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 4: Packages, Interfaces and Multithreading

Dr. Mahesh G Huddar

Dept. of Computer Science and Engineering

Packages - Introduction

- 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
    - user-defined package
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Packages - Introduction

- Using a package in Java Program is very simple. Just include the command package at the beginning of the program.
- The syntax for declaring the package is
- **package name_of_package**
- This package statement defines the name space in which the classes are stored. If we omit the package then the default classes are put in the package that has no name. Basically Java creates a directory and the name of this directory becomes the package name.

Packages - Introduction

- For example - In your program, if you declare the package as -package My_Package; then we must create the directory name My_Package in the current working directory and the required classes must be stored in that directory.

- Note that the name of the package and the name of the directory must be the same. This name is case sensitive.

Packages - Introduction

- We can create hierarchy of packages.
- For instance if you save the required class files in the subfolder MyPkg3 and the path for this subfolder is C: \ MyPkg1 \ MyPkg2 \ MyPkg3 then the declaration for the package in your java program will be -
- package MyPkg1.MyPkg2.MyPkg3;

Packages - Advantages

- 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.

Creating and Accessing Packages

- In this section we will discuss how to develop a program which makes use the classes from other package.
- **Step 1 :** Create a folder named My_Package.
- **Step 2 :** Create one class which contains two methods. We will store this class in a file named A.java. This file will be stored in a folder My_Package. The code for this class will be as follows - **Java Program[A.java]**
- package My_Package;
- //include this package at the beginning public class A
- methods defined must be public

Creating and Accessing Packages

```
//Java Program[A.java]
package My_Package; //include this package at the beginning
public class A
{
    int a;
    public void set_val(int n)
    {
        a = n;
    }
    public void display()
    {
        System.out.println("The value of a is: "+a);
    }
}
```

Creating and Accessing Packages

- Note that this class contains two methods namely - set_val and display. By the set_val method we can assign some value to a variable. The display function displays this stored value.
- **Step 3 :** Now we will write another java program named PackageDemo.java. This program will use the methods defined in class A. This source file is also stored in the subdirectory My_Package. The java code for this file is-

Creating and Accessing Packages

- **Step 4 :** Now, open the command prompt and issue the following commands in order to run the package programs
- D:\> set CLASSPATH=.;D:\;
- D:\ >cd My_Package
- Setting the class path
- D:\My_Package>javac A.java
- Creating the classes A.class anf package Demo.Class files
- D:\Mypackage>javac PackageDemo.java
- D:\Mypackage>java PackageDemo
- The value of a is: 10
- D: \Mypackage >

Creating and Accessing Packages

Java Program[PackageDemo.java]

```
import My_Package.A;
//The java class A is referenced here by import statement
class PackageDemo
{
        public static void main(String args[]) throws NoClassDefFoundError
        {
                A obj=new A();          //creating an object of class
                obj.set_val(10);     //Using the object of class A, the methods present
                obj.display();          //in class A are accessed
        }
}
```

Concept of CLASSPATH

- The packages are nothing but the directories.
- For locating the specified package the Java run time system makes use of current working directory as its starting point.
- Thus if the required packages is in the current working directory then it will found.

Concept of CLASSPATH

- Otherwise you can specify the directory path setting the CLASSPATH statement.
- For instance - if the package name My_Package is present at prompt D: \ > then we can specify
- set CLASSPATH=.;
- D:\; D:\ >cd My_Package
- D:\My_Package\> Now you can execute the required class files from this location

Access Protection

- If public or private specifier is not used then by default the classes, methods, data fields are assessable by any class in the same package.
- This is called package-private or package-access.
- A package is essentially grouping of classes.

Access Protection

```
package Test;
public class class1
{
        public int a;
        int b;
        private int c;
        public void funl() { }
        void fun2() { }
        private void fun3() { }
}
```

```
public class class2
{
        void My_method()
        {
                classl obj=new classl();
                obj.a;    //allowed
                obj.b;    //allowed
                obj.c;    //error: cannot access
                obj.funl();   //allowed
                obj.fun2();  //allowed
                obj.fun3();   //error cannot access
        }
}
```

Access Protection

```
package Test
public class class3
{
        void My_method()
        {
                class1 obj=new class1();
                obj.a;   //allowed
                obj.b;    // error:cannot access
                obj.c;  //error:cannot access
                obj.funl();  //allowed
                obj.fun2(); //erroccannot access
                obj.fun3();  //erroncannot access
        }
}
```

Access Protection

In above example,

- We have created two packages are created namely - Test and another Test.
- Inside the package Test there are two classes defined - classi and class2
- Inside the package another Test there is only one class defined and i.e. class3.
- There are three data fields - a, b and c. The data field a is declared as public, b is defined as default and C is defined as private.

Access Protection

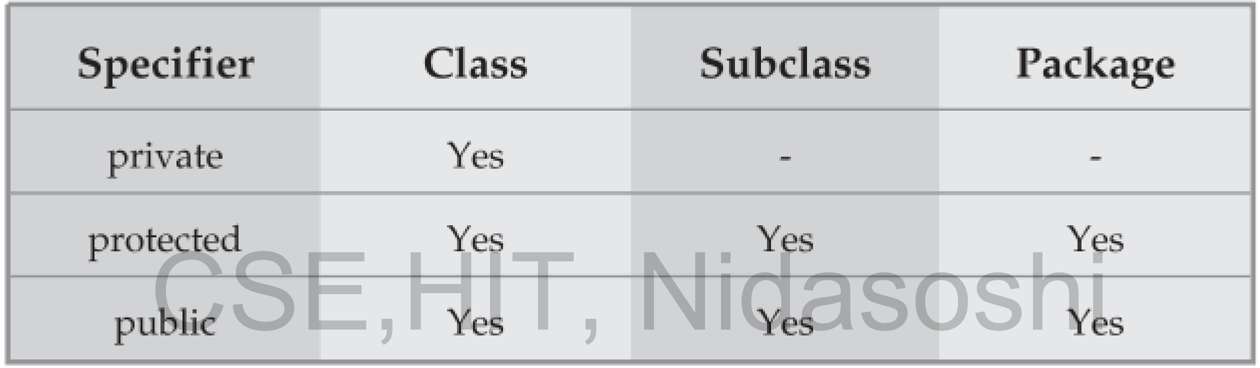
- The variable a and method fun10 both are accessible from the classes class2 and class3 (even if it is in another package). This is because they are declared as public.
- The variable b and method fun20 both are accessible from class2 because class2 lies in the same package. But they are not accessible from class3 because class3 is defined in another package.
- The variable c and method fun3() both are not accessible from any of the class, because they are declared as private.

Access Protection

- Protected mode is another access specifier which is used in inheritance. The protected mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.

Access Protection

- The effect of access specifiers for class, subclass or package is enlisted below,



Specifier	Class	Subclass	Package
private	Yes	-	-
protected	Yes	Yes	Yes
public	Yes	Yes	Yes

- For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

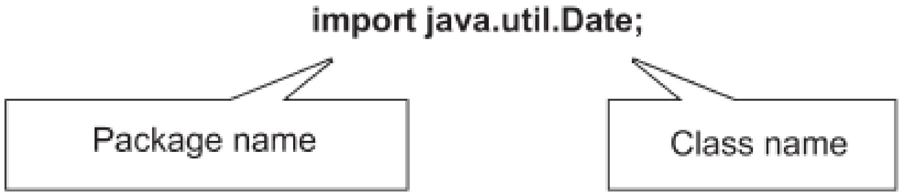
Importing Packages

- All the standard classes in Java are stored in named packages.
- There is no standard class present in Java which is unnamed. But it is always complicated to write the class name using a long sequence of packages containing dot operator. Hence the import statement is needed.
- The import statement can be written at the beginning of the Java program, using the keyword import.
- There are two ways of accessing the classes stored in the core package.

