UNIT 2

Interface

- Interface is a kind of class, except the **class** keyword is replaced with the **interface** keyword. It supports inheritance.
- **A class in Java cannot have a method without definition**. Java supports this by providing a feature called Interface.
- **Specify what a class must do, but not how it does it.**
- Designed to support dynamic method resolution at run time.

```
scope interface <interface Name>
{
    type final Var_Name1 = Value;
    type final Var_Name2 = Value;
    ::::::::::::::::::::::::::;
    returnType MethodName( Arg List);
```

- Scope is public or not used. If the interface is declared as public, than by default all data's and methods declared within interface are public.

- Methods are declared with out definition. Definition must be written within the implemented class. (i.e., an interface has a list of abstract method declarations)
- Any number classes can implement an interface. One class can implement any number of interfaces.
- By default all data members declare within the interface are final and static. So the implemented class can't modify them.
- **Final and static methods are not allowed within an interface.**
- The methods that are implemented from an interface must be declared as **public**.
- We can only declare an object for interface.

  i.e., interfaceName RefName; is possible.

  But, RefName = new interfaceName(); is not possible.

- We can initialize the interface Reference variable by its implemented class.
- Multiple inheritances can be implemented through interface.
- Both interface and abstract class provide a specification of what the subclass must do. But the main difference is that the interface is not allowed to provide any implementation details either in the form of data fields or implemented methods.

Note: Here implementation means providing definition to the methods declared in the interface.

To extend an Interface

```
interface <interface Name> extends <existingInterface>
{
    :::::::::::::::::::    // body of interface.
```

```
interface <interface Name> extends <Interface1>, <Interface2> ….
{
    :::::::::::::::::::    // body of interface.
}
```

To Implement an Interface

```
class <class Name> implements <Interface Name>
{
    :::::::::::::::::::    // body of the class.
```

```
class <class Name> extends <super class> implements
                                    <Interface1>, <Interface2>….
{
```

```
interface Inter1
    {    void show(int value);        }

class Demo implements Inter1
    {  public void show(int V)
        {  System.out.println("Value is :"+ V);        }
    }
// we can also declare additional methods within the class Demo
class TestInter1
  {
```

```
    public static void main(String arg[])
      {     Inter1 I = new Demo();
            I.show(90);
                (OR)
            Demo D = new Demo();
            D.show(90);
      }
 }

interface Inter2
 {
      void show(int value);
 }
class Demo1 implements Inter2
 {
     public void show(int V)
       {  System.out.println("Value is :"+ V);        }
 }
class Demo2 implements Inter2
 {
     public void show(int V)
```

```
        {  System.out.println("Square  is :" + (V*V));    }
  }
class TestInter1
  {       public static void main(String arg[])
             {
                    Inter2  I = new Demo1();
                    I.show(25);
                    Demo2  D = new Demo2();
                    I = D;    // object1 now point to class Demo2.
                    I.show(25);
                    Inter2 d =new Demo2();
                    d.show(25);
          }
   }
```

Output: Value is: 25 Square is: 625 Square is: 625

Note: Each class that includes an interface must implement all the methods defined by the interface. Otherwise the class must be declared as abstract as follows:

```
interface Inter3
  {
      void show(int value);
  }
class Demo1 implements Inter3
   {
      public void show(int V)
        {    System.out.println("Value is :"+ V);       }
   }
abstract class Demo2 implements Inter3
   {
      // Demo2 doesn't define show() method, so it is declared as abstract.
     // Any class that inherits Demo2 must implement show() or be
      declared abstract itself.
   }
class TestInter3
  {
      public static void main(String arg[])
```

```
        {       Inter3 I = new Demo1 ();
                I.show(90);
                    (OR)
                Demo1 d = new Demo1();
                Inter3 I; I = d;     I.show(20);
        }
  }
```

```
interface InterA
 {       void show1( );          }

interface InterB extends InterA
 {      void show2( );          }

class Demo implements InterB
 {
      public void show1( )
        {
             System.out.println("Inside Method show1()");
        }
```

```
    public void show2( )
      {
           System.out.println("Inside Method show2()");
       }
}
```

class TestInter4

```
  {
       public static void main(String arg[])
          {
                Demo D = new Demo ();
                D.show1();   D.show2();
                     (OR)
                InterB IB = new Demo();
                IB.show1();   IB.show2();
          }
  }
```

Try: Design an interface **ColorCodes**, which contains code for various colors (ex: int RED =1; int GREEN =2; etc). Create one

class **Colors** that implements the interface ColorCodes. It contain one method getColr(int) to get the color for a corresponding code. Display the name of the color depends upon the given code.

Note:

Consider a class that implements multiple interfaces with constants defined in them. **There is a possibility that two interfaces can have constants with same name**.

For example the interface **"ColorCodes"** had constants like RED, GREEN, etc. suppose if you have an interface called **"TrafficSignals"** that may also have constants like RED, GREEN, etc.

A **class that implements both these interfaces** will have access to all the constants defined in both interfaces.

When they refer to a constant RED, this constant will be available in both interfaces and **there may be conflict**.

To solve the problem refer the "ColorCodes" RED as **ColorCodes.RED** and the "TrafficSignal" RED as **TrafficSignal.RED**. In this way you can resolve the ambiguity in the constants with same name.

Difference between abstract class and interface

```
abstract class AbsDemo
{    abstract void show();
     void show();
     // Error: missing method body, or declare abstract
     int A;
     abstract void show1(){  }
     // Error: abstract methods can't have a body.
    void display()
       {    A =20;
            System.out.println("Inside Abstract class");
            System.out.println("Value of A is :" + ++A);
        }
}
class Derived extends AbsDemo
{  void show()
       { System.out.println("Inside Derived class");  }
}
```

```
class TestAbs
{   public static void main(String arg[])
    {          Derived D = new Derived();
               D.show();          D.display();
               AbsDemo d = new Derived();
               d.show();          d.display();
    }
}
```

OUTPUT:

```
Inside Derived class
Inside Abstract class
Value of A is: 21
Inside Derived class
Inside Abstract class
Value of A is: 21
```

```
interface InterA
  {       void show1( ) {   }
          int A;
  }
interface InterB extends InterA
  {    void show2( );          }
class Demo implements InterB
  {   public void show1( )
        {  System.out.println("Inside Method show1()");     }
      public void display( )
        { System.out.println("Inside dispaly()");          }
  }
class TestInter
 {       public static void main(String arg[])
            {        Demo D = new Demo ();
                     D.show1();   D.show2();
                     D.display();
                      //  (OR)
                     InterB IB = new Demo();
                     IB.show1();   IB.show2();
```

```
                    IB.display();
    }
}
```

Errors:

1. TestInter.java:3: interface methods cannot have body

   void show1( ) { }

2. TestInter.java:4: = expected

   int A;

3. TestInter.java:11: Demo should be declared abstract; it

   does not define show2() in Demo

   class Demo implements InterB

4. TestInter.java:33: cannot resolve symbol

   symbol : method display ()

   location: interface InterB

   IB.display();

Why IB.display() is not possible?

