Java Programming Tutorial

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

Content

- ❑ Classes and Objects
- ❑ Inheritance
- ❑ Packages & Interfaces
- ❑ Exception Handling
- ❑ Utility Classes & Interfaces
- ❑ Event Handling Using Swing
- ❑ I/O Streams
- ❑ Networking

Key Founders

- ❑ Java was the brainchild of:
  - ❑ James Gosling
  - ❑ Patrick Naughton
  - ❑ Chris Warth
  - ❑ Ed Frank &
  - ❑ Frank Sheridan
- ❑ The origin of Java can be traced back to the fall of 1992, and was initially called **Oak**.
- ❑ Oak was renamed as **Java** in **1995**.

Design Goal

❑ Platform-neutral language for embedded software in devices.

❑ The goal was to move away from platform and OS-specific compilers that would compile source for a particular target platform to a language that would be portable, and platform-independent.

The Java Buzzwords

- ❑ Simple
  - ❑ Small language [large libraries]
  - ❑ Small interpreter (40 k), but large runtime libraries (175 k)
- ❑ Object-Oriented and software technologies
  - ❑ Supports encapsulation, inheritance, abstraction, and polymorphism.
  - ❑ structured error-handling
  - ❑ garbage collection
- ❑ Distributed
  - ❑ Libraries for network programming
  - ❑ Remote Method Invocation
- ❑ Architecture neutral
  - ❑ Java Bytecodes are interpreted by the JVM.

The Java Buzzwords (Contd.).

- Secure
  - Difficult to break Java security mechanisms
  - Java Bytecode verification
  - Signed Applets.
- Portable
  - Primitive data type sizes and their arithmetic behaviour specified by the language
  - Libraries define portable interfaces
- Multithreaded
  - Threads are easy to create and use
- Dynamic
  - Finding Runtime Type Information is easy

A First Java Program

```
public class Hello {
    public static void main(String [] args)    {
        System.out.println(“Hello World”);
    }
}
```

- Class members can be made **static**.
- Static members are shared by all objects of a class.
- Interpreter use static method before object creation.

Class, Object and Encapsulation

- ❑ A class is a template for an object.
  - ❑ Class defines a new data type.
  - ❑ This new type can be used to create object of that type.
- ❑ A class has a class-name, a set of attributes and a set of services or actions.
- ❑ The object is the instance of the class.
- ❑ Encapsulation is the ability of an object to be a container/capsule
  - ❑ related **properties** (ie. data variables) and **methods** (ie. functions).
- ❑ Encapsulation makes it easy to maintain and modify code.
  - ❑ If method signature remains unchanged, client code is not affected with any internal change in method

Function Overloading and Constructor

- ❑ Identical name functions are said to be **overloaded**.
- ❑ Overloaded function calls are resolved using function signatures which constitutes the function name, the number, order and the data type of the arguments.
- ❑ A constructor function has the same name as the class name.
  - ❑ A class can contain more than one constructor.
  - ❑ Facilitates multiple ways of initializing an object
  - ❑ Constructors can be overloaded.

The final Keyword

- Many programming languages have a way of telling the compiler that a piece of data is “constant”.
- A constant is useful for two reasons:
  - It can be a compile-time constant that will never change.
  - It can be a value initialized at runtime that you don’t want changed.
- A variable can be declared as **final**.
- Must initialize a **final** variable when it is declared
  - **final float PI = 3.142857**;
- This prevents its contents from being modified.
- **Thus a final variable is essentially a named constant.**

Access Specifiers

- ❑ Access specifiers help implement:
  - ❑ Encapsulation by hiding implementation-level details in a class
  - ❑ Abstraction by exposing only the interface of the class to the external world
- ❑ The **private** access specifier is generally used to encapsulate or hide the member data in the class.
- ❑ The **protected** is used by the class itself, also by the subclasses of the class and by all the classes in the same package.
- ❑ The **public** access specifier is used to expose the member functions as interfaces to the outside world.

Java’s Cleanup Mechanism – The Garbage Collector

- Objects on the heap must be deallocated or destroyed, and their memory released for later reallocation.
- Java handles object deallocation automatically through **garbage collection.**
- Objects without references will be reclaimed.