Importing Packages

Method 1:

- We import the java package class using the keyword import.
- Suppose we want to use the Data class stored in the java.util package then we can write the import statement at the beginning of the program. It is as follows -



Importing Packages

2. Method:

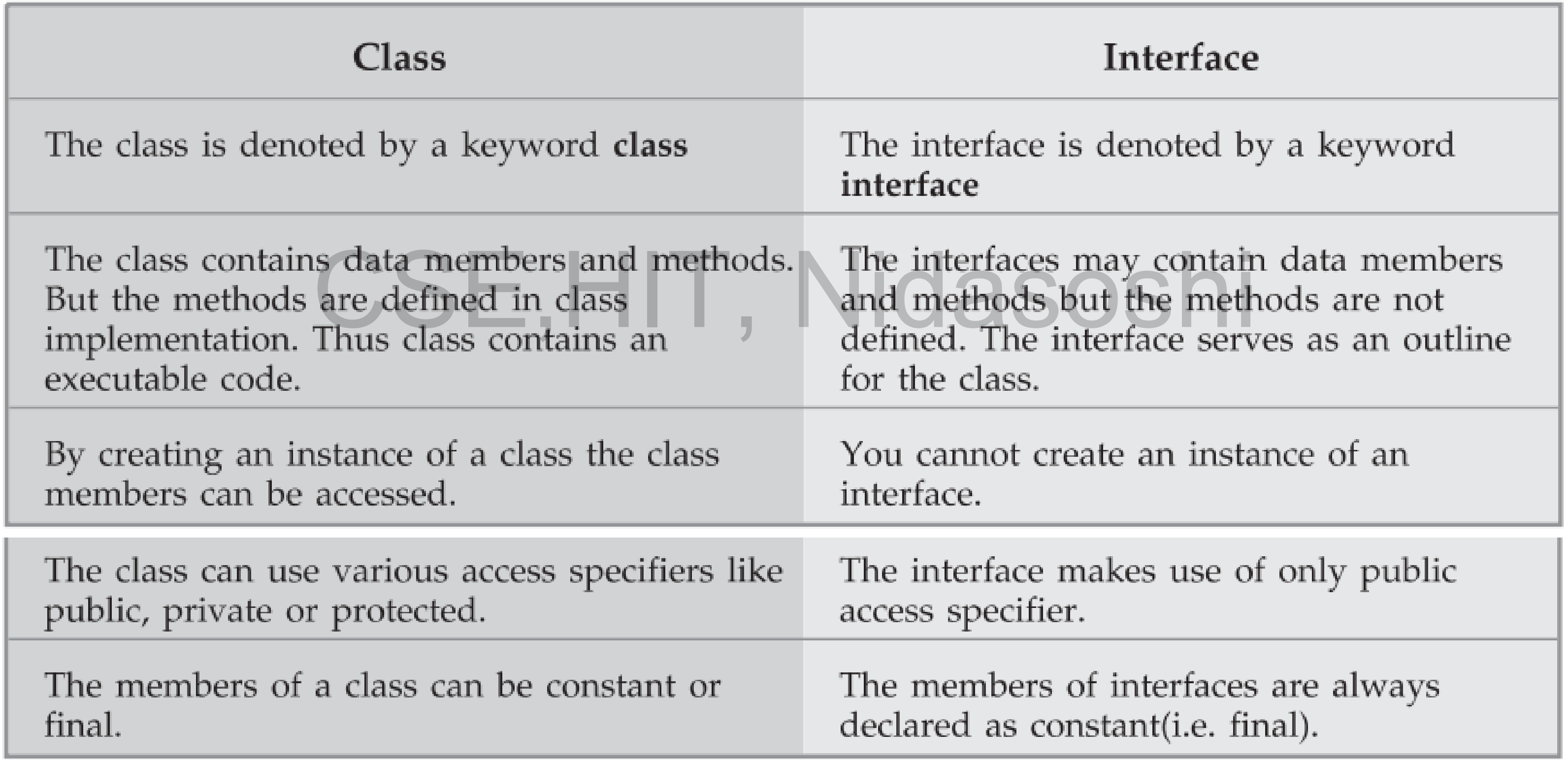
- There are some situations in which we want to make use of several classes stored in a package.
- Then we can write it as

import java.util.*

- Here * means any class in the corresponding package.

Interfaces

An interface is similar to a class but there lies some difference between the two.



Class	Interface
The class is denoted by a keyword **class**	The interface is denoted by a keyword **interface**
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.
By creating an instance of a class the class members can be accessed.	You cannot create an instance of an interface.
The class can use various access specifiers like public, private or protected.	The interface makes use of only public access specifier.
The members of a class can be constant or final.	The members of interfaces are always declared as constant(i.e. final).

Interfaces – Syntax

The interface can be defined using following syntax

```
access_modifier interface name_of_interface
{
    return_type method_name1(parameter1, parameter2, ... parametern);
    return_type method_name2(parameter1, parameter2, ... parametern);
    type static final variable_name=value;
}
```

Extending Interfaces

- Interfaces can be extended similar to the classes.
- That means we can derive subclasses from the main class using the keyword extend , similarly we can derive the sub-interfaces from main interfaces by using the keyword extends.
- The syntax is

```
interface Interface_name2 extends interface_name1
{
        //Body of interface
}
```

Extending Interfaces

For example

```
interface A
{
        int val=10;
}
interface B extends A
{
        void print_val()
        {
                System.out.println(“The value val is “+val);
        }
}
```

That means in interface B the display method can access the value of variable val. Similarly more than one interfaces can be extended.

Implementation of Interface

- It is necessary to create a class for every interface.
- The class must be defined in the following form while using the interface

```
class Class_name extends superclass_name
implements interface_name1,interface_name2,...
{
    //body of class
}
```

Implementation of Interface

- Let us learn how to use interface for a class
- Step 1 : Write following code and save it as my_interface.java

```
public interface my_interface
{
        void my_method(int val);
}
```

-
- Do not compile this program. Simply save it.

Implementation of Interface

Step 1 : Write following code and save it as **my_interface.java**

```
public interface my_interface
{
        void my_method(int val);
}
```

Do not compile this program. Simply save it.

Implementation of Interface

Step 2 : Write following code in another file and save it using **InterfaceDemo.java**

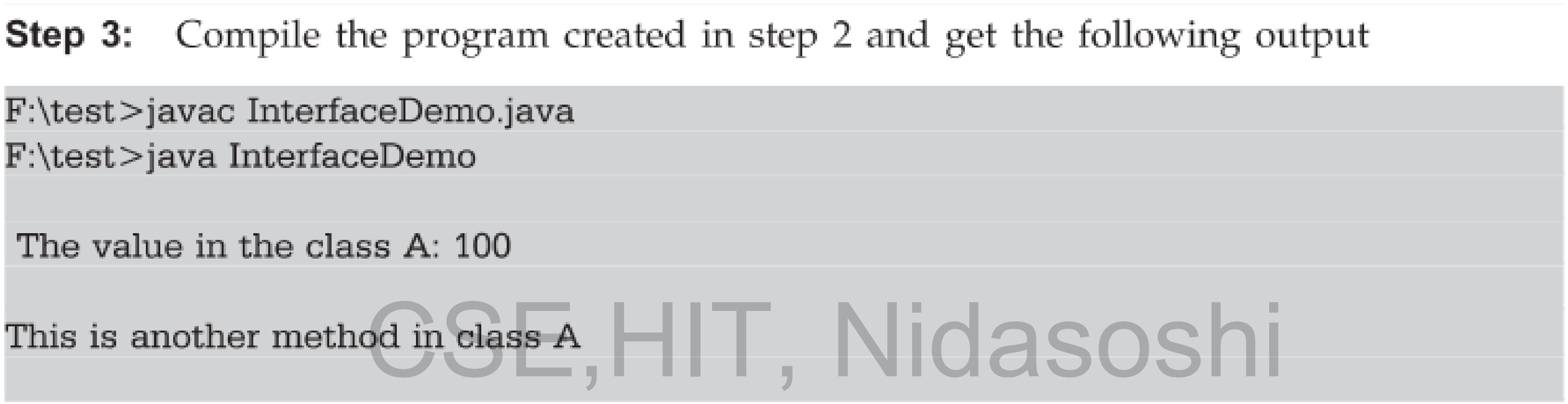
Java Program[InterfaceDemo.java]

```
class A implements my_interface
{
 public void my_method(int i)
 {
       System.out.println("\n The value in the class A: "+i);
 }
 public void another_method() //Defining another method not declared in interface
 {
       System.out.println("\nThis is another method in class A");
 }
}
class InterfaceDemo
{
 public static void main(String args[])
 {
       my_interface obj=new A();
       //or A obj=new A() is also allowed
          A obj1=new A();
       obj.my_method(100);
       obj1.another_method();
 }
}
```

