AWT in Java

MSc (IT) Semester 4th

Gurmeet Singh

PG Department of Computer Science & IT
Hans Raj Mahila Maha Vidyalaya, Jalandhar
gurmeethmv@gmail.com

Syllabus

MIT-401 Advanced Java Technology

Section A

Java I/O: I/O Basics, Streams, reading Console input and writing console output, Print Writer Class, Reading & Writing Files, Byte Streams, Character Streams & Serialization.

Section B

Multithreaded Programming: The Java Thread Model, Thread Priorities, Synchronization, Interthread communication, Suspending Resuming and Stopping Threads.

Applets: Applet Basics, Applet Architecture, Applet: Display, Repaint, Parameter Passing.

Section C

Event Handling: The Delegation Event Model, Event Classes, Event Listener Interfaces AWT: Window Fundamentals, Working with Frame Windows, Graphics, Color and Fonts.

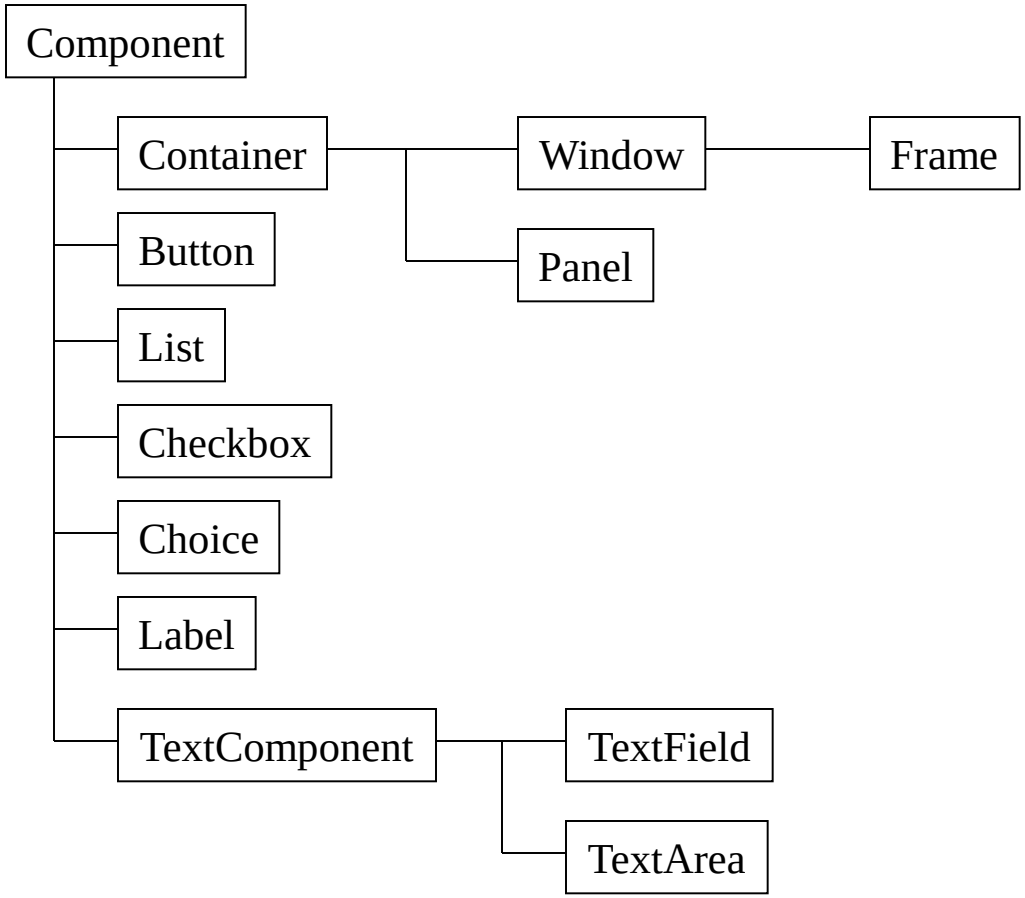
Section D

Servlets: Life Cycle of a Servlet, The Servlet API, Reading Servlet Parameters, Handling HTTP Requests and Responses, Cookies & Session Tracking.

AWT (Abstract Window Toolkit)

- The AWT is roughly broken into three categories
  - Components
  - Layout Managers
  - Graphics
- Many AWT components have been replaced by Swing components
- It is not a good idea to mix Swing components and AWT components. Use one or the other.

