Day - 27:

Interfaces:

1. *Interfaces* are basically **used to develop user defined data types**.
2. With respect to *interfaces* we can **achieve the concept of multiple inheritances**.
3. With *interfaces* we can **achieve the concept of** *polymorphism*, *dynamic binding* and hence we can **improve the performance of a JAVA program** in turns of memory space and execution time.

An interface is a construct which contains the collection of purely undefined methods or an interface is a collection of purely abstract methods.

Syntax for defining an **interface:**

```
Interface <interface name>
{
	Variable declaration;
	Method declaration;
}
```

Interface is a **keyword** which is **used for developing user defined data types**. **Interface name** represent a JAVA valid variable name and it is **treated as name of the interface**. With respect to *interface* we **cannot create an object directly but we can create indirectly**.

Variable declaration represents the **type of data members which we use a part of interface**. Whatever the *variables* we use in the *interface* **are meant for general purpose** (the variables like PI, e, etc.).

Whatever the *variables* we write in the *interface*, they are **by default belongs to**:
`public static final xxx data members`
`xxx` represents data type, variable name and variable value.

All *variables* **must be initialized** (**otherwise it will be compilation error**). Method declaration represents the **type of methods we use as a part of interface**. All the **methods of interface** are **undefined methods** and to **make those methods as abstract, the JAVA programmer need not to write a keyword abstract explicitly** before the *declaration* of *interface method.*

Since all the **methods of interface are meant for general purpose hence they must give universal access**. **To make it as universal access** the JAVA programmer **need not to write a keyword public explicitly** before the *method declaration*. Hence, by **default all the methods of interfaces belong to public abstract methods**.

For example:

```
Interface i1
{
	Int a; //invalid since, initializing is mandatory
	Int b=20;
	Void f1 ();
	Void f2 ();
}
```

Day - 28:

NOTE:

Whenever we compile an interface, we get `<interface name>.class` as an intermediate file, if no errors are present in interface.

Syntax-**1** for reusing the features of **interface(s) to class**:

```
[abstract] class <clsname> implements <intf 1>,<intf 2>.........<intf n>
{
      variable declaration;
      method definition or declaration;
};
```

In the above syntax clsname represents name of the class which is inheriting the features from 'n' number of interfaces. 'Implements' is a keyword which is used to inherit the features of interface(s) to a derived class.

NOTE:

When we inherit 'n' number of abstract methods from 'n' number of interfaces to the derived class, if the derived class provides definition for all 'n' number of abstract methods then the derived class is known as concrete class. If the derived class is not providing definition for at least one abstract class and it must be made as abstract by using a keyword abstract.

i. One class can extend only one class.
ii. One class can implement 'n' number of interfaces.
iii. One interface can extends more than one interface.
iv. Interface cannot implements or extends a class. Since defined things cannot be made as undefined things.

Syntax-**2** inheriting **'n' number of interfaces to another interface**:

```
interface <intf 0 name> extends <intf 1>,<intf 2>.........<intf n>
{
      variable declaration cum initialization;
      method declaration;
};
```

For example:

```
interface I1
{
      int a=10;
      void f1 ();
};
interface I2 extends I1
{
      int b=20;
      void f2 ();
};
```

If one interface is taking the features of another interface then that inheritance is known as interface inheritance

Syntax-**3**:

```
[abstract] class <derived class name> extends <base class name>
      implements <intf 1>,<intf 2>.........<intf n>
{
      variable declaration;
      method definition or declaration;
};
```

Whenever we use both extends and implements keywords as a part of JAVA program we must always write extends keyword first and latter we must use implements keyword.

Important points:

i. An object of interface cannot be created directly since it contains 'n' number of abstract methods. An object of interface can be created indirectly. An object of interface = an object of that class which implements that interface.
ii. An object of base interface contains the details about those methods which are declared in that interface only but it does not contain details about those methods which are specially available in either in derived classes or in derived interfaces.
iii. Interfaces should not be final.
iv. An interface does not contain Constructors.

Day - 29:

PACKAGE

A *package* is a collection of classes, interfaces and sub-packages. A sub-package in turns divides into classes, interfaces, sub-sub-packages, etc.

Learning about JAVA is nothing but learning about various packages. By default one predefined package is imported for each and every JAVA program and whose name is java.lang.*.

Whenever we develop any java program, it may contain many number of user defined classes and user defined interfaces. If we are not using any package name to place user defined classes and interfaces, JVM will assume its own package called NONAME package.

In java we have two types of packages they are **predefined** or **built-in** or **core packages** and **user** or **secondary** or **custom defined packages**.

PREDEFINED PACKAGES

Predefined packages are those which are developed by SUN micro systems and supplied as a part of JDK (Java Development Kit) to simplify the task of java programmer.