Defining the method declared in the interface

Object of class A is of type my_interface.Using this object method can be accessed

Implementation of Interface



Step 3: Compile the program created in step 2 and get the following output

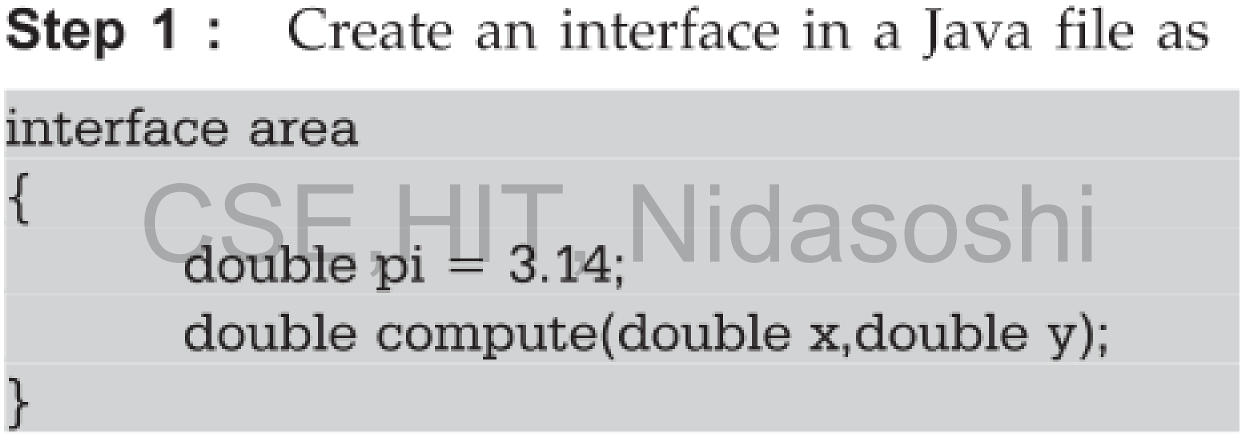
```
F:\test>javac InterfaceDemo.java
F:\test>java InterfaceDemo

 The value in the class A: 100

This is another method in class A

```

- Program Explanation
- In above program, the interface my_interface declares only one method i.e. my_method. This method can be defined by class A. There is another method which is defined in class A and that is another method. Note that this method is not declared in the interface my_interface. That means, a class can define any additional method which is not declared in the interface.

Implementation of Interface

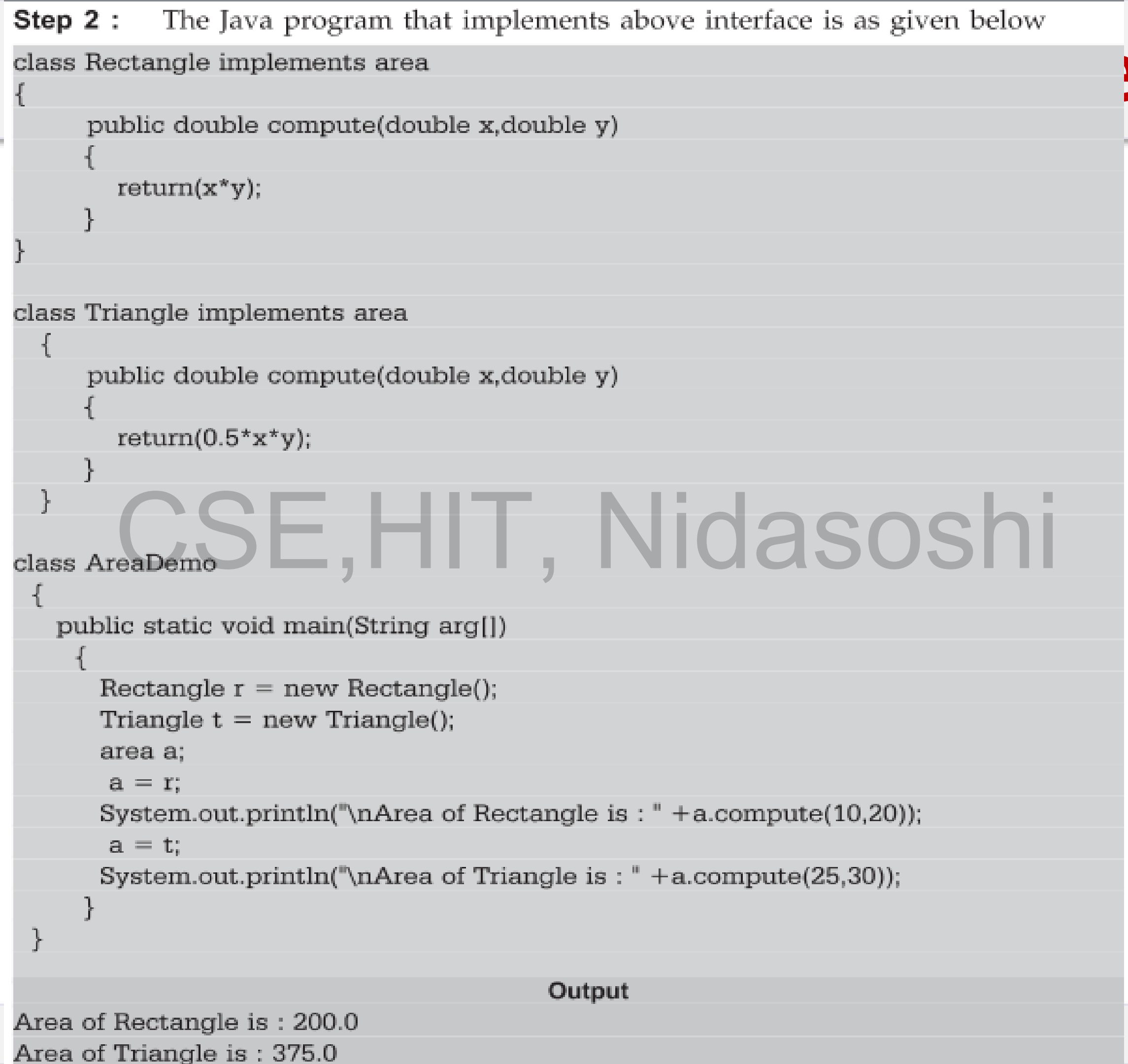
- Write a Java program to define an interface called Area which contains a method called Compute() and calculate areas of rectangle(l*b) and triangle(1/2*b*h) using classes Rectangle and Triangle. **VTU : July-18, Marks 8**

Implementation of Interface



Step 1 : Create an interface in a Java file as

```
interface area
{
        double pi = 3.14;
        double compute(double x,double y);
}
```



Step 2 : The Java program that implements above interface is as given below

```
class Rectangle implements area
{
        public double compute(double x,double y)
        {
           return(x*y);
        }
}

class Triangle implements area
   {
        public double compute(double x,double y)
        {
           return(0.5*x*y);
        }
   }

class AreaDemo
  {
     public static void main(String arg[])
       {
         Rectangle r = new Rectangle();
         Triangle t = new Triangle();
         area a;
          a = r;
         System.out.println("\nArea of Rectangle is : " +a.compute(10,20));
          a = t;
         System.out.println("\nArea of Triangle is : " +a.compute(25,30));
        }
  }
```

Output

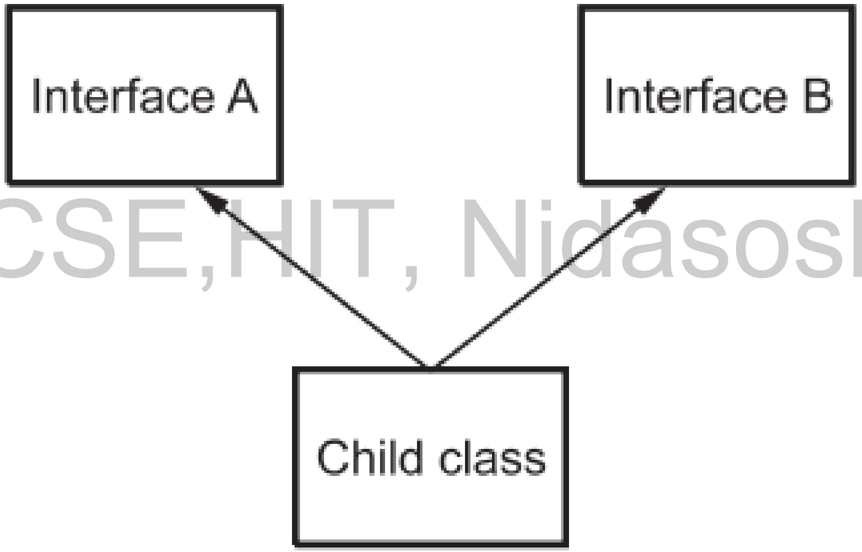
```
Area of Rectangle is : 200.0
Area of Triangle is : 375.0
```

