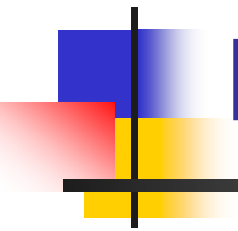
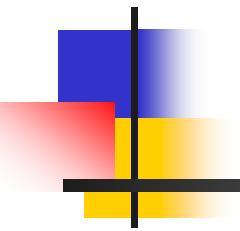


Block I – Unit 5

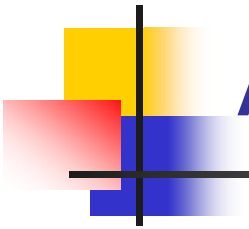
Swing, AWT and Applets



Section 1

Swing and AWT Packages

Based on Budd chapter 13 and section 6 of the IDE



Aims of this unit

- The aims of this unit are to:
  - Investigate Java user-interface components, especially Swing components.
  - Introduce applets.
  - Study the classes for inputting and outputting data.

Swing and AWT Packages

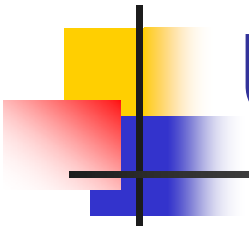
- The Abstract Windowing Toolkit (AWT) provides basic facilities for drawing graphics.
- Like the AWT, Swing provides GUI components — in fact they have similar names and functions.
- The main difference is that while the AWT components vary according to the underlying operating system, Swing components do not.
- For example, if you create an AWT button it will look like a Windows button on Windows based PC and a Macintosh button on a Macintosh.
- On the other hand, a Swing component will look the same on any platform. Swing is also larger and more comprehensive than AWT.

Study Activities

- Human-computer interaction.

- Factors to consider in constituting a successful HCI design.

  - Guidelines for usability. (Human factors)
  - Prototypes. (Look and feel)
  - Usability testing. (Feedback on the prototype)

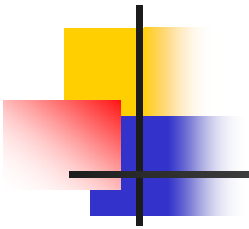


Usability Guidlines

- General interaction which includes:
  - Being consistent (menus, commands, input)
  - Minimizing dialogue and keystrokes
  - Providing help facilities
- Information display which includes:
  - Displaying relevant information
  - No overwhelming user with data

Usability Guidlines

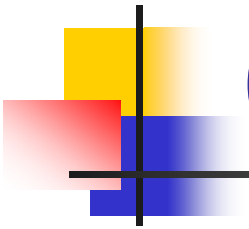
- Data input, which includes:
  - Predefined selections for the user to choose from
  - Deactivating commands that are inappropriate in the context of current actions.



IDE / GuiBuilder

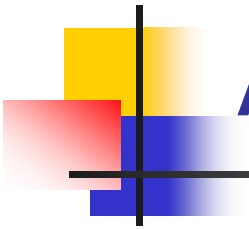
- The IDE used in this course have *GUI builders* facilities.
- This makes it much easier to create interfaces by providing a palette of components which can be ‘dragged and dropped using the mouse.
- It makes it easy to link actions to events, and so on.
- *Do section 6 in the IDE Handbook*

Hash Code and Anonymous (unnamed) classes



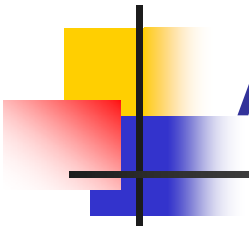
- Hash code:
  - Is an integer computed from an object.
  - Not guaranteed to be unique.
- Anonymous (unnamed) class:
  - If the name following new has the same name as an existing class, it is assumed that the anonymous class extends that class.
  - p.add(new ButtonAdapter("Quit") {

    public void pressed() {System.exit(0); }});



AWT Class Hierarchy

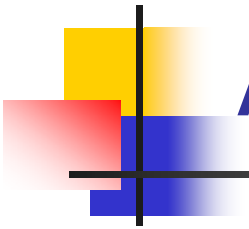
- AWT class hierarchy refer to Budd ch.13 p.212
- Class object is the parent of all classes in java. Some of its methods:
  - Equals(anObject)
  - getClass()
  - hashCode()
  - toString()



AWT Class Hierarchy

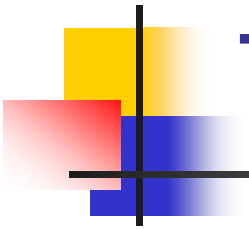
- A component is something that can be displayed and user can interact with. Some of its methods:
  - setLocation, getLocation, setSize etc.