AWT – Class Hierarchy



Component

- Component is the superclass of most of the displayable classes defined within the AWT and it is abstract.
- MenuComponent is another class which is similar to Component except it is the superclass for all GUI items which can be displayed within a drop-down menu.
- The Component class defines data and methods relevant to all Components

```
setBounds
setSize
setLocation
setFont
setEnabled
setVisible
setForeground          -- colour
setBackground          -- colour
```

Container

- Container is a subclass of Component. (ie. All containers are themselves, Components)
- Containers contain components
- To place a component on the screen, it must be placed within a Container
- The Container class defined all the data and methods necessary for managing groups of Components
  - add
  - getComponent
  - getMaximumSize
  - getMinimumSize
  - getPreferredSize
  - remove
  - removeAll

Windows and Frames

- The Window class defines a top-level Window with no Borders or Menu bar.
  - Usually used for application screens
- Frame defines a top-level Window with Borders and a Menu Bar
  - Frames are more commonly used than Windows
- Once defined, a Frame is a Container which can contain Components

```
Frame aFrame = new Frame("Hello MSCIT");
aFrame.setSize(200,100);
aFrame.setLocation(20,20);
aFrame.setVisible(true);
```

Panels

- When writing a GUI application, the GUI portion can become quite complex.
- To manage the complexity, GUIs are broken down into groups of components. Each group generally provides a unit of functionality.
- A Panel is a rectangular Container whose sole purpose is to hold and manage components within a GUI.

```
Panel aPanel = new Panel();
aPanel.add(new Button("Ok"));
aPanel.add(new Button("Cancel"));

Frame aFrame = new Frame("Button Test");
aFrame.setSize(100,100);
aFrame.setLocation(10,10);

aFrame.add(aPanel);
```

Buttons

- This class represents a push-button which displays some specified text.
- When a button is pressed, it notifies its Listeners.
- To be a Listener for a button, an object must implement the ActionListener Interface.

```
Panel aPanel = new Panel();
Button okButton = new Button("Ok");
Button cancelButton = new Button("Cancel");

aPanel.add(okButton));
aPanel.add(cancelButton));

okButton.addActionListener(controller2);
cancelButton.addActionListener(controller1);
```

Labels

- This class is a Component which displays a single line of text.
- Labels are read-only. That is, the user cannot click on a label to edit the text it displays.
- Text can be aligned within the label

```
Label aLabel = new Label("Enter password:");
aLabel.setAlignment(Label.RIGHT);

aPanel.add(aLabel);
```

List

- This class is a Component which displays a list of Strings.
- The list is scrollable, if necessary.
- Sometimes called Listbox in other languages.
- Lists can be set up to allow single or multiple selections.
- The list will return an array indicating which Strings are selected

```
List aList = new List();
aList.add(“RED");
aList.add(“GREEN");
aList.add(“BLUE");
aList.add(“YELLOW");

aList.setMultipleMode(true);
```

Checkbox

- This class represents a GUI checkbox with a textual label.
- The Checkbox maintains a boolean state indicating whether it is checked or not.
- If a Checkbox is added to a CheckBoxGroup, it will behave like a radio button.

```
Checkbox Checkbox1 = new CheckBox(“MALE");
Checkbox Checkbox2 = new CheckBox(“FEMALE");

if (Checkbox1.getState())
{
        //
}
```

Choice

- This class represents a dropdown list of Strings.
- Similar to a list in terms of functionality, but displayed differently.
- Only one item from the list can be selected at one time and the currently selected element is displayed.

```
Choice aChoice = new Choice();
aChoice.add("Calcutta");
aChoice.add("Jalandhar");
aChoice.add("Delhi");
[]..

String selectedDestination= aChoice.getSelectedItem();
```

TextField

- This class displays a single line of optionally editable text.
- This class inherits several methods from TextComponent.
- This is one of the most commonly used Components in the AWT

```
TextField emailTextField = new TextField();
TextField passwordTextField = new TextField();
passwordTextField.setEchoChar('*');
[…]

String userEmail = emailTextField.getText();
String userpassword = passwordTextField.getText();
```

TextArea

- This class displays multiple lines of optionally editable text.
- This class inherits several methods from TextComponent.
- TextArea also provides the methods: appendText(), insertText() and replaceText()

```
// 5 rows, 80 columns
TextArea fullAddressTextArea = new TextArea(5, 80);
[...]

String userFullAddress= fullAddressTextArea.getText();
```

Layout Managers

- Since the Component class defines the setSize() and setLocation() methods, all Components can be sized and positioned with those methods.

