S J P N Trust's

HIRASUGAR INSTITUTE OF TECHNOLOGY, NIDASOSHI.

Inculcating Values, Promoting Prosperity

Approved by AICTE, Recognized by Govt. of Karnataka and Affiliated to VTU Belagavi.

Accredited at 'A' Grade by NAAC

Programmes Accredited by NBA: CSE, ECE, EEE & ME

Subject: Object Oriented Concepts (18CS45)

Module 3: Classes, Inheritance and Exception Handling

Dr. Mahesh G Huddar

Dept. of Computer Science and Engineering

Fundamentals of Classes in Java

A class can be defined as an entity in which data and functions are put together.

The concept of class is similar to the concept of structure in C.

A class is declared by using the keyword class.

Fundamentals of Classes in Java

The general form of class is

```
class classname {
      type variable 1;
      type vanable2;
      type method(parameter-list)
      {
      }
      type method2(paramteter-list)
      {
      }
      type method n(paramteter-list)
      {
      }
}
```

Data Field Declaration

- The data lies within the class and the data fields are accessed by the methods of that class.
- The data fields are also called as instance variables or member variables because they are created when the objects get instantiated.
- **For example -**

```
class Test
{
        int a;
        int b;
}
```

Method Field Declaration

- In object oriented programming any two objects communicate with each other using methods.
- All the methods have the same general form as the method main(). Most of the methods are specified as either static or public.
- Java classes do not need the method main, but if you want particular class as a starting point for your program then the main method is included in that class.

Method Field Declaration

- The general form of method is –

```
type method(parameter-list)
{
}
```

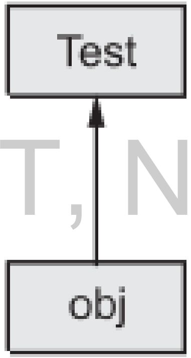
- The type specifies the type of data returned from the method. If the method does not return anything then its data type must be void.
- For returning the data from the method the keyword return is used.

Declaring Objects

- Objects are nothing but the instances of the class. Hence a block of memory gets allocated for these instance variables.
- For creating objects in Java the operator new is used.
- Test obj // Declaration of object obj.
- obj = new Test () ; // obj gets instantiated for class Test
- We instantiate one object obj; it can be represented graphically as -

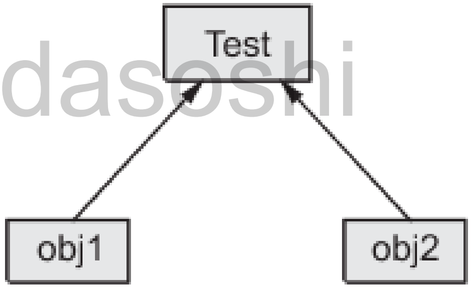
Declaring Objects

- It can be represented graphically as -



Declaring Objects

- Now we can two objects and instantiate them,
- Test obj1, obj2;
- obj1 = new Test ()
- obj2 = new Test ()
- It can be represented graphically as -



Accessing Variables and Methods

- There are two types of class members - The data members and the method.
- These members can be accessed using dot operator.
- For accessing the data members the syntax is –

name_of_object.variable_name = value;

- For accessing the method of the class the syntax is

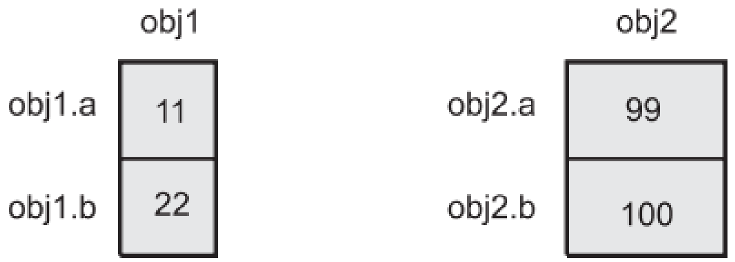
name_of_object.method_name(parameter_list);

Accessing Variables and Methods

- For example –
- obj.name="XYZ"
- obj.display();
- Suppose we wish pass the parameters to the method then first we will create the object for the class as follows –
- Test obj1=new Test();
- Test obj2=new Test();

Accessing Variables and Methods

- Now two objects are created namely obj1 and obj2.
- Obj1.get_val(11,22);
- obj2.get_val(99,100);
- Suppose by this method we assign values to two variables a and b then
- It can be graphically represented as -



Accessing Variables and Methods

```
/*This is a Java program which shows the use of class in the
program */

class Rectangle
{
            int height;
            int width;
            void area()
            {
                        int result=height*width;
                        System.out.println('The area is "+result);
            }
}
```

```
//Another class in which main() function is written.

class classDemo
{
            public static void main(String args[])
            {
                        Rectangle obj=new Rectangle();
                        obj.height=10;  //setting the attribute
values
                        obj.width=20;   //from outside of class
                        obj.area();      //using object method of
class is called
            }
}
```

OUTPUT

```
The area is 200
```

Accessing Variables and Methods

- In the above program we have used two classes one class is classDemo which is our usual one in which the main function is defined and the other class is Rectangle.
- In this class we have used height and width as attributes and one method area() for calculating area of rectangle. In the main function we have declared an object of a class as

Rectangle obj=new Rectangle();

Accessing Variables and Methods

- And now using obj we have assigned the values to the attributes of a class. The operator new is used to create an object.
- The objects access the data fields and methods of the class using the dot operator. This operator is also known as the object member access operator.
- Thus data field height and width are called as instance variables.
- And the method, area is referred as instance method. The object on which the instance method is invoked is known as calling object.

Accessing Variables and Methods

Data Hinding

- The use of class allows to hide the important data from outsider of the class.
- If the data being hidden declared as public, then only these members will be accessible from outside class.
- Thus class helps in unauthorized access to its data members.

Constructors

- It is a special method which is used to initialize the values of instance-variables at the time of creation of objects.
- **Features:**
  - Constructor will have same name as that of class name.
  - It does not specify a return type not even void.
  - It should be declared in public section.

Properties of Constructors

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the public mode.
3. The constructor gets invoked automatically when an object gets created.
4. The constructor should not have any return type. Even a void data type should not be written for the constructor.
5. The constructor can not be used as a member of union or structure.

Properties of Constructors

6. The constructors can have default arguments.
7. The constructor can not be inherited. But the derived class can invoke the constructor of base class.
8. Constructor can make use of new or delete operators for allocating or releasing memory respectively.
9. Constructor can not be virtual. Multiple constructors can be used by the same class.
10. When we declare the constructor explicitly then we must declare the object of that class.

Type of Constructors

- Types of Constructors:

  1. Default constructor: It is a constructor which do not take any argument.

  2. Parameterized constructor: It is a constructor which takes any number of parameters.

- Default constructor is automatically loaded by the compiler.