NOTE:

Core packages of java starts with **java.** (For example: java.lang.*) and *Advanced packages* of java starts with **javax.** (For example: java.sql.*)

TYPES of predefined packages:

As a part of J2SE we have nine predefined packages which are given in the following table:

Package name	Package description
java.lang.*	This package is used for achieving the language functionalities such as convertion of data from string to fundamental data, displaying the result on to the console, obtaining the garbage collector. This is the package which is by default imported for each and every java program.

Day - 29:

java.io.*	This package is used for developing file handling applications, such as, opening the file in read or write mode, reading or writing the data, etc.
java.awt.* (abstract windowing toolkit)	This package is used for developing GUI (Graphic Unit Interface) components such as buttons, check boxes, scroll boxes, etc.

java.awt.event.*	Event is the sub package of awt package. This package is used for providing the functionality to GUI components, such as, when button is clicked or when check box is checked, when scroll box is adjusted either vertically or horizontally.
java.applet.*	This package is used for developing browser oriented applications. In other words this package is used for developing distributed programs. An applet is a java program which runs in the context of www or browser.
java.net.*	This package is used for developing client server applications.
java.util.*	This package is used for developing quality or reliable applications in java or J2EE. This package contains various classes and interfaces which improves the performance of J2ME applications. This package is also known as collection framework (collection framework is the standardized mechanism of grouping of similar or different type of objects into single object. This single object is known as collection object).
java.text.*	This package is used for formatting date and time on day to day business operations.
java.lang.reflect.*	Reflect is the sub package of lang package. This package is basically used to study runtime information about the class or interface. Runtime information represents data members of the class or interface, Constructors of the class, types of methods of the class or interface.
java.sql.*	This package is used for retrieving the data from data base and performing various operations on data base.

USER DEFINED PACKAGES

A user defined package is one which is developed by java programmers to simplify the task of the java programmers to keep set of classes, interfaces and sub packages which are commonly used. Any class or interface is commonly used by many java programmers that class or interface must be placed in packages.

Syntax:
```
package pack1[.pack2[.pack3……[.packn]…..]];
```
Here, *package* is a **keyword** which is used for creating user defined packages, *pack1* represents **upper package** and *pack2* to *packn* represents **sub packages**.

For example:
```
package p1; → statement-1
package p1.p2; → statement-2
```
The statements 1 and 2 are called package statements.

RULE:

Whenever we create user defined package statement as a part of java program, we must use package statement as a first executable statement.

Day - 30:

NOTE:

Whenever we develop any JAVA program it contains 'n' number of *classes* and *interfaces*. **Each and every** *class* and *interface* which are developed by the programmer **must belong to a**

package (according to industry standards). If the **programmer is not keeping** the set of *classes* and *interfaces* in a *package*, **JVM will assume its own package** called **NONAME package**.

NONAME package will **exist only for a limited span of time until the program is completing**.

<u>**STEPS** for developing a **PACKAGE**</u>**:**

i. **Choose the appropriate package name**, the *package name* **must be a JAVA valid variable name** and we showed ensure the **package statement must be first executable statement**.
ii. Choose the appropriate *class name* or *interface name* and whose **modifier must be public**.
iii. The **modifier of Constructors of a class must be public**.
iv. The **modifier of the methods of class name or interface name must be public**.
v. **At any point of time** we **should place either** a *class* or an *interface* in a *package* and give the **file name** as *class name* or *interface name* with **extension .java**

<u>For example</u>:

```
// Test.java
package tp;
public class Test
{
	public Test ()
	{
		System.out.println ("TEST - DEFAULT CONSTRUCTOR");
	}
	public void show ()
	{
		System.out.println ("TEST - SHOW");
	}
}
//ITest.java
package tp;
public interface ITest
{
	void disp ();
}
```

<u>**Syntax** for **compiling a package**</u>**:**

```
javac –d . filename.java
```

<u>For example</u>:

```
javac –d . Test.java
```

Here, **-d** is an **option or switch** which **gives an indication to JVM** saying that **go to Test.java program take the package name** and **that package name is created as directory automatically** provides no errors are present in Test.java. When **Test.java is not containing any errors we get Test. class file and it will be copied automatically into current directory which is created recently** i.e., tp (package name). The above program cannot be executed since it doesn't contain any main method.

<u>**Day -31**</u>**:**

<u>**How to use PACKAGE CLASSES and INTERFACES in another java program**</u>**:**

In order **to refer** *package classes* and *interfaces* in JAVA we **have two approaches**, they are **using import statement** and **using fully qualified name approach**.

<u>Using **import statement**</u>**:**

Import is a **keyword** which is **used to import either single class or interface or set of classes and interfaces all at once**.

