GUI Programming

Introduction to GUI Design

- We will look at…
  - The world of GUI's
  - Definition for GUI
  - Some basic GUI heuristics
  - Introduction to SWING

- The course is 50% Exam and 50% CA (notes online may change and often do)
- The repeat strategy is 100% exam
- CA will be split as follows into 10 labs of 50 points each
- Each 50 point lab will contain various challenges of various difficulty levels
- There will be two lab submission dates and then **self-assessment** followed by lecturer assessment in labs

Background to GUI's

What is a GUI?

- The acronym GUI stands for **G**raphical **U**ser **I**nterface.
- pronounced "gooey".
- There are lots of definitions out there. Each attempts to wrap up in a single statement an exact definition for GUI.
- Lets look at a few

Definitions

- A graphical (rather than purely textual) user interface to a computer.
- A computer terminal interface, such as Windows, that is based on graphics instead of text.
- refers to a software front-end meant to provide an attractive and easy to use interface between a computer user and application.
- a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.

Problems with the definitions

- GUI's not limited to PC's
  - ATM's
  - Mobile Phones
  - Television sets
  - MP3 players
  - Air conditioning system on my car
- Many interfaces use graphics and visual elements to allow user interaction.
- In this course we will concentrate on developing GUI components for computer systems.

What's in a GUI?

- We can use a variety of elements (widgets) to create GUI's. These include
  - Windows& Frames
  - Tabs
  - Toolbars & Menus
  - Buttons
  - Dialogue Boxes
  - Lists
  - Password boxes
  - Text fields& Text Areas
  - Sliders
  - Graphics and Icons
  - Media
  - And many many more!!!

GUI's in JAVA

GUI DEVELOPMENT IN JAVA

- When JAVA 1.0 was introduced, it contained a class library which SUN called the Abstract Window Toolkit or AWT.
- AWT was used for basic GUI programming. The way AWT library deals with user interface elements is to delegate their creation and behaviour to the native GUI toolkit on each target platform (Windows, MAC, Linux etc).
- For example a simple application that displayed a button would look different when run on Windows to the same application run on the Macintosh (size of button, shape, fill, text, behaviour).
- JAVA would just be responsible for creating the peers. The idea was that you could "write once and run anywhere".

- The peer based approach worked well for simple applications but it soon became apparent that it was very difficult to create complex GUI JAVA based applications that could run on a variety of platforms.
- User interface elements such as menus, scrollbars and text fields can have subtle differences or behaviours on different platforms.
- As a result applications built in JAVA with AWT did not look as nice/slick as native Windows or Macintosh applications.
- More seriously, different bugs existed in the AWT UI library for different platforms. Developers complained that they needed to test their applications on each platform , a practice called "write once debug everywhere".

The arrival of SWING

- In 1996 Netscape created a GUI library called the IFC (Internet Foundation Classes) that used and entirely different approach.
- User interface elements such as buttons, menus and so on where painted onto blank windows. The only peer functionality needed was a way to put up windows and paint on those windows.
- Thus Netscape's IFC widgets looked and behaved the same no matter what platform they were run on.
- Sun worked with Netscape to perfect this approach creating a user interface library with the codename "SWING" (sometimes called the SWING set).
- The full JFC is vast and far more that just the SWING GUI toolkit. The JFC also features components for accessibility, a 2D API and a drag and drop API.

The benefits of SWING

- SWING based elements will be somewhat slower to appear on a users screen than those created using AWT. However on today's powerful platforms this has become a very minor issue.
- SWING offers developers a number of advantages including
  - Swing has a much richer and convenient set of user interface elements.
  - Swing depends much less on the underlying platform, therefore is much less prone to platform specific bugs.
  - SWING will give a consistent user experience across platforms.