The interface reference variable can only access the methods implemented from the interface and not of the implemented class methods.

Try: public interface Shape

```
{       void read();
        void FindArea();
        void ShowArea();
}
```

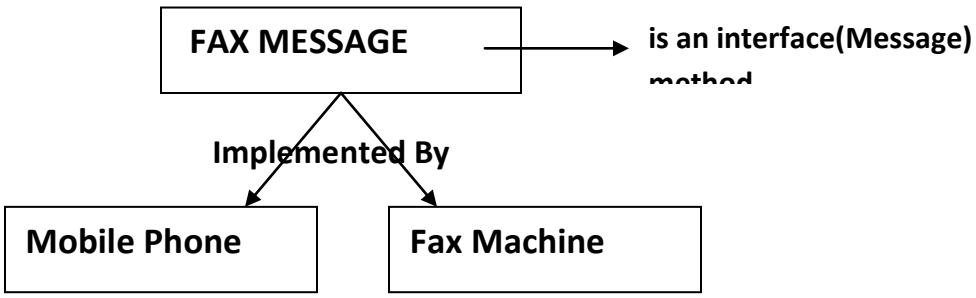
class Rectangle implements Shape { :::::::::::::: }

class Triangle implements Shape { :::::::::::::: }

Example 1:

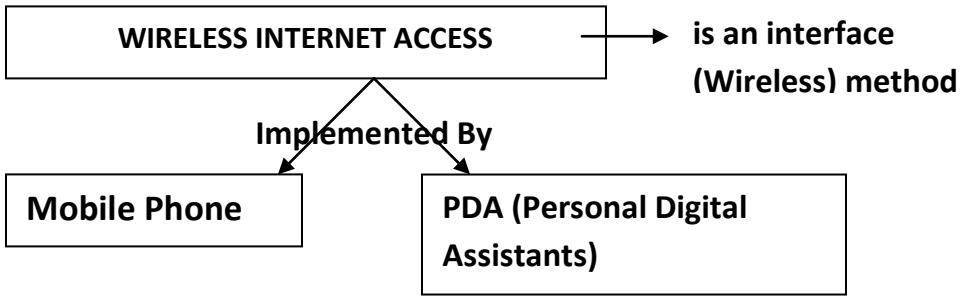
All automobiles have a **fuel tank** and **consume fuel on moving**. **Type of the fuel and the way in which you fill the fuel in its tank are different for each type of the automobile**. A car can have its own way, a truck has its own way and etc. the way in which it is done can be defined only when the specific type of vehicle is defined. In this case, automobile declare the **method "FillFuel"**, but **it can not define the method of in which it is fill.**

Example 2:



Here Mobile phone and Fax Machine have to define the method of implementation. But the Fax machine has its own way and the Mobile phone has its own way to send/receive a fax message.

Example 3:



- Here Mobile phone and PDA devices implements “wireless internet access”, but each one has its own way to access the internet.
- Finally your Mobile phone should define the methods of Fax Interface and Wireless Internet Access interface.

```
class MobileNokia implements FaxMessage,
                             WirelessInternetAccess
{
    ----------------------------
```

Packages

- A package is a group of related classes and interfaces. (Packages are similar to class library in other languages.)
- C and C++ → Header File → include
- Java → package → import
- Packages are otherwise called as class containers. Java provides a powerful means of grouping of related classes and interfaces together in a single unit called package.
- The Java API itself is implemented as a group of packages. All related classes will be put in a package and that will be made available to the programmers who are working in the related field.
- **Advantage:** Code Reusability.
- All built-in classes in java are stored in packages.

Package Declaration:

To create a package, include a "package" command as the first statement in the java source file.

```
package < package name >;

public class < class name >

{

                    // body of the
```

Example:

package mypack; where ‘mypack’ is the package name.

Note: Java uses file system directories to store packages. The directory name must match the package name exactly. A package declared as ‘**package java.awt.image**;’ needs to be stored in the ‘**java\awt\image**’ on your windows.

Import statement

- A package is included in a program using **"import"** statement.
- The import statement enables the user to import classes from other packages into a compilation unit.
- The user can import individual classes (or) entire packages of classes at the same time if desired.

❖ import packageName.*;
- import entire package, where * is the wild card character

❖ import packageName.className;
- import individual class

❖ import packageName.subpackage.*;

❖ **Example:** import mypack.*;

Access Protection:

Accessible From	Public	Private	Protected	Default (friendly)
Same Class	Yes	Yes	Yes	Yes

Same package Sub Class	**Yes**	**No**	**Yes**	**Yes**
Same package Non Sub Class	**Yes**	**No**	**Yes**	**Yes**
Another package Sub Class	**Yes**	**No**	**Yes**	**No**
Another package Non Sub Class	**Yes**	**No**	**No**	**No**

Example:

```
package P1;
    class AAA
        {   private ; public ; protected ; default ;  }
    class BBB extends AAA
        {   public ; protected ;  default ;          }
    class CCC
        {   public ; protected ;  default ;          }

package P2;
import P1.*;
    class DDD extends AAA
      {  public ; protected;  }
    clsss EEE
      {  public  ;}
```

Note: The basic language functions are stored in the 'java.lang' package. The java compiler implicitly imports the java.lang

package for all programs. So 'import java.lang.*;' being at the top of all your programs.

```
package Pack;

public class Test
  {
    public void show()
       {  System.out.println(" Inside  Package Show()"); }
  }
```

