CS 431: Programming Language Lab
Java Tutorial

Department of CSE, IIT Guwahati

Tutorial Overview

- Basics of Java
- Object Oriented programming concepts with respect to Java
- Event handling and GUI programming in Java

JAVA : Overview

- Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE])
- The latest release of the Java Standard Edition is Java SE 8. Other than Standard Edition, multiple configurations were built for e.g. J2EE for Enterprise Applications, J2ME for Mobile Applications.
- Java is guaranteed to be Write Once, Run Anywhere.

Java : Features

1. **Object Oriented :** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
2. **Platform Independent / Architectural-neural :** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
3. **Simple and Secure**
4. **Multithreaded :** Java supports multi-tasking using its multithreading architecture.
5. **Distributed**
6. **Dynamic**

Link to download JAVA SE : https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

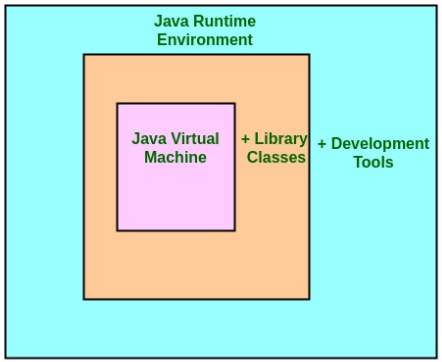
Java : Environment

When we talk about the JAVA environment following terminologies will hit our ear very often. These are:

- JDK (Java Development Kit)
- JRE (Java Runtime Environment)
- JVM (Java Virtual Machine)

JDK (Java Development Kit)

- The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets.
- It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.
- JDK is only used by Java Developers.



JRE (Java Runtime Environment)

- The Java Runtime Environment provides the minimum requirements for executing a Java application.
- It consists of the *Java Virtual Machine (JVM)*, *core classes*, and *supporting files*.
- It is only used for running JAVA program (execution) and not for development purpose.

JVM (Java Virtual Machine)

- **Java Virtual machine**(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for **executing the java program line by line** hence it is also known as interpreter.
- JVM becomes an **instance of JRE at runtime** of a Java program. It is widely known as a runtime interpreter.
- JVM largely helps in the abstraction of inner implementation from the programmers who make use of libraries for their programmes from JDK.

Setting up Environment

- CLASSPATH is an important 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.
- **Steps for CLASSPATH settings: Windows**
    - 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
- **Steps for CLASSPATH settings: MacOS/Linux (Just follow the commands below)**
    - vim .bash_profile
    - export JAVA_HOME=$(/usr/libexec/java_home)
    - source .bash_profile

First Java Program

```java
public class Program1 {
    public static void main(String []args) {
        System.out.println("Hello World"); // prints Hello World
    }
}
```

Let's look at how to save the file, compile, and run the program. Please follow the subsequent steps :

- Open notepad or any editor and add the code as above.
- Save the file as: Program1.java.
- Open a command prompt window and go to the directory where you saved the class. Assume it's E:\.
- Type '**javac** Program1.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line. (Compilation step)
- Now, type ' **java** Program1 ' to run your program. (execution step)
- You will be able to see ' Hello World ' printed on the window.

Key Points

About Java programs, it is very important to keep in mind the following points.

- **Case sensitive :** Java is case sensitive, which means identifier **Program** and **program** would have different meaning in Java.
- **Class Name :** For all class names the first letter should be in Uppercase. If several words are used to form a name of the class, each inner word's first letter should be in Uppercase.
- **Method Names :** All method names should start with a Lowercase letter. If several words are used to form the name of the method, then each inner word's first letter should be in Uppercase.
- **Program File Name :** Name of the program file should exactly match the class name.
- **public static void main(String args[]) :** Java program processing starts from the main() method which is mandatory part of every Java program.

Java - Object & Class

Object: It can be naively described as an instance of a class which has states and behaviours. In real world, we can find many objects such as monkey which has states : color, breed etc and behaviours : eating, jumping etc.

Class: A class can be defined as a template or a blueprint that describes the behavior/state that the object of its type support. An example of class is given below :

```
1  public class Monkey
2  {
3      String breed;
4      int age;
5      String color;
6      void jumping()
7      {
8          ...
9      }
10     void eating()
11     {
12         ...
13     }
14 }
```

Java - Object & Class

A class can contain any of the following variable types.