SWING versus AWT

- AWT was built initially as a quick solution to the need for "write once run anywhere"…but relied on peer components in the specific architectures too much, i.e. there were too many heavyweight components
- As a result there were inconsistencies between platforms
- Heavyweight components are those that are associated with a native peer component and are rendered in their own native opaque window
- Swing has replaced previously heavyweight components with corresponding lightweight components

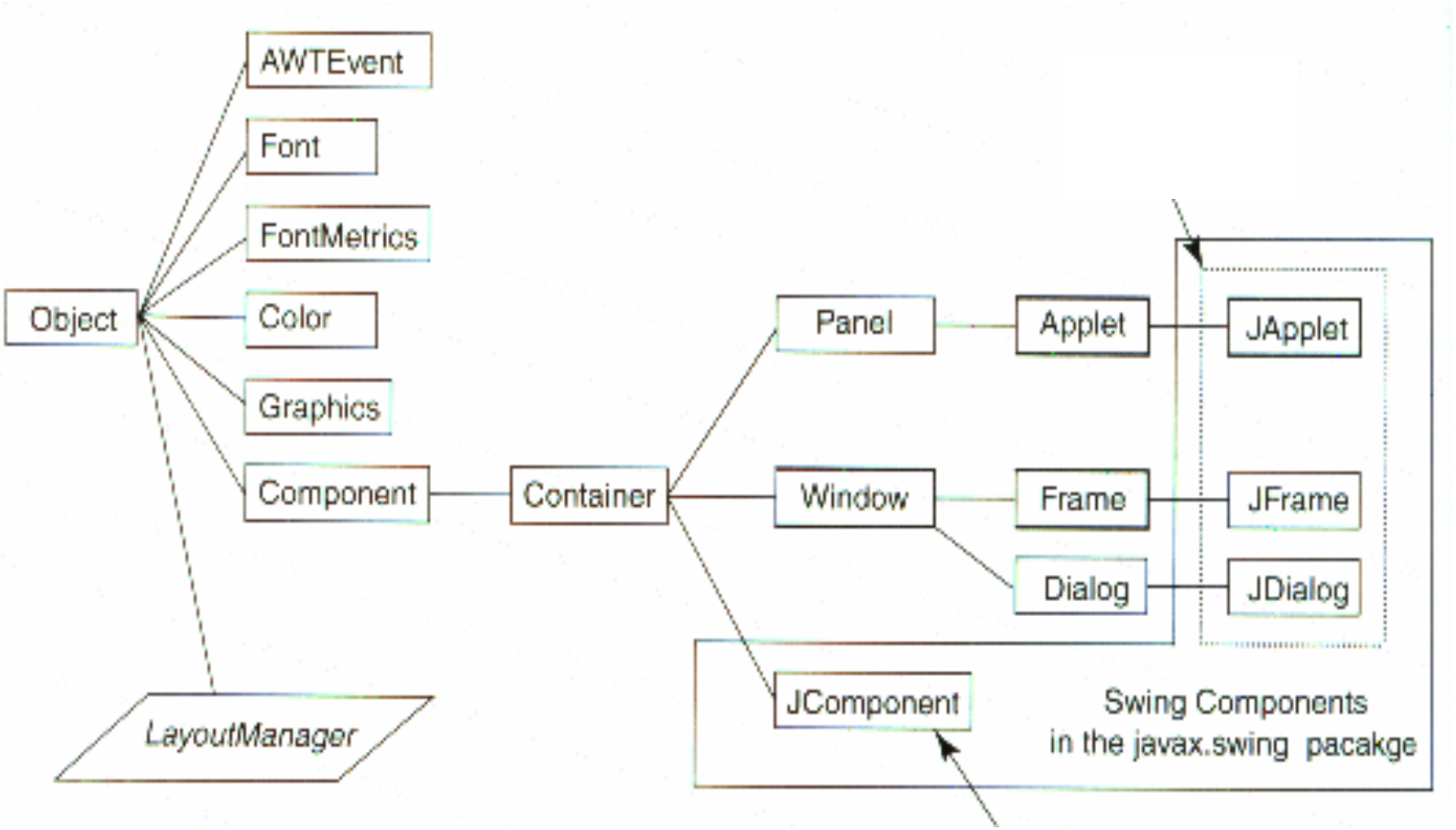
SWING versus AWT (2)

- Since Swing was released AFTER AWT, Swing has also removed many of the software bugs that were in the AWT
- Swing as well as providing lightweight replacements for AWT heavyweight components also provides about four times the components that were contained in the AWT
- Swing components, even though they mimic the AWT behaviors also add new features e.g. buttons with images
- It is important to remember that Swing extends a lot of the AWT core components…so it's worth studying AWT

SWING and Lightweight Components

- Lightweight components are components that are not associated with a native peer
- Lightweight components make the "write once run anywhere" goal more achievable since there is no platform dependence
- Lightweight components must be contained in a heavyweight container, i.e. are not standalone components
- For example the AWT Button component was heavyweight, JButton is lightweight
- Lightweight components can also have non-rectangular appearance