- A container is a component that can nest other components. Some of its methods:
  - setLayout, add (component), remove (component)



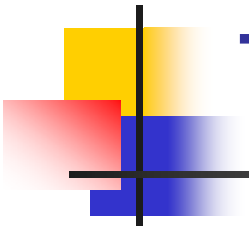
AWT Class Hierarchy

- A window is a type of Container that can be displayed, stacked on the top of other widows or moved to the front of back of visible windows. Some of its methods:
  - Show(), toFront(), toBack()
- A Frame is a type of Window with title bar, a menu bar, a border and others. Some of its methods:
  - setTitle(String), setMenuBar(MenuBar) and so on.
- *More methods are in Budd ch.13 p.212-214*



The Swing Class Hierarchy

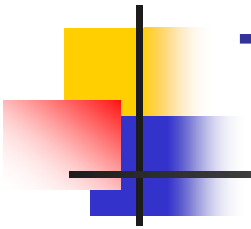
- AWT has been augmented with a newer library called Swing.
- The Swing class JFrame is a subclass of the earlier AWT class Frame.
- This can be shown by finding the definition of JFrame and Frame and check out the packages they are in.
- Swing class hierarchy refer to Budd ch.13 p.218



The Layout Manager

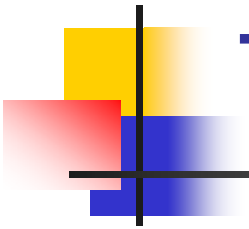
- The idea of a layout manager (technique used by the AWT) is to assign locations of components within a container.

- It is in charge of assigning positions on the surface of a container to the components held in that container.



The Layout Manager

- There are a variety of layout managers, each places components in different ways.
- Programmer creates an instance of a layout manager and hands it to the container.
- The container holds the layout manager as a data field.



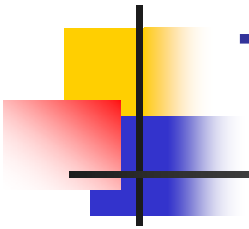
The Layout Manager

- Illustration of polymorphic variable.

- While Container thinks it is maintaining a value of type LayoutManager, it is in fact holding a value of type GridLayout that is implementing LayoutManager interface.

The Layout Manager

```
private JPanel makeScrollBars() {
            JPanel p = new JPanel();
                p.setLayout(new GridLayout(1, 3, 3, 3));
                        ……..
public class JPanel extends JComponent implements Accessible
{……..
public abstract class JComponent extends Container implements
    Serializable
{……..
public class Container extends Component
{
……..
 LayoutManager layoutMgr;
……..
```

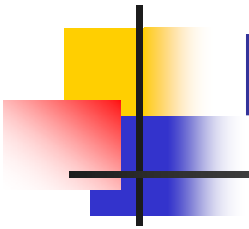


The Layout Manager

```
public void setLayout(LayoutManager mgr) {
    layoutMgr = mgr;
    ........
}
```

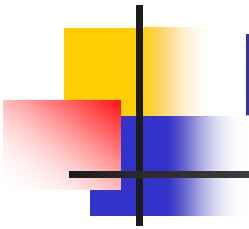
- The type of layout manager could be changed dynamically.

- Programmer can develop alternative layout managers since Layout Manager is an interface



Layout Manager Types

- There are five types:
- BorderLayout: (Default)
  - Can manage no more than 5 components.
- GridLayout:
  - Rectangular array of components.
  - Programmer specify rows, columns and spaces between them.



Layout Manager Types

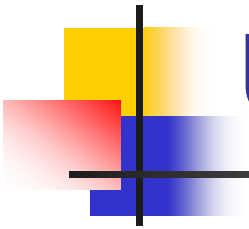
- FlowLayout:
  - Places components in rows left to right and top to bottom.
- CardLayout:
  - Stacks components vertically. Only one can be visible at one time.
- GridBagLayout:
  - For non-uniform grid of squares and places components in different positions.