Understanding CLASSPATH

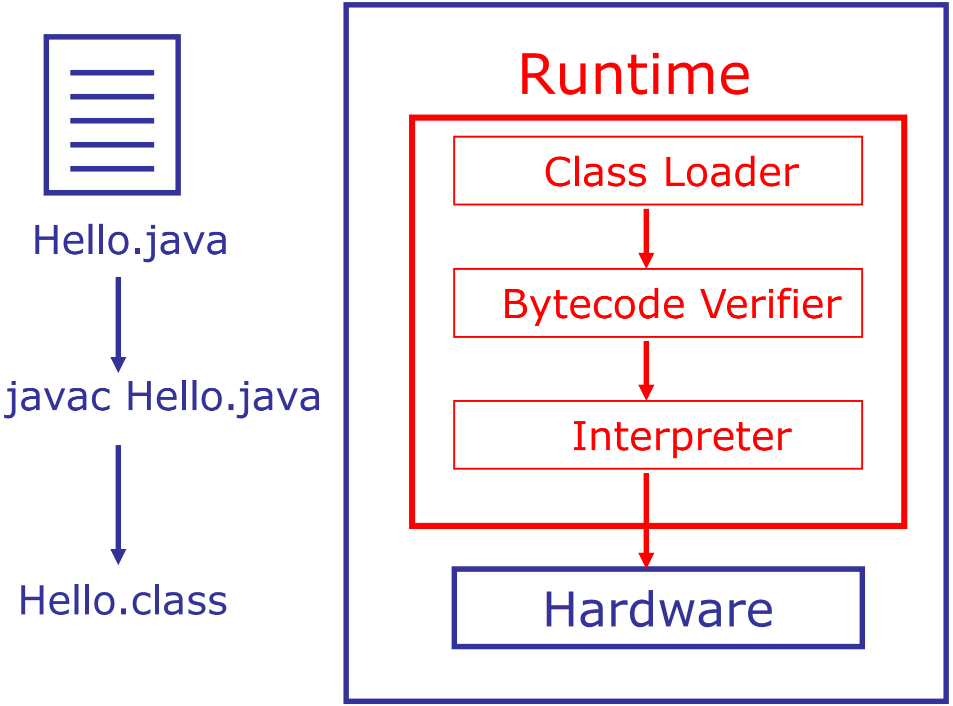
- ❑ What is **CLASSPATH**?
- ❑ **CLASSPATH is an environment variable that tells the Java runtime system where the classes are present.**
- ❑ When a packages is not created, all classes are stored in the default package.
- ❑ **The default package is stored in the current directory**.
- ❑ **The current directory is the default directory for CLASSPATH**.

Steps for CLASSPATH Setting

- **Windows 10**
  - Select Start, select Control Panel. double click System and Security, then System and select the Advanced System settings tab.
  - Click Environment Variables. ...
  - In the Edit System Variable (or New System Variable) **window**, **specify** the value of the PATH environment variable by Ex- C:\Program Files\Java\jdk1.8.0_141\bin
  - Reopen Command prompt **window**, and run your **java** code.

A First Java Program

First Example



Inheritance

- ❑ Inheritance is one of OOP concepts it allows for the creation of hierarchical classifications.
- ❑ Using inheritance, you can create a general class at the top.
- ❑ This class may then be inherited by other, more specific classes.
- ❑ Each of these classes will add only those attributes and behaviours that are unique to it.

Generalization/Specialization

- ❑ In keeping with Java terminology, a class that is inherited is referred to as a **superclass**.
- ❑ The class that does the inheriting is referred to as the **subclass**.
- ❑ **Each instance of a subclass includes all the members of the superclass.**
- ❑ **The subclass inherits all the properties of its superclass.**

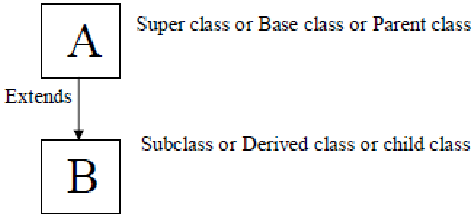
What is an Interface?

- ❑ **Definition:** An interface is a named collection of method declarations (without implementations)
  - ❑ An interface can also include constant declarations.
  - ❑ Interfaces are an integral part of Java.
  - ❑ **An interface is a complex data type similar to class in Java**.
  - ❑ An interface is syntactically similar to an abstract class.
  - ❑ An interface is a collection of abstract methods and final variables.

