UNIT – III

INTERFACES, PACKAGES AND MULTITHREADED PROGRAMMING

INTERFACE **[2 Marks]**

- Interface is a collection of final fields and abstract methods.
- Interface looks like class but it is not a class.
- An interface can have methods and variables just like the class
- The methods declared in interface are by default abstract (only method signature, no body).
- The variables declared in an interface are public, static & final by default.
- The interface in java is **a mechanism to achieve fully abstraction**.
- Java Interface also **represents IS-A relationship**.
- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

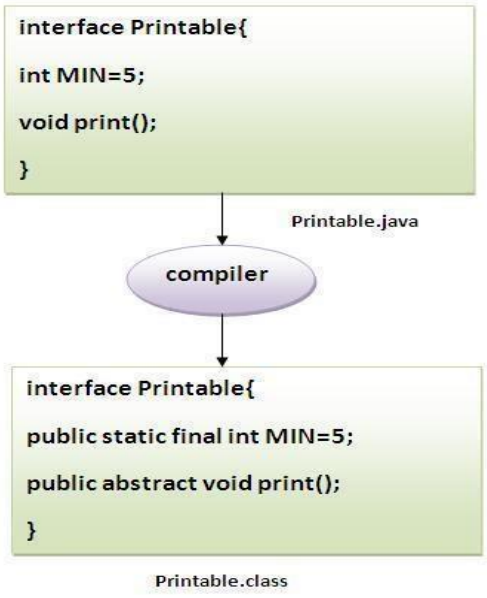
Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

In other words, Interface fields are public, static and final bydefault, and methods are public and abstract.



Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.

DIFFERENCES BETWEEN A CLASS AND AN INTERFACE [5 Marks]

Class	Interface
The fields declared in a class can be constant or variable	The fields declared in interface are always constant.
The methods declared in class can be abstract or non abstract	The methods declared in interface are always abstract
We can create object of class	We cannot create object of an interface
It can be various access specifier like public, private, protected	It can only use public access specifier
Using class we cannot use multiple inheritance	Using interface we can achieve multiple inheritance
We can declare constructor in a class	We cannot declare constructor in interface

Declaration

Interfaces are declared by specifying a keyword "interface". E.g.:

```
interface MyInterface
{
        /* All the methods are public abstract by default
        * Note down that these methods are not having body
        */
        public void method1();
        public void method2();
}
```

EXTENDING INTERFACES: [5 Marks]

An interface can extend another interface, similarly to the way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Syntax

```
interface name2 extends name1
{
        Body of name2;
}
```

Example

```
interface A
{
        int rno=10;
        String name="Sachin";
}

interface B extends A
{
        void display();
}
```

Extending Multiple Interfaces:

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.
The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

Example

```
interface A
{
        int rno=10;
        String name="Sachin";
}


interface B extends A
{
        void display();
}
interface C extends A,B
{
}
```

INTERFACE IMPLEMENTATION **[5 Marks]**

This is how a class implements an interface. It has to provide the body of all the methods that are declared in interface.
Note: Class implements interface but an interface extends another interface.

Syntax

```
interface <interface-name>
{
}
class <class-name> implements <interface-name>
{
}
```

example

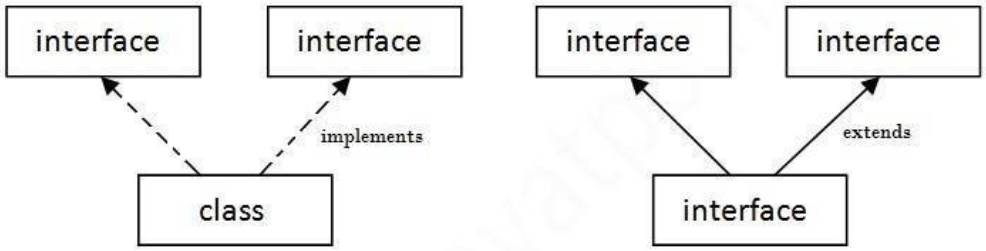
```
interface A
{
        public void method1(); public
        void method2();
}

class XYZ implements A
{
        public void method1()
        {
                System.out.println("implementation of method1");
        }
        public void method2()
        {
                System.out.println("implementation of method2");
         }
}

class Interface
{
        public static void main(String args[])
        {
                XYZ obj = new
                XYZ(); obj.
                method1();
                obj.method2();
        }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac Interface.java
C:\Program Files\Java\jdk1.7.0\bin>java Interface
implementation of method1
implementation of method2
```

