 the answer is 5.00 and cannot be rounded up to 6.

A: make sure to use division on `double`s. For example use `(double)4202/800` here.

Q46

Q: In Press.request, can I re-throw IOExceptions?

A: No. This method should only throw IllegalArgumentException (if the given bookID is invalid) and no other type of exceptions.

If the call to Press.print throws an IOException then this must be caught in a try/catch block. In that case your request method should just return an empty list of books.

Q47

Q: Will I lose marks if I were to use two different pattern objects in order to find the title, author, and content?

A: No, use whatever you like. (my solution does not use REs at all)

Q48

Q: About the exam...?

A: This **class test**

- will be on Canvas, on **12 May at 12 noon and close at 1pm**
- is worth 15% of the module grade
- covers questions about all 11 weeks of content
- is very similar to the weekly quizzes

Summary

We looked at...

- Basics of Graphical User Interfaces (GUIs)
- The Model-View-Controller pattern
- Java Swing: JFrames and other widgets

This Thursday

- No new materials (just Q&A)

Next up, week 12

- Tuesday: Recap, Java-v-Python & other fun stuff
- Thursday: No lecture
- No in-person labs
- Final quiz, on Friday at 12noon!