Syntax **-1:**
```
Import pack1 [.pack2 [.………[.packn]]].*;
```

For example:
```
Import p1.*;       ---1
Import p1.p2.*;    ---2
Import p1.p2.p3.*; ---3
```

When **statement 1 is executing we can import or we can access all the classes and interfaces of package p1 only** but not its *sub packages* p2 and p3 *classes* and *interfaces*.

When **statement 2 is executing we can import as the classes and interfaces of package p2 only** but not p1 and p3 *classes* and *interfaces*.

When **statement 3 is executing we can import as the classes and interfaces of package p3 only** but not p1 and p2 *classes* and *interfaces*.

Syntax**-2:**
```
Import pack1 [.pack2 [.…………[.packn]]].class name/interface name;
```

For example:
```
Import p1.c1;           ---4
Import p1.p2.c3;   ---5
```

When statement 4 is executing can import c1 class of package p1 only but not other classes and interfaces of p1 package, p2 package and p3 package.

Write a JAVA program which illustrates the usage of package classes?

Answer:

Import approach:
```
import tp.Test;
class PackDemo
{
	public static void main (String [] args)
	{
		Test t1=new Test ();
		t1.show ();
	}
};
```

When we compile the above program we get the following error "**package tp does not exist**". To avoid the above error we must set the classpath as., `SET CLASSPATH = %CLASSPATH%;.;`

This is the alternate technique for import statement:
```
p1.c2 o2=new p1.c2 ();
p1.p2.p3.c4 o4=new p1.p2.p3.c4 ();
p1.p2.i3 o3=new p1.p2.p3.c4 ();
```

Fully qualified approach:
```
class PackDemo
{
	public static void main (String [] args)
	{
		tp.Test t1=new tp.Test ();
		t1.show ();
	}
};
```

<u>NOTE</u>:

1. Whenever we develop user defined packages, to use the classes and interfaces of user defined packages in some other program, we must set classpath before there usage.
2. In order to set the classpath for predefined packages we must use the following statement:

`D:\core\set classpath=C: \Program Files\Java\jdk1.5.0\lib\rt.jar;.;` [rt.jar contains all the .class files for the predefined classes which are supplied as a port of predefined packages by the SUN micro systems.]

<u>Day - 32</u>:

When two classes or an interface belongs to the same package and if they want to refer those classes or interfaces need not to be referred by its package name.

<u>For example</u>:

```
// I1.java
// javac -d . I1.java

package ip;
public interface I1
{
	public void f1 ();
	public void f2 ();
};

// C01.java
// javac -d . C01.java

package cp;
public abstract class C01 implements ip.I1
{
	public void f1 ()
	{
		System.out.println ("F1 - C01");
	}
};

// C02.java
// javac -d . C02.java

package cp;
public class C02 extends C01
{
	public void f2 ()
	{
		System.out.println ("F2 - C02");
	}
	public void f1 ()
	{
		super. f1 ();
		System.out.println ("F1 - C02 - OVER RIDDEN");
	}
};

// PackDemo.java
// javac PackDemo.java

import ip.I1;
class PackDemo
{
	public static void main (String [] args)
```

```
      {
            System.out.println ("CP.C02");
            cp.C02 o2=new cp.C02 ();
            o2.f1 ();
            o2.f2 ();
            System.out.println ("CP.C01");
            cp.C01 o1=new cp.C02 ();
            o1.f1 ();
            o1.f2 ();
            I1 io;
            io=o1; //new cp.C02 ();
            io.f1 ();
            io.f2 ();
      }
};
```

In order to run the above program we must run with respect to package name.

1→ `javac –d . PackDemo1.java`

2→ `java mp.PackDemo1 or java mp/PackDemo1`

ACCESS SPECIFIERS in java

In order to use the data from one package to another package or within the package, we have to use the concept of access specifiers. In JAVA we have four types of access specifiers. They are **private, default** (not a keyword)**, protected** and **public**.

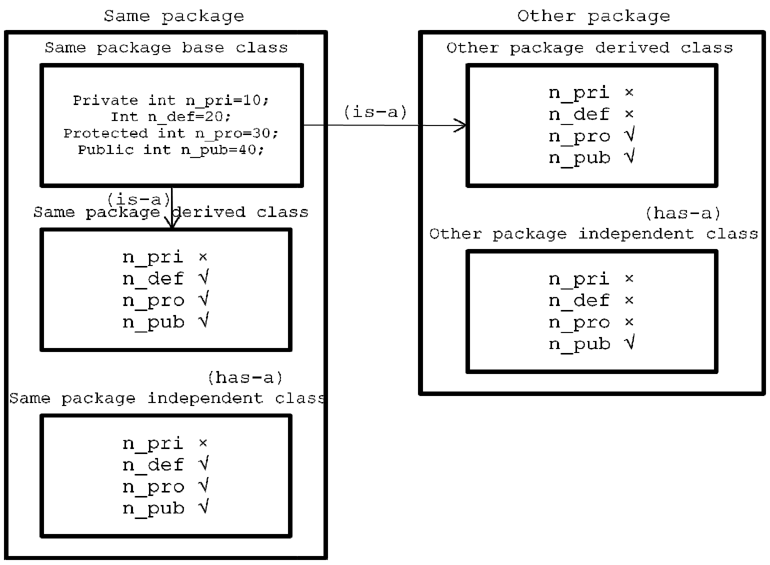
Access specifiers makes us to understand how to access the data within the package (class to class, interface to interface and interfaces to class) and across the package (class to class, interface to interface and interfaces to class). In other words access specifiers represent the visibility of data or accessibility of data.