Multiple inheritance using Interface

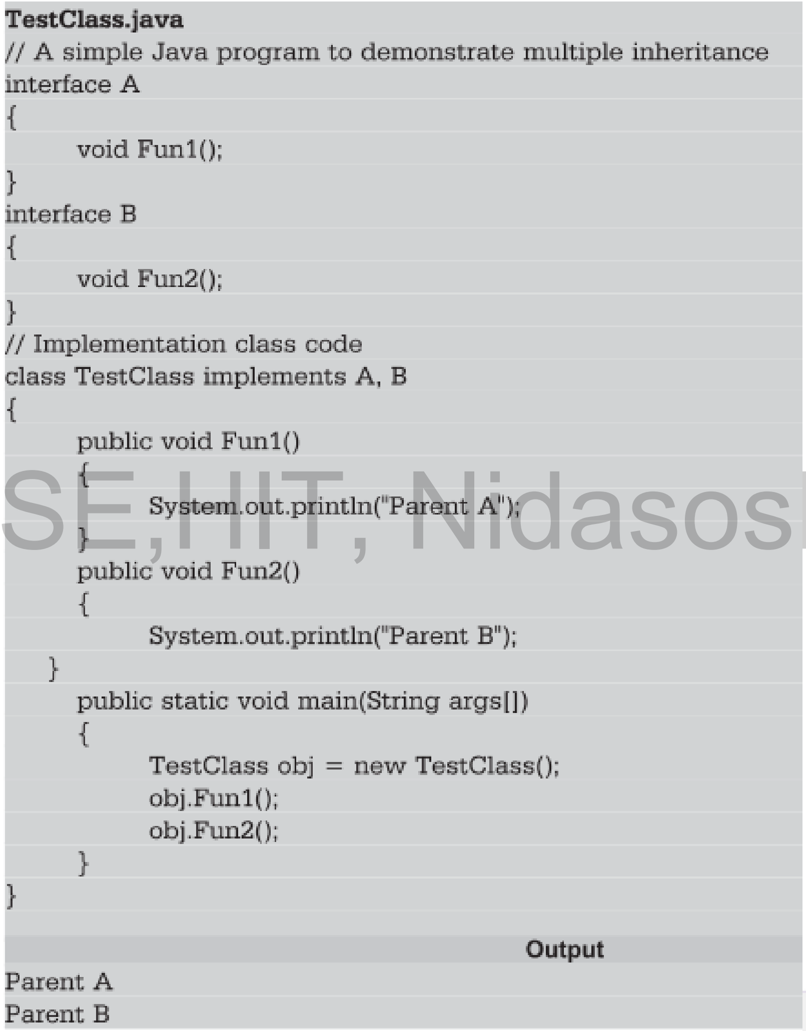
- Briefly explain the role of interfaces while implementing multiple inheritances in Java. **July 17, 6 Marks**
- Java does not support the concept of multiple inheritance using typical class hierarchy. However, it is possible to implement multiple inheritance using interface.

Multiple inheritance using Interface

- The multiple inheritance can be represented as follows:



- The simple Java code to implement multiple inheritance is as follows



TestClass.java

```
// A simple Java program to demonstrate multiple inheritance
interface A
{
        void Fun1();
}
interface B
{
        void Fun2();
}
// Implementation class code
class TestClass implements A, B
{
        public void Fun1()
        {
                System.out.println("Parent A");
        }
        public void Fun2()
        {
                System.out.println("Parent B");
    }
        public static void main(String args[])
        {
                TestClass obj = new TestClass();
                obj.Fun1();
                obj.Fun2();
        }
}
```

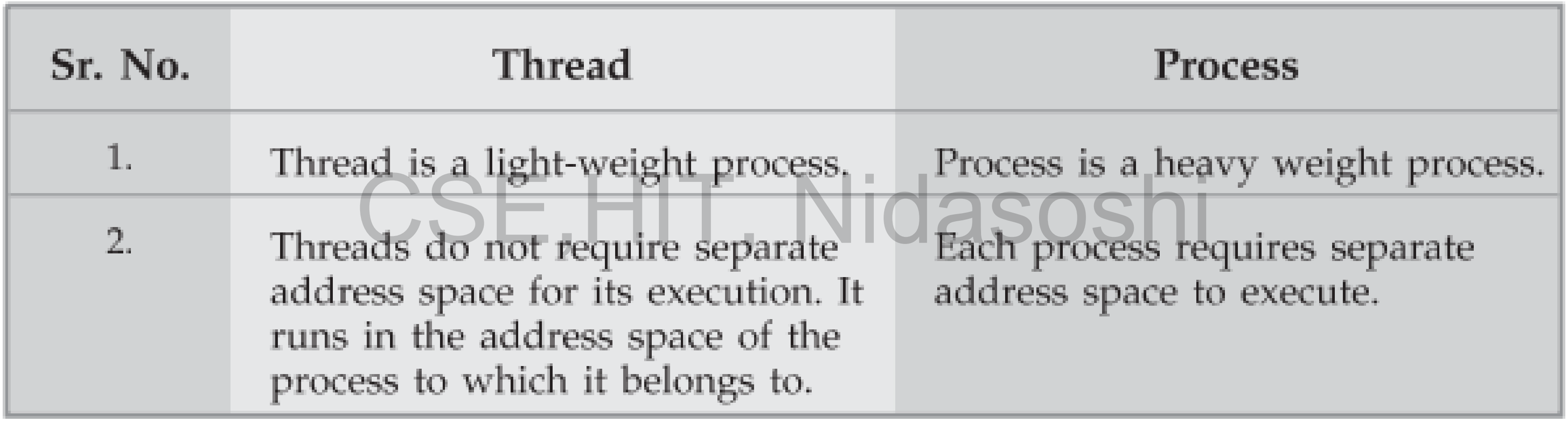
Output

```
Parent A
Parent B
```

What are Threads…?