Save: Test.java under Pack directory

```
import Pack.*;

class TestPack
 {
   public static void mian (String arg[ ])
       {    Test T = new Test();
            T.show();
       }
 }
```

Program ***Part (1)***

```
package mypack.general;
public class MyUtil
  {    public static int MAX(int A, int B)
        {
             System.out.println("Inside MyUtil Class");
                  if ( A > B) return A;
             return B;
        }
       public static int MAX(int A, int B, int C)
        {    return MAX(A,MAX(B,C));    }
 }
```

***Save*:** d:/mypack/general/MyUtil.java

Part (2)

```
package mypack.type;
public class Distance
 {
       int feet; double inch;
```

```
    public Distance()
        {   System.out.println("Inside Distance Clsss");
            feet = 0; inch = 0.0;
        }
    public Distance (int f, double i)
      {        feet = f;  inch = i ;              }
    public String toString()
     { return ("Feet:" + feet +"Inch:" + inch);  }
} Save: D:/mypack/type/Distance.java
```

Part (3)

```
import mypack.general.*;
import mypack.type.*;
class TestPack1
  {   public static void main(String arg[ ])
      {     int A = 30,B =40, C =45;
            System.out.println("Big of  A & B is:"+MyUtil.MAX(A,B));
            System.out.println("Big of  A ,B & C is:"+MyUtil.MAX(A,B,C));
            Distance  D1= new Distance ();
            System.out.println("Default Constructor value is :" +D1);
```

```
        Distance  D2 = new Distance (4,7);

        System.out.println("Two Arg Constructor value is :" +D2);

    }

} Save: D:/TestPack1.java
```

Compiling and Running: E:\cd mypack

E:\mypack>cd general

E:\mypack\general>javac MyUtil.java

E:\mypack>cd type

E:\mypack\type>javac Distance.java

E:\javac TestPack1.java

E:\java TestPack1

TRY:

1. Create a Package called **"LIBRARY"**, which contains one class **"BOOK"** to maintain all the book information. Create two classes **"STAFF"** (to maintain the staff information) and **"STUDENT"** (to maintain the student information) from the super class "BOOK". Import the package "LIBRARY" and create object for the classes STAFF and STUDENT and call the corresponding methods.

2.
```
package OS.License;
public class Windows
    {  // details of Windows OS          }
```

```
package OS.Opensource;
public class Linux
```

```
        {  // details of Linux OS                }


import OS.License.*;
import OS.Opensource.*;
class TestOS
 {
    public static void main(String arg[])
      { // Process the OS information   }
  }
```

Packages are used to avoid naming conflicts:

```
package PK1;
public class Student
 {   public int A;
     public void show()
       {  System.out.println("Inside 'PK1' show method");   }
     static public void display()
       {  System.out.println("Inside 'PK1' display");        }
 }
package PK2;
public class Student
 {   public static int A;
     public void show()
       { System.out.println("Inside 'PK2' package");  }
 }
import PK1.*;
import PK2.*;
class TestPack
{   public static void main(String arg[])
      {
```

```
        PK1.Student S = new PK1.Student();
        S.show();
        S.A=100;
        System.out.println("Value of A is: "+S.A);
      PK1.Student .display();
      PK2.Student K = new PK2.Student();
        K.show();
      PK2.Student.A=100;
        System.out.println("Value of A is:"+PK2.Student.A);
    }
}
```

Hence the three main purposes of packages are:

1. Reduce problems in name conflicts.
2. Control the visibility of classes, interfaces, methods and data defined within them.
3. Code reusability.

Note:

- Java implements encapsulation by the use of Packages.
- Only one package statement is allowed in a program.
- A java file can't contain more than one public class. If a java file contains more than one class, then the file name is same as the name of the public class.