User Interface Components

- Labels:
  - JLabel lab = new JLabel(“score: 0 to 0”);
  - getContentPane()add (“South”, lab);
    - setText(String) and getText(String)

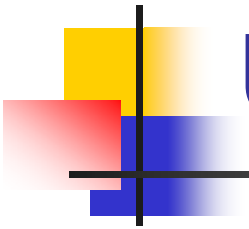
- Button:
  - Respond to user interaction by attaching an ActionListener object.

```
JButton b = new JButton (“do it!”);
b.addActionListener (new DoIt());
.....
Private class DoIt implements ActionListener {
    Public void actionPerformed (ActionEvent e) {
    ........
    }
}
```



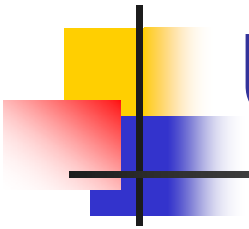
User Interface Components

- Scroll Bars:
  - A JScrollBar is a slider used to specify integer values over a wide and specified range.
- Text Components:
  - It is used to display editable text.
  - JTextField and TextArea



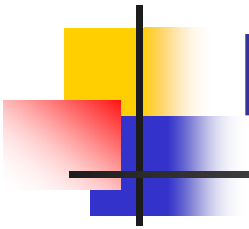
User Interface Components

- Checkbox:
  - JCheckBox maintains and displays an on/off state.
- ButtonGroup:
  - Like a radio button group, one can be selected
- JComboBox:
  - A pop up menu allows you to choose from a selection



User Interface Components

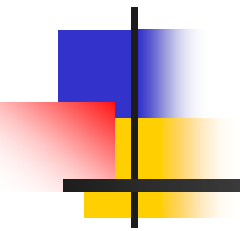
- JList:
  - Similar to JComboBox, but selection can be displayed at one time.



Panels / JScrollPane

- A JPanel is a container acts like a component and hold its own layout manager.

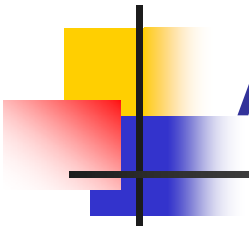
- A JScrollPane is similar to JPanel, it can hold only one component and does not have a layout manager.



Section 2

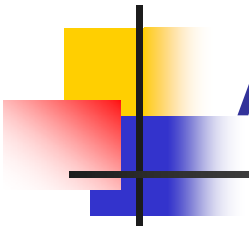
Applets

Based on Budd chapter 21 and section 7 of the IDE



Applets and HTML

- Do section 7 of the IDE Handbook.
- Applets are applications written for the WWW.
- Applets are attached to the documents distributed over the WWW.
- These documents are written using the HyperText Markup Language (HTML)



Applets and HTML

- A web browser that includes a Java processor will then retrieve and execute the Java program.

- Two HTML tags are used <applet> and <param>.
- Example on the next slide

Applets and HTML

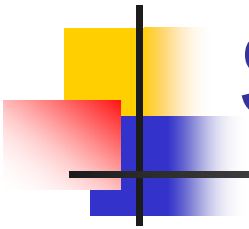
<applet codebase=http://www.sun.com code=Main
width=300 height=200>

<param name=name1 value="value1">

You do not have a Java enabled browser

</applet>

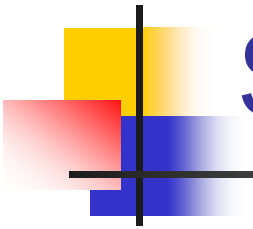
- Specifying: Address, URL, Class and space and parameters



Security Issues

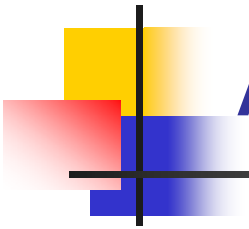
- Applets are loaded from a remote computer (server) and executed locally.

