PROGRAMMING IN JAVA

By ,

Hamsashree M K

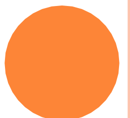
Asst.Professor

Dept.ECE BGSIT,B G Nagara

OVERVIEW OF JAVA

- Java is a programming language developed by JAMES GOSLING at SUN MICROSYSTEMS and released in 1995.

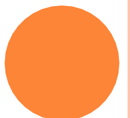
- The java source code is converting into virtual code called as **byte code.**



WHAT IS JAVA

- Java is a class based,object –oriented programming language.

- Java applications are typically compiled to byte code that can run on any Java Virtual Machine(JVM)



Why we use Java?

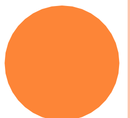
- Java can be used to create complete application.

- Java technology is an object oriented /platform independent , multithreaded programming environment .

- Java is used to create 2 types of programs

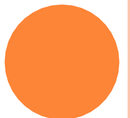
  Applets

  Application



Applications of JAVA

- Desktop GUI Applications
- Mobile Applications
- Enterprise Applications
- Scientific Applications
- Web based and Embedded Applications



JAVA FEATURES/BUZZ WORDS:

Simple

Robust

Dynamic

Secured

Interpreted

Distributed

Features

Portable

Object Oriented

High Performance

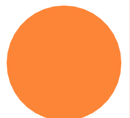
Architectural Neutral

Multi Threading

Tutorial4us.com

Platform Independent

Tutorial4us.com



Simple:

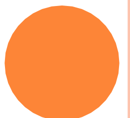
- Java is free from pointer
- Rich set of API
- Garbage Collector

Object Oriented:

- Java is easily exteded since it is based on the object model.

Robust:

- Java is a strictly typed language, it checks user code at compile time and run time.



Portability:

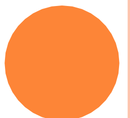
- If any language supports platform independent and architectural neutral feature known as portable.

Security:

- Java is more secured language compare to other language.

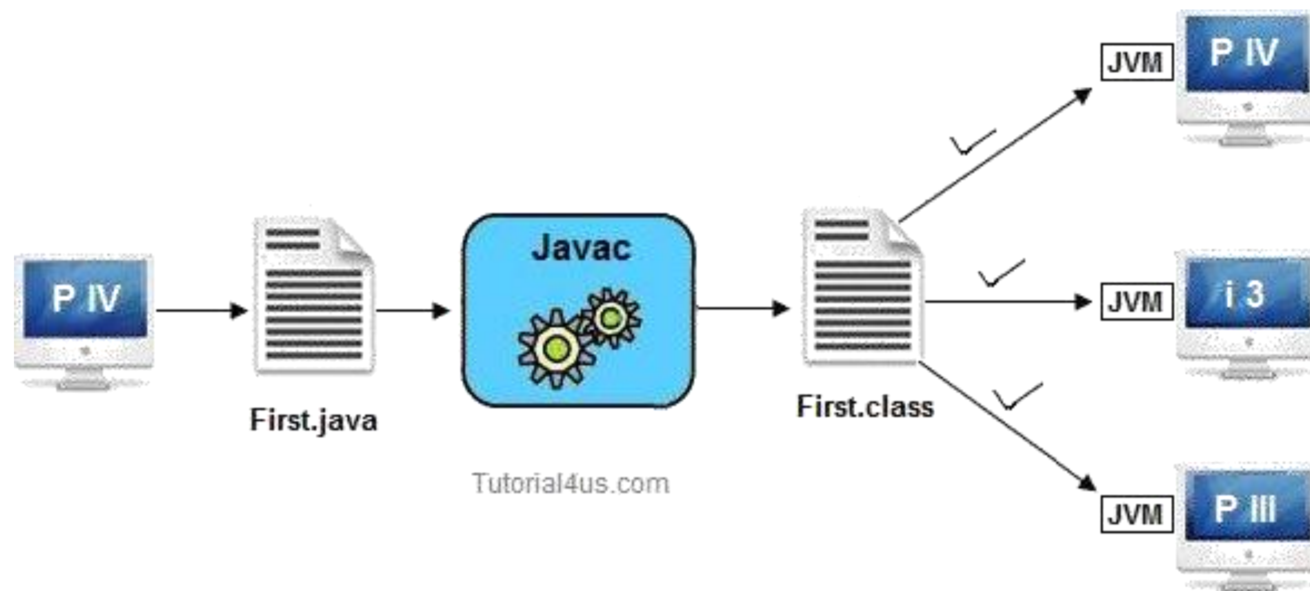
Multithread:

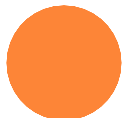
- When any Language execute multiple thread at a time that language is known as multithreaded Language



Architectural Neutral:

- The goal of java designers to develop "**write once, run anywhere**, anytime, forever" so that a program can be independent of the architecture of the system in which it is running.



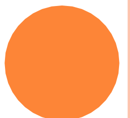


High Performance:

- Java uses Bytecode
- Garbage collector
- No pointer
- Supports Multithreading

Distributed:

- We can create distributed applications in java. RMI and EJB are used for creating distributed applications.



Distributed Application

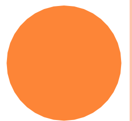
Client 7
Client 8
Client 1
Client 6
Application
Server 1
Client 2
Client 5
Client 4
Client 3

Client 7
Client 8
Client 1
Client 6
Application
Server 2
Client 2
Client 5
Client 4
Client 3

Internet

Client 7
Client 8
Client 1
Client 6
Application
Server 4
Client 2
Client 5
Client 4
Client 3

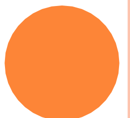
Client 7
Client 8
Client 1
Client 6
Application
Server 3
Client 2
Client 5
Client 4
Client 3



Dynamic:

- Java programming support Dynamic memory allocation due to this memory wastage is reduce and improve performance of application.

- New operator is used to allocate the memory dynamically.

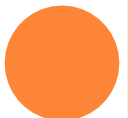


JAVA BYTECODE

- The java source code is converting into virtual code called as **byte code.**

- A bytecode is a set of highly optimized set of instructions which can be executed by JVM.

- The use of bytecode enable JVM to execute programs much faster.

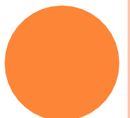


JAVA ENVIRONMENT VARIABLE/JAVA RUNTIME ENVIRONMENT(JRE):

Java environment is a collection of tools and classes, methods.

1) Java Development kit (JDK).

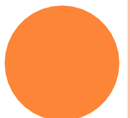
2) Application programming interface (API)/Java Standard library(JSL)



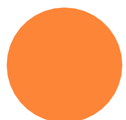
JAVA DEVELOPMENT KIT – (JDK)

The jdk consists of a collection of tools as

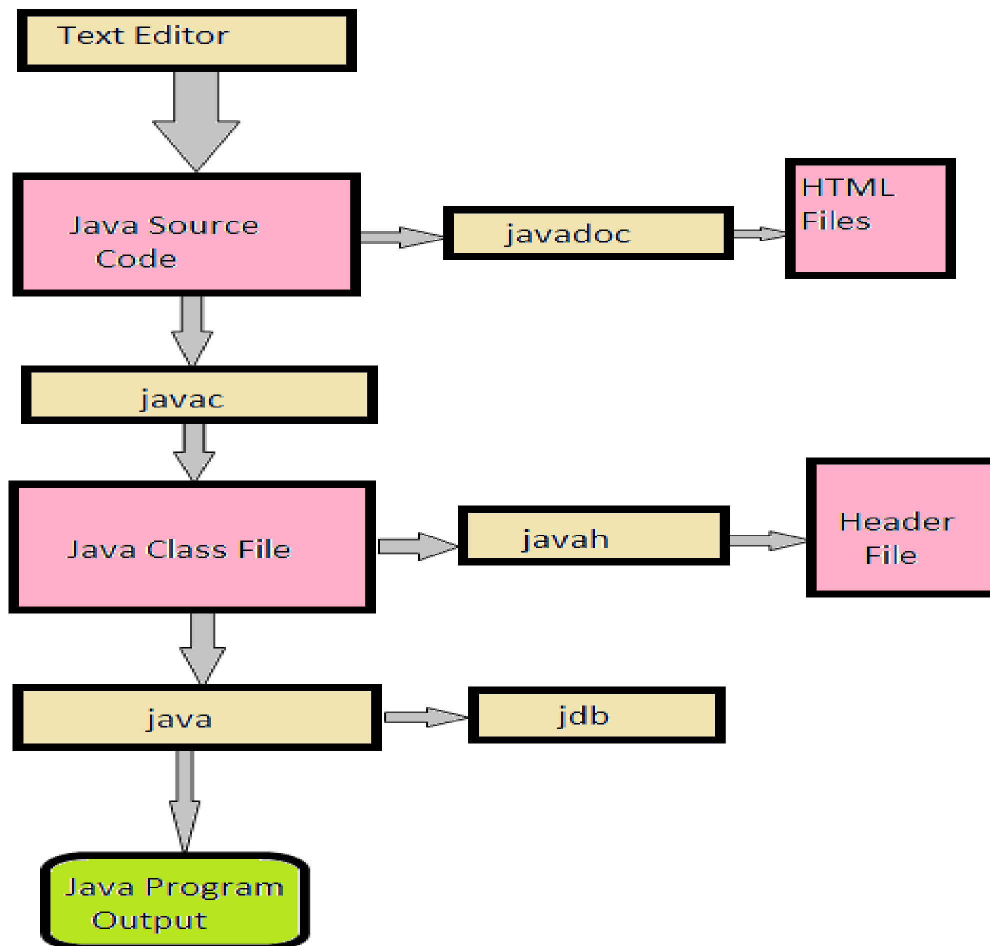
- appletviewer (to launch applets)
- javac (java compiler)
- java (java interpreter)
- javap (java disassembler)
- javah (for C header files)
- javadoc (for HTML documents)
- jdb (java debugger)