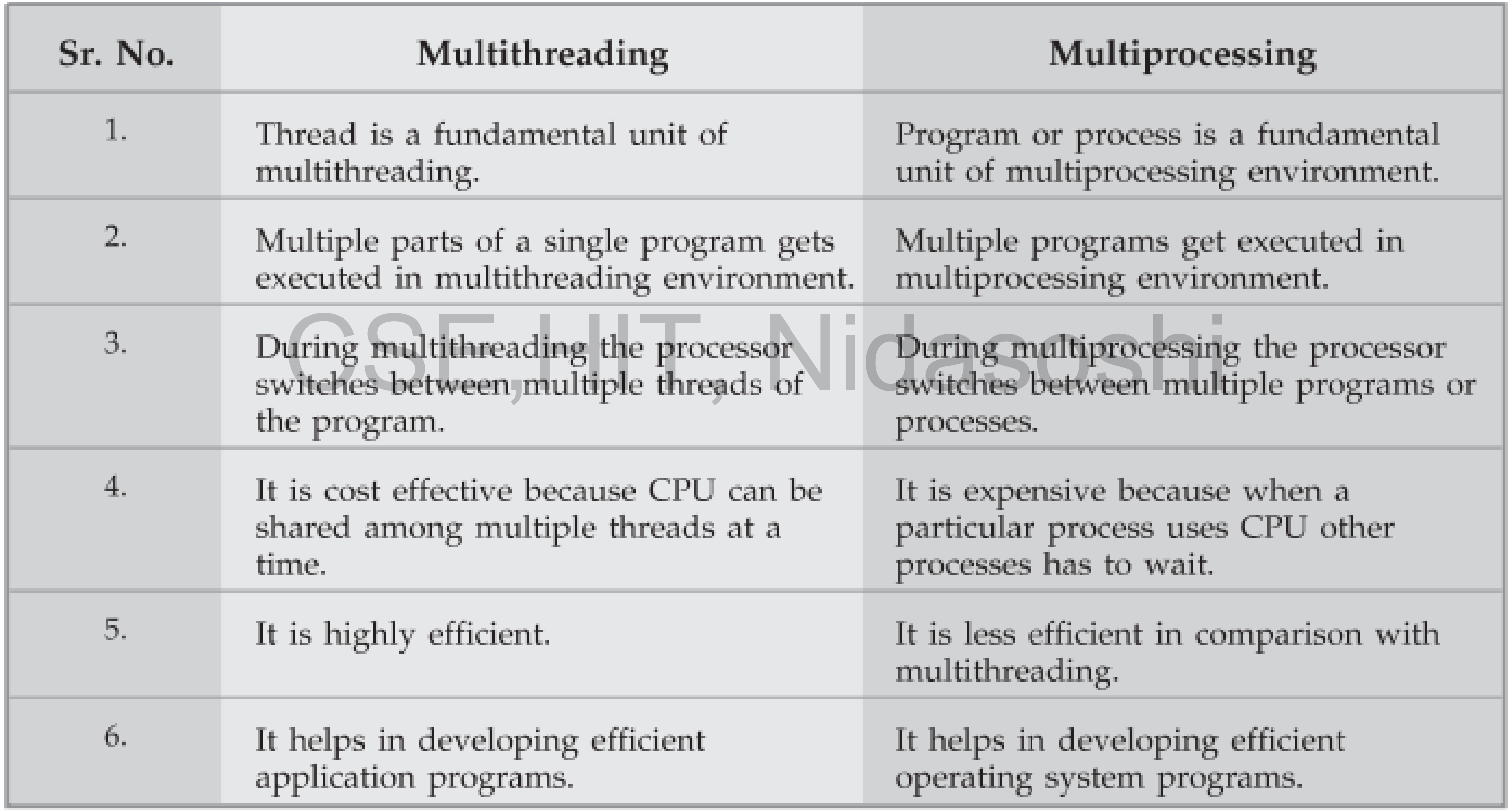
- One of the exciting features of Windows operating system is that - It allows the user to handle multiple tasks together. This facility in Windows operating system is called multitasking.
- In Java we can write the programs that perform multitasking using the multithreading concept. Thus Java allows to have multiple control flows in a single program by means of multithreading.
- Definition of thread : Thread is a tiny program running continuously. It is sometimes called as light-weight process. But there lies differences between thread and process.

Thread vs. Process



Sr. No.	Thread	Process
1.	Thread is a light-weight process.	Process is a heavy weight process.
2.	Threads do not require separate address space for its execution. It runs in the address space of the process to which it belongs to.	Each process requires separate address space to execute.

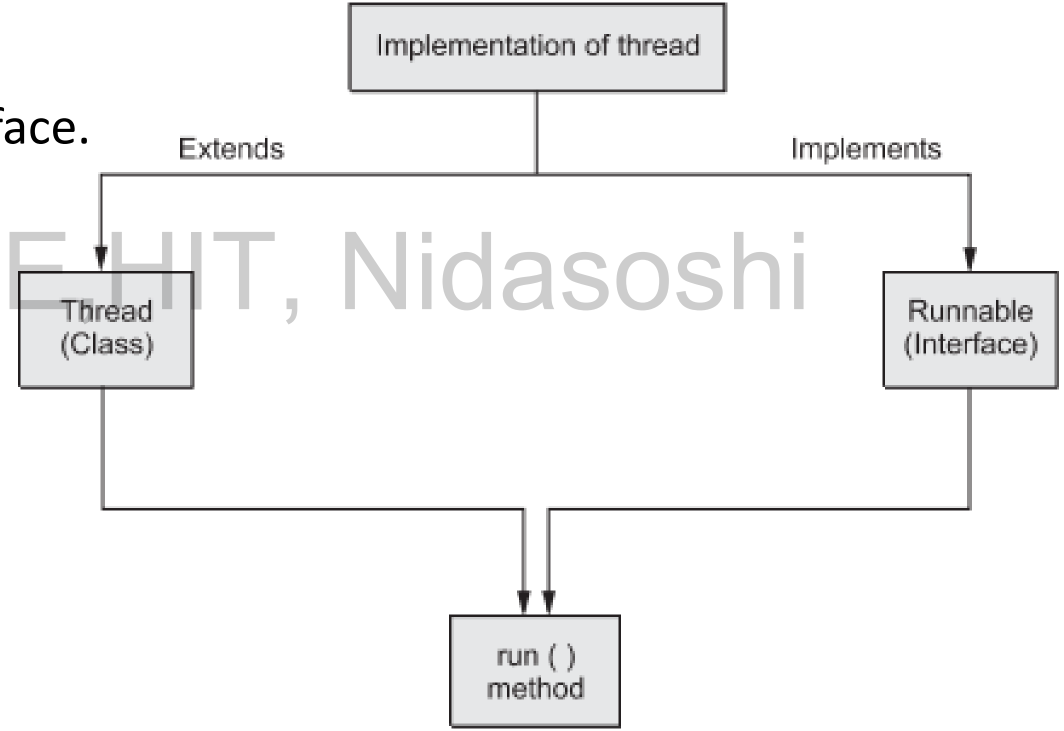
Difference between Multiple Processes and Multiple Threads



Sr. No.	Multithreading	Multiprocessing
1.	Thread is a fundamental unit of multithreading.	Program or process is a fundamental unit of multiprocessing environment.
2.	Multiple parts of a single program gets executed in multithreading environment.	Multiple programs get executed in multiprocessing environment.
3.	During multithreading the processor switches between multiple threads of the program.	During multiprocessing the processor switches between multiple programs or processes.
4.	It is cost effective because CPU can be shared among multiple threads at a time.	It is expensive because when a particular process uses CPU other processes has to wait.
5.	It is highly efficient.	It is less efficient in comparison with multithreading.
6.	It helps in developing efficient application programs.	It helps in developing efficient operating system programs.

How to make classes threadable

- In Java we can implement the thread programs using two approaches –
  - Using Thread class
  - Using runnable interface.



How to make classes y threadable

As given in Fig., there are two methods by which we can write the Java thread programs one is by extending thread class and the other is by implementing the Runnable interface.

1. The **run() method** is the most important method in any threading program. By using this method the thread's behaviour can be implemented. The run method can be written as follows –

```
public void run()
{
     Statement for implementing thread
}
```

How to make classes y threadable

2. For invoking the thread's run method the object of a thread is required. This object can be obtained by creating and initiating a thread using the start() method.

Extending Threads

- The **Thread class** can be used to create a thread.
- Using the **extend** keyword your class extends the Thread class for creation of thread.
- For example if I have a class named A then it can be written as

class A extends Thread

- **Constructor of Thread Class:** Following are various syntaxes used for writing the constructor of Thread Class.

Extending Threads

- Thread()
- Thread(String s)
- Thread(Runnable obj)
- Thread(String s, Runnable obj)

Extending Threads

Various commonly used methods during thread programming are as given below -

- **start() -** The thread can be started and invokes the run method.
- **run() -** Once thread is started it executes in run method.
- **setName() -** We can give the name to a thread using this method.
- **getName() -** The name of the thread can be obtained using this name.
- **join() -** This method waits for thread to end

Extending Threads

```
class MyThread extends Thread
{
        public void run()
        {
                System.out.println("Thread is created!!!");
        }
}
class ThreadProg
{
        public static void main(String args[])
        {
                MyThread t =new MyThread();
                t.start();
        }
}
```

Extending Threads

Program Explanation:

- In above program, we have created two classes.
- One class named MyThread extends the Thread class.
- In this class the run method is defined.
- This run method is called by t.start() in main() method of class ThreadProg
- The thread gets created and executes by displaying the message Thread is created.

Extending Threads

Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming"

```
class MyThread extends Thread
{
        String str="; //data member of class MyThread
        MyThread(String s)//constructor
        {
                this.str=s;
        }
        public void run()
        {
                System.out.println(str);
        }
}
```

Extending Threads

```
class ThreadProg
{
        public static void main(String args[])
        {
                MyThread t = new MyThread("You are Welcome to Thread Programming");
                t.start();
        }
}
```

Implementing Runnable

- The thread can also be created using runnable interface.
- Implementing thread program using Runnable interface is preferable than implementing it by extending the thread class because of the following two reasons –

1. If a class extends a thread class then it can not extends any other class which may be required to extend.
2. If a class thread is extended then all its functionalities get inherited. This is an expensive operation.