- **Local variables :** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables :** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables :** Class variables or static variables are variables declared within a class, outside any method, with the **static** keyword. There would only be one copy of each class variable per class, regardless of how many objects are created from it.

Constructor

- Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.
- Each time a new object is created, at least one constructor will be invoked.
- The main rule of constructors is that they should have the same name as the class.
- A class can have more than one constructor.

```
1   public class Monkey
2   {
3       void Monkey()       // Default Constructor
4       {
5           ...
6       }
7       void Monkey(String name)    //Overloaded Constructor
8       {
9           ...
10      }
11  }
```

Java - Modifiers

Modifiers are keywords that you add to those definitions to change their meanings or usage policy. To use a modifier, you include its keyword in the definition of a class, method, or variable. Java language has a wide variety of modifiers, including the following :

1) **Java Access Modifiers:** Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are −
    - Visible to the package, the default. No modifiers are needed.
    - Visible to the class only (**private**).
    - Visible to the world (**public**).
    - Visible to the package and all subclasses (**protected**).
1) **Non Access Modifiers:** Java provides a number of non-access modifiers to achieve many other functionality.
    - The ***static*** modifier for creating class methods and variables.
    - The ***final*** modifier for finalizing the implementations of classes, methods, and variables.
    - The ***abstract*** modifier for creating abstract classes and methods.
    - The ***synchronized*** and ***volatile*** modifiers, which are used for threads.

JAVA Object Oriented

Following are the key concepts under JAVA Object Oriented:

1. Inheritance
2. Overriding
3. Polymorphism
4. Abstraction
5. Encapsulation
6. Interfaces
7. Packages

JAVA - Inheritance

- Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.
- The class which inherits the properties of other is known as **subclass** (derived class, child class) and the class whose properties are inherited is known as **superclass** (base class, parent class).
- **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

```
1  class Super
2  {
3      ....
4  }
5
6  class Sub extends Super
7  {
8      ...
9  }
```

JAVA- Inheritance (Example)

```
class Bicycle {
            public int gear, speed;
public Bicycle(int gear, int speed) {
        this.gear = gear;
        this.speed = speed;
             }
             public String toString() {
        return("No of gears are "+gear
                +"\n"
                + "speed of bicycle is "+speed);
             }
}
class MountainBike extends Bicycle {
    public int seatHeight;
    public MountainBike(int gear,int speed,
                        int startHeight) {
        super(gear, speed);
        seatHeight = startHeight;
    }
          public String toString() {
      return(super.toString()+"\nseat height is:
      "+seatHeight);
            } }

public class Test
{
    public static void main(String args[])
    {

        MountainBike mb = new MountainBike(3,
100, 25);
        System.out.println(mb.toString());

    }}
```