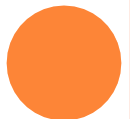


Tool	*Description*
appletviewer	Enables us to run Java applets (without actually using a Java-compatible browser).
java	Java interpreter, which runs applets and applications by reading and interpreting bytecode files.
javac	The Java compiler, which translates Java sourcecode to bytecode files that the interpreter can understand.
javadoc	Creates HTML-format documentation from Java source code files.
javah	Produces header files for use with native methods.
javap	Java disassembler, which enables us to convert bytecode files into a program description.
jdb	Java debugger, which helps us to find errors in our programs.



PROCESS OF BUILDING AND RUNNING JAVA APPLICATION

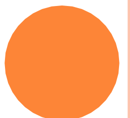




APPLICATION PROGRAMMING INTERFACE(API):

API is a collection of classes and methods are grouped into several different packages.

- **import java.lang.*;**
- **import java.lang.*;**
- **import java.io.*;**
- **import java.net.*;**
- **import.java.awt.*;**
- **import java.applets.*;**



JAVA IS INTERPRETED LANGUAGE

Java source code (.java)

javac filename.java

Java compiler

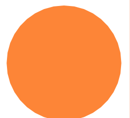
Bytecode file (.class file)

java filename

Java interpreter

JIT compiler

Native Code



Java Virtual Machine

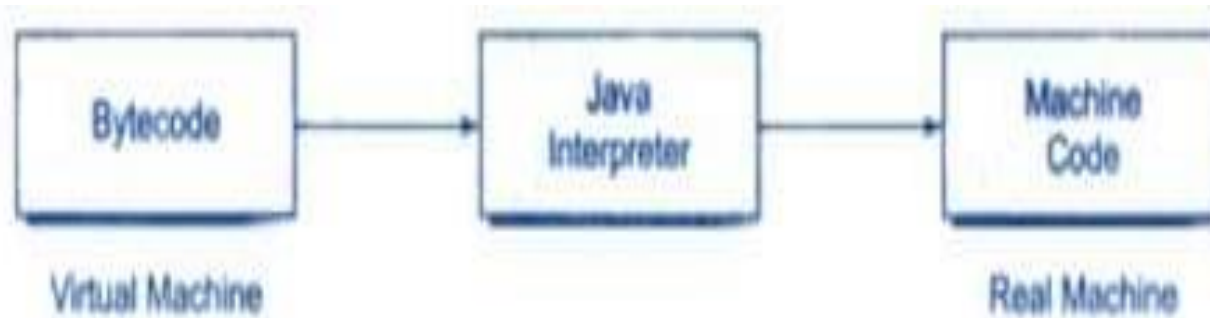
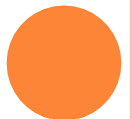


Fig. 3.7 Process of converting bytecode into machine code



THE INTERNAL ARCHITECTURE OF THE JAVA VIRTUAL MACHINE.

class files

class loader subsystem

method area

heap

Java stacks

pc registers

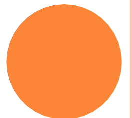
native method stacks

runtime data areas

execution engine

native method interface

native method libraries



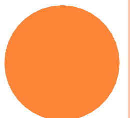
OOP'S FEATURES

- Abstraction:

  It is the property of hiding the information and presenting only the necessary.

Encapsulation:

- It is a mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.

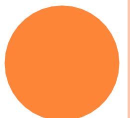


- **Inheritance**

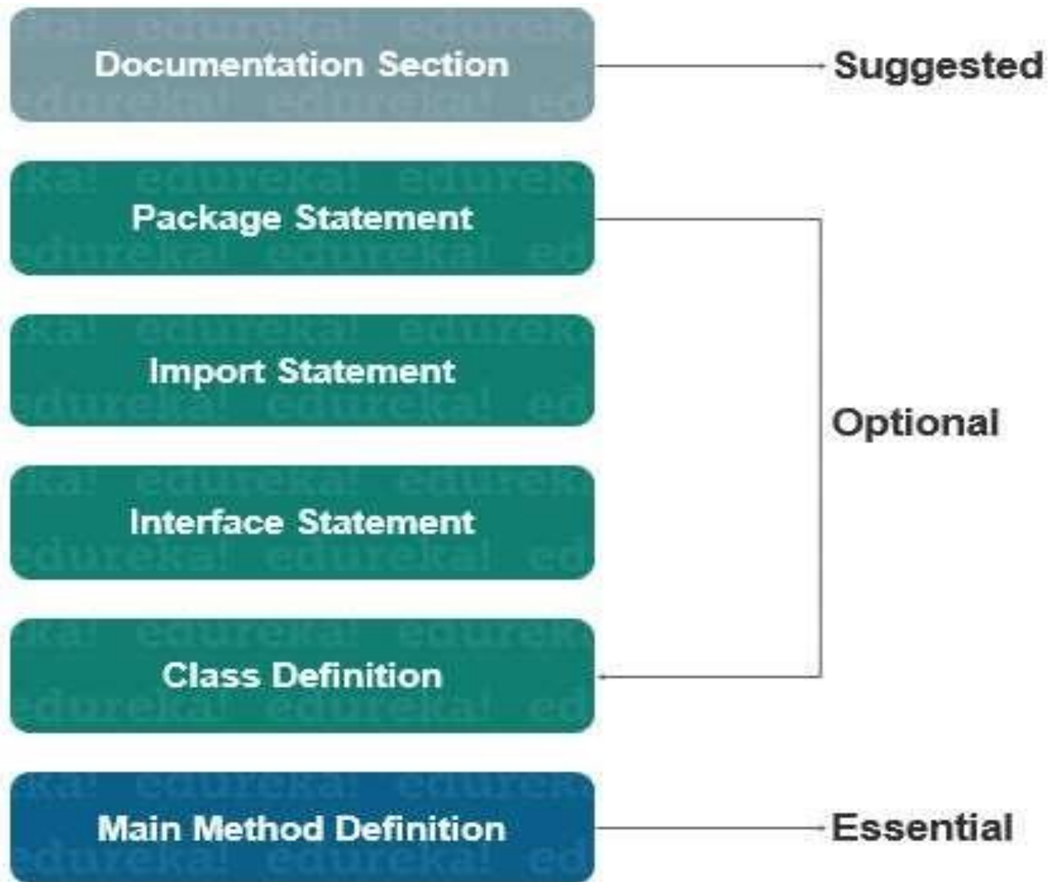
  It is the process by which one object acquires the properties of another object.

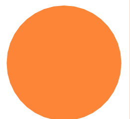
Polymorphism

The concept of polymorphism is “one interface, multiple methods”, i.e. , it is possible to design a generic interface to a group of related activities.



GENERAL STRUCTURE OF JAVA





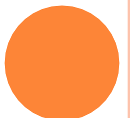
Document Section:

It is use create comment for each section

Package Statement:

- A package is a group of classes that are defined by a name.
- If you want to declare many classes within one element then you can declare it within a package.

```
package package_name;
```

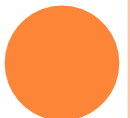


- Import Statement:

It is used to importing the class of another package.

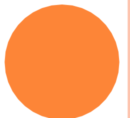
Interface Statement:

It can be used when programmers want to implement multiple inheritances within a program

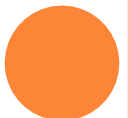


A SIMPLE JAVA PROGRAM

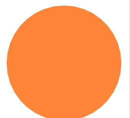
```
/*
     This is a simple Java Program  call
      this file "Example.java"
*/
Class Example{
            //Your program begins with a call to main()
Public static void main(String args[ ])
{
    System.out.println("First java program");
}
}
```



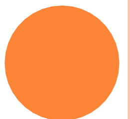
- Java code is case sensitive
- You must have to define the class first.
- The name of the class in java is the name of the java program.