SWING Classes



SWING and AWT

- As can be seen from the diagram on the previous page:
- Swing is built on top of AWT, it **DOES NOT REPLACE AWT**
- Swing uses AWT graphics, colors, fonts, and the LayoutManagers.
- Swing does not use AWT components with the exception of extending the Frame, Window and Dialog heavyweight components to create JFrame, JWindow and JDialog (these are still heavyweight in Swing)
- Like all good Object Oriented systems Swing **RE-USES** the best of AWT in order to build more lightweight components

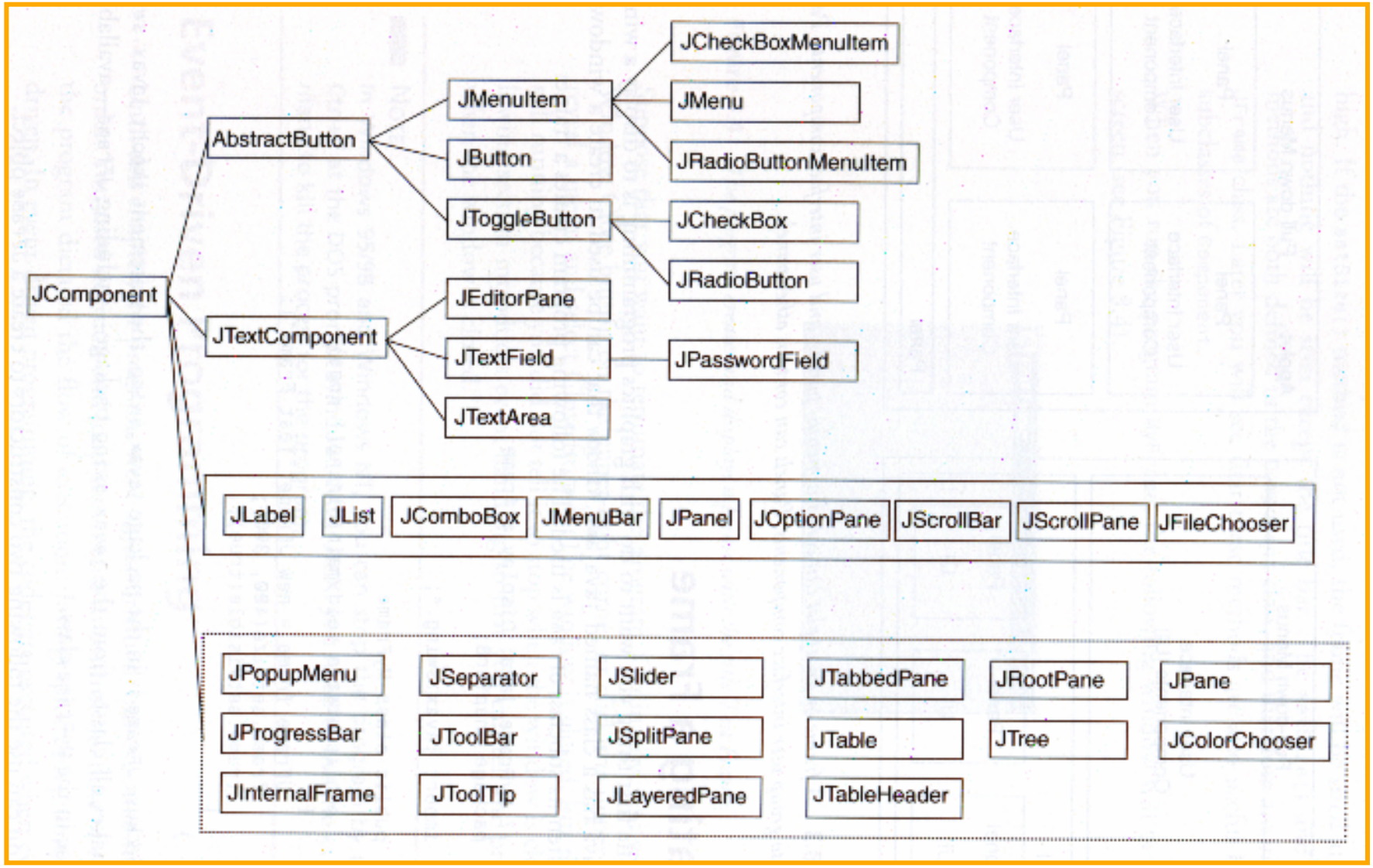
Mixing SWING and AWT

- This can be problematic as mixing Heavyweight and LightWeight components can cause problems when being laid out e.g. a heavyweight button may cover a lightweight button even though the lightweight was added first.