- Problem: the parameters provided to those methods are defined in terms of pixels. Pixel sizes may be different (depending on the platform) so the use of those methods tends to produce GUIs which will not display properly on all platforms.

- Solution: Layout Managers. Layout managers are assigned to Containers. When a Component is added to a Container, its Layout Manager is consulted in order to determine the size and placement of the Component.

Layout Managers (cont)

- There are several different LayoutManagers, each of which sizes and positions its Components based on an algorithm:
  - FlowLayout
  - BorderLayout
  - GridLayout
- For Windows and Frames, the default LayoutManager is BorderLayout. For Panels, the default LayoutManager is FlowLayout.

Flow Layout

- The algorithm used by the FlowLayout is to lay out Components like words on a page: Left to right, top to bottom.
- It fits as many Components into a given row before moving to the next row.

```
Panel aPanel = new Panel();
aPanel.add(new Button("Ok"));
aPanel.add(new Button("Add"));
aPanel.add(new Button("Delete"));
aPanel.add(new Button("Cancel"));
```

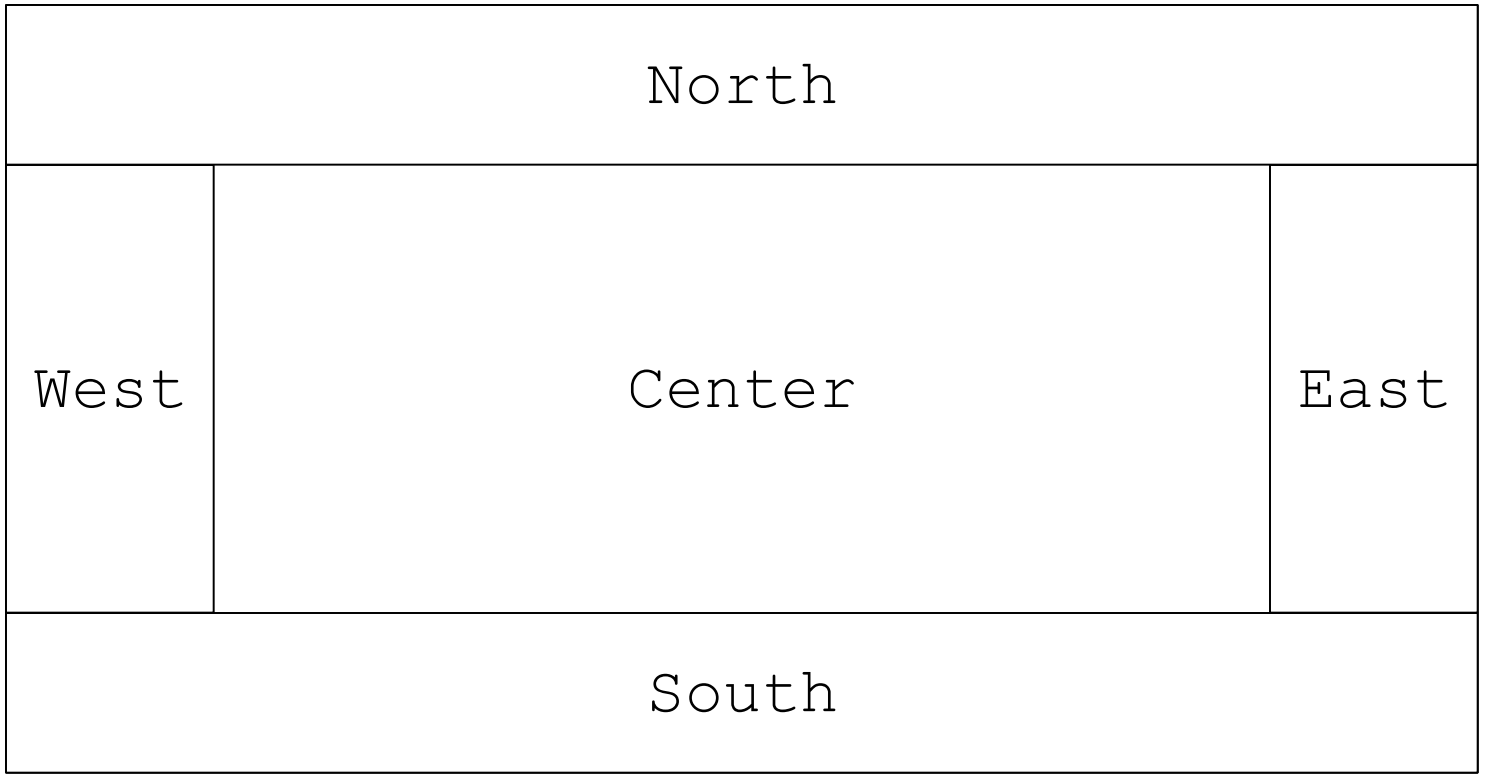
Border Layout

- The BorderLayout Manager breaks the Container up into 5 regions (North, South, East, West, and Center).
- When Components are added, their region is also specified:

```
Frame aFrame = new Frame();
aFrame.add("North", new Button("Ok"));
aFrame.add("South", new Button("Add"));
aFrame.add("East", new Button("Delete"));
aFrame.add("West", new Button("Cancel"));
aFrame.add("Center", new Button("Recalculate"));
```

Border Layout (cont)

- The regions of the BorderLayout are defined as follows:



Grid Layout

- The GridLayout class divides the region into a grid of equally sized rows and columns.
- Components are added left-to-right, top-to-bottom.
- The number of rows and columns is specified in the constructor for the LayoutManager.

```
Panel aPanel = new Panel();
GridLayout theLayout = new GridLayout(2,2);
aPanel.setLayout(theLayout);

aPanel.add(new Button("Ok"));
aPanel.add(new Button("Add"));
aPanel.add(new Button("Delete"));
aPanel.add(new Button("Cancel"));
```

What if I dont want a LayoutManager?

- LayoutManagers have proved to be difficult and frustrating to deal with.
- The LayoutManager can be removed from a Container by invoking its setLayout method with a null parameter.

```
Panel aPanel = new Panel();
aPanel.setLayout(null);
```

Graphics

- It is possible to draw lines and various shapes within a Panel under the AWT.
- Each Component contains a Graphics object which defines a Graphics Context which can be obtained by a call to getGraphics().
- Common methods used in Graphics include:

drawLine
drawOval
drawPolygon
drawPolyLine
drawRect
drawRoundRect
drawString
draw3DRect
fill3DRect
fillArc

•fillOval
•fillPolygon
•fillRect
•fillRoundRect
•setColor
•setFont
•setPaintMode
•drawImage