MULTIPLE INHERITANCE IN JAVA BY INTERFACE **[5 Marks]**

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

Example of multiple inheritance in java using interface.

```
interface A
{
        public void method1();
}

interface B
{
        public void method2();
}
class C implements A,B
{
        public void method1()
        {
                System.out.println("implementation of method1");
        }
        public void method2()
        {
                System.out.println("implementation of method2");
        }
}

class MultipleInheritance
{
        public static void main(String args[])
        {
                C obj = new
                C();            obj.
                method1();
        }       obj.method2();
```

OUTPU
T

```
C:\Program Files\Java\jdk1.7.0\bin>javac MultipleInheritance.java
C:\Program Files\Java\jdk1.7.0\bin>java MultipleInheritance
implementation of method1
implementation of method2
```

JAVA PACKAGES

One of the requirements of an Object Oriented language is that it provides encapsulation.Encapsulation is a technique by which multiple related objects can be grouped under one object. Java implements encapsulation by the use of Packages.

Package - A package is a collection of related classes, interfaces and sub packages. [2 Marks]

Benefits of Packages

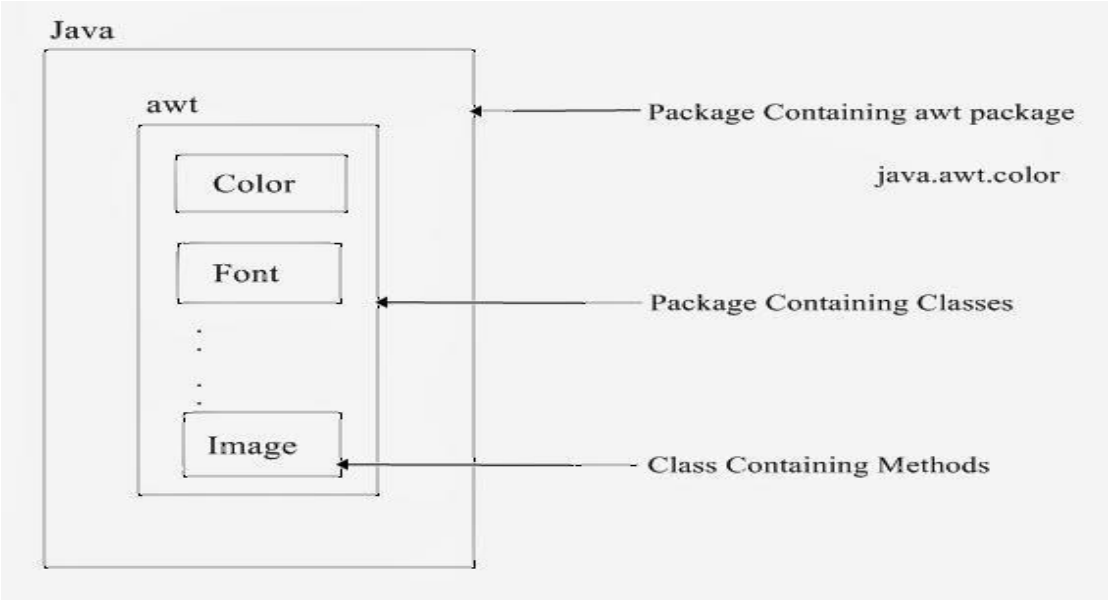
1. Classes contained in packages of other programs can be easily reused.
2. Two classes in two different packages can have the same name. They can be uniquely identified by packagename.classname.
3. Packages also provide a way for separating "design" from "coding".
4. You can define classes inside a package that are not accessible by code outside that package. You can also define class members that are only exposed to other members of the same package.

In practical applications, we may have to build our own classes and use existing classes libraries for designing use interfaces.