Java’s Inheritance Model

- ❑ Java uses the single inheritance model.
- ❑ **In single inheritance, a subclass can inherit from one and one only one superclass.**
- ❑ **Code Syntax for Inheritance:**
  - ❑ **class derived-class-name extends base-class-name**
  - ❑ **class derived-class-name implements interface-name**

Types of Inheritance



A

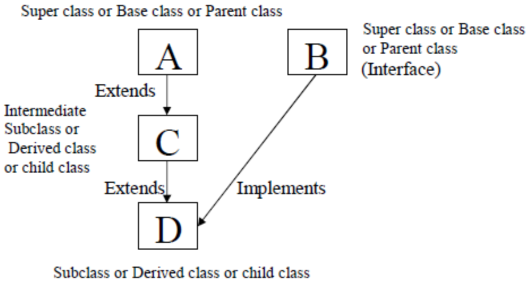
Super class or Base class or Parent class

Extends

B

Subclass or Derived class or child class

Fig: Single Inheritance



Super class or Base class or Parent class

A

B

Super class or Base class or Parent class (Interface)

Extends

Intermediate Subclass or Derived class or child class

C

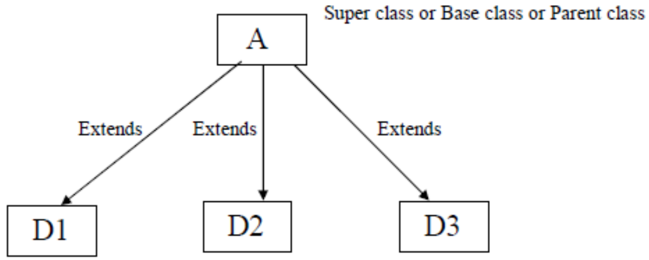
Extends

Implements

D

Subclass or Derived class or child class

Fig: Hybrid Inheritance



A

Super class or Base class or Parent class

Extends

Extends

Extends

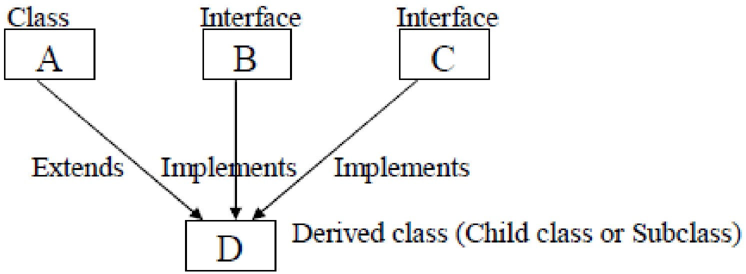
D1

D2

D3

D1, D2, D3 are the Subclass or Derived class or child class of A.

Fig: Hierarchical Inheritance



Class

A

Interface

B

Interface

C

Extends

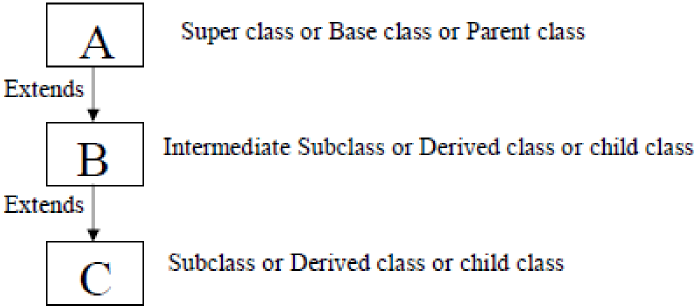
Implements

Implements

D

Derived class (Child class or Subclass)

Fig: Multiple Inheritance



A

Super class or Base class or Parent class

Extends

B

Intermediate Subclass or Derived class or child class

Extends

C

Subclass or Derived class or child class

Fig: Multilevel Inheritance

Using super in Inheritance

- Super has two uses.
  - The first is the call to the superclass’ constructor from the subclass constructor.
  - The second usage is to access a member of the superclass that has been overridden by a subclass.
- A subclass constructor can call a constructor defined in its immediate superclass by employing the following form of
  - super(parameter-list);
- **super( )** must always be the first statement executed inside a subclass’ constructor.
- It clearly tells you the order of invocation of constructors in a class hierarchy.
- **Constructors are invoked in the order of their derivation.**

Method Overriding

- ❑ When a method in a subclass has the same prototype as a method in the superclass, then the method in the subclass is said to override the method in the superclass.
- ❑ When an overridden method is called from an object of the subclass, it will always refer to the version of that method defined by the subclass.
- ❑ The version of the method defined by the superclass is hidden or overridden.