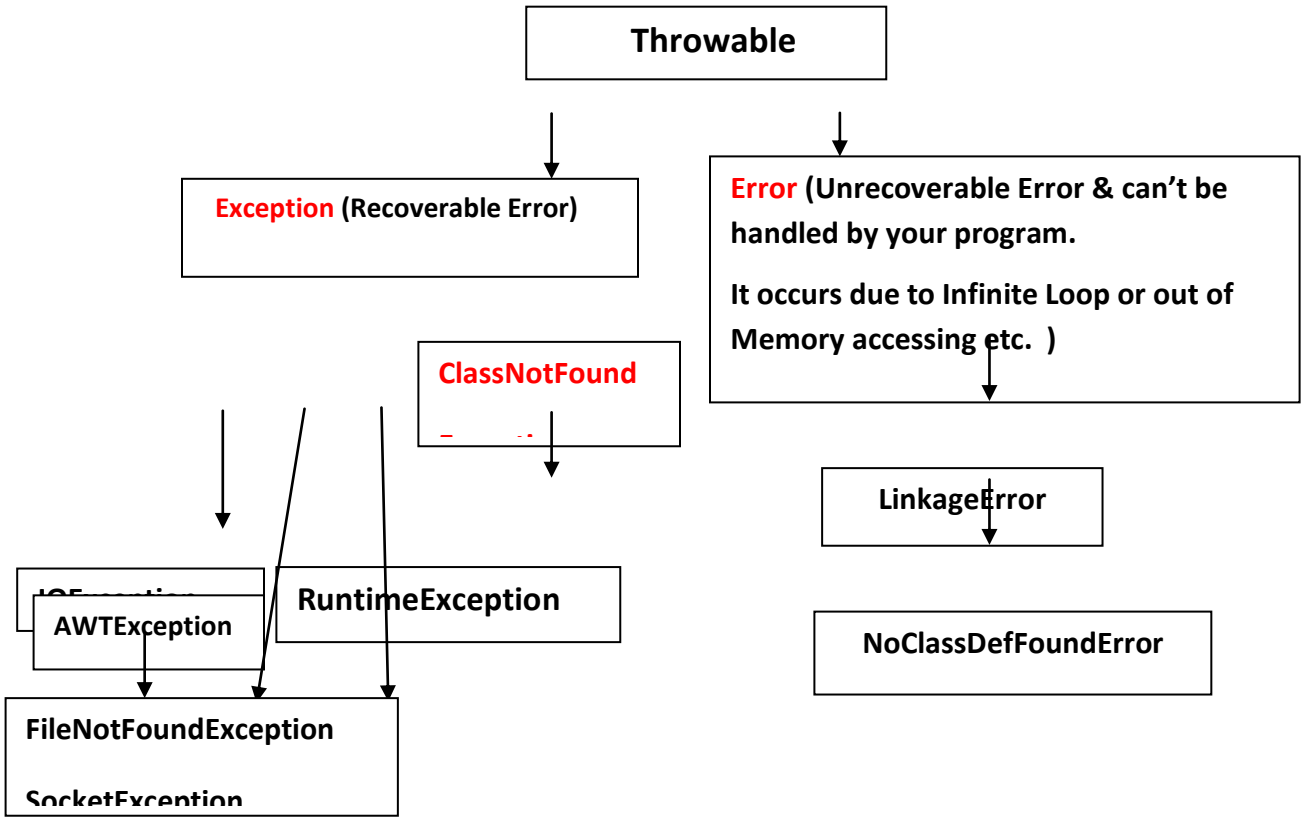
Exception Handling

Exception is an abnormal condition that arises in a code sequence at the runtime. Exceptions are otherwise called as Runtime errors.

Compile time error - due to syntax error

Runtime error - due to logical error

Hierarchy of Exception classes:



NumberFormatException - Invalid conversion of string to a numeric format

InterruptedException – One thread has been interrupted by another thread

ArithmeticException – Arithmetic error, such as divide –by– zero

IllegalArgumentException – Illegal argument used to invoke a method

NegativeArraySizeException – Array created with a negative size

ArraryIndexOutOfBoundsException – Array index out of bounds

StringIndexOutOfBoundsException – String index out of bounds

ArrayStoreException – Assignment to an array element of an incompatible type

IllegalThreadStateException– Requested operation not compatible with current thread state.

NullPointerException – Invalid use of a null reference

NoSuchFieldException – A requested field does not exist

General Format:

```
try
 {
      ::::::::::
 }
catch (ExceptionType1 object )
```

- **Key words:** try, catch, throw, throws and finally.
- Program statements that you want to monitor for exceptions are contained within the try block. If any exception arises within the try block, then the corresponding catch block will be executed.
- System generated exceptions are automatically thrown by Java Runtime System.
- To throw a manual exception use the keyword **"throw"**.
- Any exception thrown out of the method must be specified by the keyword **"throws"**.
- A try block contain more than one catch block.
- A code within the finally block will be always executed.

getMessage () method: Return a description about the Exception and it is declared within the "**Throwable"** class.

Note: Any Exception that is not catch by your program will ultimately processed by the default exception handler, it display a string representation of the exception and terminates the program.

Finally Block:

- The code within the finally block will be executed after a try/catch block has completed and before the code following the try/catch block.
- The finally block is optional.
- This can be useful for closing file handlers and freeing up any other resources that might have been allocated at the beginning of the method.
- Each try block requires at least one catch block or a finally block.

Uses: Allows you to fix the errors - prevents the program from automatically terminating

Note: Once an exception is thrown, the control transfer to the catch block. After executing the catch block statements, the control transfer to the next line followed by the try/catch block.

```
class Exce1
 {  public static void main(String as[])
```

```
{       int a=0;       int res;
        try
        {       System.out.println("Inside try block ");
                 res =20/a;
                System.out.println("Res is:"+res);
        }
      catch(ArithmeticException AE)
       {  System.out.println(AE.getMessage());                    }
      catch(Exception E)
         { System.out.println("Inside Exce1 class: "+ E);          }
      finally
        { System.out.println("Finally  block always executed..."); }
      System.out.println("Main function execution completed...");
  }
} OUTPUT:      Inside try block
                / By zero
                Finally block always executed...
                Main function execution completed...
```

Note: When you use multiple catch statements, it is important to remember that exception subclasses are come before any of their

super classes. Otherwise unreachable code is an error. Since a super class will catch exceptions of that type plus any of its subclasses.

"throw" statement:

By using the 'throw' statement we can throw an exception explicitly. 'throw' statement is used to raise user defined exceptions.

```
throw ThrowableInstance;
```

Where, 'ThrowableInstance' must be an object of type Throwable or a subclass of Throwable.

Note: All built-in Runtime Exception classes have two constructors. One with No Argument and another one with One Argument of type String.

```
class  Exce5
  {   public static void main(String arf[])
       { try
           {   throw (new InterruptedException("Exce5"));       }
        catch(InterruptedException e)
           {   System.out.println("throw demo");
```

```
            System.out.println("caught: "+ e);
        }
    }
}
```

"throws":

- A throws clause lists the type of exceptions that a method might throw.
- We should add the 'throws' keyword after the signature of the method.

```
returnType MethodName (Arg List) throws ExceptionType
  {
        …… // body of the method
```

Creating user defined exception subclasses:

The user can also create own exception types to handle situations specific to their applications. To do this, just define a subclass of Exception.

```
class MoneyException extends Exception
{       MoneyException()
```

```
        {     super();      }
    MoneyException(String s)
        {     super(s);     }
}

class money
{   int rs,ps;
    money()
     { rs=0; ps=0;  }
    money(int r,int p) throws MoneyException,ArithmeticException
     {    if(p>100)
              throw new MoneyException("ps value too high");
          rs=r;
          ps=p;
     }
    public String toString()
     {  return ("Rs: "+rs+" Ps: "+ps) ;  }
}
class TestMyExce1
{   public static void main(String arg[])
```

```
    {   money m1=new money();
        money m2=new money();
        try
         { m1 = new money(10,20);      }
        catch(MoneyException ae)
         {    System.out.println("Error:" +ae.getMessage());     }
        System.out.println("Object one is:\t" + m1);


        try
         {  m2 = new money(12,120);    }
        catch(MoneyException ae)
         {   System.out.println("Error:" +ae.getMessage());    }
        System.out.println("Object two is:\t" + m2);
    }
}
```

OUTPUT:

```
Object one is     :  Rs: 10 Ps: 20
Error             : ps value too high
Object two is     :  Rs: 0 Ps: 0
```

Program:

1. **To get the student roll number and raise an exception "InvalidRollNoNumberException", if the roll number is not valid. Sample roll number format: 16BCS201**
2. **To get the employee number and raise an exception "InvalidEmpNoNumberException", if the employee number is not valid. Sample employee number format: 17MCS201**

Multithreading

Uses:

- Allows running multiple tasks of a process concurrently. Each task or job is called as a "thread".
- Reduce the ideal time of the CPU.