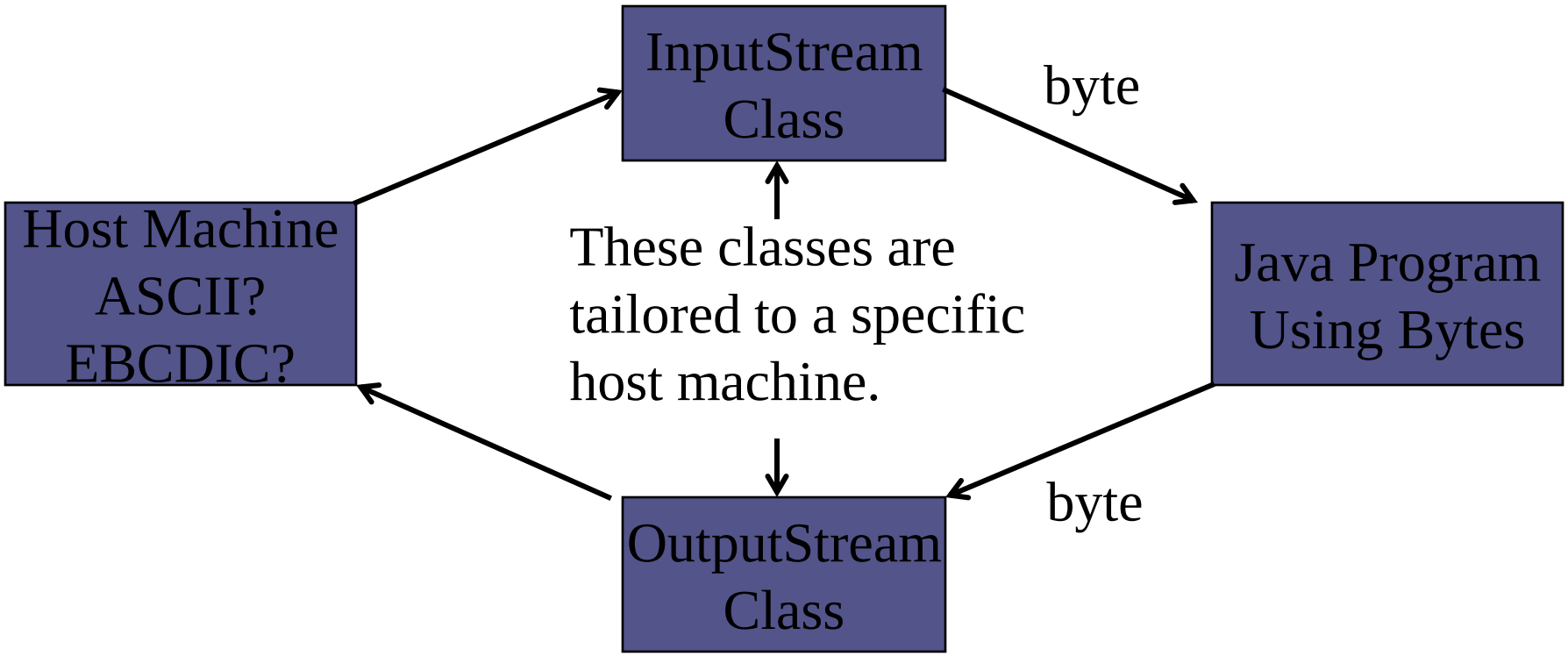
Java Input/Output

- I/O libraries are hard to design, and everyone can find something to complain about.
- Java's I/O library is extensive, and it seems like you need to know 100 things before you can start using it.
- Today, let's learn just five things and still do something useful.