Role of the Keyword final in Inheritance

❑ Using final to Prevent Overriding

- ❑ While method overriding is one of the most powerful feature of object oriented design, there may be times when you will want to prevent certain critical methods in a superclass from being overridden by its subclasses.
- ❑ Rather, you would want the subclasses to use the methods as they are defined in the superclass.
- ❑ This can be achieved by declaring such critical methods as **final**.

❑ Using final to Prevent Inheritance

- ❑ Sometimes you will want to prevent a class from being inherited.
- ❑ This can be achieved by preceding the class declaration with **final**.
- ❑ Declaring a class as **final** implicitly declares all of its methods as **final** too.
- ❑ It is illegal to declare a class as both **abstract** and **final** since an abstract class is incomplete by itself and relies upon its subclasses to provide concrete and complete implementations.

Inheritance example

Organizing Classes into Packages

- ❑ Packages are containers for classes and interfaces.
- ❑ To avoid namespace collision, we put the classes into separate containers called **packages.**
- ❑ Whenever you need to access a class, you access it through its package by prefixing the class with the package name.
- ❑ It can be made easy by giving a star(*****) at the end of the import statement. For example:
  - ❑ **`import package1.*;`**

Access Protection

- ❑ Packages facilitate access-control.
- ❑ **Once a class is packaged, its accessibility is controlled by its package**.
- ❑ That is, whether other classes can access the class in the package depends on the **access specifiers** used in its class declaration.
- ❑ There are four visibility control mechanisms packages offer:
  - ❑ private
  - ❑ no-specifier (default access)
  - ❑ protected
  - ❑ public

Packages & Access Control

Specifier	Accessibility
private	Accessible in the same class only
protected	Subclasses and non-subclasses in the same package, and subclasses in other packages
No-specifier (default access)	Subclasses and non-subclasses in the same package
public	Subclasses and non-subclasses in the same package, as well as subclasses and non-subclasses in other packages. In other words, total visibility

A Package Example

What is an Exception?

- ❑ In procedural programming, it is the responsibility of the programmer to ensure that the programs are error-free.
- ❑ Errors have to be checked and handled manually by using some error codes.
- ❑ Java provides an excellent mechanism for handling runtime errors.
- ❑ Java’s exception handling is managed using the following keywords:
    - ❑ **try, catch, throw, throws and finally.**

```
try {
 // code comes here
 }
catch(TypeofException  obj) {
      //handle the exception
  }
      finally
 {
          //code to be executed before the program ends
 }
```

Using finally

- ❑ When an exception occurs, the execution of the program takes a non-linear path, and could bypass certain statements.
- ❑ A program establishes a connection with a database, and an exception occurs.
- ❑ The program terminates, but the connection is still open.
- ❑ To close the connection, **finally** block should be used.
- ❑ The **finally** block is guaranteed to execute in all circumstances.

  Example Exception

Event Handling and GUI based programming

- ❑ In console-based programs, the program:
  - ❑ may prompt the user for some input
  - ❑ processes the user input and displays the result
- ❑ In a GUI-based program, the user initiates the interaction with the program through GUI events such as a button click.
- ❑ In a GUI environment, the user drives the program.

Event Handling and GUI based programming(Contd.).

- Whenever a user interacts with these GUI controls:
  - some event is generated
  - some action implicitly takes place that validates or adds functionality to the event
- This type of programming is called **event-driven** programming, where the program is driven by the events.
- Events are best used in GUI-based programs.

What is an Event?

- When an event occurs, the GUI run-time system:
  - intercepts the event
  - notifies the program that some event has occurred
- Thus, an **event** is a signal from the GUI run-time system to the program that a user interaction has taken place.
- This specific signal has to be interpreted by the program, and it must take appropriate action on the occurrence of the specific event.