Default Constructors

```
class B
{
        int x;
        B( )
        {
                System.out.println("Constructing Box");
                x = 10;
        }
        void show( )
        {
             System.out.println("x = " + x);
        }
}
```

```
class MB
{
        public static void main(String args[])
         {
                B b1 = new B( );
                b1.show( );
        }
}
```

Output:

```
Constructing Box
x = 10
```

Parameterized Constructors

```
class B
{
    int x, y;
    B(int a, int b)
    {
        x = a;
        y = b;
    }
    void show( )
    {
        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }
}
```

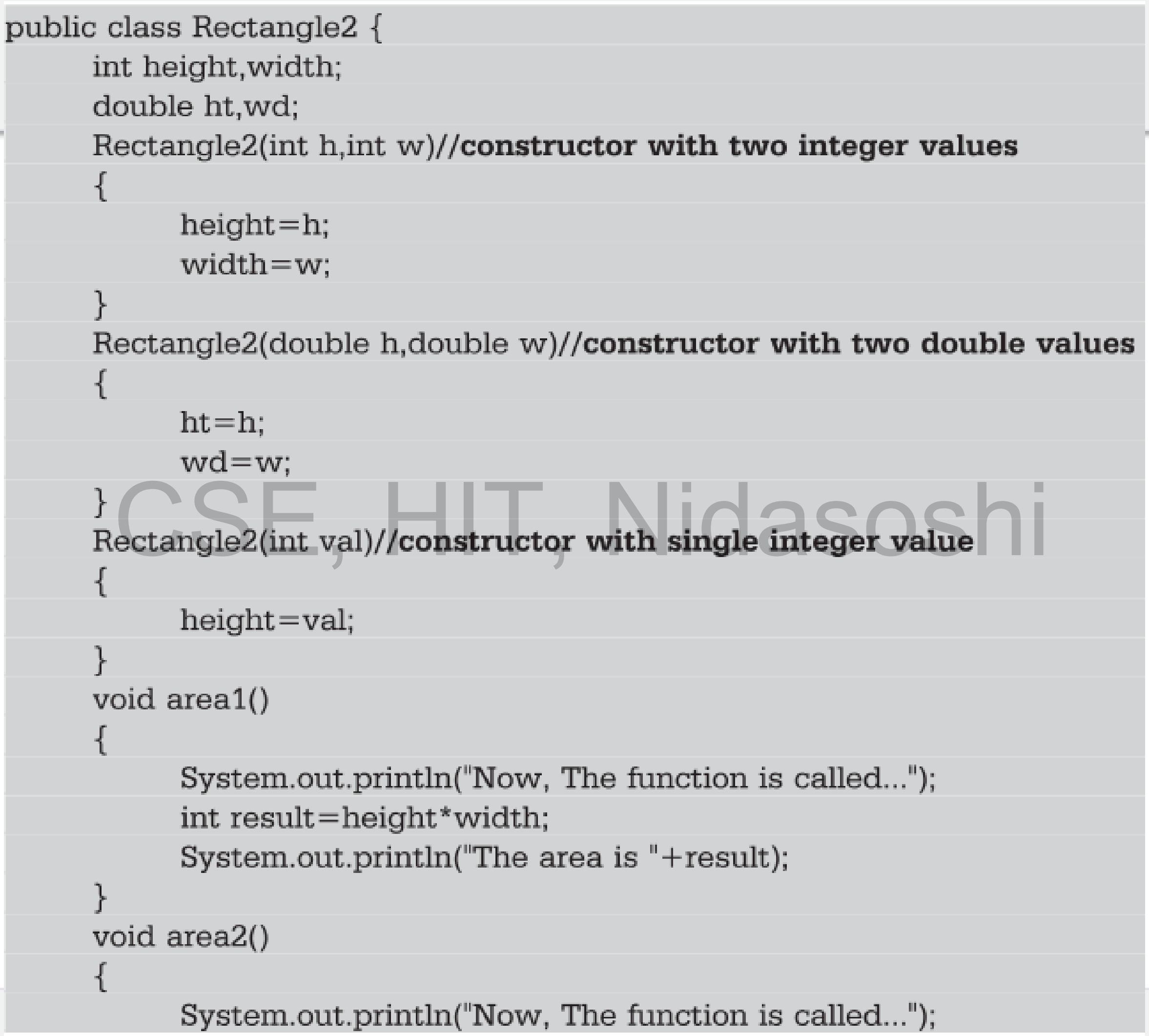
```
class MB
{
    public static void main(String args[])
    {
        B b1 = new B(4, 5);
        b1.show( );
    }
}
```

