GUI Programming

Session 1
Introduction to GUI Design

Lecturer: Luke Raeside, PhD

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

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

- Why limited to computers
  - 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 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!!! ☺

- 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 (IPod & ITunes)
- 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.

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.

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

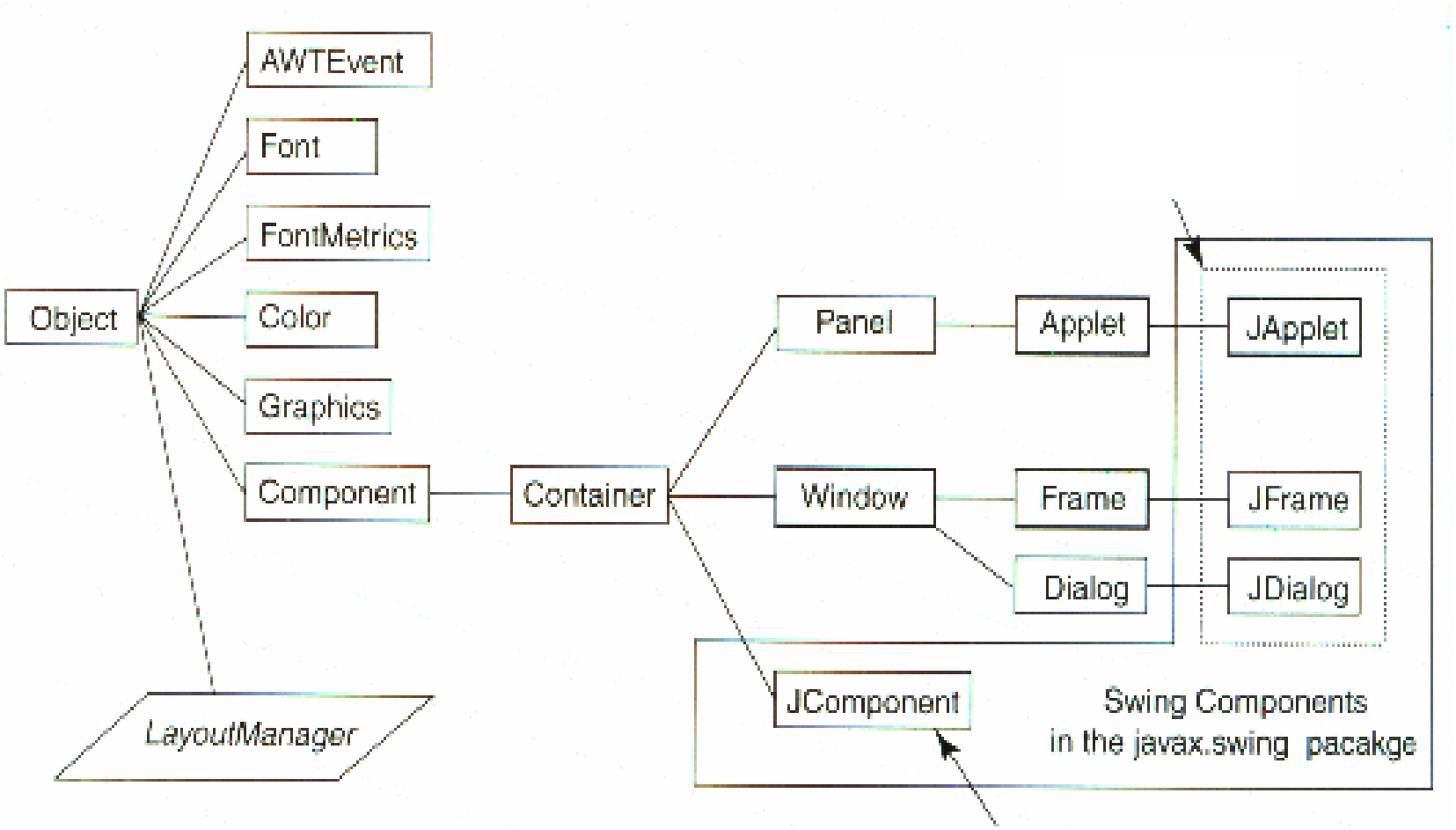
SWING versus AWT (2)

- 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 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