Public class Hello	**This creates a class called Hello**
Comments	It can be used anywhere in the program.
Braces	Two curly brackets are used to group all the commnads
Public static void main	Public it means that it can also be used by code outside of its class. Static used when we want to access a method without creating its object Void indicates that a method does not return a value. Main is a method this is a starting point of a java program



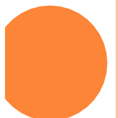
String[ ] args	You can pass the input parameter and main() method takes it as input
System.out.println();	This statement is used to print text on the screen as output.



DATA TYPES

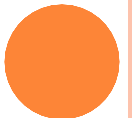
DATA TYPES IN JAVA

- PRIMITIVE (Intrinsic)
  - NUMERIC
    - INTEGER
    - FLOATING POINT
  - NON-NUMERIC
    - CHARECTER
    - BOOLEAN
- NON-PRIMITIVE (Derived)
  - CLASSES
  - INTERFACE
  - ARRAYS



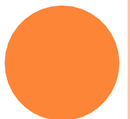
Control statements

- If statement
- If else
- Else if
- switch



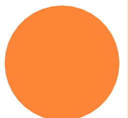
ITERATION STATEMENTS

- While
- Do while
- For
- Foreach



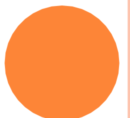
There are various tokens used in Java

- Reserved keywords
- Identifiers
- Literals
- Operators
- Seperators



INTRODUCING CLASSES

- Class fundamentals
- Declaring objects
- Constructors
- This key word
- Garbage collection
- Overloading methods
- Access control
- Final key word
- Nested and inner classes.



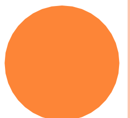
CONSTRUCTORS

A constructor is a method with the same name as that of the class, which is invoked automatically during the creation of an object.

Types of java constructors

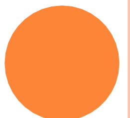
There are two types of constructors:

- Default constructor (no-arg constructor)
- Parameterized constructor.



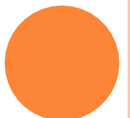
What is class?

- A class is a template or blueprint that is used to create objects.
- Class representation of objects and the sets of operations that can be applied to such objects.
- A class consists of data members and methods.



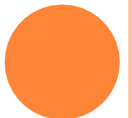
Syntax:

```
Public class class_name
{
       type        instance-variable1;
        type       instance-variable2;
        type instance-variableN;
}
        type methodname1(parameter-list){
         //body of method
}
       type methodnameN(parameter-list){
        //body of method
}
```

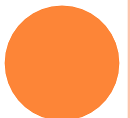


Example

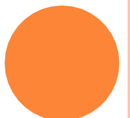
```
class Box
{
   double width;
   double height;
   double depth;
}
```



```
//This class declares an object of type Box
Class BoxDemo{

    public static void main(String args[ ]){  Box
      mybox=new Box();
      double vol;

//assign values to mybox's instance variables
     mybox.width=10;
     mybox.height=20;
     mybox.depth=15;
```

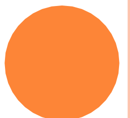


```
//compute volume of box

Vol=mybox.width*mybox.height*mybox.width;

System.out.println(“Volume is” + vol);

}
}
```

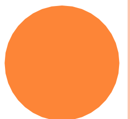


OBJECT

- In real-world an entity that has state ad its behaviour is known as an object.
- A pen is an object.
- Its name is parker,color is silver etc.known as its state.
- It is used to write,so writing is its behaviour.



```
Pen    mypen; //declare reference to object

mypen=new Pen( )://allocate a Pen object
```



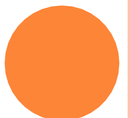
CONSTRUCTORS

A constructor is a method with the same name as that of the class, which is invoked automatically during the creation of an object.

Types of java constructors

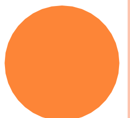
There are two types of constructors:

- Default constructor (no-arg constructor)
- Parameterized constructor.



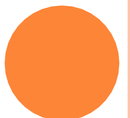
Default Constructor Syntax

```
class className
{
className ()
{
Block of statements;       // Initialization
}
.....
}
```



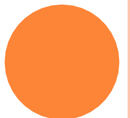
EXAMPLE PROGRAM FOR DEFAULT CONSTRUCTOR

```
class A
{
A()
{
int a=10,b=20;
System.out.println("I am from default Constructor...");
System.out.println("Value of a: "+a);
System.out.println("Value of b: "+b);}
public static void main(String ar[])
{
A a1=new A();
}
}
```

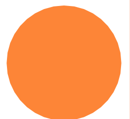


PARAMETERIZED CONSTRUCTOR SYNTAX

```
class ClassName
{
.......
ClassName(list of parameters) //parameterized
   constructor{
.......
}
.......
}
```

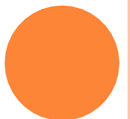


```
class A
{int a=10,b=20;
A(int a,int b)          // variable as a parameter
{
System.out.println("Parameter as variable to Constructor...");
System.out.println("Value of a: "+a);  System.out.println("Value of b:
"+b);
}
A(A a1)      // object as parameter to constructor
{
System.out.println("Parameter as object to Constructor...");
System.out.println("Value of a: "+a1.a);  System.out.println("Value
of b: "+a1.b);
}
public static void main(String ar[])
{A a1=new A(2,3);
A a2=new A(a1);
}
}
```

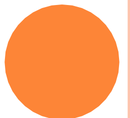


Rules or properties of a constructor

- Constructor will be called automatically when the object is created.
- Constructor name must be similar to name of the class.
- Constructor should not return any value even void also.

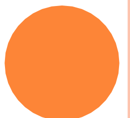


- Constructor definitions should not be static.
- Constructor should not be private provided an object of one class is created in another class
- Constructors will not be inherited from one class to another class



THIS KEYWORD

- this can be used inside any method to refer to the current object
- this is always a reference to the object on which the method was invoked
- this can be passed as an argument in the method call.

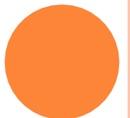


- this() can be used to invoke current class constructor.

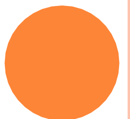
- this keyword can be used to invoke current class method (implicitly)

- this can be passed as argument in the constructor call.

- this keyword can also be used to return the current class instance.



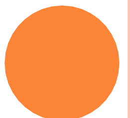
```
Box(double w,double h,double d)
{

      this.width=w;
      this.height=h;
      this.depth=d;

}
```



INSTANCE VARIABLE

- Instance variables are used to overlap the names of the local variable.

```
Box(double width,double height,double depth)
{
  this.width=width;
  this.height=height;
  this.depth=depth;
}
```

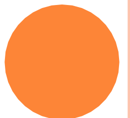


Garbage collection

- New keyword

- Delete Keyword

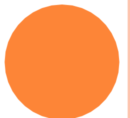
Java takes different approach to de-allocate the memory is called garbage collection



FINAL KEYWORD

In java language final keyword can be used in following way.

- Final at variable level
- Final at method level
- Final at class level

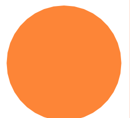


Final at variable level

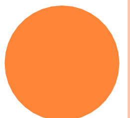
- Final keyword is used to make a variable as a constant. This is similar to const in other language.
- A variable declared with the final keyword cannot be modified by the program after initialization.

Example:

```
class A
{
    final    int a=10;
   public static void main(String ar[])
{
   System.out.println("static variable a="+a1.a); //
    no error
  System.out.println("static variable a="+a1.a++);
    //final variable cannot be modified (error)
}
}
```

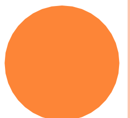


- **Final at method level**
- It makes a method final, meaning that sub classes cannot override this method.
- The compiler checks and gives an error if you try to override the method.
- When we want to restrict overriding, then make a method as a final.



Example:

```
class A
{
final void add()
{
System.out.println("sum="+(2+3));
}
}
class B extends A
{
void add()        // error because final method cannot override
{
System.out.println("sum="+(2+3));
}
public static void main(String ar[])
{ A a1=new A();
a1.add();
}
}
```

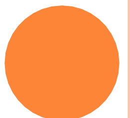


Final at class level

It makes a class final, meaning that the class cannot be inheriting by other classes. When we want to restrict inheritance then make class as a final.

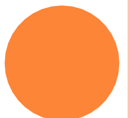
Example:

```
final class A
{
void add()
{
System.out.println("sum="+(2+3));
}
}
class B extends A // error because final class cannot inherited.
{
public static void main(String ar[])
{
A a1=new A();
a1.add();
}
}
```



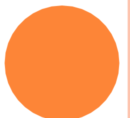
OVERLOADING METHODS

```
//Demonstrate method overloading

Class Overload Demo{
      void test()    {
       System.out.println("No parameters");
                  }
//overload test for one integer parameter
Void test(int a) {
  System.out.println("a:        "  +a);

}
```



CONSTRUCTOR OVERLOADING

```
class A
{
int a=10,b=20;
A(int a,int b)
{
System.out.println("Value of a: "+a);
System.out.println("Value of b: "+b);
      }
A(int a,int b,int c)
{S
ystem.out.println("Value of a: "+a);
System.out.println("Value of b: "+b);
System.out.println("Value of b: "+c);
}
public static void main(String ar[])
{
A a1=new A(2,3)
;A a2=new A(2,3,4);}

}
```

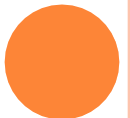


Access control

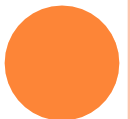
Which parts of a program can access the members of a class.