JAVA PACKAGES ARE CLASSIFIED INTO TWO TYPES. [2 Marks]

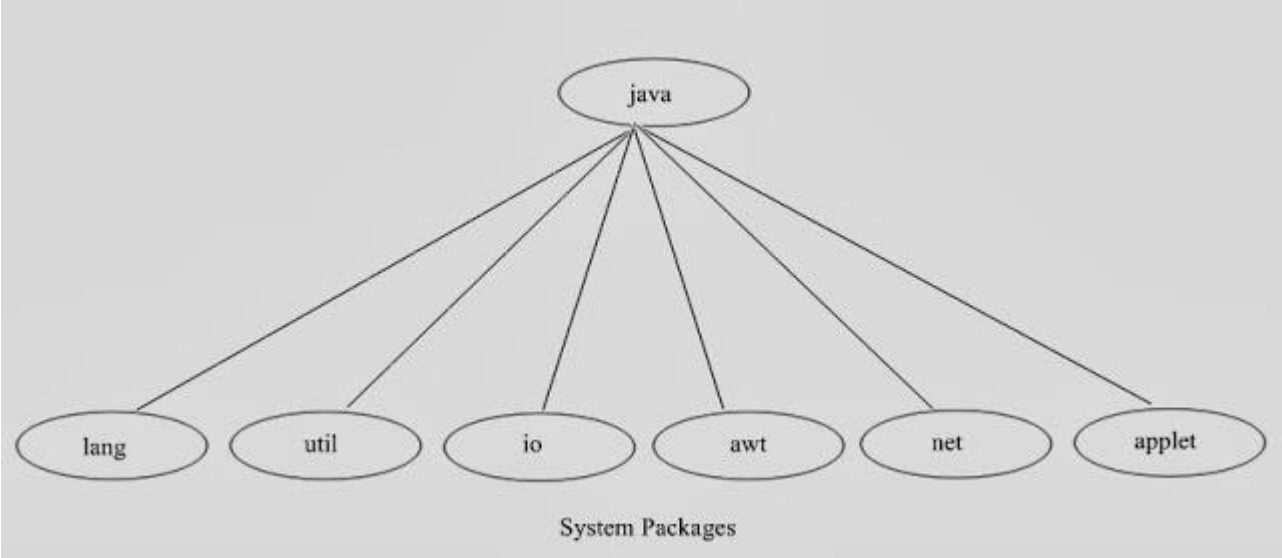
1. Java API package
2. User defined packages

Using Java API Packages



SYSTEM PACKAGES **[2 Marks]**

Java API provides a large number of classes grouped into different packages according to functionality. Most of the time in our programming we use the Java System Packages. The following are the frequently used system packages, showing the functional breakdown.



System Packages

The following are the brief description of Java System Packages and their Classes: [2 Marks]

Java.lang	Lang means to language. Lang classes are used by java compiler itself and therefore they are automatically imported. Classes that are required for primitive types, strings main functions, threads and exceptions included in it.
Java.util	Util stands for language utility. It include classes such as vectors, has tables random numbers, date etc.
Java.io	Input/Output support classes are included in it. They provide the facilities for the input and output of data.
Java.awt	Awt stands for abstract window toolkit. It is used for implementing graphical user interface in the program. They include classes for windows, buttons, Menu , checkbocx etc.
Java.net	Net stands for networking. Therefore the classes required for communicating with local computers as well as with internet swervers are included in it.
Java.applet	This pacake includes the classes for creating and implementing applets.

Naming Conventions **[2 Marks]**

Package names are written in all lower case to avoid conflict with the names of classes or interfaces.

CREATING A PACKAGE **[10 Marks]**

Creating a package in java is quite easy. We must first declare the name of the package using the **package** keyword followed by a package name. It must be a first statement in java source file.

```
package <package-name>;
public class <class-name>
{
 ...statement;
}
```

```
Ex- package package1
   Public class ClassA
   {
   }
```

Here the package name is package1. The class ClassA is now considered a part of this package. This is saved as a file called ClassA.java and located in a directory name package1. When a source file is compiled, java will create .class file and store it in the same directory.
Remember that the .class file must be located in a directory that has the same name as the package. The following steps are used to create a package

1. Declare the package at the beginning of a file using the form
   package packagename;
2. Define the class that is to be put in the package and define it public
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates .class file in the subdirectory.

Java also support the concept of package hierarchy. This is done by specifying multiple names in a package statements, separated by dots(.)
Example : package package1.secondpackage;

```
Package package1;
Public class ClassA
{
        Public void display();
        System.out.println("I am in package1 class A");
        }
}
```

ACCESSING PACKAGE **[2 Marks]**

import keyword is used to import built-in and user-defined packages into your java source file. So that your class can refer to a class that is in another package by directly using its name.