- The biggest issues arises when trying to add a heavyweight component into a lightweight component

- So the upshot is don't mix SWING and AWT components in your software (i.e. don't use a SWING button in an AWT interface and visa versa)

SWING Classes



SWING Components to examine

- We will be examining some of the more common GUI components used in SWING. These include

JLabel	An area where un-editable text or icons can be displayed.
JTextField	An area in which the user inputs data from the keyboard. The area can also display information.
JButton	An area that triggers an event when clicked.
JCheckBox	A GUI component that is either selected or not selected.
JComboBox	A drop down list of items from which the user can make a selection by clicking an item on the list.
JList	A drop down list of items from which the user can make a selection by clicking once on any element in the list. Double clicking on any element in the list generates an action event.
JPanel	A container in which components can be placed.

SWT

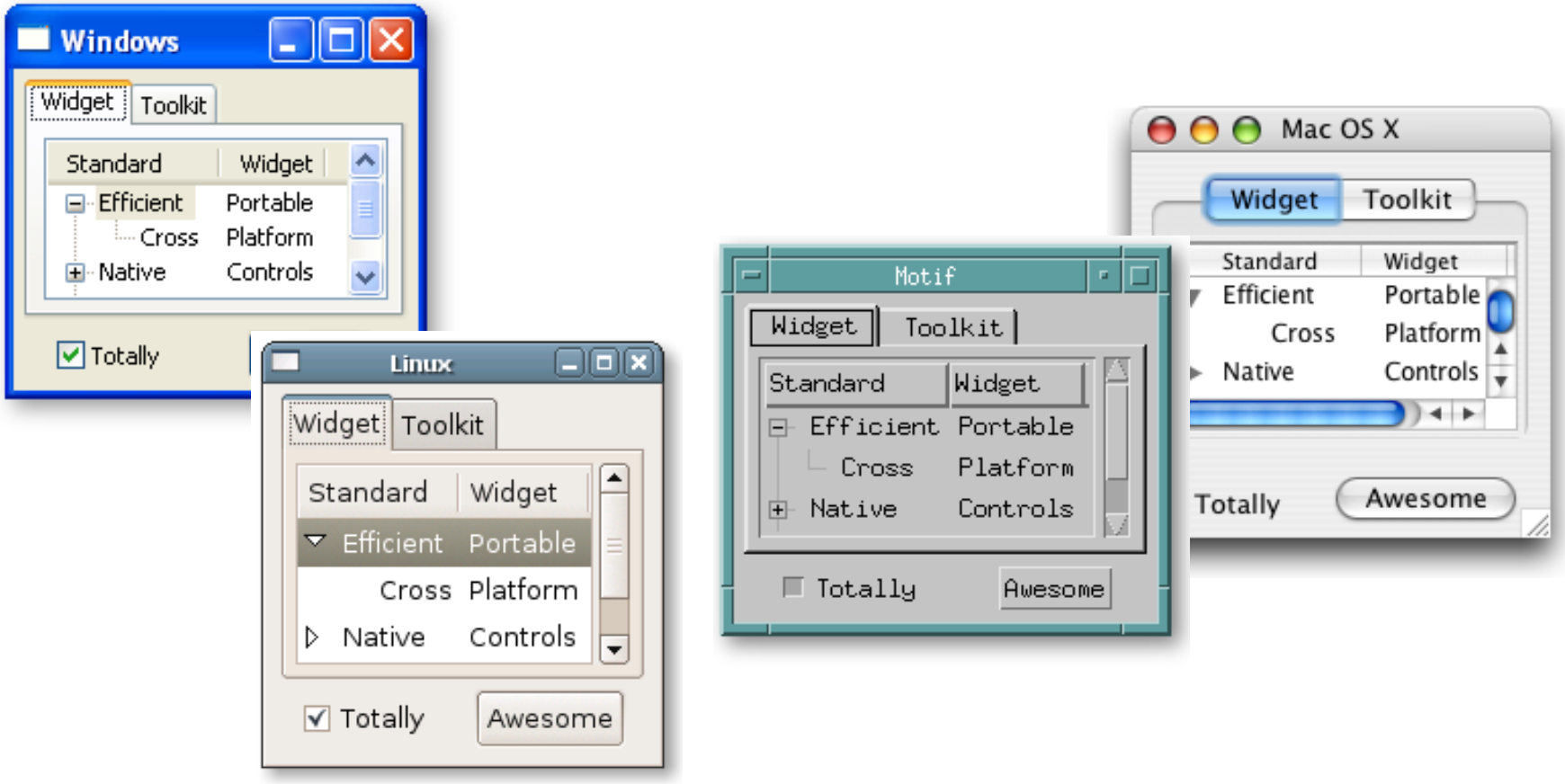
- SWT (the Standard Widget Toolkit) is an alternative (competing) GUI component technology that is part of the Eclipse project. There is a large and growing community that advocates SWT over Swing for Java GUI programming.