Access specifies are

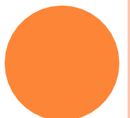
1. Public
2. Private
3. Protected



```
Class Test
{
    int a;
    public int b;
    private int c;

//methods to access c
Void setc(int i)
{
   c=i;
}
Int getc()
{
   return c;
}
}
```



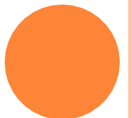
```
Class      AccessTest{
Public static void main(String args[ ]){  Test
            ob=new Test();
             ob.a=10;
             ob.b=20;
             ob.c=100//Error


             ob.setc(100);
             System.out.println("a,b and c:" ob+a          +"
   "+ob.b.+ " " +ob.getc());
}
}
```



Nested and inner class

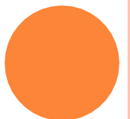
- If one class is existing within another class is known as inner class or nested class

- Inner class properties can be accessed in the outer class with the object reference but not directly.



- Outer class properties can be access directly within the inner class.

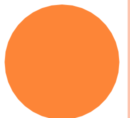
- Inner class properties can't be accessed directly or by creating directly object.



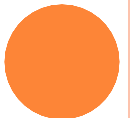
Syntax

Syntex:

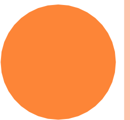
```
Class A
{
  Class B
          {
          }
 }
```

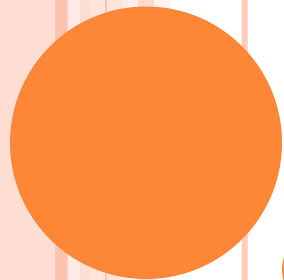
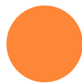
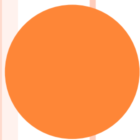


```
class A
{
   int a=10;
void add()
      {
int a=10,b=20;
int c=a+b;
System.out.println("sum="+c);
     }
 class b
{
void display()
{
     System.out.println("outer class variable a="+a);  add();
}
public static void main(String ar[])
{
       A a1=new A();
       a1.add();
}
 }
```



Thank you



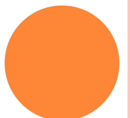




MODULE 2

By
Hamsashree M K
Asst.Professor
Dept.Of ECE
BGSIT,B G Nagara

INHERITANCE

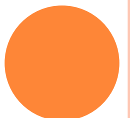
- Inheritance Basics
- Using super
- Creating a multilevel Hierarchy
- When constructors are called
- Method overriding
- Dynamic method Dispatch
- Using Abstract class
- Using final with inheritance
- The object class



INHERITANCE

- The process of obtaining the data members and methods from one class to another class is known as **inheritance**.

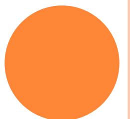
- It is one of the fundamental features of object-oriented programming.



Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
  {
     //methods and fields
  }
```

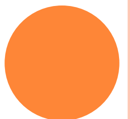
The **extends keyword** indicates that you are making a new class that derives from an existing class.



KEY POINTS

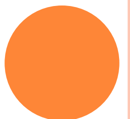
- In the inheritance the class which is give data members and methods is known as base or super or parent class.

- The class which is taking the data members and methods is known as sub or derived or child class.



- The data members and methods of a class are known as features.

- The concept of inheritance is also known as re-usability or extendable classes or sub classing or derivation.

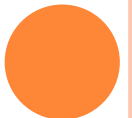


WHY WE USE INHERITANCE

- For Method Overriding

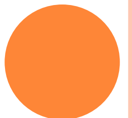
- For method overloading

- For code Re-usability

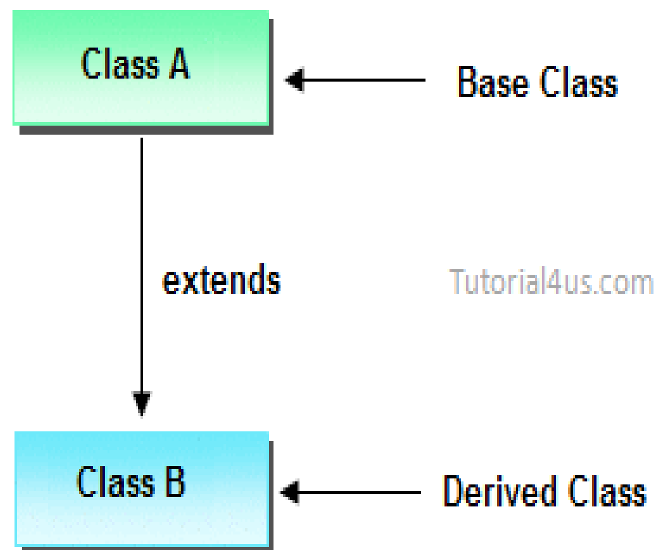


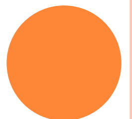
TYPES OF INHERITANCE

- Single inheritance
- Multiple inheritance(Interface)
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



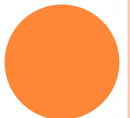
SINGLE INHERITANCE



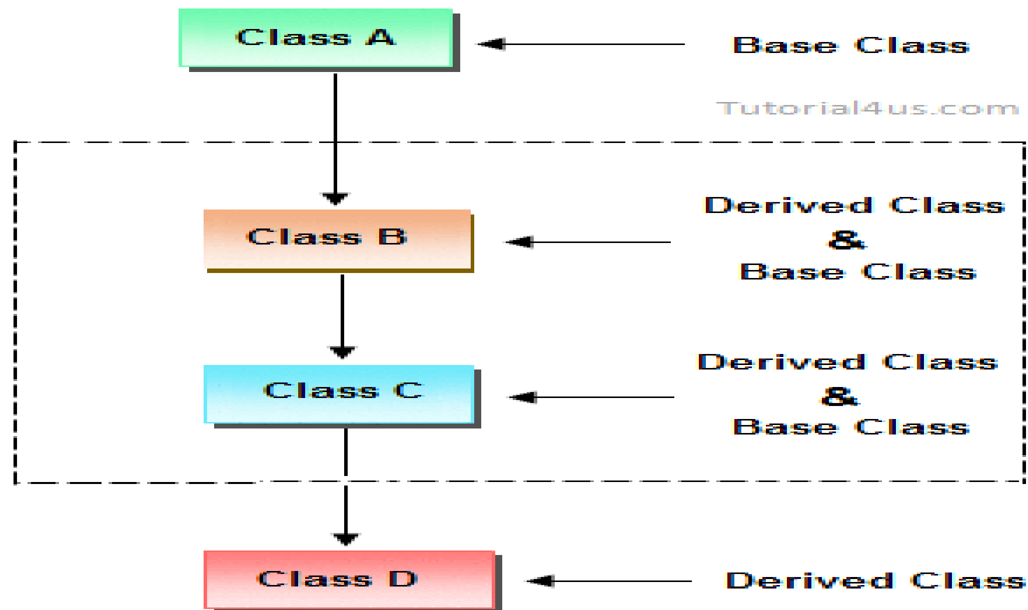


EXAMPLE

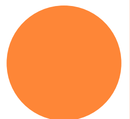
```
class A
{
void display()
{
System.out.println("base class method");
}
}
class B extends A
{
void display2()
{
System.out.println("sub class methods");
}
Public static void main(String ar[])
{
B a1=new B();
a1.display();
a1.display2();
}}
```



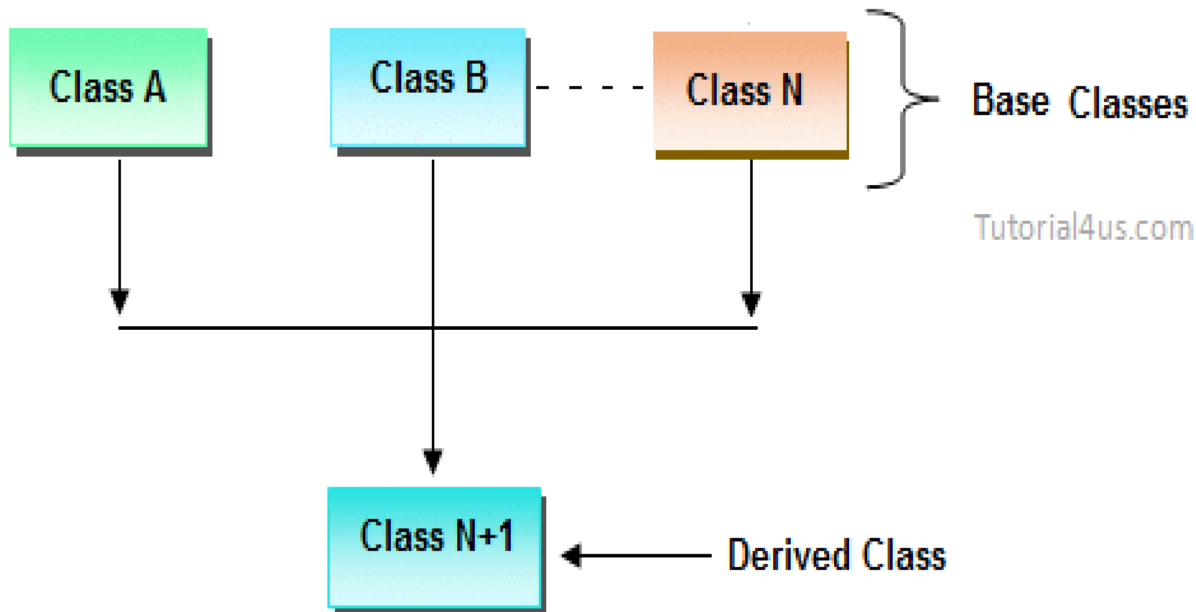
MULTILEVEL INHERITANCE



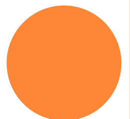
In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.



Multiple inheritance



In multiple inheritance there exist multiple classes and single derived class.



HYBRID AND HEIRARCHICAL

Single Inheritance

Class A

Hierarchical Inheritnce

Class A

Class B

Hierarchical Inheritnce

Class C

Class B

Multiple Inheritance

Class C

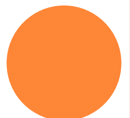
Class D

Class D

Valid by both
Classes and Interface

Tutorial4us.com

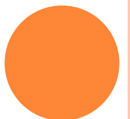
Invalid by Classes
Valid by Interface



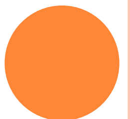
SUPER KEYWORD

- The super is java keyword. As the name suggest super is used to access the members of the super class. It is used for two purposes in java.
- The first use of keyword super is to access the hidden data variables of the super class hidden by the sub class.

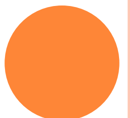
super.member;