Output:

```
x = 4
y = 5
```

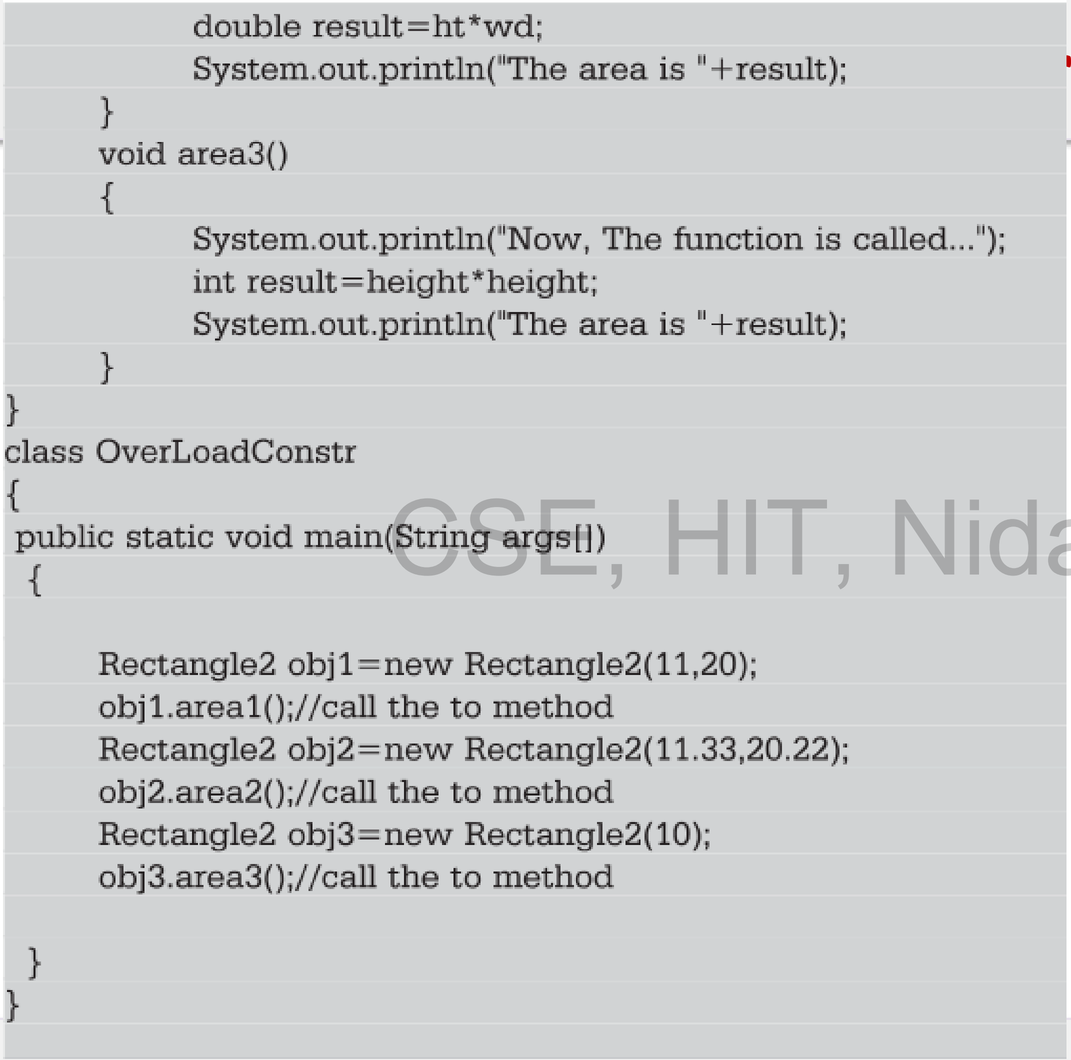
Constructor Overloading

- Overloading is one of the important concept in Object Oriented Programming.
- Similar to methods the constructors can also be overloaded.
- Constructor overloading in Java allows to have more than one constructor inside one Class.
- Multiple constructor with different signature with only difference that Constructor doesn't have return type in Java.
- Those constructor will be called as overloaded constructor.



```
public class Rectangle2 {
        int height,width;
        double ht,wd;
        Rectangle2(int h,int w)//constructor with two integer values
        {
                height=h;
                width=w;
        }
        Rectangle2(double h,double w)//constructor with two double values
        {
                ht=h;
                wd=w;
        }
        Rectangle2(int val)//constructor with single integer value
        {
                height=val;
        }
        void area1()
        {
                System.out.println("Now, The function is called...");
                int result=height*width;
                System.out.println("The area is "+result);
        }
        void area2()
        {
                System.out.println("Now, The function is called...");
```

loading



```
            double result=ht*wd;
            System.out.println("The area is "+result);
        }
        void area3()
        {
            System.out.println("Now, The function is called...");
            int result=height*height;
            System.out.println("The area is "+result);
        }
}
class OverLoadConstr
{
 public static void main(String args[])
  {

        Rectangle2 obj1=new Rectangle2(11,20);
        obj1.area1();//call the to method
        Rectangle2 obj2=new Rectangle2(11.33,20.22);
        obj2.area2();//call the to method
        Rectangle2 obj3=new Rectangle2(10);
        obj3.area3();//call the to method

  }
}
```

Output:

```
Now, The function is called...
The area is 220
Now, The function is called...
The area is 229.09259999999998
Now, The function is called...
The area is 100
```

Constructor Overloading

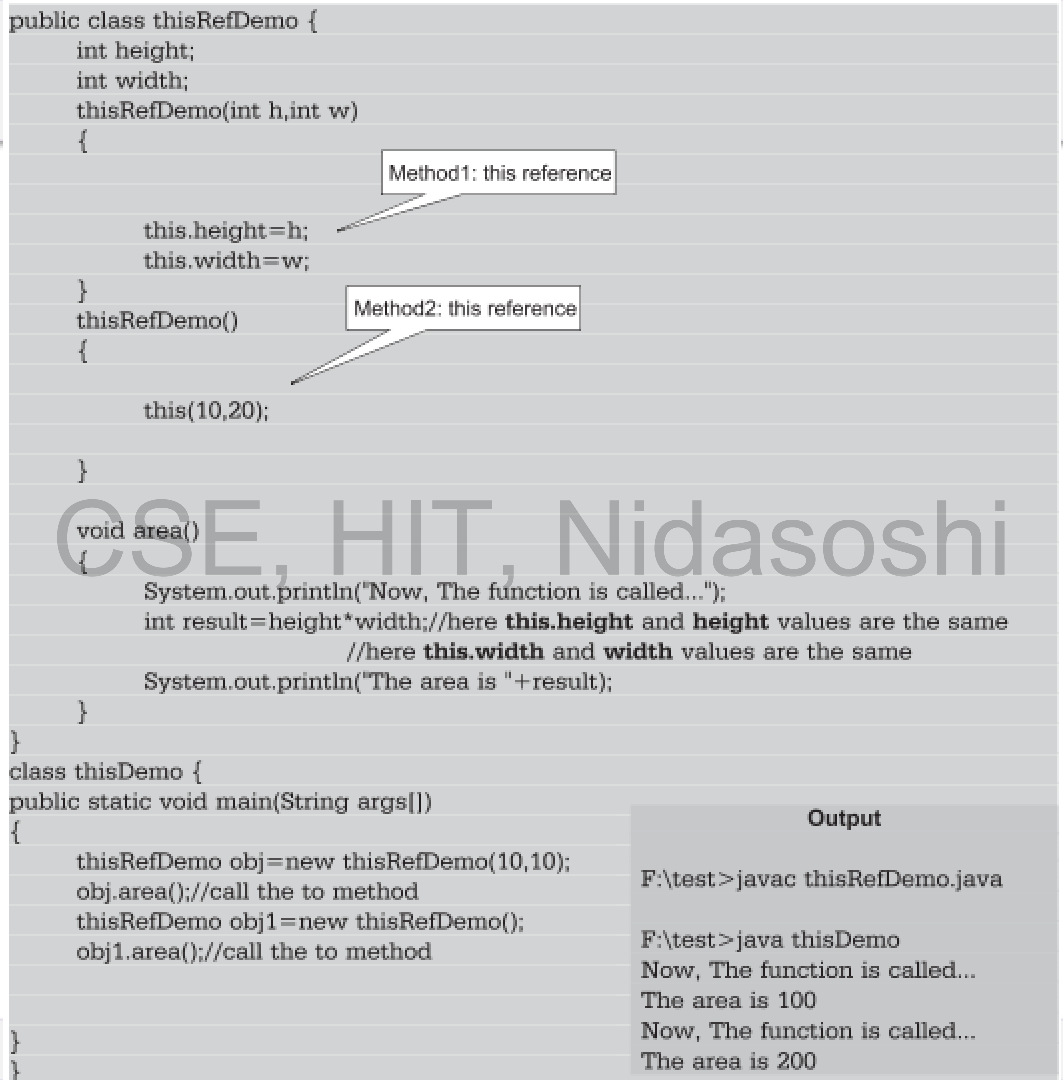
- In above program we have defined three constructors; there are two integer parameters that are passed to the first constructor.
- This constructor invokes the method area1. To the second constructor the two double values are passed as arguments.
- This constructor invokes the method area2.
- Then the third constructor is defined which has only one argument passed to it.
- This constructor invokes the method area3.
- Depending upon the parameters passed the appropriate constructor gets invoked.
- This mechanism is called constructor overloading.

this Keyword

- When a calling object wants to refer its own values then the this reference is used.
- The this is a keyword used for making the this reference.
- Using this reference we can refer to class's hidden data fields.
- For example

```
this.a = a;
```

- Means, assign the value of a to data field a of the calling object.