Types of Inheritance

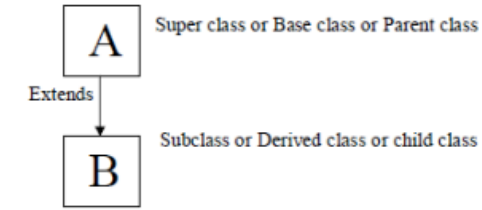


Fig: Single Inheritance

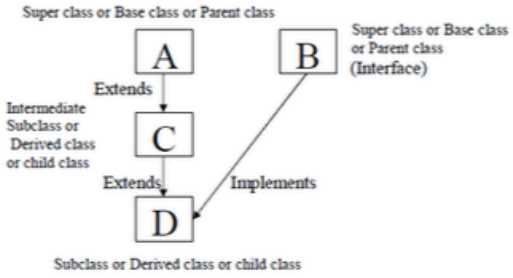


Fig: Hybrid Inheritance

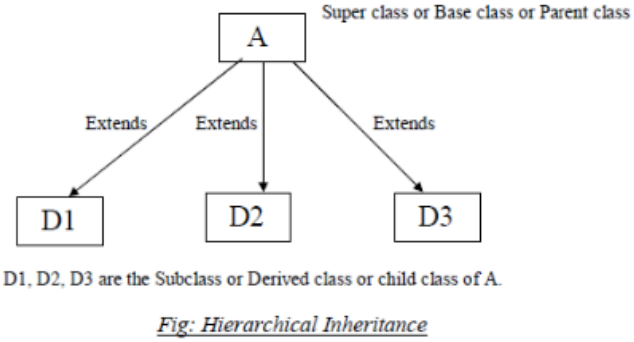


Fig: Hierarchical Inheritance

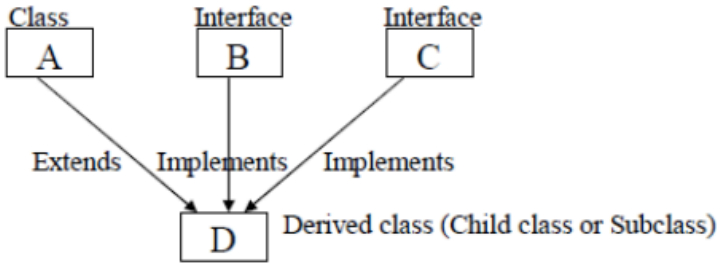


Fig: Multiple Inheritance

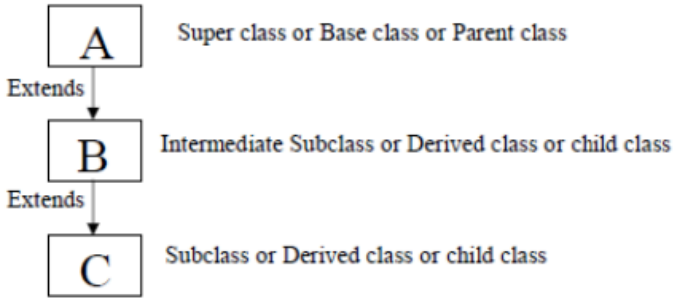


Fig: Multilevel Inheritance

JAVA - Overriding

- In object-oriented terms, overriding means to override the functionality of an existing method.
- If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final.
- The benefit of overriding is: ability to define a behaviour that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

JAVA - Overriding

```java
class Automobile {
   public void move() {
      System.out.println("Automobile can move");
   }
}

class Car extends Automobile {
   public void move() {
      System.out.println("Car can move");
   }
}

public class TestCar {

   public static void main(String args[]) {
      Automobile a = new Automobile();   // Automobile reference and object
      Automobile b = new Car();   // Automobile reference but Car object

      a.move();   // runs the method in Automobile class
      b.move();   // runs the method in Car class
   }
}
```

```
Output :
Automobile can move
Car can move
```

JAVA - Polymorphism

- Polymorphism means to process objects differently based on their data type.
- In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)
- This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

JAVA - Polymorphism

```java
class MultiplyFun {
    static int Multiply(int a, int b)
    {
        return a * b;
    }
    static int Multiply(int a, int b, int c)
    {
        return a * b * c;
    }
}

class Main {
    public static void main(String[] args)
    {
        System.out.println(MultiplyFun.Multiply(2, 4));

        System.out.println(MultiplyFun.Multiply(2, 7, 3));
    }
}
```

```
output:
8
42
```

JAVA- Abstraction

Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

In JAVA, abstraction is achieved using the class and interface (will be shown later)

Abstract Class

It needs to be extended and its method implemented.

Key Points

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.

JAVA- Abstraction

Abstract Method in Java

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- **Bike** is an abstract class that contains only one abstract method run. Its implementation is provided by the **Honda** class.
- The **Honda** class inherits all properties from its parent Bike but have to provide it's own implementation of **run()** method.

```
abstract class Bike{
  abstract void run();
}
class Honda extends Bike{
void run()
{
    System.out.println("running safely"); }
    public static void main(String args[]){
        Bike obj = new Honda4();
        obj.run();
}
}
```

JAVA - Encapsulation

- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.
- To achieve encapsulation in Java :
  - Declare the variables of a class as **private**.
  - Provide public **setter** and **getter** methods to modify and view the variables values.
- Benefits of Encapsulation :
  - The fields of a class can be made read-only or write-only.
  - A class can have total control over what is stored in its fields.

JAVA - Encapsulation (Example)

The public setXXX() and getXXX() methods are the access points of the instance variables of the EncapAnimal class. Normally, these methods are referred as getters and setters. Therefore, any class that wants to access the variables should access them through these getters and setters.

```java
/* File name : EncapTest.java */
public class EncapAnimal {
   private String name;
   private int age;

   public int getAge() {
      return age;
   }
   public String getName() {
      return name;
   }
   public void setAge( int newAge) {
      age = newAge;
   }

   public void setName(String newName) {
      name = newName;
   }
}
```

The variables of the EncapAnimal class can be accessed using the following program –

```java
public class RunEncapAnimal {
   public static void main(String args[]) {
      EncapAnimal encap = new EncapAnimal();
      encap.setName("Ashwin");
      encap.setAge(22);
      System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());
   }
}
```

Output

```
Output:
Name : Ashwin  Age : 22
```

JAVA - Interface

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.