Implementing Runnable

- Following Java program shows how to implement Runnable interface for creating a single thread.

```
class MyThread implements Runnable
{
        public void run()
        {
                System.out.println("Thread   is
created!");
        }
}
```

```
class ThreadProgRunn
{
        public static void main(String args[])
        {
                MyThread obj=new MyThread();
                Thread t=new Thread(obj);
                t.start();
        }
}
```

Implementing Runnable

Program Explanation:

- In above program, we have used interface Runnable.
- While using the interface, it is necessary to use implements keyword.
- Inside the main method
    1. Create the instance of class MyThread.
    2. This instance is passed as a parameter to Thread class.
    3. Using the instance of class Thread invoke the start method.
    4. The start method in-turn calls the run method written in MyThread.
- The run method executes and display the message for thread creation.

Implementing Runnable

Create a thread by implementing runnable interface. Also make use of constructor and display message "You are Welcome to Thread Programming"

Implementing Runnable

```
class MyThread implements Runnable
{
        String str;
        MyThread(String s)
        {
                this.str=s;
        }
        public void run()
        {
                System.out.println(str);
        }
}
```

```
class ThreadProgRunn
{
        public static void main(String args[])
        {
                MyThread obj=new MyThread("You are Welcome to Thread Programming");
                Thread t =new Thread(obj);
                t.start();
        }
}
```

Output

```
You are Welcome to Thread Programming
```

University Questions

1. What is Thread ? Explain two ways of creation of thread. **VTU : Jan.-18, Marks 5**
2. Elucidate the two ways of making a class threadable, with examples. **VTU : July-18, Marks 8**
3. What is thread ? Explain two ways of creating a thread in JAVA with example. **VTU : Jan.-19, Marks 8**
4. Explain the concepts of multithreading in Java. Explain the two ways of making class threadable with examples. **VTU : July-19, Marks 10**

Creating Multiple Threads

- Multiple threads can be created by extending the Thread class and by implementing the runnable interface.

Creating Multiple Threads using Threads Class

```
class A extends Thread
{
        public void run()
        {
                for (int i=1; i<=5; i++)
                {
                        System.out.println("Thread - 1");
                }
        }
}
```

Creating Multiple Threads using Threads Class

```
class B extends Thread
{
        public void run()
        {
                for (int i=1; i<5; i++)
                {
                        System.out.println("Thread - 2");
                }
        }
}
```

Creating Multiple Threads using Threads Class

```
public class MultipleThreadsUsingThreadCLass
{
        public static void main(String args[])
        {
                A t1 = new A();
                B t2 = new B();
                t1.start();
                t2.start();
        }
}
```

Creating Multiple Threads using Threads Class

SAMPLE OUTPUT:

```
Thread - 1
Thread - 2
Thread - 2
Thread - 2
Thread - 2
Thread - 1
Thread - 1
Thread - 1
Thread - 1
```

Creating Multiple Threads by Implementing Runnable Interface

Java program for creating multiple threads by implementing the Runnable Interface

Creating Multiple Threads by Implementing Runnable Interface

```
class AA implements Runnable
{
        public void run()
        {
                for (int i=1; i<=5; i++)
                {
                        System.out.println("Thread - 1");
                }
        }
}
```

Creating Multiple Threads by Implementing Runnable Interface

```
class BB implements Runnable
{
        public void run()
        {
                for (int i=1; i<5; i++)
                {
                        System.out.println("Thread - 2");
                }
        }
}
```

Creating Multiple Threads by Implementing Runnable Interface

```
public class MultipleThreadsUsingRunnable
{
        public static void main(String args[])
        {
                AA obj1 = new AA();
                BB obj2 = new BB();
                Thread t1 = new Thread(obj1);
                Thread t2 = new Thread(obj2);
                t1.start();
                t2.start();
        }
}
```

Creating Multiple Threads by Implementing Runnable Interface

SAMPLE OUTPUT:

```
Thread - 1
Thread - 1
Thread - 1
Thread - 2
Thread - 2
Thread - 1
Thread - 1
Thread - 2
Thread - 2
```

Multiple Threads – Synchronization

- When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization. The synchronization is the concept which is based on monitor. Monitor is used as mutually exclusive lock or mutex. When a thread owns this monitor at a time then the other threads can not access the resources. Other threads have to be there in waiting state.
- In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword synchronized. When a particular method is in synchronized state then all other threads have to be there in waiting state.

Multiple Threads – Synchronization

- In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword synchronized.
- When a particular method is in synchronized state then all other threads have to be there in waiting state.
- There are two ways to achieve the synchronization –
  1. Using Synchronized Methods
  2. Using Synchronized Blocks (Statements).

Synchronization – Using Synchronized Method

- Let us make the method synchronized to achieve the synchronization by using following Java program -

Synchronization – Using Synchronized Method

```
class Sync
{
        synchronized void display(int n)
        {
                System.out.println("The table for "+n);
                for (int i=1;i<=10; i++)
                {
                        System.out.print(" "+n*i);
                }
                System.out.println("\nEnd of table");
        }
}
```

Synchronization – Using Synchronized Method

```
class thread1 extends Thread
{
        Sync s1;
        thread1(Sync s)
        {
                s1 = s;
        }
        public void run()
        {
                s1.display(2);
        }
}
```

Synchronization – Using Synchronized Method

```
class thread2 extends Thread
{
        Sync s2;
        thread2(Sync s)
        {
                s2 = s;
        }
        public void run()
        {
                s2.display(10);
        }
}
```

Synchronization – Using Synchronized Method

```
public class SynchronizedMethod
{
        public static void main(String args[])
        {
                Sync obj = new Sync();
                thread1 t1 = new thread1(obj);
                thread2 t2 = new thread2(obj);
                t1.start();
                t2.start();
        }
}
```

Synchronization – Using Synchronized Method

Sample Output:

```
The table for 2

 2 4 6 8 10 12 14 16 18 20

End of table

The table for 10

 10 20 30 40 50 60 70 80 90 100

End of table
```

Synchronization – Using Synchronized Method

Program Explanation:

- In above program we have written one class named Sync. Inside this class the synchronized method named display is written. This method displays the table of numbers.
- We have written two more classes named thread1 and thread2 for executing the thread. The constructors for class thread1 and Class thread2 are written and to initialize the instance variables of class Sync as s1 (as a variable for class thread1) and s2 (as a variable for class thread2)
- Inside the run methods of these classes we have passed number 2 and 10 respectively and display method is invoked.
- The display method executes firstly for thread s1 and then for s2 in synchronized manner.

Synchronization – Using Synchronized Block

Using Synchronized Block

- When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized.
- Synchronized statements must specify the object that provides the intrinsic lock.

Syntax The syntax for using the synchronized block is –

```
synchronized (object references)
{
 // Block of code to be synchronized
}
```

Synchronization – Using Synchronized Block

```
class Sync
{
        void display(int n)
        {
                synchronized (this)
                {
                        System.out.println("The table for "+n);
                        for (int i=1;i<=10; i++)
                        {
                                System.out.print(" "+n*i);
                        }
                        System.out.println("\nEnd of table");
                }
        }
}
```

Synchronization – Using Synchronized Block

```
class thread1 extends Thread
{
        Sync s1;
        thread1(Sync s)
        {
                s1 = s;
        }
        public void run()
        {
                s1.display(2);
        }
}
```

Synchronization – Using Synchronized Block

```
class thread2 extends Thread
{
        Sync s2;
        thread2(Sync s)
        {
                s2 = s;
        }
        public void run()
        {
                s2.display(10);
        }
}
```

Synchronization – Using Synchronized Block

```
public class SynchronizedMethod
{
        public static void main(String args[])
        {
                Sync obj = new Sync();
                thread1 t1 = new thread1(obj);
                thread2 t2 = new thread2(obj);
                t1.start();
                t2.start();
        }
}
```

Synchronization – Using Synchronized Block

Sample Output:

```
The table for 2
 2 4 6 8 10 12 14 16 18 20
End of table
The table for 10
 10 20 30 40 50 60 70 80 90 100
End of table
```

Concept of static synchronization

- The static synchronization can be achieved by making the static method as synchronized.
- In static synchronization the lock will be on the class and not on the instance.
- That means while execution of a static method the whole class is blocked.
- So other static synchronized methods are also blocked.