Method1: this reference
Method2: this reference

```
public class thisRefDemo {
        int height;
        int width;
        thisRefDemo(int h,int w)
        {


                this.height=h;
                this.width=w;
        }
        thisRefDemo()
        {

                this(10,20);

        }

        void area()
        {
                System.out.println("Now, The function is called...");
                int result=height*width;//here this.height and height values are the same
                                        //here this.width and width values are the same
                System.out.println("The area is "+result);
        }
}
class thisDemo {
public static void main(String args[])
{
        thisRefDemo obj=new thisRefDemo(10,10);
        obj.area();//call the to method
        thisRefDemo obj1=new thisRefDemo();
        obj1.area();//call the to method


}
}
```

Output

```
F:\test>javac thisRefDemo.java

F:\test>java thisDemo
Now, The function is called...
The area is 100
Now, The function is called...
The area is 200
```

this Keyword

- In above program, we have written a simple method area for computing the area of rectangle.
- For calculating the area of rectangle we need the height and width values. Each time an object is created to invoke the method area.
- While using the first object obj the parameterized constructor is used.
- The values passed as parameter are assigned to the data fields height and width using this reference.
- Similarly the simple constructor is used to create another object obj1.
- In this constructor the this reference is used to pass the values to height and width.

Inheritance

- **Inheritance is the process by which one object acquires the properties of another object.**
- Using inheritance, you can create a general class that defines traits common to a set of related items.
- This class can then be inherited by other, more specific classes, each adding those things that are unique to it.
- In the terminology of Java, a class that is inherited is called a **superclass .**
- The class that does the inheriting is called a **subclass .**

Inheritance

- Inheritance is a property in which data members and member functions of some class are used by some other class.

Advantages of Inheritance

One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses.

1. **Reusability:** The base class code can be used by derived class without any need to rewrite the code.
2. **Extensibility**: The base class logic can be extended in the derived classes.
3. **Data hiding:** Base class can decide to keep some data private so that it cannot be altered by the derived class.
4. **Overriding:** With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Inheritance Types

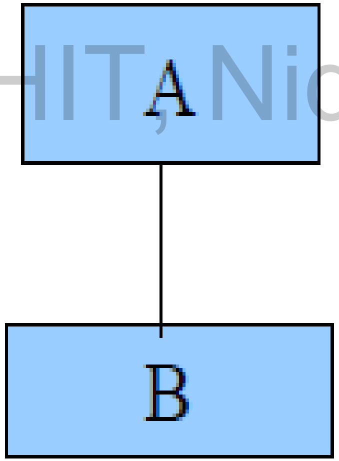
```
class Superclassname

{

        ……….

}

class subclassname extends Superclassname

{

        ………..

}
```

Inheritance Types

1. Single Inheritance.
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance

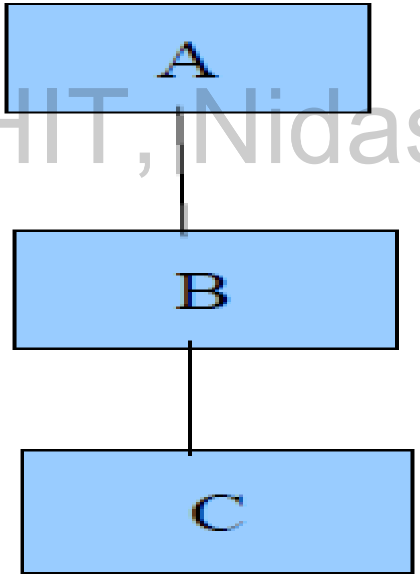
1. Single Inheritance

- In single inheritance there is one parent per derived class. This is the most common form of inheritance.



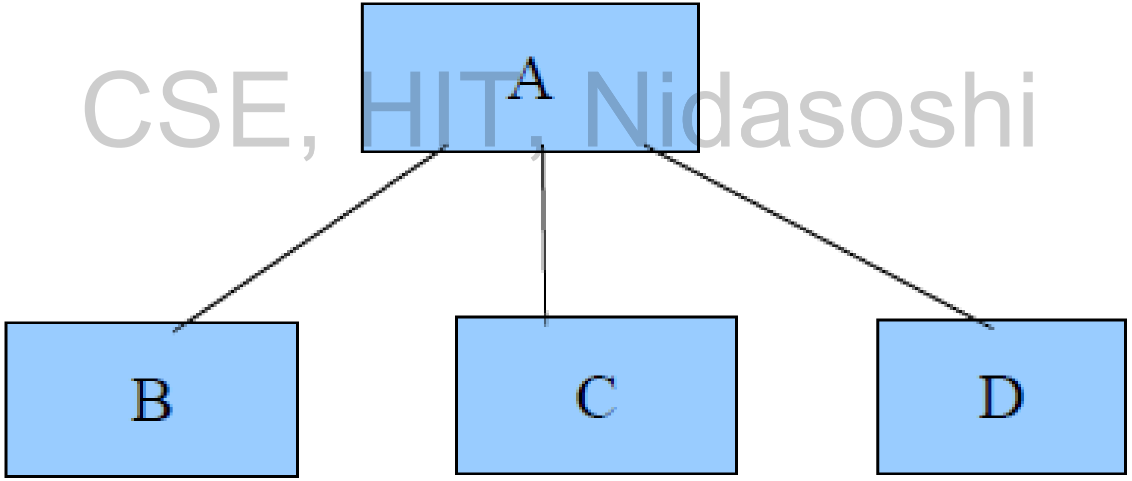
2. Multilevel Inheritance

- When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.