- Growing in popularity.

- The good news for developers is that its coding is very similar to SWING. So if you can program in SWING, you will have no problems moving to SWT (if required)

- For this course we will focus on SWING.

Applications that we develop in JAVA can be run on many different platforms.



Developing GUI's

- Lots of development tools
  - Swing Designer
  - NetBeans
  - JFormDesigner
  - FormLayoutMaker
  - Foam
- Once you have defined your GUI, in what format does the GUI builder save it?
  - Pure code. The tool generates Java code to create the GUI, and parses Java code to read it into the visual editor. It does not save any files other than the Java code.
  - Code and metadata. The tool generates metadata describing the forms, as well as Java code to display the forms. An example is NetBeans, which saves the form definition in .form files and generates code in .java files. The .form files live in the same directory as the corresponding .java file.

- We won't be making much use of the tools
- All hand coded (notepad / textpad)
- Best for developers to understand code rather than tools
- You can learn a tool at any time!

- The SWING tutorial helps you get started with SWING and understanding SWING

- http://download.oracle.com/javase/tutorial/uiswing/start/index.html

Developing our first GUI in SWING

- We will look at…

- Creating Frames in SWING
- JPanel
- Labels
- Buttons
- Basic Text

Basic Components

Creating a Window (Frame)

- Before we can draw a SWING component on screen we need a window to place it into.
- A top level window (that is, a window that is not contained inside of another window) is called a frame in JAVA.
- The AWT library has a peer based class called Frame.
- The SWING version of this class is called JFrame.
- JFrame extends the frame class and is one of the few SWING components that is not drawn on the canvas.
- Frames are examples of containers. This means that a frame can contain other user interface components such as buttons and text fields.

Creating a Frame

- Look at the sample program contained in sample 1 folder.

Conclusions

- JAVA swing packages are contained in the javax.swing package. The package name javax indicates a JAVA extension package and not a core package.
- In the program we define a class, FirstTestFrame, that behaves exactly like the class JFrame in the javax.swing package with two exceptions.
- The constructor of the of FirstTestFrame sets the title bar to the string "FirstFrame" and the size of the frame to 300 X 200 pixels.
- The frame has no other methods.
- The main method just creates an instance of the frame, set the visibility and exits. Note that it does not terminate the program, just the main thread.