Syntax

import package1.package2.classname;

USING PACKAGE **[10 Marks]**

```
package p1;
public class ClassA
{
	public void display()
	{
		System.out.println("Welcome to Package");
	}
}
```

Save this program **ClassA.java** in bin folder(directory)

Compilation : compile ClassA.java file using following command
C:\Program Files\Java\jdk1.7.0\bin>javac -d . ClassA.java

When you compile above program p1 package is created in bin. In that p1 package ClassA.class file also created.

```
import p1.ClassA;
class Pack
{
	public static void main(String args[])
	{
		ClassA a=new ClassA();
		a.display();
	}
}
```

Save this program **Pack.java** in bin folder(directory)

Compilation : compile Pack.java file using following command
C:\Program Files\Java\jdk1.7.0\bin>javac Pack.java **Execution** :
C:\Program Files\Java\jdk1.7.0\bin>java Pack
Welcome to Package

ACCESS PROTECTION **[2 Marks]**

Access modifier / Access Location	public	Protected	private	Default
Same class	Yes	Yes	Yes	Yes
Subclass in same class	Yes	Yes	No	Yes
Other classes in same class	Yes	Yes	No	Yes
Subclass in other package	Yes	Yes	No	No
Non-sub class in other package	Yes	No	No	No

Lab 8 : Program to demonstrate a)Packages b)Interfaces. **[5 Marks]**

```
// Save this program A.java

package p1;
public interface A
{
        int a=10,b=20;
}

/* Compilation : compile A.java file using following command
C:\Program Files\Java\jdk1.7.0\bin>javac -d . A.java

When you compile above program p1 package is created in bin. In that p1 package A.class file also created. */

// Save this file B.java
package p2;
public class B
{
        public void display()
        {
                System.out.println("I am Package p2 and in Class B");
        }
}
/* Compilation : compile B.java file using following command
C:\Program Files\Java\jdk1.7.0\bin>javac -d . B.java

When you compile above program p2 package is created in bin. In that p2 package B.class file also
created. */
```

MULTITHREADED PROGRAMMING

Thread - Thread is a small block of code, which executes particular task [2 Marks]

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.

Multithreading in java is a process of executing multiple threads simultaneously. **[2 Marks]**

Advantage of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.
2) You **can perform many operations together so it saves time**.
3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

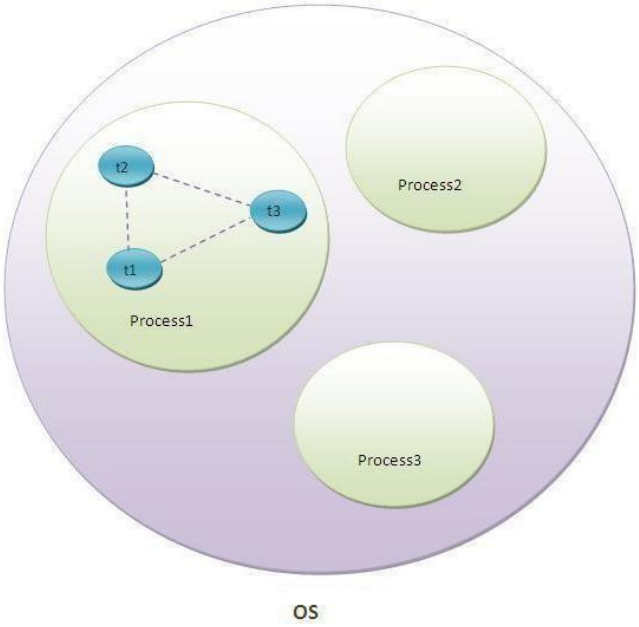
1) Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

Note: At least one process is required for each thread.



As shown in the above figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

Note: At a time one thread is executed only.

CREATING THREADS **[2 or 5 or 10 Marks]**

Threads are implemented in the form of objects that contains a method called run(). The run() method is the heart and soul of any thread.

In run() method only we can implement the threads behavior .

Syntax-

```
Public void run()
{
        Body of thread
}
```

There are two ways to create a thread:

1. By extending Thread class -: define class that extends thread class and override its run() method.
2. By implementing Runnable interface-: define class that implements Runnable interface.

EXTENDING THREAD CLASS **[5 Marks]**

We can make our class runnable by extending thread class java.lang.Thread. This gives us access to all the thread methods directly.

It includes the following steps.

1. Declare the class as extending the Thread class
2. Implement the run() method
3. Create a thread object and call the start() method to initiate the thread execution.