- To ensure that applets do not do any damage to the local computer, they are restricted and can perform the following:



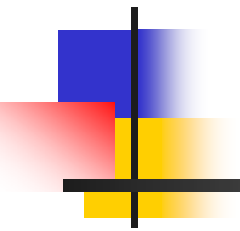
Security Issues

- Applets are not permitted to run any local executable program.
- Cannot read or write to the local computer's file system.
- Can only communicate with the server from which they originate
- Can learn the name and version of the OS, but not the user's name or e-mail address.



Applets and Applications

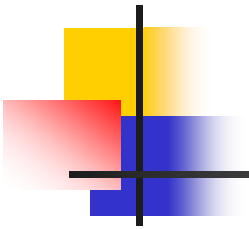
- init(), start(), stop() and destroy().
- First init() and start(), if a link is clicked, the applet is overwritten, but still available to come back to.
- When user returns, start() is invoked but not init().
- When user exit the page containing the applet, destroy() is invoked.



Section 3

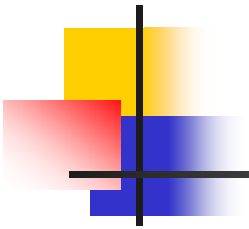
Input and Output Streams

Based on Budd chapter 14



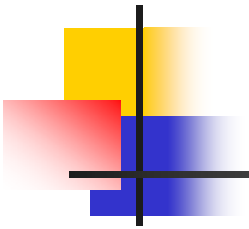
Input and Output Stream

- Binary representation:
  - Data type int uses four bytes
  - short uses two bytes
  - Floating point uses four bytes and so on.
  - ASCII character uses one byte of 256 different characters
  - Unicode uses two bytes to internationalize the character set.



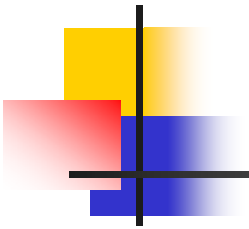
Input and Output Stream

- Sources and Sinks:
  - Sources are places/devices from which data is fetched
  - Sinks or destination are places to which data is sent.
  - Stream is a communication channel between the program and source or sink. It is a narrow conduit for passing data.



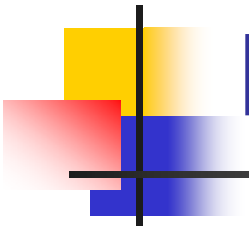
Input and Output Stream

- There are two large systems involved in I/O InputStream and OutputStream
- The hierarchy rooted in the above two classes are used to read and write 8-bit unit.
- Alternative hierarchy rooted in the classes Reader and Writer are used to read and write 16-bit Unicode character values.



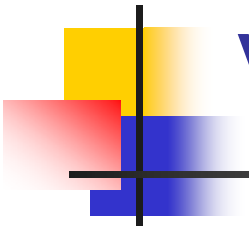
Input and Output Stream

- The class InputStream is an abstract class, parent of 10 subclasses. Budd p. 240.
- There are two variety of input stream classes:
  - Physical: These read values from byte array or a file.
  - Virtual: Reads values from another input stream and allows program to unread characters into the input stream.



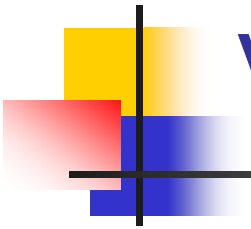
Physical Input Streams

- Three input stream classes that read from data areas.
- They are distinguished by their names and the arguments used in their constructors.
  - ByteArrayInputStream (byte [ ] buffer);
  - ByteArrayInputStream(byte [ ] buffer, int offset, int count);
  - FileInputStream (File f);
  - FileInputStream (String fileName);
  - PipedInputStream (PipedOutputStream p);



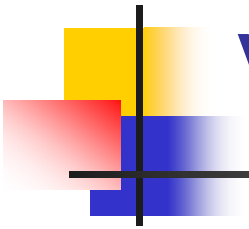
Virtual Input Stream

- These classes rely on one or more input streams for their data source.
- SequenceInputStream takes two or more input streams and places them end to end.
  - InputStream f1 = new FileInputStream (“file1.text”);
  - InputStream f2 = new FileInputStream (“file2.text”);
  - InputStream f3 = new SequenceInputStream (f1, f2);