Two types of multitasking:

1. Process based: (O/S Level)

- Process based multitasking allows your computer to run two or more programs concurrently.
- Example: Windows (running the Java compiler, Text editor, etc. at the same time)

2. Thread Based: (Application Level)

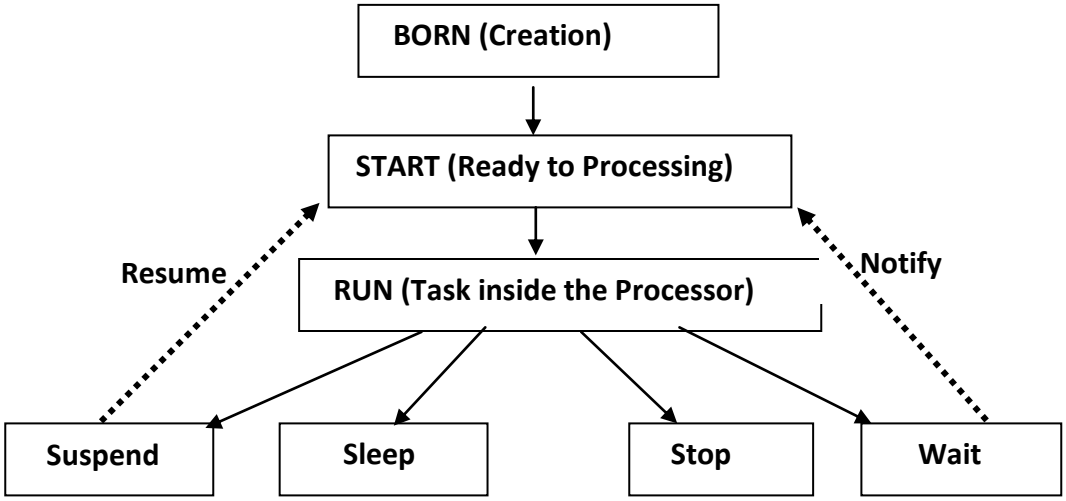
- Thread based multitasking allows you to execute multiple task simultaneously. Thread based multitasking is called as multithreading.

- Example: Ms-Word (carry out typing, auto save, background printing etc. at the same time)

Processed Based	Thread Based
Operating system level multitasking	Application level multitasking
Heavy Weight Task	Low Weight Task
Each task have separate address space	Each task have same address space
Switch to one task to another is very difficult.	Switch to one task to another is very easy.

Note: To create a thread, your program will either extend "Thread" class (Or) implements "Runnable" interface.

Life Cycle of a Thread



- **start()**
  - Start a thread by calling its run method.
  - General Format: **public synchronized void start();**
  - Example: Thread T = new Thread (this); T.start ();

- **run()**
  - Entry point for the thread (task enter to the processor). Run method can call all other methods.
  - General Format: **public void run()**

```
{
    ::::::::::::// set of statements
}
```

- **stop()**
  - Terminate the execution of a thread. At any time a thread can be terminated. Once terminated, a thread can't be resumed.
  - General Format: **public final void stop();**

- **sleep()**
  - Suspend a thread for a period of time (ms). This method may throw an Interrupted Exception.

static void sleep(long milliseconds) throws InterruptedException

static void sleep(long ms, int nanoseconds) throws InterruptedException

  - There is no command from sleep to start. If the specified sleep time is finish, then it is automatically go to start.

- **getName():** Return the name of the thread.
  General Format: **final String getName();**

- **setName():** Set the name of the thread.
  final void setName(String ThdName);

- **currentThread()**
  - Returns a reference to a thread in which it is called.
  - General Format: **static Thread currentThread();**

- **isAlive()**
  - This function return true, if the thread is still running. Otherwise return false.
  - General Format: **final boolean isAlive();**

- **join()**
  - Waits until the thread on which it is called terminates.
  - G . F: **final void join() throws InterruptedException**

```
class Thd1
{       public static void main(String arg[])
        {       Thread T=Thread.currentThread();
                System.out.println("Current Thread is:"+ T);
                System.out.println("Thread Name is:"+T.getName());
                T.setName("My Thread");
                System.out.println("New Thread Name is:"+
                                                        T.getName());
                Thread T2 = new Thread("One Arg Constructor");
                System.out.println("Current Thread is:"+ T2);
        }
}
```

Output

```
Current Thread is: THREAD [main, 5, MAIN]
THREAD NAME IS: MAIN
```

NEW THREAD NAME IS: MY THREAD

CURRENT THREAD IS: THREAD [ONE ARG CONSTRUCTOR, 5, MAIN]

```
class MyThread extends Thread
  {    MyThread()
        {    super("MyThread");
             System.out.println("Child thread..:"+this);
             start();
        }
       public void run()
        {    try
             {    for(int i=4;i>0;i--)
                       {    System.out.println("Child Thd:"+i);
                            Thread.sleep(1000);
                       }
             }catch(InterruptedException e)
             { System.out.println("Child Interrupted...");  }
```

```
            System.out.println("Child thread exists...");
        }
    }
class Thd3
  {  public static void main(String arg[])
      {       new MyThread();
         try
          { for(int i=4;i>0;i--)
              {
               System.out.println("Main Thd:"+ i);
               Thread.sleep(100);
              }
           }catch(InterruptedException e)     {          }
         System.out.println("Main thread exists...");
      }
  }
```

OUTPUT:

Child thread..: Thread[MyThread,5,main]

Main Thd:4

Main Thd:3

Main Thd:2

Main Thd:1

Main thread exists...

Note: Draw back of the above program**:** the main thread is terminated before the child threads completion.