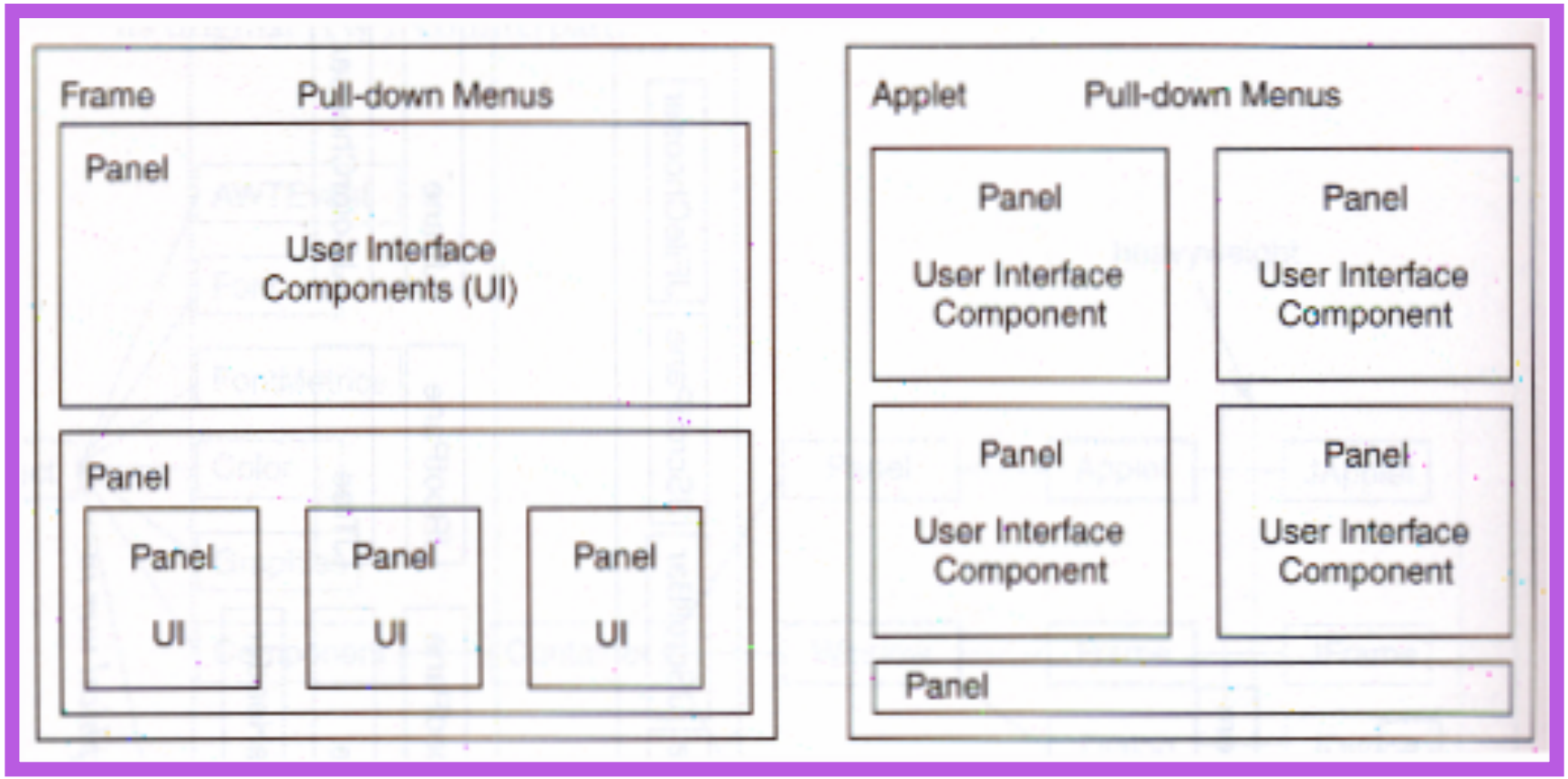
Creating a Frame (2)

- To actually show a frame, you perform the following steps.
  - Create a frame object by a call to new.
  - Optionally position it on screen by using the setLocation method.
  - Call the setVisible(true) method to make the frame visible and to bring it to the front if it behind another window.
- The key line in the main method is
  - JFrame frame = new FirstTestFrame();

  This line makes a new frame by giving all the necessary information to the underlying windowing system to display a window.
- Remember that new does not display the frame. We must use the setVisible method to make it appear on screen (frames are invisible by default).
- But we're not finished. The default size for a new frame is 0x0 pixels. We must use the setSize method to give the frame (window) a size.
- We haven't taken into account a method for terminating the window. We'll do that later when we examine event handling!

Frame Contents

- A frame or an applet can contain various components including menus, panels, and user interface components.
- Panels are used to group user interface components usually within the frame.
- Panels can contain other panels making an organized hierarchy of components.



Drawing on a Frame?

- It would be possible to draw on or place buttons, labels etc, directly onto a frame. However this is not considered good programming practice. In JAVA frames are designed to be containers for components.
- Normally we draw/place objects on another component which we call a panel. This is then added to the frame.
- A frame is made up of many panes (menu bar, content, root etc). For the time being we are only concerned with the content pane.
- When developing, we add components into the content pane, using code such as the following…you need to get the content pane reference using the get method

```
Container contentPane = frame.getContentPane();
Component c = ……;
contentPane.add(c);
```

Panels inside frames - JPanel

- Within a frame there can be many components (the more complex the GUI the more components are added)
- A JPanel is used as a grouping component, we can better organize complex GUI's by grouping into panels:
- Note the use of a JPanel in the sample program:
  - LabelFrame.java.
- To add a component to a JPanel use the *add* method and the name of the component to add to it (see the sample code).