Syntax for **declaring a variable** along with **access specifiers**:

```
[Access specifiers] [Static] [Final] data type v1 [=val1], v2 [=val2] …vn [=valn];
```

For example:

```
Public static final int a=10;
Protected int d;
```
`Int x;` [If we are not using any access specifier it is by **default** treated as default access specifier]
```
Private int e;
```



	Private	Default	Protected	Public
Same package base class	√	√	√	√
Same package derived class	×	√	√	√
Same package independent class	×	√	√	√
Other package derived class	×	×	√	√
Other package independent class	×	×	×	√

Day - 33:

NOTE:
1. Private access specifier is also known as native access specifier.
2. Default access specifier is also known as package access specifier.
3. Protected access specifier is also known as inherited access specifier.
4. Public access specifier is also known as universal access specifier.

Write a JAVA program which illustrates the concept of access rules?
Answer:
```
// Sbc.java
// javac -d . Sbc.java

package sp;
public class Sbc
{
	private int N_PRI=10;
	int N_DEF=20;
	protected int N_PRO=30;
	public int N_PUB=40;
	public Sbc()
	{
		System.out.println ("VALUE OF N_PRIVATE = "+N_PRI);
		System.out.println ("VALUE OF N_DEFAULT = "+N_DEF);
		System.out.println ("VALUE OF N_PROTECTED = "+N_PRO);
		System.out.println ("VALUE OF N_PUBLIC = "+N_PUB);
	}
};

// Sdc.java
// javac -d . Sdc.java

package sp;
public class Sdc extends Sbc //(is-a relation & within only)
{
	public Sdc()
	{
//		System.out.println ("VALUE OF N_PRIVATE = "+N_PRI);
		System.out.println ("VALUE OF N_DEFAULT = "+N_DEF);
		System.out.println ("VALUE OF N_PROTECTED = "+N_PRO);
```

```
            System.out.println ("VALUE OF N_PUBLIC = "+N_PUB);
      }
};

// Sic.java
// javac -d . Sic.java

package sp;
public class Sic
{
      Sbc so=new Sbc(); // (has-a relation & within only)
      public Sic()
      {
//          System.out.println ("VALUE OF N_PRIVATE = "+so.N_PRI);
            System.out.println ("VALUE OF N_DEFAULT = "+so.N_DEF);
            System.out.println ("VALUE OF N_PROTECTED = "+so.N_PRO);
            System.out.println ("VALUE OF N_PUBLIC = "+so.N_PUB);
      }
};

// Odc.java
// javac -d . Odc.java
package op;
public class Odc extends sp.Sbc // (is-a relation & across)
{
      public Odc ()
      {
//          System.out.println ("VALUE OF N_PRIVATE = "+N_PRI);
//          System.out.println ("VALUE OF N_DEFAULT = "+N_DEF);
            System.out.println ("VALUE OF N_PROTECTED = "+N_PRO);
            System.out.println ("VALUE OF N_PUBLIC = "+N_PUB);
      }
};

// Oic.java
// javac -d . Oic.java

package op;
public class Oic
{
      sp.Sbc so=new sp.Sbc (); // (has-a relation & across)
      public Oic ()
      {
//          System.out.println ("VALUE OF N_PRIVATE = "+so.N_PRI);
//          System.out.println ("VALUE OF N_DEFAULT = "+so.N_DEF);
//          System.out.println ("VALUE OF N_PROTECTED = "+so.N_PRO);
            System.out.println ("VALUE OF N_PUBLIC = "+so.N_PUB);
      }
};

// ASDemo.java
// javac ASDemo.java

import sp.Sbc;
import sp.Sdc;
import sp.Sic;
class ASDemo
{
      public static void main (String [] args)
      {
            // import approach

            System.out.println ("WITH RESPECT TO SAME PACKAGE BASE CLASS");
            Sbc so1=new Sbc();
            System.out.println ("WITH RESPECT TO SAME PACKAGE DERIVED CLASS");
            Sdc so2=new Sdc();
            System.out.println ("WITH RESPECT TO SAME PACKAGE INDEPENDENT CLASS");
            Sic so3=new Sic();

            //fully qualified name approach

            System.out.println ("WITH RESPECT TO OTHER PACKAGE DERIVED CLASS");
            op.Odc oo1=new op.Odc();
            System.out.println ("WITH RESPECT TO OTHER PACKAGE INDEPENDENT CLASS");
            op.Oic oo2=new op.Oic();
      }
};
```

Nameless object approach:

Sometimes there is no necessity for the JAVA programmer to create an object with some name. In such situations we can use the concept of nameless object.