Creating multiple threads

```
public void run()
  {    try
       {     for(int i=4;i>0;i--)
                {  System.out.println("Child Thd:"+name +i);
                   Thread.sleep(500);
                }
        }
       catch(InterruptedException e)
        { System.out.println("Child Interrupted...");  }
       System.out.println("Child thd exist...");
 }
```

```
class Thd4
{  public static void main(String arg[])
   {  new MyThread("One");
      new MyThread("Two");
      new MyThread("Three");
      try
         {for(int i=4;i>0;i--)
           {          System.out.println("Main Thd:"+ i);
                      Thread.sleep(1000);
            }
         }catch(InterruptedException e)     {          }
       System.out.println("Main thd exist...");
  }
}
```

```
Child thread..: Thread [One, 5, main]
Child thread..: Thread [Two, 5, main]
Child thread..: Thread [Three, 5, main]
Main Thd:  4
Child Thd: One4        Child Thd: Two4        Child Thd: Three4
Child Thd: One3        Child Thd: Two3        Child Thd: Three3
Main Thd:3
```

Note: In the above program the main thread's sleep time is greater than the child thread's sleep time. So that it ends after the completion of the child threads execution.

For thread priority, join () and isAlive ()

```
class MyThread implements Runnable
{    String name;    Thread t;
     MyThread(String name)
      {   this.name=name;
          t=new Thread(this, name);
          System.out.println("Child thread..:"+t);
          t.start();
      }
     public void run()
      { try
```

```
        {  for(int i=5;i>0;i--)
            {        System.out.println("Child Thd "+name +  i);
                     Thread.sleep(1000);
            }
        }catch(InterruptedException e)
           {System.out.println("Child Interrupted...");  }
        System.out.println("Child thread "+ name + " : exist...");
     }
 }
class Thd5
{  public static void main(String arg[])
    {  MyThread m1=new MyThread("One");
       MyThread m2=new MyThread("Two");
       MyThread m3=new MyThread("Three");
       m1.t.setPriority(Thread.MIN_PRIORITY+2);
       m2.t.setPriority(m1.t.getPriority()+2);
       m3.t.setPriority(Thread.MAX_PRIORITY);
       System.out.println("Thd m1 alive is :"+m1.t.isAlive());
       try
          {  System.out.println("Waiting for child threads completion");
```

```
            m1.t.join();
            m2.t.join();
            m3.t.join();
          }catch(InterruptedException e){          }
        System.out.println("Thd m1 is alive:"+m1.t.isAlive());
        System.out.println("Main thread exists...");
    }
}
```

Note: In the above program the join() method is used to stop the termination of the main thread until the child threads completion.

OUTPUT:

```
Child thread..:Thread[One,5,main]
Child thread..:Thread[Two,5,main]
Child thread..:Thread[Three,5,main]
    Child Thd Three5
Thd m1 alive is :true
Waiting for chile threads to finish...
Child Thd Two5      Child Thd One5
Child Thd Three4    Child Thd Two4      Child Thd One4
Child Thd Three3    Child Thd Two3      Child Thd One3
```

Thread Priority

setPrioity ()

- Set priority to a thread. The higher priority threads are executed always first.
- General format: **final void setPriority (int level);**
- Where ***level*** specify the new priority for the calling thread.
  - **public static final int MIN_PRIORITY** (1)
  - **public static final int NORM_PRIORITY** (5)
  - **public static final int MAX_PRIORITY** (10)
- The value of ***level*** must between MIN_PRIORITY (1) to MAX_PRIORITY (10).

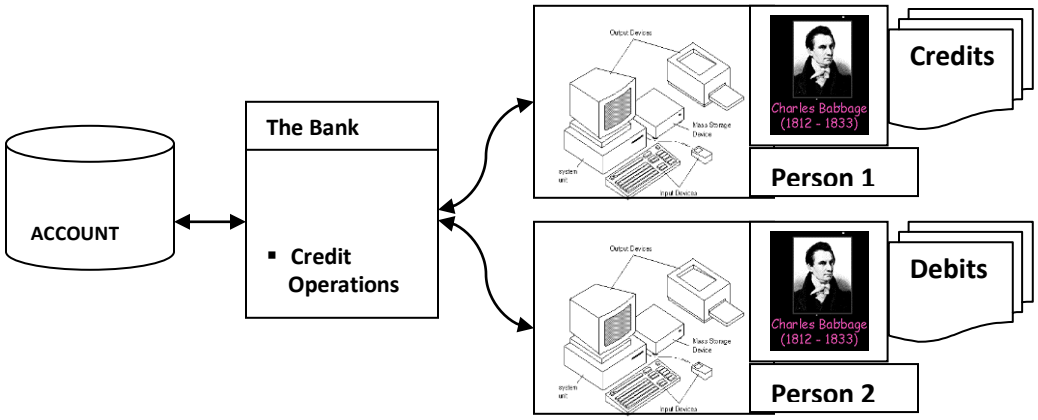
getPriority ()

- Return the priority of the thread.
- General Format: **final int getPriority ();**

Thread Synchronization

When two or more threads need to access shared resources, they need some way to gives the resource. i.e., only one thread can access the resource at a time. The process by which this is achieved is called synchronization.

When we need synchronization?



We can synchronize your code by two ways:

1. Synchronized the code at the method level – this involves synchronizing methods.
2. Synchronized the code at the block level – this involves synchronizing blocks.

1. Synchronize method: Method declaration has been modified with the synchronized keyword. When a method is declared as synchronized, then only one thread can access the method at a time.

```
synchronized ReturnType MethodName (Arg List)
 {
        …… // body of the synchronized method
```

2. Synchronized Statement (or) Block:

The need of synchronized block,

- ☺ You have one source code, that was written by a third party and the code does not have any synchronized methods.
- ☺ You do not have authority to modify the source code and you have the authority to create object for the class and call the methods.
- ☺ Suppose you want to access the code in a synchronized way. Is it achievable? Not achievable. The reason you can't add synchronized to the appropriate methods within the class.
- ☺ The solution to this problem is very simple: You simply put the object inside a synchronized block and access the code in a synchronized way.

```
synchronized (SyncObject)
 {
        :::::::::::::::: // body of the synchronized block
```

- Where “SyncObject” is a reference to the object being synchronized. If we want to synchronize a single statement, then the curly braces are not needed.