What is the difference between static synchronized and synchronized methods ?

- Static synchronized methods synchronize on the class. If one thread is executing a static synchronized method, all other threads trying to execute any static synchronized methods will be blocked.
- Non-static synchronized methods synchronize on the instance of the class. If one thread is executing a synchronized method, all other threads trying to execute any synchronized methods will be blocked.

Synchronization – static synchronization

```
class Sync
{
        synchronized static void display(int n)
        {
                System.out.println("The table for "+n);
                for (int i=1;i<=10; i++)
                {
                        System.out.print(" "+n*i);
                }
                System.out.println("\nEnd of table");
        }
}
```

Synchronization – static synchronization

```
class thread1 extends Thread
{
        Sync s1;
        thread1(Sync s)
        {
                s1 = s;
        }
        public void run()
        {
                s1.display(2);
        }
}
```

Synchronization – static synchronization

```
class thread2 extends Thread
{
        Sync s2;
        thread2(Sync s)
        {
                s2 = s;
        }
        public void run()
        {
                s2.display(10);
        }
}
```

Synchronization – static synchronization

```
public class SynchronizedMethod
{
        public static void main(String args[])
        {
                Sync obj = new Sync();
                thread1 t1 = new thread1(obj);
                thread2 t2 = new thread2(obj);
                t1.start();
                t2.start();
        }
}
```

Synchronization – static synchronization

Sample Output:

```
The table for 2

 2 4 6 8 10 12 14 16 18 20

End of table

The table for 10

 10 20 30 40 50 60 70 80 90 100

End of table
```

University Questions

1. How synchronization can be achieved for threads in Java ? Explain with syntax. VTU: July-17, Marks 6
2. Discuss briefly Synchronization in Java (2). VTU: July-18, Marks 2
3. Write an example Program for implementing static synchronization in Java. VTU: July-18, Marks 6
4. What is the need of synchronization? Explain with an example how synchronization is implemented in JAVA. VTU: Jan.-19, Marks 8
5. What is synchronization? When do we use it? VTU: Jan.-18, Marks 5

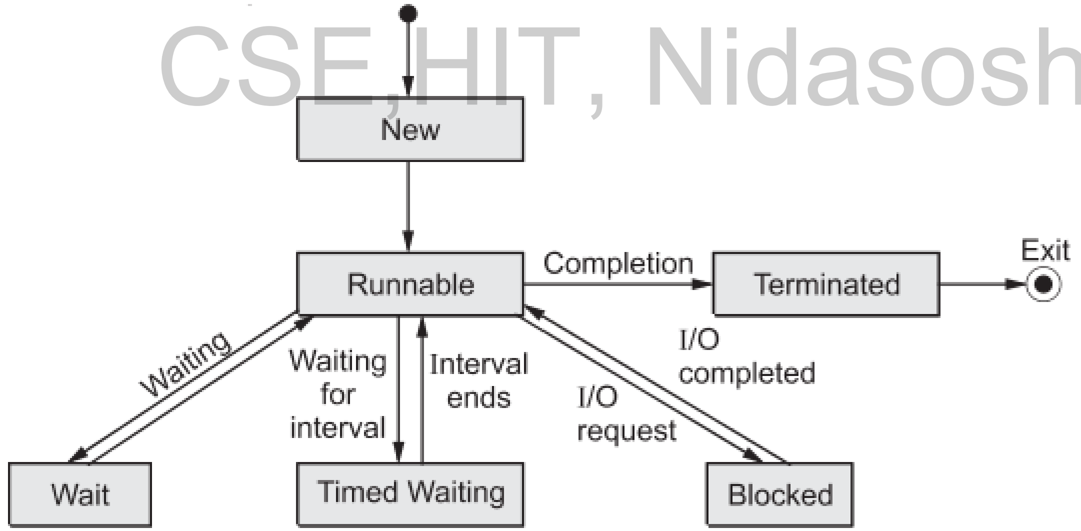
Changing the State of the Thread

Thread always exists in any one of the following states

1. New or create state
2. Runnable state
3. Waiting state
4. Timed waiting state
5. Blocked state
6. Terminated state.

Changing the State of the Thread – Life Cycle

Thread life cycle specifies how a thread gets processed in the Java program by executing various methods. Following Fig. represents how a particular thread can be in one of the state at any time



Changing the State of the Thread – Life Cycle

New state

When a thread starts its life cycle it enters in the new state or a create state.

Runnable state

This is a state in which a thread starts executing.

Waiting state

Sometimes one thread has to undergo in waiting state because another thread starts executing.

Changing the State of the Thread – Life Cycle

Timed waiting state

There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state.

Blocked state

When a particular thread issues an Input/Output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state.

Changing the State of the Thread – Life Cycle

Terminated state

After successful completion of the execution the thread in runnable state enters the terminated state.

Suspending and Stopping the thread

1. **Stopping a thread**: A thread can be stopped from running further by issuing the following statement - **th.stop();** By this statement the thread enters in a dead state. From stopping state a thread can never return to a runnable state.
2. **Blocking a thread:** A thread can be temporarily stopped from running. This is called blocking or suspending of a thread.

Changing the State of the Thread – Life Cycle

Following are the ways by which thread can be blocked –

1. **sleep()** By sleep() method a thread can be blocked for some specific time. When the specified time gets elapsed then the thread can return to a runnable state.
2. **suspend -** By suspend() method the thread can be blocked until further request comes. When the resume() method is invoked then the thread returns to a runnable state.

Changing the State of the Thread – Life Cycle

3. wait - The thread can be made suspended for some specific conditions using wait() method.

- When the notify method is called then the blocked thread returns to the runnable state.
- The difference between the suspending and stopping thread is that if a thread is suspended then its execution is stopped temporarily and it can return to a runnable state. But in case , if a thread is stopped then it goes to a dead state and can never return to runnable state.
- Note that while using these methods, the exception **InterupttedException** must be used.

Changing the State of the Thread – Life Cycle

```
/* Program for demonstrating the use of suspend() and resume() methods using two thread  */
public class Suspend_ResumeDemo extends Thread
{
        public void run()
        {
                try
                {
                        for( int i = 0; i < 5; i+ +)
                        {
                                Thread.sleep(500);
                                System.out.println( this.getName() + ": " + i );
                        }
                }
                catch( InterruptedException e )
                {
                        e.printStackTrace();
                }
        }
```

Changing the State of the Thread – Life Cycle

```
public static void main( String args[ ] )
{
        Suspend_ResumeDemo t1 = new Suspend_ResumeDemo();
        Suspend_ResumeDemo t2 = new Suspend_ResumeDemo();
        t1.setName("A");
        t2.setName("B");
        t1.start();
        t2.start();
        try
        {
                Thread.sleep( 1000 );
                t1.suspend();
                System.out.println("Suspending A Thread");
                Thread.sleep( 1000 );
                t1.resume();
                System.out.println("Resuming A Thread");
```

Changing the State of the Thread – Life Cycle

```
                        Thread.sleep(1000);
                        t2.suspend();
                        System.out.println("Suspending B Thread");
                        Thread.sleep(1000);
                        t2.resume();
                        System.out.println("Resuming B Thread");
                }
                catch(InterruptedException e)
                {
                        e.printStackTrace();
                }
        }
}
```

Output

```
A: 0
B: 0
Suspending A Thread
B: 1
B: 2
Resuming A Thread
A: 1
B: 3
A: 2
B: 4
Suspending B Thread
A: 3
 A: 4
Resuming B Thread
```

University Questions

1. What are the differences between suspending and stopping the threads ? VTU : July-17, Marks 5
2. Define the concept of multithreading in Java and explain the different phases in the life cycle of a thread, with a neat sketch. VTU : July-18, Marks 8

isAlive() and join() Methods

- There are two ways that determine whether the thread has finished or not.
- These methods are **isAlive()** and **join().**
- The isAlive() returns **true** upon which it is called is still running. It returns **false** otherwise.
- The syntax of isAliv() method is as follows - **final boolean isAlive()**
- The method that is used commonly to wait for a thread to finish is called **join().**
- The syntax of join() is as follows - **final void join( ) throws InterruptedException**
- This method waits until the thread on which it is called terminates.