Declaring the class

```
class MyThread extends Thread
{
        Body
}
```

Implementing the run() method

The run() method has been inherited by the class MyThread. public

```
void run()
{
    Body
}
```

Starting new thread

To start our thread we have to create object of thread class and using object we have to call our start() method.

```
MyThread obj=new MyThread();
obj.start();
```

Example of Creating threads using thread class

```
class A extends Thread
{
    public void run()
    {
        System.out.println("Starts Thread A");
        for(int i=1;i<=5;i++)
        {
            System.out.println("From thread A : i = " +i);
        }
        System.out.println("Exit from Thread A");
    }
}
class B extends Thread
{
    public void run()
```

```
        {
                System.out.println("Starts Thread B");
                for(int j=1;j<=5;j++)
                {
                        System.out.println("From thread B : j = " +j);
                }
                System.out.println("Exit from Thread B");
        }
}

class C extends Thread
{
        public void run()
        {
                System.out.println("Starts Thread C");
                for(int k=1;k<=5;k++)
                {
                        System.out.println("From thread C : k = " +k);
                }
                System.out.println("Exit from Thread C");
        }
}

class MyThread
{
        public static void main(String args[])
        {
                A a=new A();
                B b=new B();
                C c=new C();
                a.start();
                b.start();
                c.start();
        }
}
```

Output

```
C:\Program Files\Java\jdk1.7.0\bin>javac MyThread
C:\Program Files\Java\jdk1.7.0\bin>java MyThread
Starts Thread A
From thread A : i = 1
From thread A : i = 2
From thread A : i = 3
```

From thread A : i = 4
From thread A : i = 5
Exit from Thread A
Starts Thread C From
thread C : k = 1 From
thread C : k = 2 From
thread C : k = 3 From
thread C : k = 4 From
thread C : k = 5 Exit
from Thread C Starts
Thread B From thread
B : j = 1 From thread
B : j = 2 From thread
B : j = 3 From thread
B : j = 4 From thread
B : j = 5 Exit from
Thread B

STOPPING AND BLOCKING A THREAD **[2 Marks]**

Stopping a thread

Whenever we want to stop a thread from running we may do so by calling its stop() method.
Example - : a.stop();

This statement causes the thread to move to dead state. A thread will also move to the dead state automatically when it reaches the end of its method.

Blocking a Thread

A thread can also be temporarily suspended or blocked from entering into the runnable and running state by using following methods.

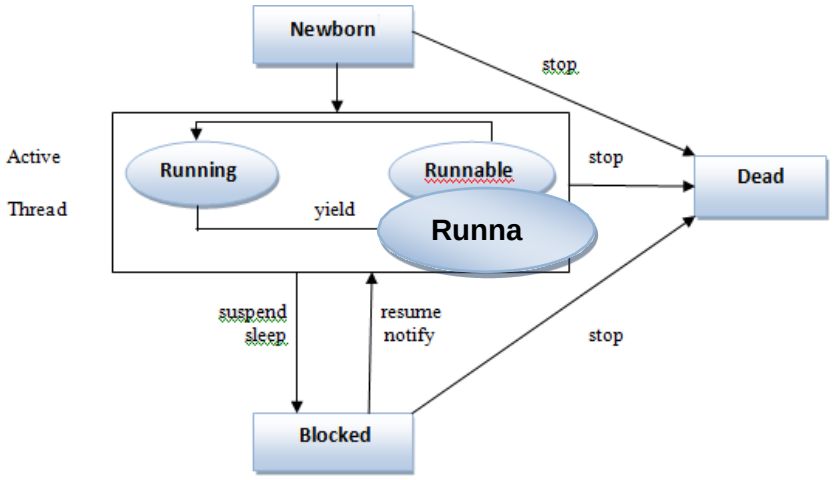
sleep() // blocked for a specified time
suspend() //blocked until further orders
wait() //blocked until certain condition occurs

These methods cause the thread to go into the blocked state. The thread will return to runnable state when specified time is elapsed in the case of sleep().
The resume() method is invoked in the case of suspend(). notify() method is called in the case of wait().

LIFE CYCLE OF A THREAD **[5 Marks]**

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. Newborn
2. Runnable
3. Running
4. Blocked/ Non-Runnable
5. Dead/Terminated



1 Newborn state - :

When we create thread object, the thread is born and it is said to be newborn state.the thread is not yet scheduled for running. At this stage we can do only one of the following things with it

1. Schedule it for running by calling start() method
2. Kill it using stop() method

2 Runnable State - :

The runnable state means that the thread is ready for execution and waiting for the availability of the processor. i.e the thread has joined queue and waiting for execution**.**

3 Running - :

Running means that the processor has given its time to the thread for its execution. At this stage we can do only only one of the following thing with it.