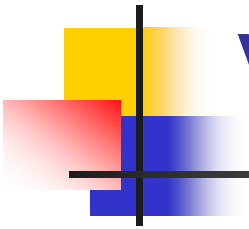
Virtual Input Stream

- Another example:
  - Vector fv = new Vector ();
  - fv.addElement (f1); fv.addElement (f2);
  - InputStream f4 = new SequenceInputStream (fv.elements());



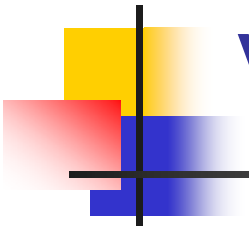
Virtual Input Stream

- The class ObjectInputStream is used to provide serialization.
- Object can be converted into representation that could be transmitted as a sequence of 8-bit values.
- These values are then read from or written to a stream.



Virtual Input Stream

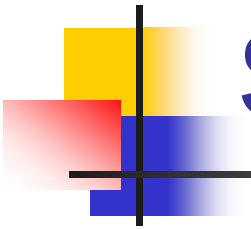
- The PushbackInputStream allows a single character to be unread.
  - Imagine reading the textual representation of the a numeric value 456. We would know that all digits have been seen after reading the first nondigit character.
- A BufferedInputStream reads values from an input stream in large blocks.
  - Responds to read operation by returning characters from its internal buffer.



Virtual Input Stream

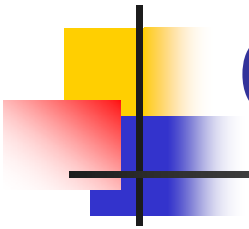
- The subclasses of FilterInputStream sit between the client (code) and the physical input stream producing values.

- Methods of the subclass DataInputStream read bytes from the source and return them as primitive data type.



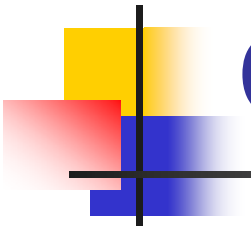
Stream Tokenizer

- Used to break apart string values.
- It can recognize words (sequence of letters separated by nonletter characters) and numbers.



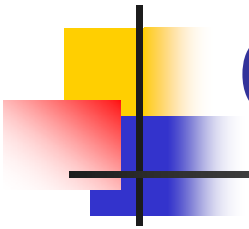
OutputStream

- The class hierarchy rooted in the class OutputStream is given in Budd p.245
- Many of the subclasses of OutputStream will buffer their values.
- They are divided into two categories:
  - Classes that characterize the physical location
  - Classes that add more behavior to an OutputStream



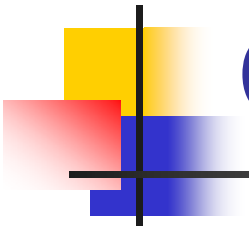
OutputStream

- The first category:
  - ByteArrayOutputStream:
    - Writes values into an in-memory byte array
  - FileOutputStream:
    - Writes values on to an external file
  - PipedOutputStream:
    - Writes values on to a pipe



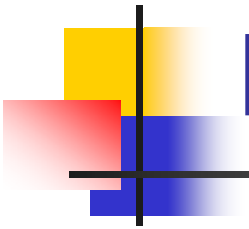
OutputStream

- The second category:
- ObjectOutputStream:
  - Designed for writing object values to a stream so it can be read back in using ObjectInputStream
- FilterOutputStream and its subclasses:
  - BufferedOutputStream
  - DataOutputStream
  - PrintStream



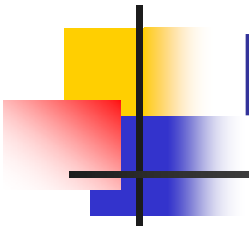
Object Serialization

- Is the process of converting an object into a representation that can be accommodated in an 8-bit units.
- It provides object persistence.
- It is essential to network programming.
- Budd p. 248 gives an illustration.



Piped Input and Output

- A pipe is a buffered data area that is used for both reading and writing.
- It can hold a limited number of values.
- A write will be suspended if the current pipe buffer is full.
- A read operation will be suspended if the buffer is empty.
- Budd p. 251 gives an illustration.



Readers and Writers

- The class hierarchies rooted at Reader and Writer are in Budd p.255

- These classes manipulate 16-bit Unicode character values.