```
class A
{
int a; int b; int c;
A(int p, int q, int r)
{
a=p; b=q; c=r;
}
}

class B extends A
{
int d;
B(int l, int m, int n, int o)
{
super(l,m,n); d=o;
}

void Show()
```



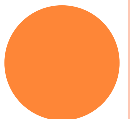
```
class A
{
int a;
float b;
void Show()
{
System.out.println("b in super class: " + b);
}
 }
class B extends A
{
int a;
float b;
B( int p, float q)
{
a = p;
super.b = q;
}
```



```
void Show()
{
super.Show();
System.out.println("b in super class: " + super.b);
System.out.println("a in sub class:  " + a);
}
}
class Mypgm
{
public static void main(String[] args)
{
B subobj = new B(1, 5); subobj.Show();
}
}
```

OUTPUT

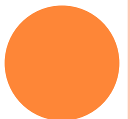
b in super class: 5.0 b in super class: 5.0 a in sub class: 1



Use of super to call super class constructor: The second use of the keyword super in java is to call super class constructor in the subclass.

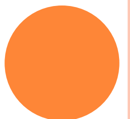
This functionality can be achieved just by using the following command.

super(param-list);



EXAMPLE

```
class A
{
int a; int b; int c;
A(int p, int q, int r)
{
a=p; b=q; c=r;
}
}

class B extends A
{
int d;
B(int l, int m, int n, int o)
{
super(l,m,n); d=o;
}

void Show()
```



```
{
System.out.println("a = " + a); System.out.println("b = " + b);
System.out.println("c = " + c); System.out.println("d = " + d);
}
}
class Mypgm
{
public static void main(String args[])
{
B b = new B(4,3,8,7);
b.Show();
}
}
```

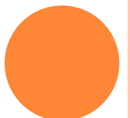
OUTPUT

a = 4

b = 3

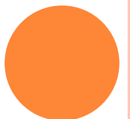
c = 8

d = 7



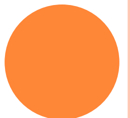
MULTILEVEL HIERARCHY

- When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance.
- The derived class is called the subclass or child class for it's parent class and this parent class works as the child class for it's just above (parent) class.
- Multilevel inheritance can go up to any number of level.



EXAMPLE

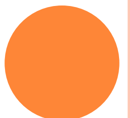
```
class A
{
int x; int y;
int get(int p, int q)
{
x=p; y=q;
return(0);
}
void Show()
{
System.out.println(x);
}
}
```



```
class B extends A
{
void Showb()
{
System.out.println("B");
}
}

class C extends B
{
void display()
{
System.out.println("C");
}
public static void main(String args[])
{
A a = new A(); a.get(5,6);
a.Show();
}
}
```

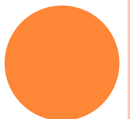
OUTPUT

5



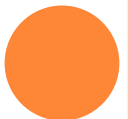
When constructors are called

- Constructors are called in order of derivation ,from super class to subclass ,
- Super() must be the first statement executed in a subclass constructor.
- If super() is not used ,then the default constructor of each super class will be executed.

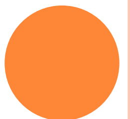


EXAMPLE

```
Class A{
A()
{
System.out.println("Inside A's constructor");
}
```



```
class B extends A{
B(){
System.out.println("Inside B's constructor");
    }
}

Class C extends B{
C()  {
System.out,println("Inside C's constructor");
   }
}
```



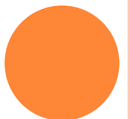
```
Class CallingCons{
Public static void mai(String args[])
{
  C c=new C();
}
}
```

Output;

Inside A's constructor

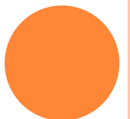
Inside B's constructor

Inside C's constructor



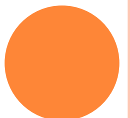
METHOD OVRRIDING

- Method overriding in java means a subclass method overriding a super class method.
- Superclass method should be non-static. Subclass uses extends keyword to extend the super class
- In overriding methods of both subclass and superclass possess same signatures.



Example

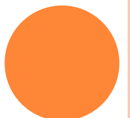
```
class A
{
int i;
A(int a, int b)
{
i = a+b;
}
void add()
{
System.out.println("Sum of a and b is: " + i);
}
}
class B extends A
{
int j;
B(int a, int b, int c)
{
super(a, b); j = a+b+c;
}
```



```
void add()
{
super.add();
System.out.println("Sum of a, b and c is: " + j);
}
}
class MethodOverriding
{
public static void main(String args[])
{
B b = new B(10, 20, 30);
b.add();
}
}
```

OUTPUT

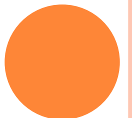
Sum of a and b is: 30 Sum of a, b and c is: 60



Method overloading

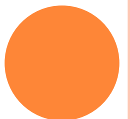
- Two or more methods have the same names but different argument lists.

- The arguments may differ in type or number, or both. However, the return types of overloaded methods can be the same or different is called **method overloading**.



EXAMPLE

```
class MethodOverloading
{
int add( int a,int b)
{
return(a+b);
}
float add(float a,float b)
{
return(a+b);
}
double add( int a, double b,double c)
{
return(a+b+c);
}
}
```



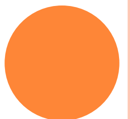
```
class MainClass
{
public static void main( String args[] )
{
MethodOverloading mobj = new MethodOverloading ();
    System.out.println(mobj.add(50,60));
    System.out.println(mobj.add(3.5f,2.5f));
    System.out.println(mobj.add(10,30.5,10.5));
}
}
```

OUTPUT

110

6.0

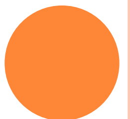
51.0



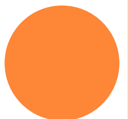
Dynamic method dispatch

- Dynamic method dispatch is the mechanism by which a call to an overridde method at run time rather than compile time.

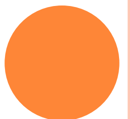
- It is important because this is how java implements run –time polymorphism



```
Class A
{
Void callme() {
System.out.printnl("Inside A's callme method");
}
}

Class B extends A{
//override callme()
Void callme() {
System.out.printl("Inside B's callme method");
}
}
```

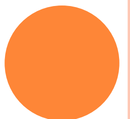


```
Class C extends A {
//override callme()
Void callme() {
System.out.println("Inside C's callme method");
}
}