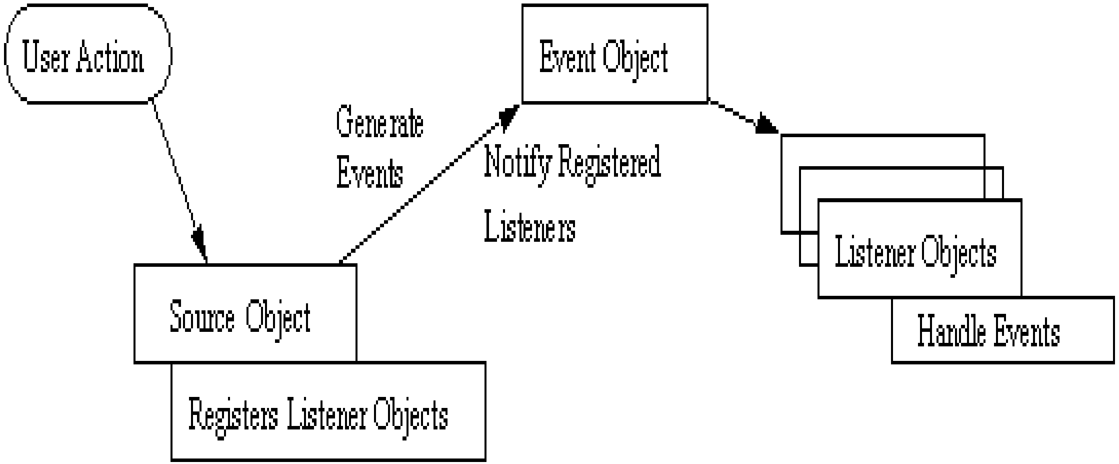
What is an Event? (Contd.).

- The GUI object on which an event is generated is called the source of the event.
- In Java, all events are implemented as classes.
- When an event occurs, an object:
  - of the respective event class is created
  - encapsulates a state change in the source that generated the event
- Thus, an event can be captured as an object that describes a state change on the source.

Designing a Basic GUI and use of Swing

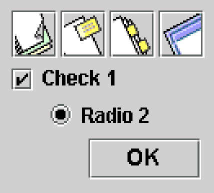
- Basic User Interface Tasks:
  - Provide help/guidance to the user
  - Allow input of information
  - Allow output of information
  - Control interaction between the user and device
- Swing objects for:
  - **Guidance**
  - **Input**
  - **Output**
  - **Control**

Delegation Event Model

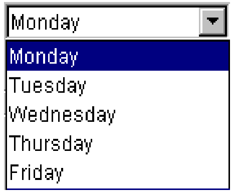


❑The methods that receive and process events are defined in a set of interfaces found in **java.awt.event** package.

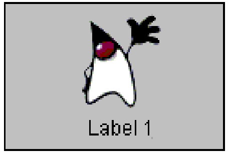
Useful Components



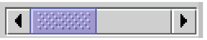
JButton



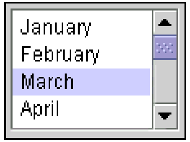
JComboBox



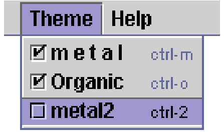
JLabel



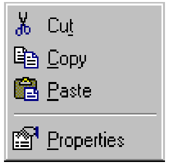
JScrollBar



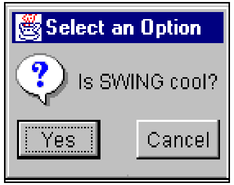
JList



JMenu

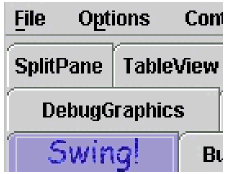


JPopupMenu

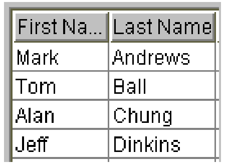


Is SWING cool?

JDialog

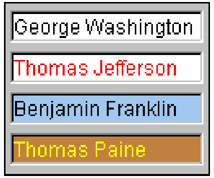


JTabbedPane



First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

JTable



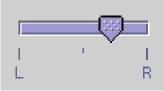
George Washington

Thomas Jefferson

Benjamin Franklin

Thomas Paine

JTextField



JSlider

AWTEvent Subclasses

Examples for events:

- ❑ **ActionEvent** – generated when:
  - ❑ a button is clicked
- ❑ **MouseEvent** – generated when the mouse is:
  - ❑ pressed, released, clicked, or when the mouse enters or exits a component
  - ❑ moved, or dragged
- ❑ **WindowEvent** – generated when the window is activated, deactivated, minimized, maximized, opened, closed or quit.