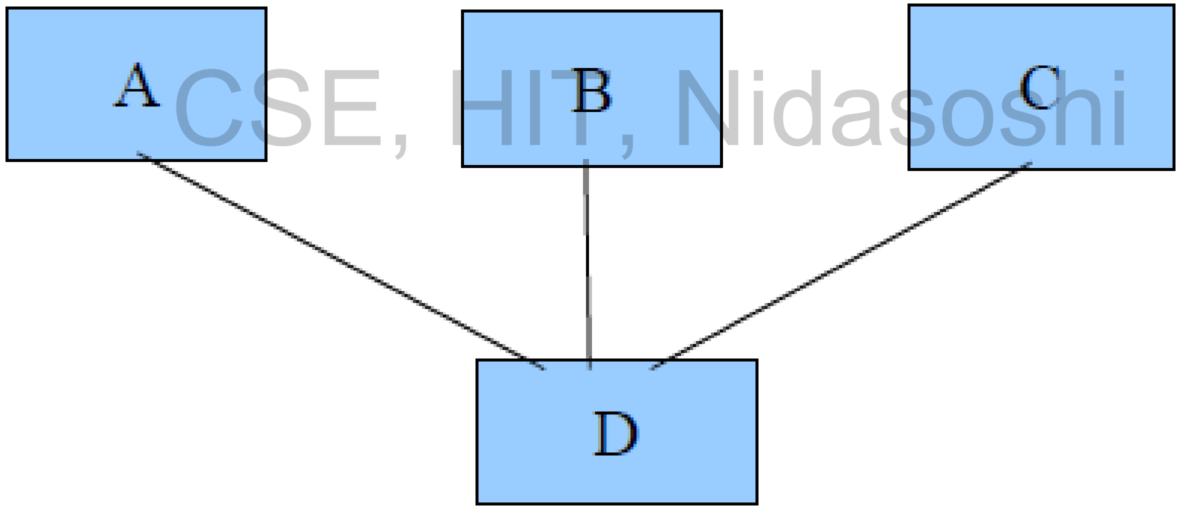
3. Hierarchical Inheritance

- The process of deriving more than one subclass from the single superclass is called as hierarchical inheritance.



4. Multiple Inheritance

- The process of deriving a single subclass from more than one super classes is called as Multiple inheritance.

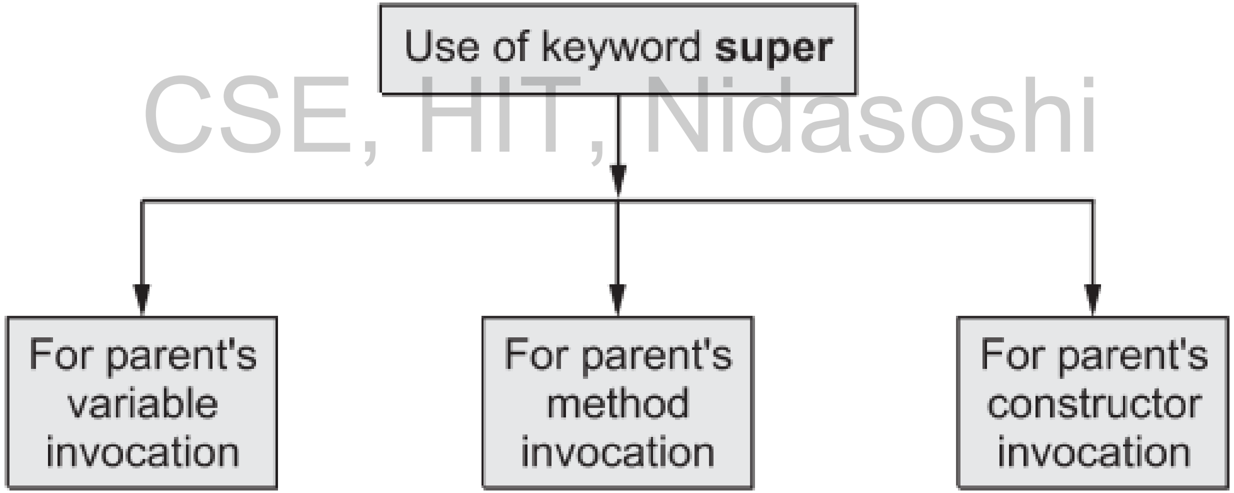


Use of Inheritance

- Inheritance means taking some properties from the parents. For instance: if your mother has blue eyes and your eyes are also blue then it is said that the color of your eyes is inherited.
- In Java inheritance means derived class borrows some properties of base class. At the same time the derived class may have some additional properties.
- The inheritance can be achieved by incorporating the definition of one class into the another using the keyword extends.
- In Object Oriented Programming, inheritance is referred as is-a relation.

The super()

- Super is a keyword used to access the immediate parent class from subclass. There are three ways by which the keyword super is used.


Use of keyword super
For parent's variable invocation
For parent's method invocation
For parent's constructor invocation

The super() is used to invoke the class variable of immediate parent class.

```
class A
{
        int x=10;
}
class B extends A
{
        int x=20;
        void display()
        {
                System. out.println(super.x);
        }
}
```

```
class superdemo
{
        public static void main(String args[])
        {
                B obj=new ();
                obj.display();
        }
}
```

Output:

```
10
```

The super() is used to invoke the class variable of immediate parent class.

- **Program Explanation:**
- In above program class A is a immediate parent class of class B.
- Both the class A and Class B has variables x.
- In class A, the value of x variable is 10 and in class B the value of variable x is 20.
- In display function if we would write System.out.println(x);
- The output will be 20 but if we user super.x then the variable x of class A will be referred.
- Hence the output is 10.

The super() is used to access the class method of immediate parent class.

```
class A
{
        void fun()
        {
                System.out.println("Method: Class
A");
        }
}
class B extends A
{
        void fun()
        {
        System.out.println("Method: Class B");
        }
```

```
        void display()
        {
                super.fun();
        }
}
class superdemo2
{
        public static void main(String args[])
        {
                B obj =new B();
                obj.display();
        }
}
```

Output:

```
Method: Class A
```

The super() is used to access the class method of immediate parent class.

- **Program Explanation:**
- In above program, the derived class can access the immediate parent's class method using super.fun().

- Hence is the output.
- You can change super.fun() to fun().
- Then note that in this case, the output will be invocation of subclass method fun.

The super() is used to invoke the immediate parent class constructor.

```
class A
{
		A()
		{
		System.out.println("Constructor of Class A");
		}
}
class B extends A
{
		B()
		{
		super();
		System.out.println("Constructor of Class B");
		}
}
```

```
class superdemo3
{
		public static void main(String args[])
		{
				B obj=new B();
		}
}
```

Output:

```
Constructor of Class A
Constructor of Class B
```

The super() is used to access the class variable of immediate parent class.

- **Program Explanation:**
- In above program, the constructor in class B makes a call to the constructor of immediate parent class by using the keyword super, hence the print statement in parent class constructor is executed and then the print statement for class B constructor is executed.

Method Overriding

- Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

Method Overriding

- The method of superclass which gets modified in subclass has the same name and type signature.
- The overridden method must be called from the subclass. Consider following Java Program, in which the method(named as fun) in which a is assigned with some value is modified in the derived class.
- When an overridden method is called from within a subclass, it will always refer to the version of that method re-defined by the subclass.
- The version of the method defined by the superclass will be hidden.