For example:

```
// named object approach
Test t1=new Test ();
t1.display ();
```

To convert the above statements into nameless object approach follow the following statements.

For example:

```
// nameless object approach
new Test ().display ();
```

Day - 34:

EXCEPTIONAL HANDLING

Whenever we develop any project in real time it should work in all circumstances (mean in any operation either in error or error free). Every technology or every programming language, if we use for implementing real time applications and if the end user commits a mistake then by default that language or technology displays system error messages which are nothing but run time errors.

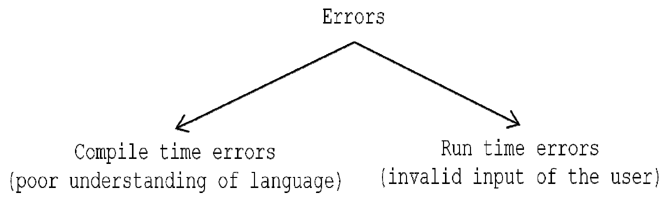
- Run time errors in JAVA are known as exceptions.
- System error messages are those which are unable to understand by end user or client.
- User friendly messages are those which are understandable by end user or client.

Exceptional handling is a mechanism of converting system error messages into user friendly messages.

Errors are of two types. They are **compile time errors** and **run time errors**.

- Compile time errors are those which are occurring because of poor understanding of the language.
- Run time errors are those which are occurring in a program when the user inputs invalid data.

The run time errors must be always converted by the JAVA programmer into user friendly messages by using the concept of exceptional handling.



In JAVA run time environment, to perform any kind of operation or task or an action that will be performed with respect to either class or an interface.

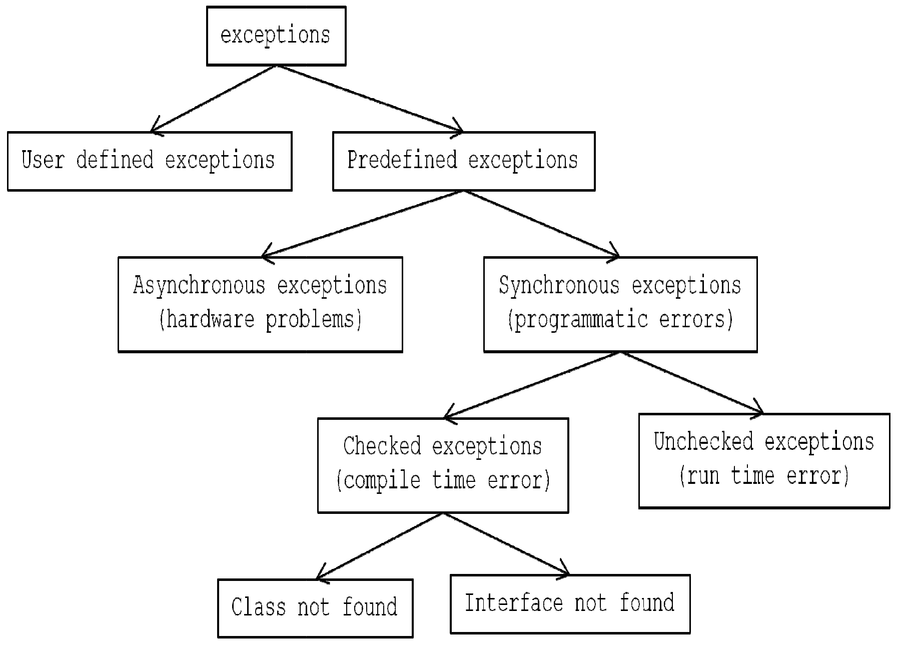
Whenever we pass invalid data as an input to the JAVA program then JAVA run time environment displays an error message, that error message is treated as a class.

<u>Types of exceptions</u>:

In JAVA we have two types of exceptions they are **predefined exceptions** and **user or custom defined exceptions**.

1. Predefined exceptions are those which are developed by SUN micro system and supplied as a part of JDK to deal with universal problems. Some of the universal problems are *dividing by zero*, *invalid format of the number*, *invalid bounce of the array*, etc.