Class Dispatch {
Public static void main(String args[ ] ) {
A a =new A();
B b=new B();
C c=new C();
A r;
```



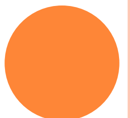
```
r=a;        //r refers to an A object
r.callme(); //calls A's version of callme


r=b;        //r refers to an B object
r.callme(); //calls B's version of callme

r=c;        //r refers to an C object
r.callme(); //calls C's version of callme
```

OUTPUT:
Inside A's callme method
Inside B's callme method
Inside C's callme method



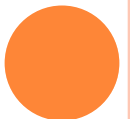
Abstract classes

- **abstract keyword** is used to make a class abstract.
- Abstract class can't be instantiated with new operator.
- We can use abstract keyword to create an abstract method; an abstract method doesn't have body.

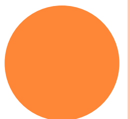


- If classes have abstract methods, then the class also needs to be made abstract using abstract keyword, else it will not compile.

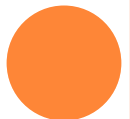
- Abstract classes are used to provide common method implementation to all the subclasses or to provide default implementation.



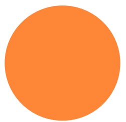
```
abstract Class AreaPgm
{
double dim1,dim2; AreaPgm(double x,double y)
{
dim1=x; dim2=y;
}
abstract double area();
}
class rectangle extends AreaPgm
{
rectangle(double a,double b)
{
super(a,b);
}
double area()
{
System.out.println("Rectangle Area"); return dim1*dim2;
}
}
```



```
class triangle extends figure
{
triangle(double x,double y)
{
super(x,y);
}
double area()
{
System.out.println("Traingle Area"); return dim1*dim2/2;
}
}
class MyPgm
{
public static void main(String args[])
{

AreaPgm a=new AreaPgm(10,10); // error, AreaPgm is a abstract class.

rectangle r=new rectangle(10,5); System.out.println("Area="+r.area());

triangle t=new triangle(10,8); AreaPgm ar;
ar=obj; System.out.println("Area="+ar.area());
}
}
```



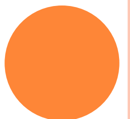
```
{
public static void main(String args[])
{

AreaPgm a=new AreaPgm(10,10); // error,
   AreaPgm is a abstract class.

rectangle r=new rectangle(10,5);
   System.out.println("Area="+r.area());

triangle t=new triangle(10,8); AreaPgm ar;
ar=obj; System.out.println("Area="+ar.area());
}
}
```



Using final with inheritance

The **final keyword** in java is used to restrict the user. The final keyword can be used in many context. Final can be:

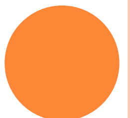
- variable
- method
- class



EXAMPLE PROGRAM

```
class Bike
{
final int speedlimit=90;//final variable
void run()
{
speedlimit=400;
}
}
Class MyPgm
{
public static void main(String args[])
{
Bike obj=new Bike(); obj.run();
}
}
```

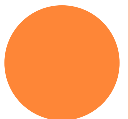
Output:
Compile Time Error



final method: If you make any method as final, you cannot override it.

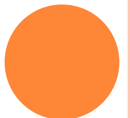
Example:

```
class Bike
{
final void run()
{
System.out.println("running");
}
}
class Honda extends Bike
{
void run()
{
System.out.println("running safely with 100kmph");
}
}
```



```
Class MyPgm
{
public static void main(String args[])
{
Honda honda= new Honda(); honda.run();
}
}
```

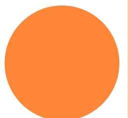
Output:Compile Time Error



final class: If you make any class as final, you cannot extend it.

Example:

```
final class Bike
{

}

class Honda extends Bike
{
void run()
{
System.out.println("running safely with 50kmph");
}
}
```



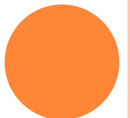
Class MyPgm

{

```
public static void main(String args[])
{
Honda honda= new Honda(); honda.run();
}
```

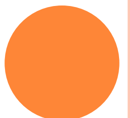
}

Output:Compile Time Error



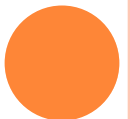
The object class

Method	
Object clone()	
Boolean equals(Object object)	
Void finalize()	
Class getClass()	
Int hashCode()	
Void notify()	
Void notifyAll()	
String toString()	
Void wait() Void wait(long milliseconds) Void wait(long milliseconds, int nanoseconds)	



Packages and Interfaces

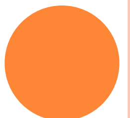
- Packages
- Access Protection
- Importing Packages
- Interfaces



PACKAGES

- A **java package** is a group of similar types of classes, interfaces and sub- packages.

- Package in java can be categorized in two form,
  - built-in package and
  - user-defined package.

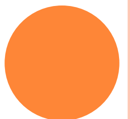


Advantage of Java Package

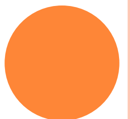
- Java package is used to categorize the classes and interfaces so that they can be easily maintained.

- Java package provides access protection.

- Java package removes naming collision.



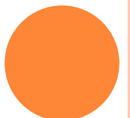
```
//save as Simple.java package mypack; public class
   Simple
{
public static void main(String args[])
{
System.out.println("Welcome to package");
}
}
```



Example of package that import the packagename.*

```
//save by A.java package pack; public class A
{
public void msg(){System.out.println("Hello");}
}

//save by B.java package mypack; import pack.*;
class B
{
public static void main(String args[])
{
A obj = new A(); obj.msg();
}
}
```

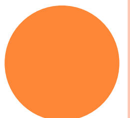
Output:Hello



Example of package by import package.classname

```
//save by A.java
package pack;
public class A
{
public void msg(){System.out.println("Hello");

    }
}

//save by B.java package mypack; import pack.A;
class B
{
public static void main(String args[])
{
A obj = new A(); obj.msg();
}
}
```

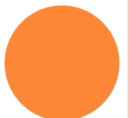
Output:Hello



Example of package by import fully qualified name

```
//save by A.java package pack; public class A
{
public void msg()
{
System.out.println("Hello");
}
}

//save by B.java package mypack; class B
{
public static void main(String args[])
{
pack.A obj = new pack.A();//using fully qualified name
obj.msg();
}
```

Output:Hello

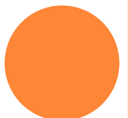


Access Modifiers/Specifiers

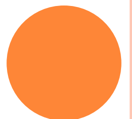
The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

- private
- default
- protected
- public

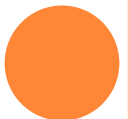


Access Modifier	within class	within package	Outside Package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

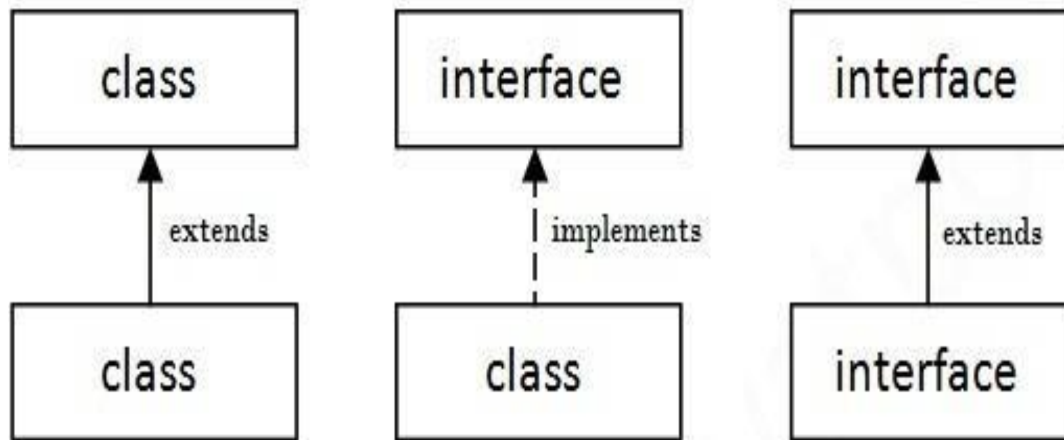


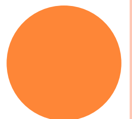
Interfaces

- An **interface in java** is a blueprint of a class. It has static final variables and abstract methods.
- It is used to achieve abstraction.
- It is used to achieve abstraction and multiple inheritance in Java.
- Interface fields are public, static and final by default, and methods are public and abstract.



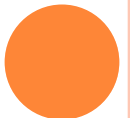
UNDERSTANDING RELATIONSHIP BETWEEN CLASSES AND INTERFACES





Example 1

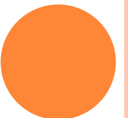
```
interface printable
{
void print();
}