ActionEvent Class

- ❑ An **ActionEvent** is generated when a button is clicked.
- ❑ **ActionEvent** object has following methods:
  - ❑ **String getActionCommand()**
    - ❑returns label on the button
  - ❑ **Object getSource( )**
    - ❑Returns a reference to the source

MouseEvent Class

- **MouseEvent** class is a subclass of the **InputEvent** class.
- A mouse event is generated when the mouse is pressed, released, clicked, entered, exited etc

Event Listener Interfaces

- ❑ The delegation event model has two parts: sources and listeners
- ❑ Listeners are created by implementing one, or more of the interfaces defined by the **java.awt.event** package
- ❑ When an event occurs, the event source
  - ❑ invokes the appropriate method defined by the listener object’s implementing interface
  - ❑ provides an event object as its argument

ActionListener Interface

- ❑ **ActionListener Interface** has a single method that is invoked when an action event occurs.
- ❑ The method takes the reference of **ActionEvent** as its argument.
  - ❑ **void actionPerformed(ActionEvent ae)**

MouseListener Interface

- The **MouseListener** interface contains:
  - five methods
  - these methods take a **MouseEvent** reference as an argument
- The methods are:
  - **void mousePressed(MouseEvent me)**
  - **void mouseReleased(MouseEvent me)**
  - **void mouseClicked(MouseEvent me)**
  - **void mouseEntered(MouseEvent me)**
  - **void mouseExited(MouseEvent me)**

WindowListener Interface

- The **WindowListener** interface has:
  - seven methods that are invoked when window events take place
- These methods take a **WindowEvent** reference as an argument.

```
void windowOpened(WindowEvent we)
void windowClosed(WindowEvent we)
void windowClosing(WindowEvent we)
void windowIconified(WindowEvent we)
void windowDeiconified(WindowEvent we)
void windowActivated(WindowEvent we)
void windowDeactivated(WindowEvent we)
```

Streams

- Java programs perform I/O through streams. A stream is:
  - an abstraction that either produces or consumes information
  - linked to a physical device by the Java I/O system
  - Stream classes are defined in the java.io package.

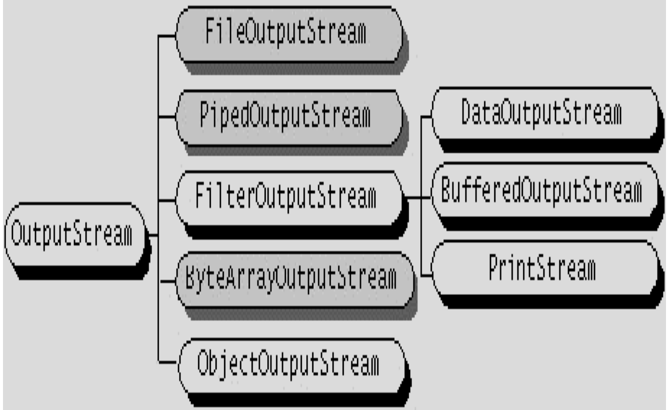
Basic types of streams:

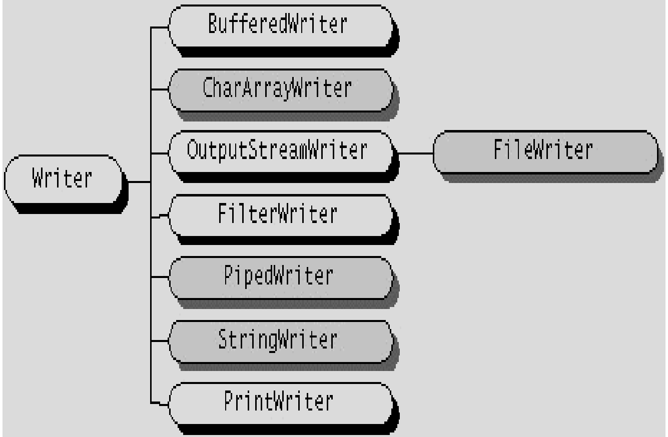
- Byte streams:
  - provide a convenient means for handling input and output of bytes
  - are used for reading or writing binary data
- Character streams:
  - provide a convenient means for handling input and output of characters
  - use **Unicode**, and, therefore, can be internationalized