Method Overriding - Example

```
class A
{
        int a=0;
        void fun(int i)
        {
                this.a=i;
        }
}
class B extends A
{
        int b;
        void fun(int i)
        {
                int c;
                b=20;
                super.fun(i+5);
                System.out.println(“The    value of a:“+a);
                System.out.println(“The value of b:"+b);
                c=a*b;
                System.out.println("The value of c: "+c);
        }
}
class OverrideDemo
{
        public static void main(String args[])
        {
                B obj_B=new B();
                obj_B.fun(10);  //function re-defined in derived class
        }
}
```

Output:

```
The value of a:15      The value of b:20      The value of c: 300
```

Method Overriding - Example

Program Explanation

- In the above program, there are two classes - class A and class B.
- Class A acts as a superclass and class B acts as a subclass.
- In class A, a method fun is defined in which the variable a is assigned with some value.
- In the derived class B, we use the same function name fun in which, we make use of a super keyword to access the variable a, and then it is multiplied by b and the result of multiplication will be printed.

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
The method overloading occurs at compile time.	The method overriding occurs at the run time or execution time.
In the case of method overloading, a different number of parameters can be passed to the function.	In function overriding the number of parameters that are passed to the function is the same.
The overloaded functions may have different return types.	In method overriding all the methods will have the same return type.
Method overloading is performed within a class.	Method overriding is normally performed between two classes that have an inheritance relationship.

The Final Keyword

The final keyword can be applied at three places

- For declaring variables
- For declaring the methods
- For declaring the class

The Final Keyword

- A variable can be declared as final.
- If a particular variable is declared as final then it cannot be modified further.
- The final variable is always a constant.

For example - final int a = 10;

- The final keyword can also be applied to the method. When final keyword is applied to the method, the method overriding is avoided.
- That means the methods those are declared with the keyword final cannot be overridden.
- Consider the following Java program which makes use of the keyword final for declaring the method -

The Final Keyword for variable - Example

```
class Test
{
        final int a =10;
        void fun()
        {
                System.out.println("\n Hello, this
function declared using final");
        }
}
class Test1 extends Test
{
        int a = 20;
}
```

```
class finaldemo
{
        public static void main(String
args[])
        {
                Test t = new Test1();
                t.fun();
        }
}
```

Output:

```
1 Error
Cannot override final variable a in class
Test1
```

The Final Keyword for method - Example

```
class Test
{
	final void fun()
	{
		System.out.println("\n Hello, this 
function declared using final");
	}
}
class Test1 extends Test
{
	final void fun()
	{
		System.out.println("\n Hello, this 
another function");
	}
}
```

```
class finaldemo
{
	public static void main(String 
args[])
	{
		Test t = new Test1();
		t.fun();
	}
}
```

Output:

```
1 Error
fun() in Test1 cannot override fun() in 
Test; overridden method is final final void 
fun()
```

Final Classes to Stop Inheritance

- If we declare particular class as final, no class can be derived from it.
- Following Java program is an example of final classes.

Final Classes to Stop Inheritance - Example

```
final class Test
{
        void fun()
        {
                System.out.println("\n Hello, this
function in base class");
        }
}
class Test1 extends Test
{
        final void fun()
        {
                System.out.println("\n Hello, this
another function");
        }
}
```

```
class finalclassdemo
{
        public static void main(String
args[])
        {
                Test t = new Test1();
                t.fun();
        }
}
```

Output:

```
1 Error
 cannot inherit from final Test class Test1
extends Test
```

Exception Handling in Java

- Exception is an unusual situation in program that may lead to crash it.
- Usually it indicates the error.
- Let us first understand the concept. In Java, exception is handled using five keywords try, catch, throw, throws and finally.
- The Java code that you may think may produce exception is placed within the try block.
- Let us see one simple program in which the use of try and catch is done in order to handle the exception divide by zero.

Exception Handling in Java

```
class ExceptionDemo
{
        public static void main(String args[])
        try
        {
                int a, b;
                a=5;
                b=a/0;
        }
        catch(ArithmeticException e)
        {
                System.out.println("Divide by Zero");
        }
        System.out.println("...Executed catch statement...");
}
```

Exception Handling in Java

- Inside a try block as soon as the statement:

b = a/0

- gets executed then an arithmetic exception must be raised, this exception is caught by a catch block.
- Thus there must be a try-catch pair and catch block should be immediate follower of try statement.
- After execution of catch block the control must come on the next line.
- These are basically the exceptions thrown by java runtime systems.

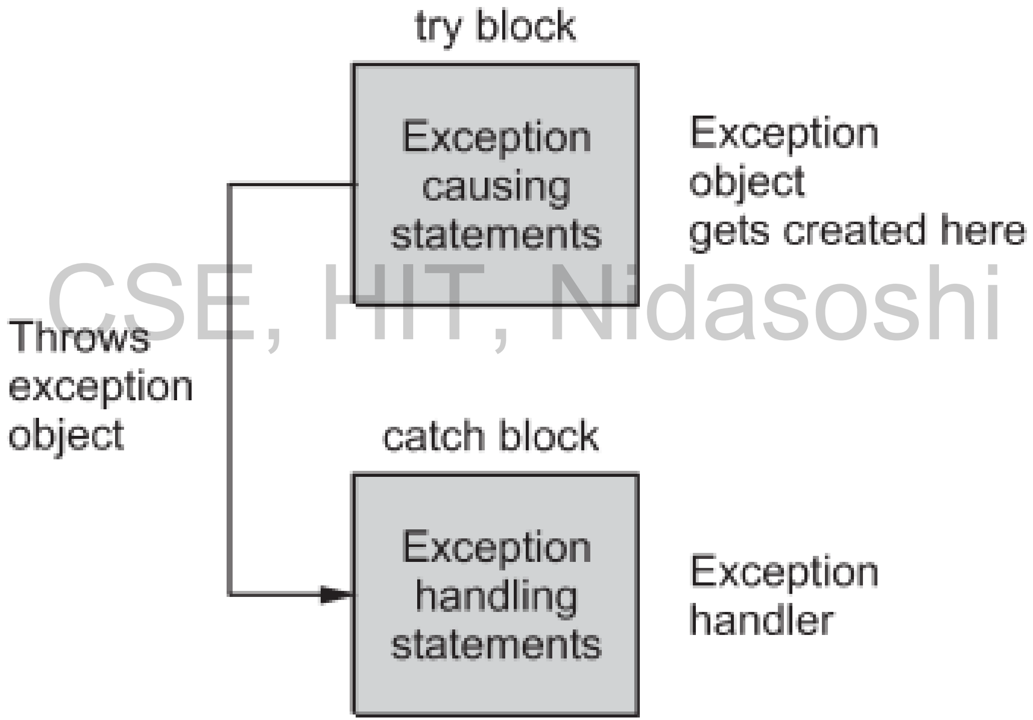
Exception Handling Syntax

- Various keywords used in handling the exception are -
- **try** - A block of source code that is to be monitored for the exception.
- **catch** - The catch block handles the specific type of exception along with the try block. Note that for each corresponding try block there exists the catch block.
- **finally** - It specifies the code that must be executed even though exception may or may not occur.
- **throw** - This keyword is used to throw specific exception from the program code.
- **throws** - It specifies the exceptions that can be thrown by a particular method.