class Pgm1 implements printable
{
public void print()
{
System.out.println("Hello");
}
}
```



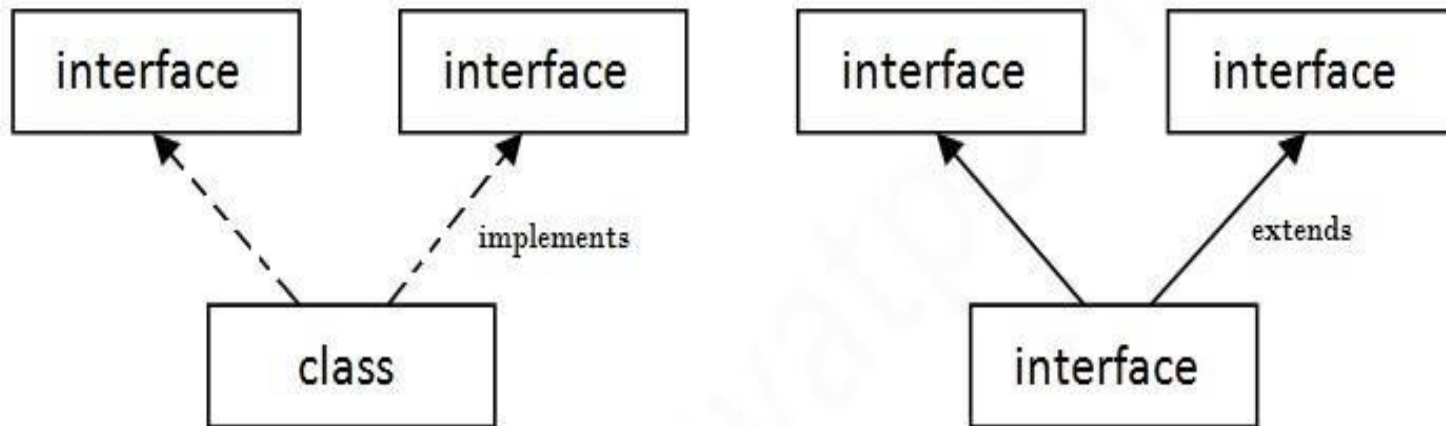
```
class IntefacePgm1
{
public static void main(String args[])
{
Pgm1 obj = new Pgm1 (); obj.print();
}
}
```

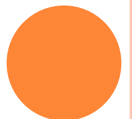
Output:

Hello



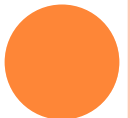
Multiple inheritance in Java by interface





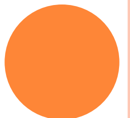
Example

```
interface Printable
{
void print();
}

interface Showable
{
void show();
}

class Pgm2 implements Printable,Showable
{
public void print()
{
System.out.println("Hello");
}
```



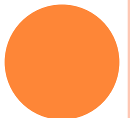
```
public void show()
{
System.out.println("Welcome");
}
}
Class InterfaceDemo
{
public static void main(String args[])
{
Pgm2 obj = new Pgm2 (); obj.print();
obj.show();
}
}
```

Output: Hello Welcome



EXAMPLE 2

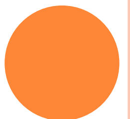
```
interface Printable
{
void print();
}

interface Showable
{
void print();
}

class InterfacePgm1 implements Printable, Showable
{
public void print()
{
System.out.println("Hello");
}
}
```



```
class InterfaceDemo
{
public static void main(String args[])
{
InterfacePgm1 obj = new InterfacePgm1 ();
obj.print();
}
}
```

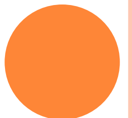
Output:

Hello

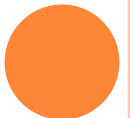


Interface inheritance

```
interface Printable
{
void print();
}

interface Showable extends Printable
{
void show();
}
```



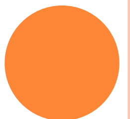
```
class InterfacePgm2 implements Showable
{
public void print()
{
System.out.println("Hello");
}
public void show()
{
System.out.println("Welcome");
}
```



```
Class InterfaceDemo2
{
public static void main(String args[])
{
InterfacePgm2 obj = new InterfacePgm2 ();
   obj.print();
obj.show();
}
}
```

Output:

Hello Welcome

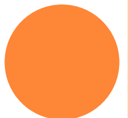


Multithreded programming

By
Hamsashree M K
Assistant Professor
Dept. Of ECE
BGSIT,B G Nagara

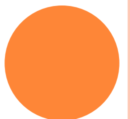
MULTITHREADED PROGRAMMING

- The Java Thread Model
- The Main Method
- Creating a Thread
- Creating Multiple Thread
- Using isAlive() and join()
- Thread Priorities
- Synchronization
- Interthread Communication
- Suspending,Resuming and Stopping Threads
- Using Multithreading

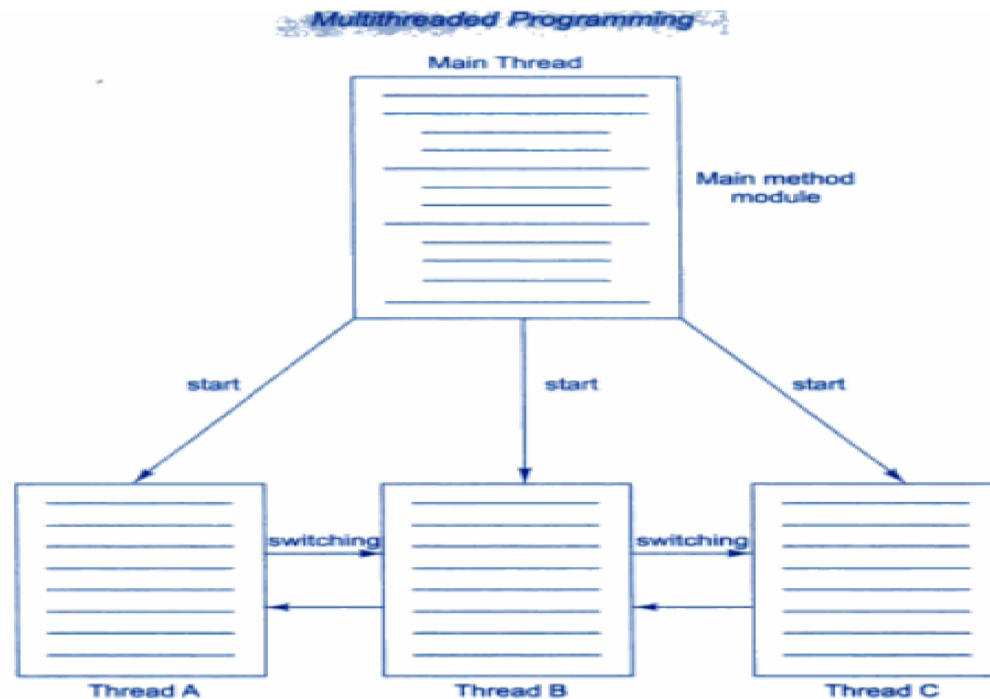


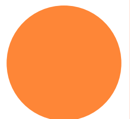
MULTITHREADING

- Multithreading is a conceptual programming paradigm where a program is divided into two or more subprogram.
- Which can be implemented at the same time in parallel.
- Multithreading is a specialized form of multitasking.



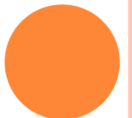
- In a *thread-based* multitasking environment, the thread is the smallest unit of dispatch able code. This means that a single program can perform two or more tasks simultaneously.





THREAD

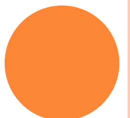
- The small unit of program or sub module is called as thread.
- Each thread defines a separate path of execution
- Main thread has the ability to create additional threads.



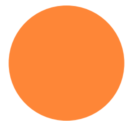
Life cycle of a thread

Thread can enter into different state during life of thread, different stages in thread are as follows:

- New born
- Runnable
- Running
- Dead
- Blocked



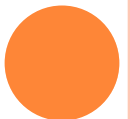
New Thread
Newborn
start
stop
Active
Thread
Running
Runnable
yield
stop
Dead
Killed
Thread
suspend
sleep
wait
resume
notify
stop
Idle Thread
(Not Runnable)
Blocked



Creating a Thread

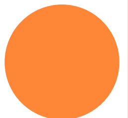
Thread can be created in two ways:

- By creating a thread class(**Extending Thread class**)
- By converting a class to a Thread Class(**Implementing Runnable interface**)



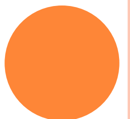
BY CREATING A THREAD CLASS(EXTENDING THREAD CLASS):

```
class class_name extends Thread
{
public void run()
{
/*Implement actual code*/
}
public static void main(String ar[])
{
class_name object=new class_name();
object.start();
}
}
```

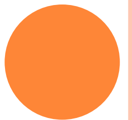


Example

```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("A class="+i);
}
}
class B extends A
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("class B="+i);
}
```

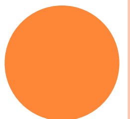


```
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.start();
b1.start();
}
}
```



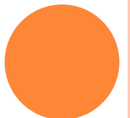
By converting a class to a Threadable Class(Implementing Runnable interface)

```
class Class_Name implements Runnable
{
public void run()
{
/* Implements operation */
}
public static void main(String ar[])
{
Class_Name object=new Class_Name();
Thread object1=new Thread(object);
object1.start();
}
}
```

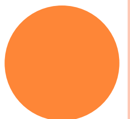


EXAMPLE

```
class A implements Runnable
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("Class A="+i);
}
}
class B implements Runnable
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println("class B="+i);
}
```

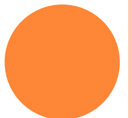


```
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
Thread t1=new Thread(a1);
Thread t2=new Thread(b1);
t1.start();
t2.start();
}
}
```



Thread Methods

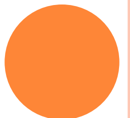
- **Yield()**
- **stop()**
- **suspend()**
- **resume()**
- **wait()**
- **notify()**
- **notifyall().**



YIELD() METHOD

- Calling **yield()** will move the current thread from running to runnable, to give other threads a chance to execute.

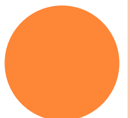
```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) yield();
}
}
}
class B
{
public static void main(String ar[])
{
A a1=new A();
a1.start();
}
}
```



STOP() METHOD

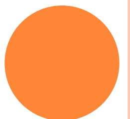
When stop() is called then processor will kill thread permanently.It means thread move to dead state.