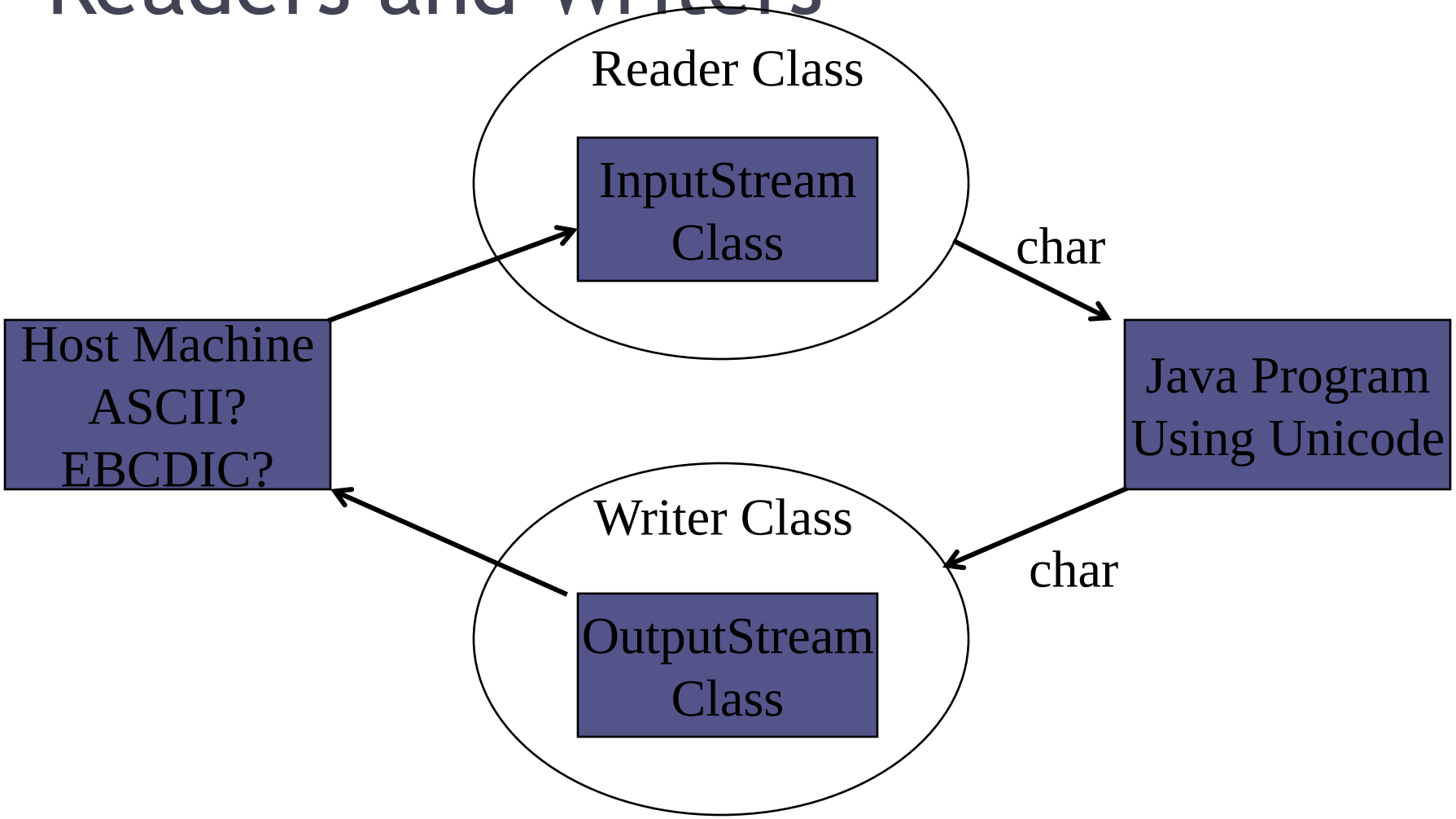
Java I/O

- Java is intended to be used on many very different machines, having
  - different character encodings (ASCII, EBCDIC, 7- 8- or 16-bit…)
  - different internal numerical representations
  - different file systems, so different filename & pathname conventions
  - different arrangements for EOL, EOF, etc.

Streams


InputStream Class
byte
Host Machine ASCII? EBCDIC?
These classes are tailored to a specific host machine.
Java Program Using Bytes
byte
OutputStream Class

Readers and Writers



Input Sources

- **System.in**, which is an **InputStream** connected to your keyboard. (**System** is **public**, **static** and **final**, so it's always there).
- A file on your local machine. This is accessed through a **Reader** and/or an **InputStream**, usually using the **File** class.
- Resources on another machine through a **Socket**, which can be connected to an **InputStream**, and through it, a **Reader**.

InputStreamReader

- This is a bridge between bytes and chars.
- The **read()** method returns an **int**, which must be cast to a **char**.
- **read()** returns -1 if the end of the stream has been reached.

Use a BufferedReader

```
public static void main(String[] args) {
    BufferedReader br =
        new BufferedReader(new InputStreamReader(System.in));
    String s;
    try {
        while ((s = br.readLine()).length() != 0)
        System.out.println(s);
    }
    catch(IOException e) {
    }
}
```

Reading From a File

- The same idea works, except we need to use a **FileInputStream**.