Exception Handling Syntax - Try-catch Block

- The statements that are likely to cause an exception are enclosed within a **try** block. For these statements the exception is thrown.
- There is another block defined by the keyword **catch** which is responsible for handling the exception thrown by the try block.
- As soon as exception occurs it is handled by the **catch** block.
- The catch block is added immediately after the **try** block.
- Following is an example of try-catch block.

Exception Handling Syntax - Try-catch Block


try block
Exception causing statements
Exception object gets created here
Throws exception object
catch block
Exception handling statements
Exception handler

Exception Handling Syntax - Try-catch Block

```
try
{
      //exception gets generated here
}
catch(Type_of_Exception e)
{
      //exception is handled here
}
```

If any one statement in the try block generates an exception then the remaining statements are skipped and the control is then transferred to the catch statement.

Exception Handling - Try-catch Block - Example

```
class RunErrDemo
{
        public static void main(String[] args)
        {
                int a,b,c;
                a=10;
                b=0;
                try
                {
                        c = a/b;
                }
                catch(ArithmeticException e)
                {
                        System.out.println("\n Divide by zero");
                }
                System.out.println("\n The value of a: "+a);
                System.out.println("\n The value of b: "+b);
                }
        }
}
```

Output:

```
Divide by zero

The value of a: 10

The value of b: 0
```

Note that even if the exception occurs at some point, the program does not stop at that point.

Finally Block

- Sometimes because of execution of try block the execution gets break off. And due to this some important code (which comes after throwing off an exception) may not get executed. That means, sometimes try block may bring some unwanted things to happen.
- The finally block provides the assurance of execution of some important code that must be executed after the try block.
- Even though there is any exception in the try block the statements assured by finally block are sure to execute. These statements are sometimes called as clean up code.

Finally Block

The syntax of finally block is

```
finally
{
        //clean up code that has to be executed finally
}
```

The finally block always executes. The finally block is to free the resources.

This is a java program which shows the use of finally block for handling exception

```
class finallyDemo
{
        public static void main(String args[])
        {
                int a=10,b=-1;
                try
                {
                        b=a/0;
                }
                catch(ArithmeticException e)
                {
                        System.out.println("In catch block: "+e);

                finally
                {
                        if(b!=-1)
                                System.out.println("Finally block executes without occurrence of exception");
                        else
                                System.out.println("Finally block executes on occurrence of exception");
                }
        }
}
```

Output

```
In catch block: java.lang.ArithmeticException: / by zero
Finally block executes on occurrence of exception
```

Finally Block

Program Explanation

In above program, on occurrence of exception in try block the control goes to catch block, the exception of instance ArithmeticException gets caught.

This is divide by zero exception and therefore / by zero will be printed as output.

Following are the rules for using try, catch and finally block

1. There should be some preceding try block for catch or finally block. Only catch block or only finally block without preceding try block is not at all possible.
2. There can be zero or more catch blocks for each try block but there must be single finally block present at the end.

Throws

- When a method wants to throw an exception then keyword throws is used.

```
method name(parameter list) throws exception list
{
}
```

- Let us understand this exception handling mechanism with the help of simple Java program.

Throws - Example

```
class ExceptionThrows
{
        static void fun(int a,int b) throws ArithmeticException
        {
                int c;
                try
                {
                        c=a/b;
                }
                catch(ArithmeticException e)
                {
                        System.out.println("Caught exception: "+e);
                }
        }
        public static void main(String args[])
        {
                int a=5;
                fun(a,0);
        }
}
```

Output

```
Caught exception:
javalang.ArittuneticException: / by zero
```

Throws - Example

- In above program the method fun is fur handling the exception divide by zero.
- This is an arithmetic exception hence we write

  static void fun(int a, int b) throws ArithmeticException

- This method should be of static type.
- Also note as this method is responsible for handling the exception the **try-catch** block should be within fun.

Throw

- For explicitly throwing the exception, the keyword throw is used.
- The keyword throw is normally used within a method.
- We can not throw multiple exceptions using throw.

Throw - Example

```
class ExceptionThrow
{
        static void fun(int a,int b)
        {
                int c;
                if(b==0)
                        throw new ArithmeticException("Divide By Zero!!!");
                else
                        c=a/b;
        }
        public static void main(String args[])
        {
                int a=5;
                fun(a,0);
        }
}
```

Difference between throw and throws

Throw	Throws
For explicitly throwing the exception, the keyword throw is used.	For declaring the exception the keyword throws is used.
Throw is followed by instance.	Throws is followed by exception class.
Throw is used within the method.	Throws is used with method signature.
We cannot throw multiple exceptions.	It is possible to declare multiple exceptions using throws.

Example Program

- Define exception. Write a program that contains one method which will throw IllegalAccessException and use proper exception handlers so that exception should be printed. **July-17. Marks 6**

Example Program

```
class Test
{
        static void fun() throws IllegalAccessException
        {
                System.out.println("Inside the function');
                throw new IllegalAccessException(“testing”);
        }
        public static void main(String args[])
        {
                try
                {
                        fun();
                }
                catch(IllegalAccessException e)
                {
                        System.out.println(e);
                }
        }
}
```

Output:

```
Inside the function
java.lang.IllegalAccessException: testing
```

Multiple Catch

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The syntax for single try and multiple catch is -

Multiple Catch

```
try
{
        ...//exception occurs
}
catch(Exception_type e)
{
        ...//exception is handled here
}
catch(Exception_type e)
{
        ...//exception is handled here
}
catch(Exception_type e)
{
        ...//exception is handled here
}
```

Multiple Catch - Example

```
class MultipleCatchDemo
{
        public static void main (String args [])
        {
                int al] = new int 13];
                try
                {
                        for (int i = 1; i <=3; i++)
                        {
                                a[il = i *i;
                        }
                        for (int i = 0; i <3; i++)
                        {
                                a[i] = i/i;
                        }
                }
                catch (ArrayIndexOutOfBoundsException e)
                {
                System.out.printin ("Array index is out of bounds");
                }
                catch (ArithmeticException e)
                {
                System.out.println ("Divide by zero error");
                }
        }
}
```

Output:

```
Array index is out of bounds
```

Note:If we comment the first for loop in the try block and then execute the above code we will get following output-

```
Divide by zero error
```

Example Program

- Write a Java program for illustrating the exception handling when a number is divided by zero and an array has a negative index value.