<u>Day -35</u>:



Predefined exceptions are divided into two types. They are **asynchronous exceptions** and **synchronous exceptions**.

Asynchronous exceptions are those which are always deals with hardware problems. In order to deal with *asynchronous exceptions* there is a predefined class called **java.lang.Error**. *java.lang.Error* class is the super class for all *asynchronous exceptions*.

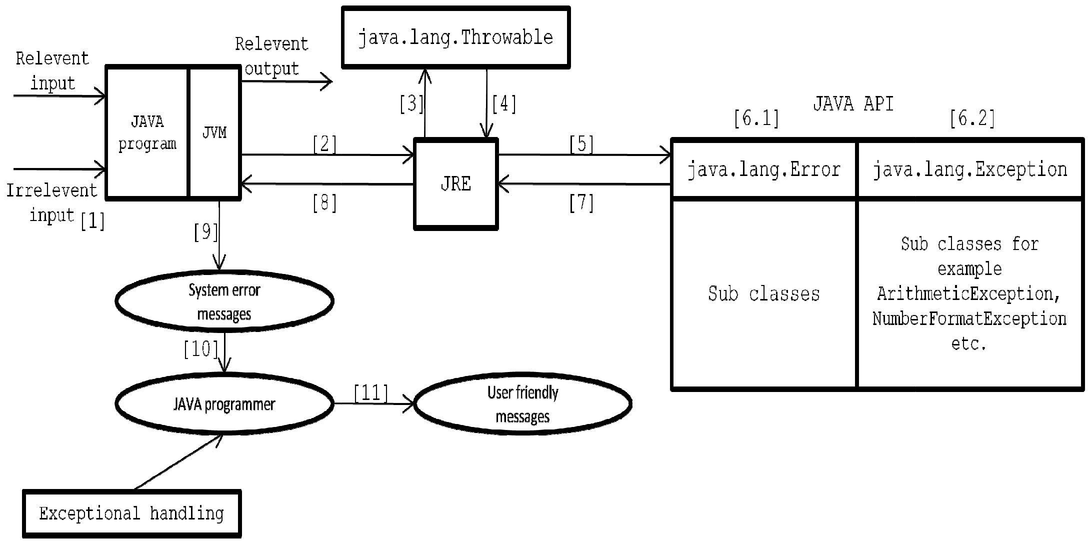
Synchronous exceptions are one which always deals with programmatic errors. In order to deal with *synchronous exceptions* we must use a predefined class called **java.lang.Exception** class.

java.lang.Exception is the super class for all *synchronous exceptions*. *Synchronous exceptions* are divided into two types. They are **checked exceptions** and **unchecked exceptions**.

i. A *checked exception* is one which always deals with compile time errors regarding **class not found** and **interface not found**.
ii. *Unchecked exceptions* are those which are always deals with programmatic run time errors such as ArithmeticException, NumberFormatException, ArrayIndexOutOfBoundsException, etc.

An exception is an object which occurs at run time which describes the nature of the message. The nature of the message can be either system error message or user friendly message.

How an EXCEPTION OCCURS in a java RUN TIME ENVIRONMENT

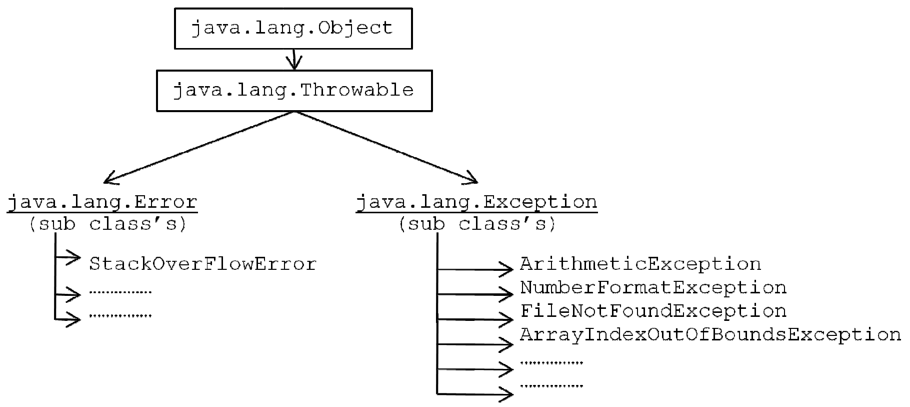


1. Whenever we pass irrelevant input to a JAVA program, JVM cannot process the irrelevant input.
2. Since JVM is unable to process by user input, hence it can contact to JRE for getting an appropriate exception class.
3. JRE contacts to java.lang.Throwable for finding what type of exception it is.
4. java.lang.Throwable decides what type of exception it is and pass the message to JRE.
5. JRE pass the type of exception to JAVA API.
6. [6.1 & 6.2] From the JAVA API either java.lang.Error class or java.lang.Exception class will found an appropriate sub class exception.
7. Either java.lang.Error class or java.lang.Exception class gives an appropriate exception class to JRE.
8. JRE will give an appropriate exception class to JVM.
9. JVM will create an object of appropriate exception class which is obtained from JRE and it generates system error message.
10. In order to make the program very strong (robust), JAVA programmer must convert the system error messages into user friendly messages by using the concept of exceptional handling.