- Its constructor takes a string containing the file pathname.

```
public static void main(String[] args) throws IOException {
    InputStreamReader isr = new
        InputStreamReader(new FileInputStream("FileInput.java"));
    int c;
    while ((c = isr.read()) != -1)
        System.out.println((char) c);
    isr.close();
}
```

Reading From a File (cont.)

- Here we check for a -1, indicating we've reached the end of the file.
- This works just fine if the file to be read is in the same directory as the class file, but an absolute path name is safer.
- The **read()** method can throw an **IOException**, and the **FileInputStream** constructor can throw a **FileNotFoundException**
- Instead of using a try-catch construction, this example shows main() declaring that it throws **IOException**.

The **File** Class

- Think of this as holding a file *name*, or a list of file *names* (as in a directory).
- You create one by giving the constructor a pathname, as in

  File f = new File("d:/mscit/java/awt/pooja/.");
- This is a directory, so now the **File f** holds a list of (the names of) files in the directory.
- It’s straightforward to print them out.

Listing Files

```
import java.io.*;
import java.util.*;
public class DirList {
    public static void main(String[] args) {
        File path = new File(".");
        String[] list;
        System.out.println(path.getAbsolutePath());
        if(args.length == 0)
            list = path.list();
        else
            list = path.list(new DirFilter(args[0]));
        for (int i = 0; i < list.length; i++)
            System.out.println(list[i]);
    }
}
```