Thread Communication

wait( )

- To control one thread form another thread.
- Tells the calling thread to go to sleep, until call notify () method.
- **final void wait ( ) throws InterruptedException**

notify( )

- Wakeup the first thread that called wait on the same object.
- General format: **final void notify ();**

suspend( )

- To pause the thread that call the suspend method.
- General format: **final void suspend ( );**

resume( )

- Restart the execution of a thread that called the suspend method.
- General format: **final void resume ( );**

Thread priority, join() and suspend() and resume()

```
class MyThread implements Runnable
{       String name;    Thread t;
        MyThread(String name)
          {    this.name=name;
               t=new Thread(this,name);
               System.out.println("Child thread..:"+t);
               t.start();
           }
        public void run()
         {  try
              {   for(int i=5;i>0;i--)
                   {
                      System.out.println("Child Thread "+name +  i);
                      Thread.sleep(1000);
```

```
                }
            }catch(InterruptedException e)
              {System.out.println("Child Interrupted...");  }
          System.out.println("Child thread "+ name + " : exist...");
        }
  }

class Thd7
{
public static void main(String arg[])
   {
      MyThread m1=new MyThread("One");
      MyThread m2=new MyThread("Two");
      MyThread m3=new MyThread("Three");
      m1.t.setPriority(Thread.MIN_PRIORITY+2);
      m2.t.setPriority(m1.t.getPriority()+2);
      m3.t.setPriority(Thread.MAX_PRIORITY);

      try{   m1.t.suspend();
             System.out.println("suspend thread m1");
```

```
            Thread.sleep(5000);
            m1.t.resume();
            System.out.println("Resume thread m1");
        }catch(Exception e) {      }

    try
      {   System.out.println("Waiting for chile threads to finish...");
            m1.t.join();
            m2.t.join();
            m3.t.join();
        }catch(InterruptedException e) {          }
    System.out.println("Main thread exists...");
  }
}
```

OUTPUT:

```
Child thread..:Thread[One,5,main]
Child thread..:Thread[Two,5,main]
Child thread..:Thread[Three,5,main]
Child Thread Three5
suspend thread m1
Child Thread Two5
```

Child Thread Three4

Child Thread Two4

Child Thread Three3

Child Thread Two3

Child Thread Three2

Child Thread Two2

Child Thread Three1

Child Thread Two1

Child thread Three : exist...

Resume thread m1

Waiting for chile threads to finish...

Child thread Two : exist...

Child Thread One5

Child Thread One4

Child Thread One3

Child Thread One2

Child Thread One1

Child thread One : exist...

Main thread exists...

For synchronized method

```
class Base
{    synchronized static void show(String str)
        {
            System.out.print('[' + str);
            try
                { Thread.sleep(1000); }
            catch(Exception e) {    }
            System.out.println(']');
        }
}
class Sync extends Thread
{       String str;
        Sync(String s)
         {    super("Sync Thread..");
              str=s;
              start();
         }
        public void run()
```

```
        {   Base.show(str);   }
}
class TestSync
{   public static void main(String arg[])
      {    Sync a,b,c;
           a=new Sync("Java is a pure opps language...");
           b=new Sync("Java support multithreading...");
           c=new Sync("Java support UNICODE character set...");
           try{ a.join(); b.join(); c.join();   }
           catch(Exception e){    }
           System.out.println("Main thread exists...");
      }
}
```

OUTPUT:

```
[Java is a pure opps language...]
[Java support multithreading...]
[Java support UNICODE character set...]
Main thread exists...
```

OUTPUT with out Synchronized:

[Java support multithreading...[Java support UNICODE character set...[Java is a

pure opps language...]

]

]

Main thread exists...

Program for synchronized block

```
class Base
 {  public void show(String str)
       {   System.out.print('[' + str);
           try { Thread.sleep(100); }
           catch(Exception e) {    }
           System.out.println(']');
      }
  }
class Sync extends Thread
  {      String str;
         Sync(String s)
```

```
        {     str=s;
              start();
        }
      public void run()
        {     Base B=new Base();
              synchronized(B)    // synchronized block
                  { B.show(str); }
        }
  }
class TestSync1
 {  public static void main(String arg[])
      {       Sync a=new Sync("Welcome to ...");
              Sync b=new Sync("Dr.MCET...");
              Sync c=new Sync("By CSE…");
              try  { a.join(); b.join(); c.join();  }
              catch(Exception e){    }
              System.out.println("Main thd exist...");
     }
}
```

Thread Synchronization

When two or more threads need to access shared resources, they need some way to access the resource [only one thread can access at a time].

"monitor": an Object - used as a mutually exclusive lock (MUTEX) - Only one thread can won a monitor at a time and that thread can access the shared resource. All other thread attempting to enter the locked monitor will be suspended until the first thread exist the monitor.

Two ways to synchronize the code:

3. **Synchronized the code at the method level – synchronizing methods.**
4. **Synchronized the code at the block level – synchronizing blocks.**

3. Synchronize method: Method declaration has been modified with the synchronized keyword.

```
synchronized ReturnType MethodName (Arg List)
 {
        :::::::::::::::: // body of the synchronized method
```

4. Synchronized Statement (or) Block:

The need of synchronized block,

- ☺ You have a source code, that was written by a third party and the code does not have any synchronized methods.
- ☺ You do not have authority to modify the source code and you have the authority to create object for the class and to call the methods.
- ☺ Suppose you want to access the code in a synchronized way. Is it achievable? Not achievable. The reason you can't add synchronized to the appropriate methods within the class.

☺ Solution: simply put the object inside a synchronized block and access the code in a synchronized way.

```
synchronized (SyncObject)
  {
        :::::::::::::: // body of the synchronized block
```

- **"SyncObject" - reference to the object being synchronized**

Thread Communication

- **final void wait ( ) throws InterruptedException**
- Control the calling thread from another thread.
- Tells the calling thread to go to sleep, until call notify().

- **final void notify ();**
- Control the calling thread from another thread.
- Wakeup the first thread that called wait on the same object.

- **notifyall() –** wakeup all the threads in waiting state - highest priority thread will run first.