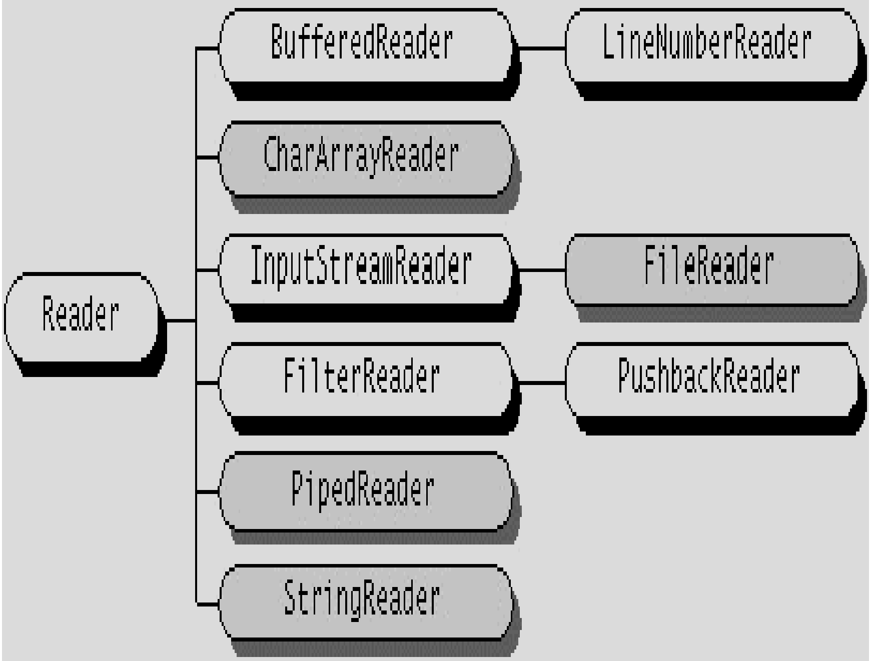
The Predefined Streams

- **System** class of the java.lang package contains three predefined stream variables, **in**, **out**, and **err.**
- These variables are declared as **public** and **static** within **System**:
  - **System.out** refers to the standard output stream which is the console.
  - **System.in** refers to standard input, which is the keyboard by default.
  - **System.err** refers to the standard error stream, which also is the console by default.

Byte Streams, Character Streams







Reading Console Input - Stream Wrapping

- ❑ The preferred method of reading console input in Java 2 is to use a character stream.
- ❑ **InputStreamReader** class acts as a bridge between byte and character streams.
- ❑ Console input is accomplished by reading from **System.in**
- ❑ To obtain a character-based stream that is attached to the console, you wrap **System.in** in a **BufferedReader** object, to create a character stream.

Writing Console Output

- ❑ Console output is accomplished with **print( )** and **println( )**
- ❑ These methods are defined by the class **PrintStream**, which is the type of object referenced by **System.out**
- ❑ **System.out** is a byte stream used to write bytes.
- ❑ Since **PrintStream** is an output stream derived from **OutputStream**, it implements the low-level method **write( )**
- ❑ Thus, **write( )** can be used to write to the console.

Network Programming

- ❑ The package java.net provides support for sockets programming (and more).

- ❑ Typically you import everything defined in this package with:

```
import java.net.*;
```

Class

`InetAddress`

`Socket`

`ServerSocket`

`DatagramSocket`

`DatagramPacket`

Constructor

- ❑ Constructor creates a TCP connection to a named TCP server.
  - ❑ There are a number of constructors:

```
Socket(InetAddress server, int port);

Socket(InetAddress server, int port,
            InetAddress local, int localport);

Socket(String hostname, int port);
```

Methods

```
void close();

InetAddress getInetAddress();

InetAddress getLocalAddress();

InputStream getInputStream();

OutputStream getOutputStream();
```

- Lots more (setting/getting socket options, partial close, etc.)
- `I/O is same as Stream ( see Example)`

Thank You