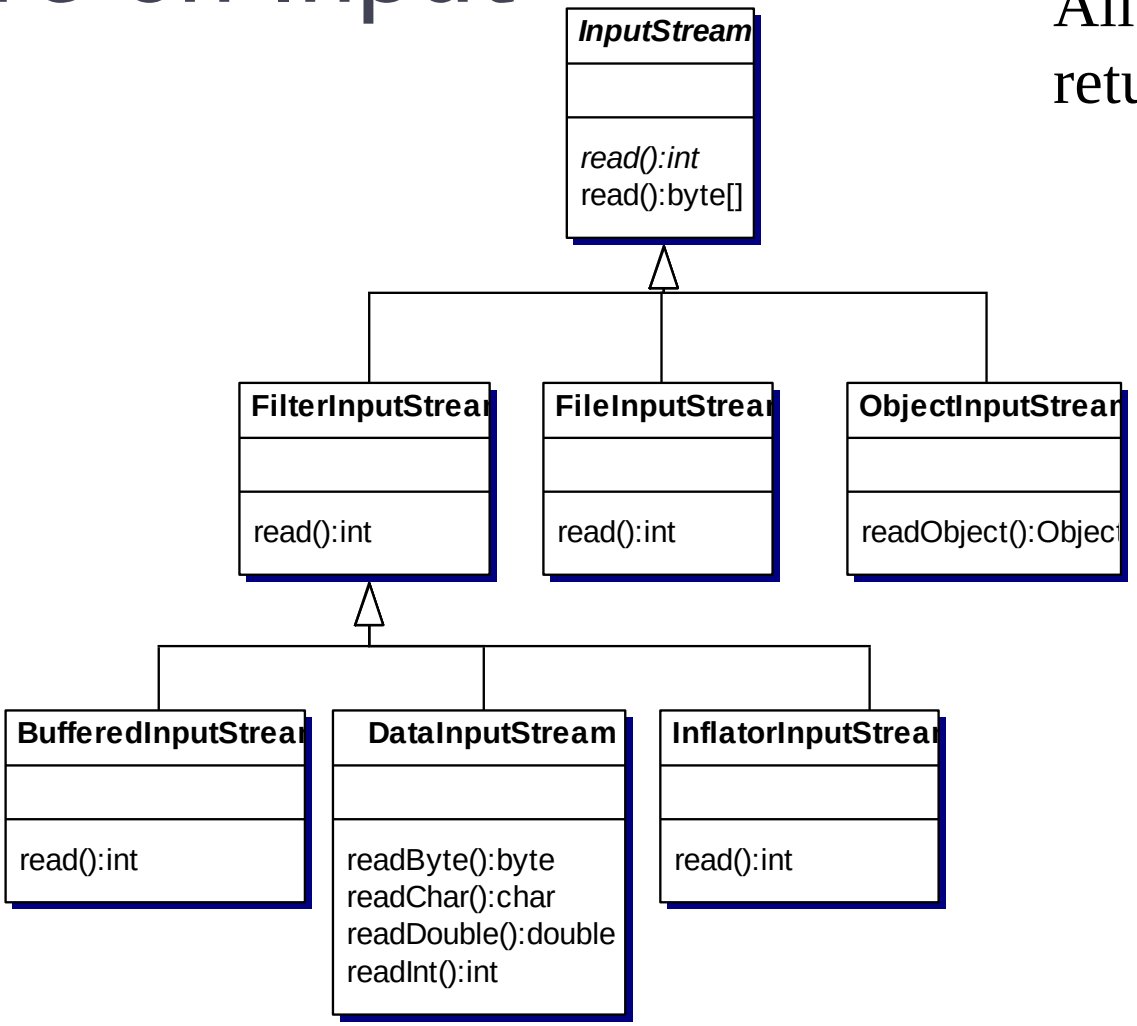
Other **File** Methods

- canRead()
- canWrite()
- exists()
- getParent()
- isDirectory()
- isFile()
- lastModified()
- length()

File Methods for Modifying

- createNewFile()
- delete()
- makeDir()
- makeDirs()
- renameTo()
- setLastModified()
- setReadOnly()