  VTU : July-18, Marks 6

Example Program

```
class test
{
        public static void main (String args[])
        {
                int a[] = new int [3];
                try
                {
                        for (int i = 1; i <=3; i++)
                        {
                                int j=-1;
                                a[j] = i *i;
                        }
                        for (int i = 0; i <3; i++)
                        {
                                a[i] = i/i;
                        }
                }
                catch (ArrayIndexOutOfBoundsException e)
                {
                        System.out.println ("Array index is
Negative");
                }
                catch (ArithmeticException e)
                {
                        System.out.println ("Divide by zero
error");
                }
        }
}
```

Output:

Array index is Negative Note that if we comment first for loop in the try block then we get following output Divide by zero error

Benefits of Exception Handling

Following are the benefits of exception handling –

1. Using exception the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. The working code and the error handling code can be separated out due to exception handling mechanism. Using exception handling, various types of errors in the source code can be grouped together.
4. Due to exception handling mechanism, the errors can be propagated up the method call stack i.e. problems occurring at the lower level can be handled by the higher up methods.

Benefits of Exception Handling

Following are the benefits of exception handling –

1. Using exception the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. The working code and the error handling code can be separated out due to exception handling mechanism. Using exception handling, various types of errors in the source code can be grouped together.

Garbage Collection

Garbage collection is a method of automatic memory management.

It works as follows -

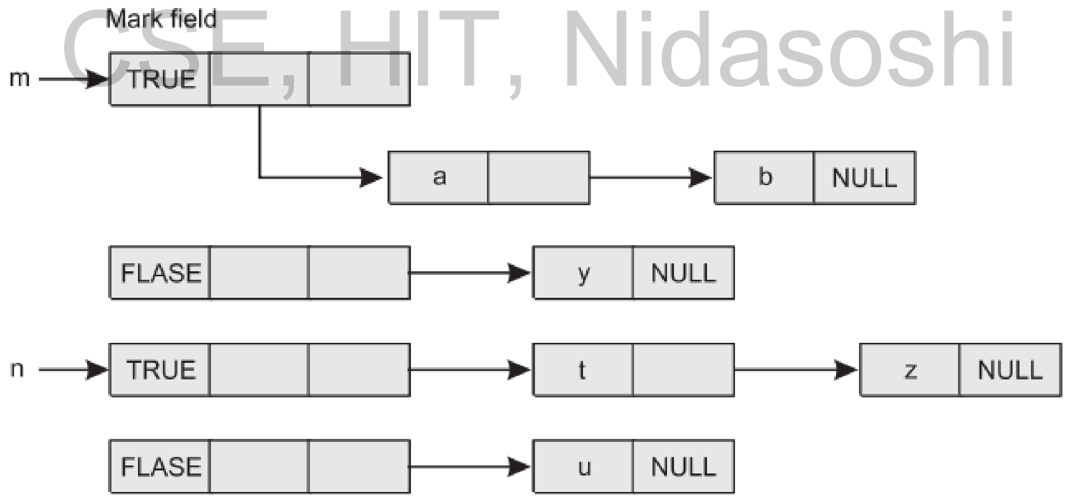
1. When an application needs some free space to allocate the nodes and if there is no free space available to allocate the memory for these objects then a system routine called garbage collector is called.
2. This routine then searches the system for the nodes that are no longer accessible from an external pointer. These nodes are then made available for reuse by adding them to available pool. The system can then make use of these free available space for allocating the nodes.

Garbage Collection

- Garbage collection is usually done in two phases - marking phase and collection phase. In marking phase, the garbage collector scans the entire system and marks all the nodes that can be accessible using external pointer.
- During collection phase, the memory is scanned sequentially and the unmarked nodes are made free.

Garbage Collection

- **Marking phase:**
- For marking each node, there is one field called mark field. Each node that is accessible using external pointer has the value TRUE in marking field.
- For example



Garbage Collection

- **Collection phase**
- During collection phase, all the nodes that are marked FALSE are collected and made free. This is called sweeping. There is another term used in regard to garbage collection called Thrashing.

Garbage Collection

- Consider a scenario that, the garbage collector is called for getting some free space and almost all the nodes are accessible by external pointers.
- Now garbage collection routine executes and returns a small amount of space. Then again after some time system demands for some free space. Once again garbage collector gets invokes which returns very small amount of free space. This happens repeatedly and garbage collection routine is executing almost all the time. This process is called thrashing. Thrashing must be avoided for better system performance.

Garbage Collection

Advantages of garbage collection

1. The manual memory management done by the programmer (i.e. use of malloc and free) is time-consuming and error prone. Hence automatic memory management is done.
2. Reusability of memory can be achieved with the help of garbage collection.

Garbage Collection

Disadvantages of garbage collection

1. The execution of the program is paused or stopped during the process of garbage collection.

2. Sometimes situations like thrashing may occur due to garbage collection.

The finalize() Method

- Java has a facility of automatic garbage collection.
- Hence even though we allocate the memory and then forget to deallocate it then the objects that are no longer is used get freed.
- Inside the finalize() method you will specify those actions that must be performed before an object is destroyed.
- The garbage collector runs periodically checking for objects that are no longer referenced by any running state or indirectly though other referenced objects.

The finalize() Method

- Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held.
- To handle such situation **finalize()** method is used.
- **finalize()** method is called by garbage collection thread before collecting object.
- It’s the last chance for any object to perform cleanup utility.

The finalize() Method

- To add finalizer to a class simply define the finalize method.
- The syntax to finalize() the code is –

```
protected void finalize()
{
    finalization code
}
```

- Note that finalize() method is called just before the garbage collection. It is not called when an object goes out-of-scope

The finalize() Method

- ***Can the Garbage Collection be forced explicitly?***
- No, the Garbage Collection cannot be forced explicitly.
- We may request JVM for **garbage collection** by calling **System.gc()** method.
- But this does not guarantee that JVM will perform the garbage collection.

The finalize() Method

- **gc()** method is used to call garbage collector explicitly.
- However **gc()** method does not guarantee that JVM will perform the garbage collection.

- It only requests the JVM for garbage collection.
- This method is present in **System** and **Runtime** class.

The finalize() Method

```
public class Test

{

     public static void main(String[] args)

    {

        Test t = new Test();

        t=null;

        System.gc();

    }
```

```
    public void finalize()
    {
        System.out.println("Garbage
        Collected");
    }
}
```

Output :

```
Garbage Collected
```

Why does Java not support destructors and how does the finalize method help in garbage collection

- The destructor is used to free or deallocate the memory of unused variables.
- This is called cleaning up.
- Java has in built mechanism of cleaning up.
- This mechanism is called Garbage collection.
- The garbage collector automatically deallocates the memory of unused variables.
- Hence there is no need of destructor in Java.

Why does Java not support destructors and how does the finalize method help in garbage collection

- Finalization is the facility provided by the Java for the classes for cleaning up the native resources before the objects are garbage collected.
- The garbage collector is unable to control the cleaning up of native resources which are used earlier.
- Then the responsibility of cleaning up those native allocations falls on the object's finalization code.
- Thus the purpose of finalization is to clean up the native resources used earlier. The finalize() method must be run before invoking the garbage collector.