User friendly messages are understand by normal user effectively hence our program is robust.

HIERARCHY chart of EXCEPTIONAL HANDLING

java.class.Object → super class for all JAVA class's.



Day - 36:

Syntax for **exceptional handling**:

In order to handle he exceptions in JAVA we must use the following keywords. They are **try, catch, finally, throws** and **throw**.

```
try
{
Block of statements which are to be monitored by JVM at run time (or problematic errors);
}

catch (Type_of_exception1 object1)
{
      Block of statements which provides user friendly messages;
}
catch (Type_of_exception2 object2)
{
      Block of statements which provides user friendly messages;
}
.
.
.
catch (Type_of_exception3 object3)
{
      Block of statements which provides user friendly messages;
}

finally
{
      Block of statements which releases the resources;
}
```

Try block:

1. This is the block in which we write the block of statements which are to be monitored by JVM at run time i.e., try block must contain those statements which causes problems at run time.
2. If any exception is taking place the control will be jumped automatically to appropriate catch block.
3. If any exception is taking place in try block execution will be terminated and the rest of the statements in try block will not be executed at all and the control will go to catch block.
4. For every try block we must have at least one catch block. It is highly recommended to write 'n' number of catch's for 'n' number of problematic statements.

<u>Catch block</u>:

1. This is used for providing user friendly messages by catching system error messages.
2. In the catch we must declare an object of the appropriate execution class and it will be internally referenced JVM whenever the appropriate situation taking place.
3. If we write 'n' number of catch's as a part of JAVA program then only one catch will be executing at any point.
4. After executing appropriate catch block even if we use return statement in the catch block the control never goes to try block.

<u>Finally block</u>:

1. This is the block which is executing compulsory whether the exception is taking place or not.
2. This block contains same statements which releases the resources which are obtained in try block (resources are opening files, opening databases, etc.).
3. Writing the finally block is optional.

<u>For example</u>:
```
class Ex1
{
	public static void main (String [] args)
	{
		try
		{
			String s1=args[0];
			String s2=args[1];
			int n1=Integer.parseInt (s1);
			int n2=Integer.parseInt (s2);
			int n3=n1/n2;
			System.out.println ("DIVISION VALUE = "+n3);
		}
		catch (ArithmeticException Ae)
		{
			System.out.println ("DONT ENTER ZERO FOR DENOMINATOR...");
		}
		catch (NumberFormatException Nfe)
		{
			System.out.println ("PASS ONLY INTEGER VALUES...");
		}
		catch (ArrayIndexOutOfBoundsException Aioobe)
		{
			System.out.println ("PASS DATA FROM COMMAND PROMPT...");
		}
		finally
		{
			System.out.println ("I AM FROM FINALLY...");
		}
	}
};
```

<u>Day - 37</u>:

<u>Throws block</u>: This is the keyword which **gives an indication to** the **calling function to keep** the **called function under try and catch blocks**.

<u>Syntax</u>**:**
```
<Return type> method name (number of parameters if any) throws type of exception1,type of exception2,………type of exception;
```

Write a JAVA program which illustrates the concept of throws keyword?

Answer:

(**CALLED FUNCTION**)

```
package ep;
public class Ex2
{
        public void div (String s1, String s2)   throws ArithmeticException, NumberFormatException
        {
                int n1=Integer.parseInt (s1);
                int n2=Integer.parseInt (s2);
                int n3=n1/n2;
                System.out.println ("DIVISOIN = "+n3);
        }
};
```

(**CALLING FUNCTION**)

```
import ep.Ex2;
class Ex3
{
        public static void main (String [] args)
        {
                try
                {
                        String s1=args [0];
                        String s2=args [1];
                        Ex2 eo=new Ex2 ();
                        eo.div (s1,s2);
                }
                catch (ArithmeticException Ae)
                {
                        System.out.println ("DONT ENTER ZERO FOR DENOMINATOR");
                }
                catch (NumberFormatException Nfe)
                {
                        System.out.println ("PASS INTEGER VALUES ONLY");
                }
                catch (ArrayIndexOutOfBoundsException Aioobe)
                {
                        System.out.println ("PASS VALUES FROM COMMAND PROMPT");
                }
        }
};
```

Day - 38:

Number of **ways to find** details of the **exception:**

In JAVA there are three ways to find the details of the exception. They are **using an object of java.lang.Exception class, using public void printStackTrace method** and **using public string getMessage method**.

i. **Using an object of java.lang.Exception:** An object of Exception class prints the name of the exception and nature of the message.