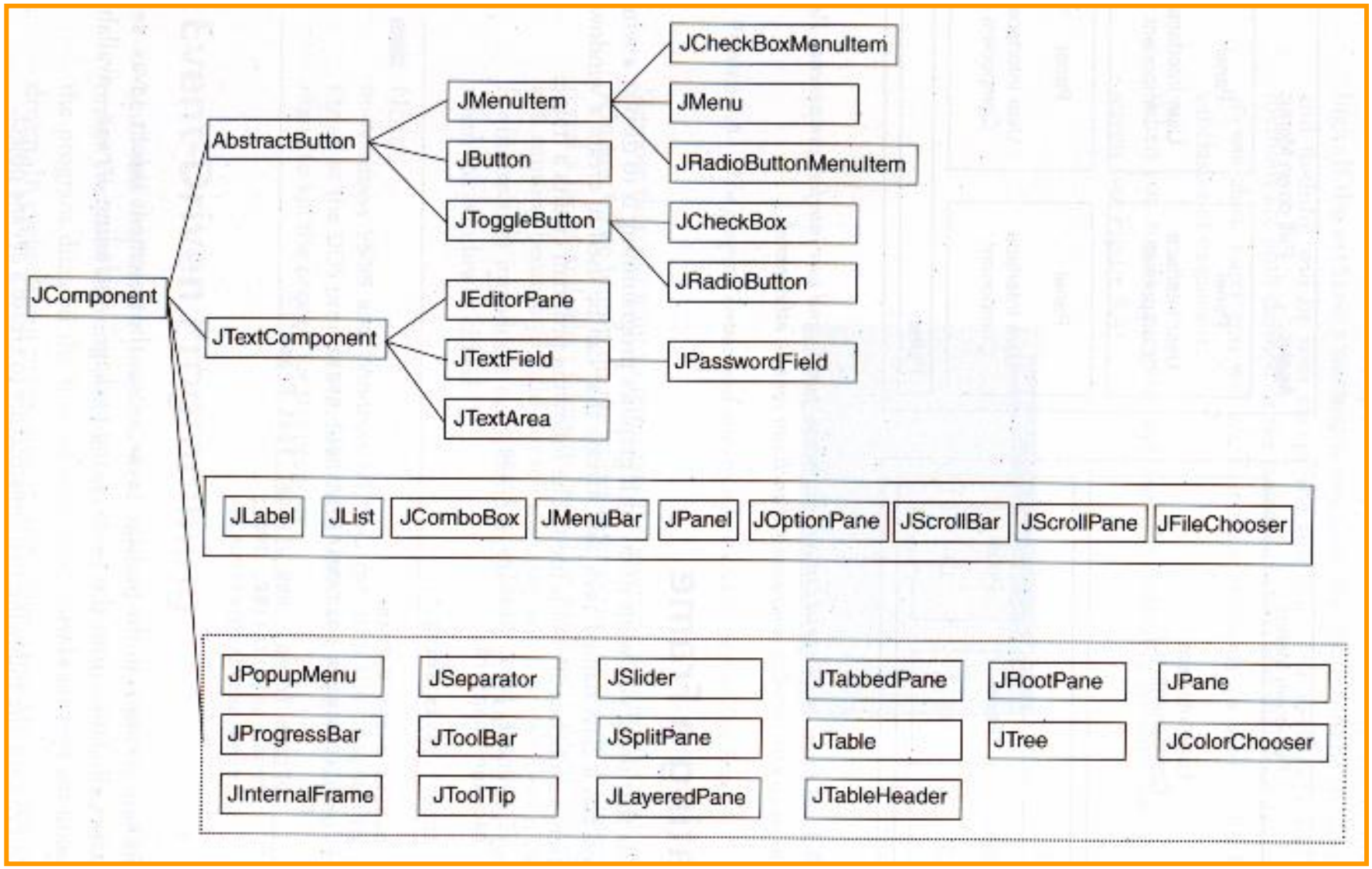
- As can be seen from the diagram on the previous page:
- Swing is built on top of AWT, it **DOES NOT REPLACE AWT**
- Swing uses AWT graphics, colors, fonts, and the LayoutManagers.
- Swing does not use AWT components with the exception of extending the Frame, Window and Dialog heavyweight components to create JFrame, JWindow and JDialog (these are still heavyweight in Swing)
- Like all good OO systems Swing **RE-USES** the best of AWT in order to build more lightweight components
- Given the "IS-A" relationship of OOP it's should be deducible from the diagram that every Swing component "IS-A" AWT Container (Since all Swing Components derive from JComponent)