Our first 'visible' component - JLabel

- Labels provide text instructions or information on a GUI. Labels are defined with the class JLabel – a subclass of JComponent.
- A JLabel is a single line label. Among many other things a JLabel has is the ability to:
  - Add an Icon (you can see an image in the label)
  - Set the vertical and horizontal position of text relative to the Icon
  - Set the relative position of contents within the component
- Our first simple JLabel program is available for review in the sample2 folder
  - LabelFrame.java.

Conclusions regarding the JLabel sample

- We use both the SWING and AWT classes.
- Our class LabelFrame is based on JFrame
- We call the super class constructor to give our frame a name in the title bar using the code

  super("My Frame"); //could use setTitle("My Frame")
- We can get the content pane by using the code Container

  Container contentPane = getContentPane();
- We create a panel for placing GUI components on by using the code

  JPanel panel = new JPanel();

Conclusions regarding the JLabel sample (1)

- We create the first label using the code

```
JLabel plainLabel = new JLabel("Plain Small Label");
panel.add(plainLabel);
```

- Notice that we add the label to the panel and not to the content pane. At a later stage we will add the panel with the GUI components to the content pane.

- We create a new font for our next label using the font class (this is part of the AWT package and so the AWT package must be imported).

```
Font fancyFont = new Font("Serif", Font.BOLD | Font.ITALIC, 32);
```

- We then set the font for the label using the code (not always needed just looks nice!)

```
fancyLabel.setFont(fancyFont);
```

Conclusions regarding the JLabel sample (2)

- We are going to add an icon to the label (again this is not always needed but just show for demo purposes). This involves two steps
  - Create an instance of the icon class
  - Assign an image to the instance of the icon class
  - Add the icon instance to the label object using the setIcon method.

```
// Create an Icon
        Icon tigerIcon = new ImageIcon("bug1.gif");

// Place the Icon in the label
        fancyLabel.setIcon(tigerIcon);

// Align the text to the right of the Icon
        fancyLabel.setHorizontalAlignment(JLabel.RIGHT);
```

Conclusions regarding the JLabel sample (3)

- We add the second label to the panel using the code

  panel.add(fancyLabel);

  Now that we have added all of our GUI components to the panel we will add the panel to the frame.

  contentPane.add(panel);

- We then set the size of the frame and display it on screen.

  setSize(300,200);
  setVisible(true);

Buttons - JButton

- A button is a component that a user clicks to trigger a specific action. A JAVA program can use several types of buttons, including command buttons, check boxes, toggle buttons and radio buttons.
- The SWING version of a button is called JButton.
- As with labels, we can specify a size, text and an icon.
- To view a simple button application view the sample contained in sample 3 - ButtonFrame.java

Expanding JButton

- As with JLabel, we can add an icon to a button.
- To view a simple button application with an icon view the sample contained in sample 4 - ButtonIcon.java
- Notice how we add the icon. We do not use the setIcon function. Instead we use the constructor by passing it values for a string and an icon (you could use setIcon instead, the effect if the same):

```
JButton myButton = new JButton("The Bug", bugIcon);
```

- JButton and JLabel are two very simple yet powerful GUI components. Just take a look at all of the methods and properties available to us through their use.
- For a full list of methods/properties make sure to check the JDK documentation online at the Oracle website (e.g. https://docs.oracle.com/javase/7/docs/api/javax/swing/JButton.html)

Text Fields – JTextField & JTextArea

- A text field is an input area where the user can type in characters. Text fields enable the user to enter variable data, such as names or descriptions (JTextField).

- A text area allows the user to enter multiple lines of text (JTextArea).