- **final void suspend ( );**
- Control the thread itself.
- To pause the thread that call the suspend method.

- **final void resume ( );**
- Control the thread itself.
- Restart the execution of a thread that called the suspend().

Points to keep in mind

- A multithreaded program contains two or more parts that can run currently. Each part of such a program is called a thread and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

- It introduces an asynchronous behavior to your program.
- Multitasking threads require less overhead than multitasking processes.
- Processes are heavy weight tasks - require their own address spaces. IPC is expensive and limited. Context switching from one process to another is also costly.
- Threads are lightweight - share the same address space - cooperatively share the same heavy weight process. Interthread communication is inexpensive. Context switching from one thread to the other is low cost.

- **Multithreading enables you to use of the CPU maximum.**
  a. Important for the interactive, networked environment - because idle time is common.

b. Used when the transmission rate of data over a network is slower than the transmission rate of the computer can process it.
c. Used when the local file system resources are read and written at a slower pace than they can be processed by the CPU.
d. Used when the input is much slower than the computer.

Thread Priorities:

- Thread priorities are integers that specify the relative priority of on that to another.
- A thread's priority is used to decide whether to switch from one running thread to next. This is called a **Context switch.**
- Rules that determine when a context switch takes place:

(1) A thread can voluntarily relinquish control:

This is done by explicitly yielding, sleeping or blocking on pending I/O. In this scenario, all other threads are examined and highest priority thread that is ready to run is given to CPU.

(2) A thread can be preempted by a higher priority thread:

In this case, a lower priority thread can be preempted (no matter what it is doing) by a highest priority thread. This is called **Preemptive multitasking.**

Caution: Problems can arise during context switch, when threads have equal priority.

Synchronization:

- Because multithreading introduces an asynchronous behavior to your programs, there must be a way for you to enforce synchronicity when you need it.

- For example, if you want two threads to communicate and share a complicated data structure, such as a linked list, you need some way to ensure that they don't conflict with each other. That is you must prevent one thread from writing data while another thread is in the middle of reading it.
- For this purpose, java implements an elegant model of Interprocess synchronization: the *monitor/semaphore*.
- You can think of a monitor as a very small box that can hold only one thread. Once a thread enters a monitor, all other threads must wait until that thread exists the monitor.
- There is no class "Monitor", instead each object has its own implicit monitor that is automatically entered when one of the object's synchronized methods is called. Once a thread is inside a

synchronized method, no other thread can call any other synchronized method on the same object.

The main Thread

- **All the programs have used a single thread of execution.**
- **When a java program starts up one thread begins running immediately. This is usually called the main thread of your program, because it is the one that is executed when your program begins.**
- **The main thread is created automatically when your program is started, it can be controlled through a Thread Object. To obtain a reference to it, call the currentThread() method.**
- **Once you have a reference to the main thread, you can control it just like any other thread.**

The main thread is important for two reasons:

(1) It is the thread from which other "child" threads will be generated.

(2) It must be the last thread to finish execution. When the main thread stops, your program terminates.

Inter Thread Communication

Polling: Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the

condition is true, appropriate action is taken. This wastes CPU time.

- **For example, consider the classic queuing problem, where one thread is producing some data and another one is consuming it. To make the problem more interesting, suppose that the producer has to wait until the consumer is finished before it generates more data.**
- **In a polling system, the consumer would waste many CPU cycles while it waited for the producer to produce.**
- **Once the producer was finished, it would start polling, wasting more cycles, waiting for the consumer to finish and so on. Clearly this situation is undesirable.**
- **To avoid polling, java includes an elegant Inter Process Communication mechanism via the wait(), notify() and notifyAll() methods.**

The Thread class:

```
public class java.lang.Thread extends java.lang.Object implements java.lang.Runnable
{
   java.lang.ThreadLocal.ThreadLocalMap threadLocals;
   java.lang.ThreadLocal.ThreadLocalMap inheritableThreadLocals;
   public static final int MIN_PRIORITY;
   public static final int NORM_PRIORITY;
   public static final int MAX_PRIORITY;
   public static native java.lang.Thread currentThread();
   public static native void yield();
   public static native void sleep(long) throws java.lang.InterruptedException;
   public static void sleep(long, int) throws java.lang.InterruptedException;
   public java.lang.Thread();
   public java.lang.Thread(java.lang.Runnable);
   public java.lang.Thread(java.lang.ThreadGroup,java.lang.Runnable);
   public java.lang.Thread(java.lang.String);
   public java.lang.Thread(java.lang.ThreadGroup,java.lang.String);
   public java.lang.Thread(java.lang.Runnable,java.lang.String);
```

```
public java.lang.Thread(java.lang.ThreadGroup,
                                    java.lang.Runnable,java.lang.String);
public java.lang.Thread(java.lang.ThreadGroup,
                                    java.lang.Runnable,java.lang.String,long);
   public synchronized native void start();
   public void run();
   public final void stop();
   public final synchronized void stop(java.lang.Throwable);
   public void interrupt();
   public static boolean interrupted();
   public boolean isInterrupted();
   public void destroy();
   public final native boolean isAlive();
   public final void suspend();
   public final void resume();
   public final void setPriority(int);
   public final int getPriority();
   public final void setName(java.lang.String);
   public final java.lang.String getName();
   public final java.lang.ThreadGroup getThreadGroup();
   public static int activeCount();
```

```
    public static int enumerate(java.lang.Thread[]);
    public native int countStackFrames();
    public final synchronized void join(long) throws
                                                java.lang.InterruptedException;
    public final synchronized void join(long, int) throws
                                                    java.lang.InterruptedException;
    public final void join() throws java.lang.InterruptedException;
    public static void dumpStack();
    public final void setDaemon(boolean);
    public final boolean isDaemon();
    public final void checkAccess();
    public java.lang.String toString();
    public java.lang.ClassLoader getContextClassLoader();
    public void setContextClassLoader(java.lang.ClassLoader);
    public static native boolean holdsLock(java.lang.Object);
    static {};
}
```

The Runnable interface:

```
public interface java.lang.Runnable      /* ACC_SUPER bit NOT set */
```

```
{
    public abstract void run();
}
```