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

SWING Classes



SWING Components to examine

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

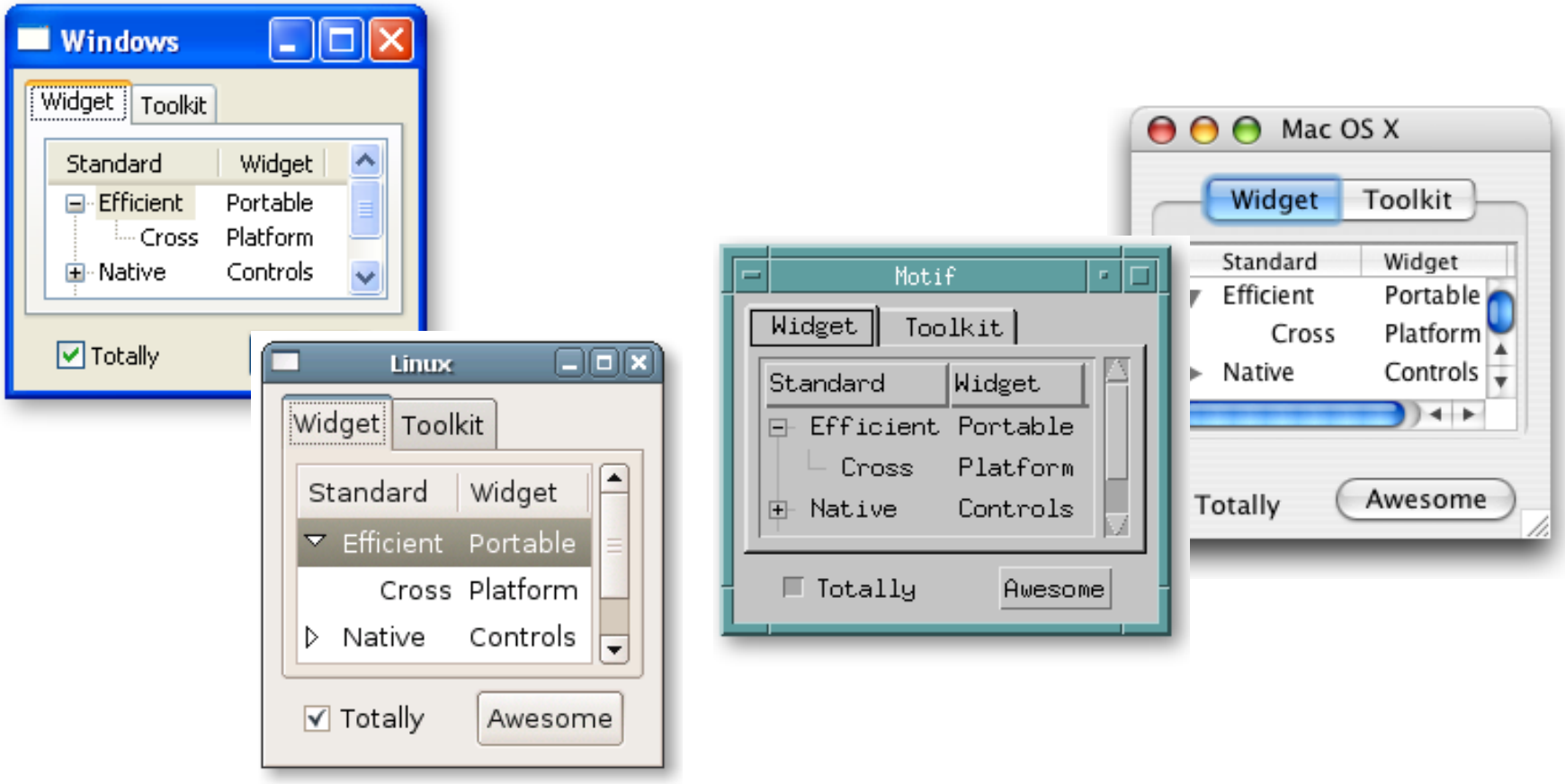
JLabel	An area where un-editable text or icons can be displayed.
JTextField	An area in which the user inputs data from the keyboard. The area can also display information.
JButton	An area that triggers an event when clicked.
JCheckBox	A GUI component that is either selected or not selected.
JComboBox	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

There is a new kid on the block.

- 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.
- It's in it's early stages but 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!

SWING SET DEMO

- The SWING set demo shows off all of the components available in SWING

- http://java.sun.com/products/plugin/1.4/demos/plugin/jfc/SwingSet2/SwingSet2.html

Labwork:

- Try out the demo mentioned in the previous slide
- Write a summary paragraph in your own words about the difference between AWT and SWING in Java