- Both of these are sub classes of JTextComponent, a generalized text class that contains all the features you would expect from a simple editor. Some of its methods include:

copy()	cut()	paste()	getSelectedText()
setSelectionStart()	setSelectionEnd()	selectAll()	replaceSelection()
getText()	setText()	setEditable()	setCaretPosition()

Text Fields – JTextField & JTextArea (2)

- JTextComponent objects in SWING can be placed in a panel.
- There are three basic subclasses of JTextComponent: JTextField, JTextArea, and JEditorPane. JPasswordField and JTextPane are sub-subclasses that are also of interest.
- If you want your users to be able to see content that exceeds the screen display area, you must place the component inside of a JScrollPane to support scrolling to the extra content.
- Other than having to add a JTextArea to a JScrollPane for scrolling, JTextField and JTextArea behave very similarly to their AWT counterparts: java.awt.TextField and java.awt.TextArea:

Text Fields – JTextField & JTextArea (3)

- To view a simple text application view the sample contained in sample 5 - TextDemo.java

- The main methods that we use are the setText methods.

```
tf.setText("TextField");
ta.setText("JTextArea\n Allows Multiple Lines");
```

- Note that we added the text area to a scroll pane using the code

```
panel.add(new JScrollPane(ta));
```

Summary

- **The SWING API was introduced to improve the process of creating graphical user interfaces.**
- **It makes use of lightweight components, many of which inherit methods/ behaviours from AWT classes.**
- SWING offers developers a number of advantages including
  - Swing has a much richer and convenient set of user interface elements.
  - Swing depends much less on the underlying platform, therefore is much less prone to platform specific bugs.
  - SWING will give a consistent user experience across platforms.

Summary (2)

- Swing is built on top of AWT, it DOES NOT REPLACE AWT.
- Some of the more common GUI components we have examined include JLabel, JButton, JTextField and JTextArea.
- Before we can draw a SWING component on screen we need a window to place it into.
- A top level window (that is, a window that is not contained inside of another window) is called a frame in JAVA.
- The SWING version of this class is called JFrame.
- Frames are examples of containers. This means that a frame can contain other user interface components such as buttons and text fields.

Summary (3)

- Normally we draw/place objects on another component which we call a panel. This is added to the frame.
- Labels provide text instructions or information on a GUI. Labels are defined with the class JLabel – a subclass of JComponent.
- A button is a component that a user clicks to trigger a specific action. A JAVA program can use several types of buttons, including command buttons, check boxes, toggle buttons and radio buttons.
- The SWING version of a button is called JButton.

Summary (4)

- A text field is an input area where the user can type in characters. Text fields enable the user to enter variable data, such as names or descriptions (JTextField).
- A text area allows the user to enter multiple lines of text (JTextArea).
- Both of these are sub classes JTextComponent. is a generalized text class that contains all the features you would expect from a simple editor.
- Extensive documentation is available on the SWING API and is available in the JAVA documentation on the Oracle website

Finding out more about the components

- Every component literally has hundreds of methods associated with them, for example, setting size, visibility, icons etc....there are too many options and methods to cover all options in a course

- All of the components we will study this semester are explained in detail in the Java documentation found on the Oracle website (e.g. JButton below):

https://docs.oracle.com/javase/7/docs/api/javax/swing/JButton.html

- If you do a search for a component name, e.g., 'JFrame' in the oracle website (or even just in google) you will be forwarded to the documentation page which contains everything you need to know about the component including all of the methods associated with the component

Basic GUI heuristics

- We don't simply throw a collection of elements on screen
- Good interface design is vital
- In IT projects, it can be a time and capital intensive phase of a project.
- A bad GUI can mean that a product is a poor seller.
- A good GUI can mean commercial success (e.g., iPhone)
- We don't always notice a well designed interface but we notice bad ones!
- Lets take a look at a few heuristics (rules of thumb) for good interface design that you can keep in mind while developing your GUI's this semester!

Ten Usability Heuristics

1. Visibility of system status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time. Examples include

- Dialogue and message boxes
- Loading/copying/please wait messages etc
- Warning signals, flashing symbols etc

2. Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

- People don't speak like programmers!
- Avoid complex language where possible
- Don't surprise people. Try to build systems that users are at ease with.

3. User control and freedom

- Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

4. Consistency and standards

– Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

- For example, an application you create for Microsoft windows, should behave like a windows application. User standard menu items like File, Edit, New etc

5. Error prevention

- Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

  - Question: Does your bank's ATM let you withdraw more money than you have?

6. Recognition rather than recall

– Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

- Assume your users have the memory of a goldfish ☺

7. Flexibility and efficiency of use

– Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

- In Windows, pressing the start key and 'm' at the same time minimises all windows!

8. Aesthetic and minimalist design

– Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

9. Help users recognize, diagnose, and recover from errors

- Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
  - e.g insufficent funds for this transaction
  - Short, simple and to the point

10. Help and documentation

– Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Labwork 4 is now on Moodle !!