isAlive() and join() Methods

```
/*Following is a Java program that demonstrates the use of these methods.*/
class A extends Thread
{
        public void run()
        {
                System.out.println("First Thread!!!");
                try
                {
                        for(int i=0;i<5;i+ +)
                        {
                                System.out.println(i);
                                Thread.sleep(1000);
                        }
                }
                catch(InterruptedException e){}
        }
}
```

isAlive() and join() Methods

```
class B extends Thread
{
        public void run()
        {
                System.out.println("Second Thread!!!");
                try
                {
                        for(int i= 10;i> 5;i--)
                        {
                                System.out.println(i);
                                Thread.sleep(1000);
                        }
                }
                catch(InterruptedException e){}
        }
}
```

isAlive() and join() Methods

```
class ThreadJoin
{
        public static void main(String args[])
        {       A t1=new A();
                t1.start();
                try
                {
                        t1.join();
                }
                catch(InterruptedException e){}
                System.out.println("IS threadl alive? “+t1.isAlive());
                B t2=new B();
                System.out.println("IS thread2 alive? "+t2.isAlive());
                t2.start();
                System.out.println("IS thread2 alive? "+t2.isAlive());
        }
}
```

Output

```
First Thread!!!
0
1
2
3
4
IS thread1 alive? false
IS thread2 alive? false
IS thread2 alive? true
Second Thread!!!
10
9
8
7
6
```

University Questions

1. With the syntax explain the use of isAlive() and Join() methods.

   VTU : July-17, Marks 6

2. With the syntax, explain the use of is Alive( ) and join( ) methods.

   VTU : Jan.-19, Marks 4

3. With a syntax, explain is Alive( ) and join( ) with suitable program.

   VTU : July-19, Marks 10

Producer Consumer Problem

- Two or more threads communicate with each other by exchanging messages. This mechanism is called inter-thread communication.
- **Polling** is a mechanism generally implemented in a loop in which certain condition is repeatedly checked.
- To better understand the concept of polling, consider the producer-consumer problem in which producer thread produces and the consumer thread consumes whatever is produced.

Producer Consumer Problem

- Both must work in coordination to avoid wastage of CPU cycles.
- But there are situations in which the producer has to wait for the consumer to finish consuming data.
- Similarly, the consumer may need to wait for the producer to produce the data. In the Polling system either the consumer will waste many CPU cycles when waiting for the producer to produce or the producer will waste CPU cycles when waiting for the consumer to consume the data.

Producer Consumer Problem

- In order to avoid polling, there are three in-built methods that take part in inter-thread communication –

notify()	If a particular thread is in the sleep mode then that thread can be resumed using the notify call.
notifyall()	This method resumes all the threads that are in the suspended state.
wait()	The calling thread can be sent into sleep mode.

Producer Consumer Problem

- **Example Program**
- Following is a simple Java program in which two threads are created one for the producer and another is for the consumer.
- The producer thread produces(writes) the numbers from 0 to 9 and the consumer thread consumes(reads) these numbers.
- The wait and notify methods are used to send a particular thread to sleep or to resume the thread from sleep mode respectively.

Producer Consumer Problem

```
class MyClass
{
        int val;
        boolean flag = false;
        synchronized int get() //by toggling the flag Synchronized read and write is performed
        {
                if(!flag)
                {
                        try  {    wait();    }
                        catch(InterruptedException e)
                        {
                                System.out.println("InterruptedException!!!");
                        }
                }
                System.out.println("Consumer consuming: " + val);
                flag = false;
                notify();
                return val;
        }
```

Producer Consumer Problem

```
        synchronized void put(int val)
        {
                if(flag)
                {
                        try
                        {
                                wait();
                        }
                        catch(InterruptedException e)
                        {
                                System.out.println("InterruptedException!!!");
                        }
                }
                this.val = val;
                flag = true;
                System.out.println("Producer producing: " + val);
                notify();
        }
}
```

Producer Consumer Problem

```
class Producer extends Thread
{
        MyClass th1;
        Producer(MyClass t)
        {
                th1 = t;
        }
        public void run()
        {
                for(int i=0; i<10; i++)
                {
                        th1.put(i);
                }
        }
}
```

```
class Consumer extends Thread
{
        MyClass th2;
        Consumer(MyClass t)
        {
                th2 = t;
        }
        public void run()
        {
                for (int i = 0;i<10; i++)
                {
                        th2.get();
                }
        }
}
```

Producer Consumer Problem

```
class ProducerConsumer
{
        public static void main(String arg[])
        {
                MyClass TObj = new MyClass();
                Producer pthread=new Producer(TObj);
                Consumer cthread=new Consumer(TObj);
                pthread.start();
                cthread.start();
        }
}
```

Output

```
Producer producing: 0
Consumer consuming: 0
Producer producing: 1
Consumer consuming: 1
Producer producing: 2
Consumer consuming: 2
Producer producing: 3
Consumer consuming: 3
Producer producing: 4
Consumer consuming: 4
Producer producing: 5
Consumer consuming: 5
Producer producing: 6
Consumer consuming: 6
Producer producing: 7
Consumer consuming: 7
Producer producing: 8
Consumer consuming: 8
Producer producing: 9
Consumer consuming: 9
```

Producer Consumer Problem

Program Explanation

- **Inside get( )** -The wait( ) is called in order to suspend the execution by that time the producer writes the value and when the data gets ready it notifies other threads that the data is now ready.
- Similarly, when the consumer reads the data execution inside get( ) is suspended. After the data has been obtained, get( ) calls notify( ). This tells the producer that now the producer can write the next data in the queue.

Producer Consumer Problem

- **Inside put( )** - The wait( ) suspends execution by that time the consumer removes the item from the queue.
- When execution resumes, the next item of data is put in the queue, and notify( ) is called.
- When the notify is issued the consumer can now remove the corresponding item for reading.

Bounded Buffer Problems

- The bounded buffer problem is a classic problem in which there is a buffer of n slots and each slot is capable of storing one unit of data.
- There are two processes running, namely, producer and consumer, which are operating on the buffer.
- The producer inserts data in the buffer and Consumer deletes the data from the buffer

Bounded Buffer Problems

```
class Buffer
{
        int Max;
        int[] Queue;
        int front, rear, QSize;
        public Buffer(int size)
        {
                Max = size;
                rear = -1;
                front = 0;
                QSize = 0;
                Queue = new int[Max];
        }
```

Bounded Buffer Problems

```
public synchronized void insert(int ch)
{
        try
        {
                while (QSize == Max)
                {
                        wait();
                }
                rear = (rear + 1) % Max;
                Queue[rear] = ch;
                QSize++;
                notifyAll();
        }
        catch (InterruptedException e) {}
}
```

Bounded Buffer Problems

```
public synchronized int delete()
{
        int ch = 0;
        try
        {
                while (QSize == 0)
                {
                        wait();
                }
                ch = Queue[front];
                front = (front + 1) % Max;
                QSize--;
                notifyAll();
        }
        catch (InterruptedException e)  {}
        return ch;
}
}
```

Bounded Buffer Problems

```
class Consumer extends Thread
{
        Buffer buffer; public Consumer(Buffer b)
        {
                buffer = b;
        }
        public void run()
        {
                while (!Thread.currentThread().isInterrupted())
                {
                        pi int c = butter.delete();
                        System.out.println(c);
                }
        }
}
```

Bounded Buffer Problems

```
class Producer extends Thread
{
        Buffer buffer;
        public Producer(Buffer b)
        {
                buffer = b;
        }
        public void run()
        {
                for(int c=0;c<10;c++)
                        buffer.insert(c);
        }
}
```

Bounded Buffer Problems

```
class BoundedBuffer
{
        public static void main(String[] args)
        {
                Buffer Q = new Buffer(5);                    // Queue of size 5
                Producer prod = new Producer(Q);             //for inserting data to Queue
                Consumer cons = new Consumer(Q);             //for deleting data from Queue
                prod.start();
                cons.start();
                try
                {
                        prod.join();
                        cons.interrupt();
                }
                catch (InterruptedException e) {}

        }
}
```

University Questions

1. Explain the role of synchronization with producer and consumer problem. VTU : Jan.-18, Marks 8
2. Explain the role of synchronization with Bounded buffer problem.

CSE,HIT, Nidasoshi