```
class A extends Thread
{

public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) stop();
}
}
}

class B
{
public static void main(String args[])
{
A a1=new A();
a1.start();
}
}
```

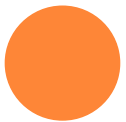


Sleep() and suspend() method

```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) try { sleep(100);}
 catch(Exception e){
 s.o.p(e) resume();}
}
}
}
class B extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
if(i==2) suspend();
}
}
}
```



```
class Mainclass
{
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.start();
b1.start();
}
}
```



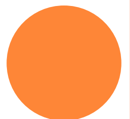
Thread priorities

- Each thread assigned a priority, which effects the order in which it is scheduled for running.

- Thread of same priority are given equal treatment by the java scheduler and there for they share the processor on FCFS basis

- Java permits us to set the priority of the thread using setPriority()methods.

Final void setPriority(int level)



- Where level specify the new priority setting for the calling thread. Level is the integer constant as follows:

MAX_PRIORITY

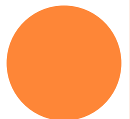
MIN_PRIORITY

NORM_PRIORITY

- The MAX_priority value is 10,MIN_PRIORITY values is 1 And NORM_PRIORITY is the default priority whose value is 5.

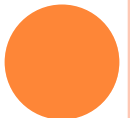
- We can also obtain the current priority setting value by calling getPriority() method of thread.

Final int getPriority()

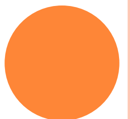


EXAMPLE

```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
s.o.p(“class a thread=”+i);
}
}
}
class B extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
s.o.p(“Class b thread=”+i);
}
}
```

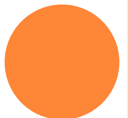


```
class MainThread
{
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.setPriority(Thread.MAX_PRIORITY);
b1.setPriority(Thread.MIN_PRIORITY);
a1.start();
b1.start();
b1.setPriority(a1.getPriority()+10);
}
}
```

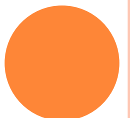


Creating multiple thread

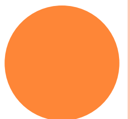
```
//Create multiple thread
Class NewThread implements Runnable{
String Name;
Thread t;

NewThread(String threaName){
 name=threadName;
 t=new Thread(this,name);
System.out.println("New thread:" +t);
t.start();
}
```



```
//This is the entry point for thread
Public void run()
{
 try{
 for(int i=5;i>0;i++)
 System.out.println(name + ":" +i);
 Thread.sleep(100);
}
}
Catch(InterruptedException e){
System.out.println(name + "Interrupted");
}
System.out.println(name + "existing.");
}
}
```

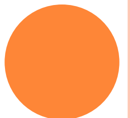


```
class MultiThreadDemo{
public static void main(string args[]){
new NewThread("One"); //start threads
new NewThread("Two");
new NewThread("Three");

try{
 //wait other threads to end
  Thread.sleep(10000);
  }catch(InterruptedException e) {
    System.out.println("Main thread Interrupted");
}
 System.out.println("Main thread exiting");
}
}
```



OUTPUT

New thread:Thread [One, 5,main]
New thread:Thread [Two, 5,main]
New thread:Thread [Three, 5,main]
One:5
Two:5
Three:5
One:4
Two:4
Three:4
One:3
Two:3
Three:3
One:2
Two:2
Three:2
One:1
Two:1
Three:1
One Exiting
Two Exiting
Three Exiting.
Main thread exiting

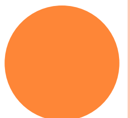


ISALIVE() AND JOIN()

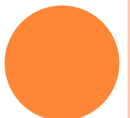
- The final **isAlive()** method returns true if the thread is still running or the Thread has not terminated.

final join()

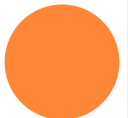
- The final **join()** method waits until thread on which it is called is terminated. For example, thread1.join() suspends the current thread until thread1 dies.
- The join() method can throw an InterruptedException if the current thread is interrupted by another thread.



```
class A extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println(“class A=”+i);
}
}
Class B extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
System.out.println(“class B=”+i);
}
}
```

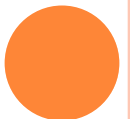


```
class C
{
public static void main(String ar[])
{
A a1=new A();
B b1=new B();
a1.start();
b1.start();
System.out.println(a1.isAlive());
System.out.println(b1.isAlive());
try
{
A1.join();
B1.join();
}
catch(Exception e)
{
System.out.println(e);
}
System.out.println(“main thread dead”);
}
}
```



Synchronization

- When two or more thread needs access to the shared resource, they need some way to ensure that the resource will be used by only one thread at a time.
- The process by which this is achieved is called synchronization.
- Key to synchronized is the concepts of monitor or semaphores.



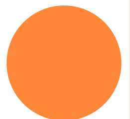
- A monitor is an object that is used as mutually exclusive lock or mutex. Only one thread can own a monitor at a given time. When one thread acquires a lock it is said to have entered the monitor.

- All other thread attempting to enter the locked monitor will be suspended until the first thread exits the monitor. These other thread are said to be waiting for monitor.

- This can be achieved by using keyword synchronized to method.

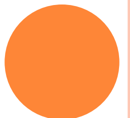
Syntex:

```
synchronized void method_name()
{  /* implementation or operation
}
```

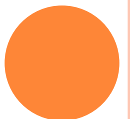


Example program

```
class A
{
Synchronized void display()
{
for(int i=0;i<10;i++)
System.out.pprintln("i="+i);
}
}
class B extends Thread
{
public void run()
{
A a1=new A();
System.out.println("class A thread");
For(int i=0;i<10;i++)
a1.display();
}
}
```



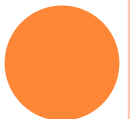
```
class C extends Thread
{
public void run()
{
A a1=new A();
Sysem.out.println("class B thread");
For(int i=0;i<10;i++)
a1.display();
}
}
class D
{
public static void main(String ar[])
{
B b1=new B();
C c1=new C();
b1.start();
c1.start();
}
}
```



INTER-THREAD COMMUNICATION

- Inter-thread communication can be defined as exchange of message between two or more threads. The transfer of message takes place before or after changes of state of thread.
- The inter-thread communication can be achieved with the help of three methods as follows:

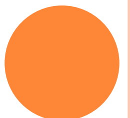
Wait(),notify() notifyall()



- **Wait()-** tells the calling thread to give up the monitor and go to sleep mode until some of other thread enters the same monitor and call the notify() methods

- **Notify()**-wakes up a thread that called wait() method on the same object.

- **Notifyall()-**wakes up all thread that called wait() methods on the same object.

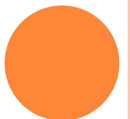


PRODUCER CONSUMER/BOUNDED BUFFER PROBLEM

- Producer thread goes on producing an item unless an until buffer is full.

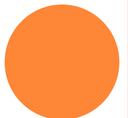
- Producer thread check before producing an item weather buffer is full or not.

- Consumer thread goes on consuming an item which is produced by the producer.

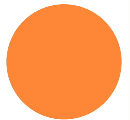


EXAMPLE

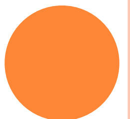
```
class A
{
int stack[]=new int[10];
int top=-1;
Synchronized void produce(int item)
{
if(top==10)
try
{
 wait();
}
catch(Exception e)
{
System.out.println(e);
}
Stack[++top]=item;
notify();
}
```



```
Synchronized void consume()
{
if(top==-1)
try
{
 wait();
}
catch(Exception e)
{
System.out.println(e);
}
item=Stacktop++];
System.out.println("consumed item is"+item);
notify();
}
}
class Producer extends Thread
{
public void run()
{
A a1=new A();
for(int i=0;i<10;i++)
a1.produce(i);
}
}
```

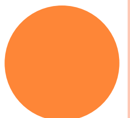


```
class Consumer extends Thread
{
public void run()
{
A a1=new A();
for(int i=0;i<12;i++)
a1.consume();
}
}
class MainThread
{
public static void main(String args[])
{
Produce p=new Produce();
Consume c=new Consume();
p.start();
c.start();
}
}
```



Reader-writer problem

- Reader thread reading an item from the buffer, Where as writer thread writing an item to buffer.
- If reader is reading then writer has to wait unless and until reading is finish.
- While writing thread writing an content then no other thread read the content unless and until writing is over.



THANK YOU