More on Input

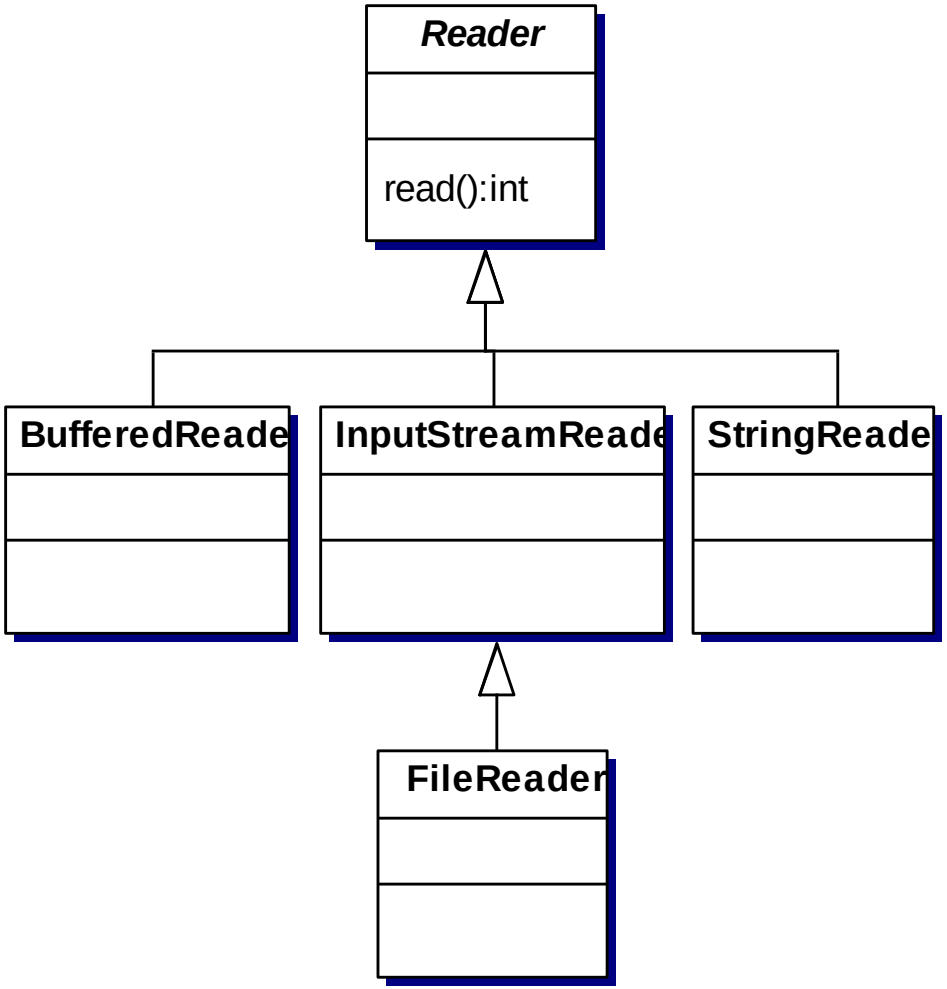


All of these return bytes!

FilterInputStream

- A FilterInputStream contains some other input stream, which it uses as its basic source of data and transforming the data to provide additional functionality.
- The class FilterInputStream itself simply overrides all methods of InputStream
- Subclasses of FilterInputStream may further override some of these methods and may also provide additional methods and fields.

Readers


Reader
read():int
BufferedReade
InputStreamReade
StringReade
FileReader

Example

```
BufferedReader br =
        new BufferedReader(new
  InputStreamReader(System.in));


InputStreamReader isr = new
        InputStreamReader(new
  FileInputStream("FileInput.java")); //slow: unbuffered


 InputStreamReader isr = new
        FileReader(" FileInput.java");
```

OutputStreams and Writers

- It is an image of **InputStream**s and **Reader**s.

e.g.,

```
BufferedWriter bw =
        new BufferedWriter(new OutputStreamWriter(System.out));
    String s;
    try {
        while ((s = br.readLine()).length() != 0) {
            bw.write(s, 0, s.length());
            bw.newLine();
            bw.flush();
        }
    }
```

FileWriter

- Again, basically the same. The constructors are
  - **FileWriter(File file)**
  - **FileWriter(FileDescriptor fd)**
  - **FileWriter(String s)**
  - **FileWriter(String s, boolean append)**
- The last one allows appending, rather than writing to the beginning (and erasing an existing file!).
- These will create files.

PrintWriter

```
PrintWriter out =
    new PrintWriter(new BufferedWriter(new FileWriter("Test.txt")));
String s;
try {
    while ((s = br.readLine()).length() != 0) {
        bw.write(s, 0, s.length());
        bw.newLine();
        bw.flush();
        out.println(s);
        //out.flush();
    }       }
catch(IOException e) {
}
out.close();
```

Thanks