JAVA - Interface

```java
import java.io.*;

interface in1
{
    final int a = 10;
    // public and abstract
    void display();
}

class testClass implements in1
{

    public void display()
    {
        System.out.println("Geek");
    }

    public static void main (String[] args)
    {
        testClass t = new testClass();
        t.display();
        System.out.println(a);
    }
}
```

```
output:
Geek
10
```

JAVA - Packages

- Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.
- Package in java can be categorized in two form, built-in package and user-defined package.

Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

The **package** keyword is used to create a package in java.

JAVA - Packages

User Defined Package

These are the packages that are defined by the user.

```
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

Built-In Package

Some of the commonly used built-in packages are -

java.lang, java.io, java.util, java.applet, java.awt, java.net

Event Handling and GUI Programming

- 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
- 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

Event

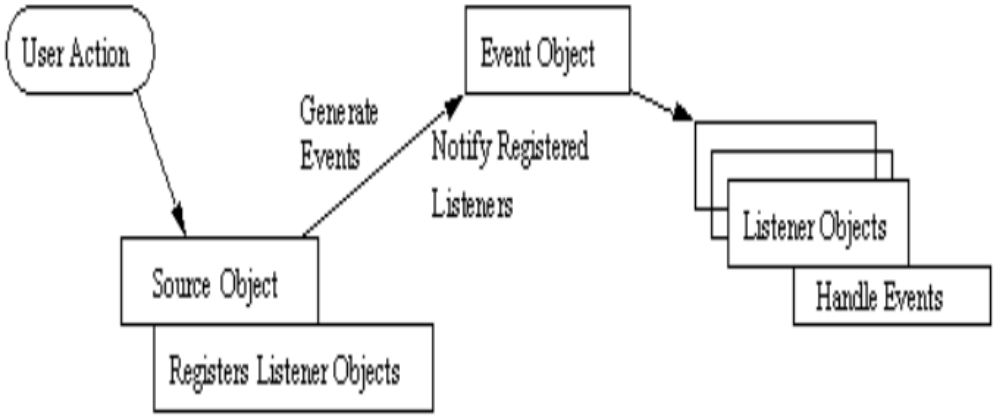
- When an event occurs, the GUI run-time system:
  - intercepts the event
  - notifies the program that some event has occurred
- Thus, an event is a signal from the GUI run-time system to the program that a user interaction has taken place.
- This specific signal has to be interpreted by the program,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

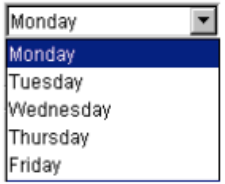
Delegation Event Model



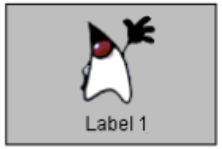
- The methods that receive and process events are defined in a set of interfaces found in **java.awt.event** package.

Useful Components

JButton

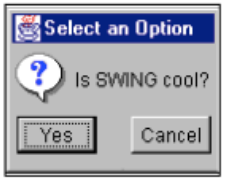


JComboBox

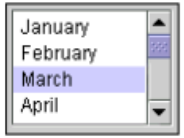


JLabel

JScrollBar

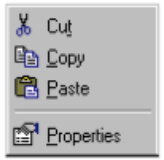


JDialog

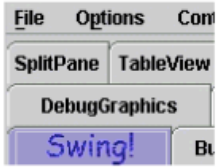


JList

JMenu



JPopupMenu



JTabbedPane

JTable

JTextField

JSlider

AWTEvent Subclasses

- **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)

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:
  - Character streams:
- The Predefined Streams
  - System class of the java.lang package contains three predefined stream variables, in, out, and err.

Streams

```
import java.io.*;
class MyInput
{
  public static void main(String[] args)
  {
    String text;
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    text = br.readLine();    //Reading String
    System.out.println(text);
  }
}
```

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.*;**

- **Classes:**
  - InetAddress
  - Socket
  - ServerSocket
  - DatagramSocket
  - DatagramPacket

Constructors

- 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();


भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Thank You