- It has been suspended using suspend() method. A suspended thread can be revived using resume() method.
- It has been made to sleep. We can put thread to sleep for a specified time period using the method sleep(time). This means that the thread is out of the queue during this time period.
- It has been told to wait until same event occurs. This is done using the wait() method. The thread can be scheduled to run again using the notify() method.

4 Blocked State - :

This is the state when the thread is still alive, but is currently not eligible to run.

5 Dead State - :

A thread is in terminated or dead state when its run() method exits.

THREAD PRIORITY **[2 Marks]**

For Executing a Program java Contains a Scheduler which Executes the Programs of java on the behalf of Priorities and because a Process or can Execute only one Thread at a time and the Priority of a Thread will determine which Thread will be Executed now and by default all the Threads have a Same Priorities. But if a Thread has a Higher Priority then this will be Executed First and then after other lower Priorities Thread will be Executed. Generally JAVA Provided us three types of Priorities those are MIN_ Priority , Max-Priority and Normal_ Priority They are used as.

1) MAX_PRIORITY
2) NORM_PRIORITY
3) MAX_PRIORITY

In this Minimum Priority of a thread has value 0 and Normal Priority has value 5 and Maximum Priority has a value 10.

The following example creates a new thread and starts it running:

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

class B extends Thread
{
	public void run()
	{
		System.out.println("Class B starts");
		for(int j=1;j<=5;j++)
		{
			System.out.println("j = "+j);
		}
		System.out.println("Class B exits");
	}
}
```

```
class C extends Thread
{
	public void run()
	{
		System.out.println("Class C starts");
		for(int k=1;k<=5;k++)
		{
			System.out.println("k = "+k);
		}
		System.out.println("Class C exits");
	}
}
class PriorityDemo
{
	public static void main(String args[])
	{
		A a=new A();
		B b=new B();
		C c=new C();
		c.setPriority(Thread.MIN_PRIORITY);
		a.setPriority(Thread.MAX_PRIORITY);
		b.setPriority(Thread.NORM_PRIORITY);
		c.start();
		a.start();
		b.start();
	}
}
```

OUTPUT

```
C:\Program Files\Java\jdk1.7.0\bin>javac PriorityDemo.java
C:\Program Files\Java\jdk1.7.0\bin>java PriorityDemo Class A
starts
i = 1
i = 2
i = 3
i = 4
i = 5
Class A exits
Class C starts
Class B starts
j = 1
j = 2
j = 3
j = 4
j = 5
Class B exits
k = 1
k = 2
k = 3
k = 4
k = 5 Class C exits
```

SYNCHRONIZATION IN JAVA **[5 Marks]**

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method. Synchronized

method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Program to show the thread synchronization by creating threads using runnable interface.

```
class Table
{
        synchronized void printTable(int n)
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println(n*i);
                        try
                        {
```

```
                                Thread.sleep(400);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        }
}
class MyThread1 implements Runnable
{
        Table t;
        MyThread1(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(5);
        }
}
 class MyThread2 implements Runnable
{
        Table t;
        MyThread2(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(100);
        }
}
class Synchronization
{
        public static void main(String args[])
        {
                Table obj = new Table();
                MyThread1 r1=new MyThread1(obj);
                Thread t1=new Thread(r1);

                MyThread2 r2=new MyThread2(obj);
                Thread t2=new Thread(r2);
```

```
                t1.start();
                t2.start();
        }
}
```

OUTPUT

```
C:\Program Files\Java\jdk1.7.0\bin>javac Synchronization.java
C:\Program Files\Java\jdk1.7.0\bin>java Synchronization
5
10
15
20
25
100
200
300
400
500
```

IMPLEMENTING RUNNABLE INTERFACE **[5 Marks]**

- Declare class as implementing the runnable interface
- Implement the run method.
- Create a thread by defining an object that is instantiated from this runnable class as the target of the thread.
- Call threads start() method to run the thread.

```
Class X implements Runnable
{
        Public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.pritnln("i=" +i);
                }
        }
}

Class RunnableDemo
{
        Public static void main(String args[])
        {
                X runnable=new X();
                Thread t=new Thread(runnbale);
                t.start();
        }
}
```

OUTPUT

```
i=1
i=2
i=3
i=4
i=5
```