For example:

```
try
{
   int x=Integer.parseInt ("10x");
}
catch (Exception e)
{
    System.out.println (e); // java.lang.NumberFormatException  :  for input string 10x
}                                  name of the exception           nature of the message
```

ii. **Using printStackTrace method:** This is the method which is defined in java.lang.Throwable class and it is inherited into java.lang.Error class and java.lang.Exception class. This method will display name of the exception, nature of the message and line number where the exception has taken place.

For example:

```
try
{
    ......;
    int x=10/0;
    ......;
}
catch (Exception e)
{
    e.printStackTrace (); // java.lang.ArithmeticException : / by zero :       at line no: 4
}                          name of the exception    nature of the message   line number
```

iii. **Using getMessage method:** This is also a method which is defined in java.lang.Throwable class and it is inherited into both Error and Exception classes. This method will display only nature of the message.

For example:

```
try
{
    ......;
    int x=10/0;
    ......;
}
catch (Exception e)
{
    System.out.println (e.getMessage ()); // / by zero
}                                          nature of the message
```

USER or CUSTOM DEFINED EXCEPTIONS

User defined exceptions are those which are developed by JAVA programmer as a part of application development for dealing with specific problems such as negative salaries, negative ages, etc.

Guide lines for developing user defined exceptions:

- Choose the appropriate package name.
- Choose the appropriate user defined class.
- Every user defined class which we have choose in step 2 must extends either java.lang.Exception or java.lang.RunTimeException class.
- Every user defined sub-class Exception must contain a parameterized Constructor by taking string as a parameter.
- Every user defined sub-class exception parameterized Constructor must called parameterized Constructor of either java.lang.Exception or java.lang.RunTimeException class by using super (string parameter always).

For example:

```
package na; // step1
public class Nage extends Exception // step2 & step3
{
        public Nage (String s) // step4
```

```
	{
		super (s); // step5
	}
};
```

Day - 39:

Write a JAVA program which illustrates the concept of user defined exceptions?

Answer:

```
package na;
public class Nage extends Exception
{
	public Nage (String s)
	{
		super (s);
	}
};

package ap;
import na.Nage;
public class Age
{
	public void decide (String s) throws NumberFormatException, Nage
	{
		int ag= Integer.parseInt (s);
		if (ag<=0)
		{
			Nage na=new Nage ("U HAV ENTERED INVALID AGE..!");
			throw (na);
		}
		else
		{
			System.out.println ("OK, U HAV ENTERED VALID AGE..!");
		}
	}
};

import na.Nage;
import ap.Age;
class CDemo
{
	public static void main(String[] args)
	{
		try
		{
			String s1=args [0];
			ap.Age Ao=new ap.Age ();
			Ao.decide (s1);
		}
		catch (Nage na)
		{
			System.out.println (na);
		}
		catch (NumberFormatException nfe)
		{
			System.out.println ("PASS ONLY INTEGER VALUES..!");
		}
		catch (ArithmeticException ae)
		{
			System.out.println ("PASS INTEGER VALUES ONLY..!");
		}
		catch (ArrayIndexOutOfBoundsException aioobe)
		{
			System.out.println ("PASS VALUES THROUGH COMMAND PROMPT..!");
		}
		catch (Exception e)
		{
			System.out.println (e);
		}
	}
};
```

NOTE: Main method should not throw any exception since the main method is called by JVM and JVM cannot provide user friendly message.

APPLET's

In JAVA we write two types of programs or applications. They are **standalone applications** and **distributed applications**.

- A *standalone application* is one which runs in the context of local disk and whose result is not sharable. Every standalone application runs from command prompt and it contains main method along with System.out.println statements.
- A *distributed application* is one which runs in the context of browser or World Wide Web and it can be accessed across the globe. Any technology which runs in the browser will have 'n' number of life cycle methods and it does not contain main methods and System.out.println statements.

In JAVA, SUN micro initially has developed a concept called applets which runs in the context of browser. "*An **applet** is a JAVA program which runs in the context of browser or World Wide Web*".

In order to deal with applets we must import a package called java.applet.*. This package contains only one class Applet whose fully qualified name is java.applet.Applet.

Since applets are running in the browser, the class Applet contains the life cycle methods. Life cycle methods are also called as loop back methods.

Day - 40:

In java.applet.Applet we have four life cycle methods. They are **public void init (), public void start (), public void stop ()** and **public void destroy ()**.

1. **Public void init ():** This is the method which is called by the browser only one time after loading the applet. In this method we write some block of statements which will perform one time operations, such as, obtaining the resources like opening the files, obtaining the database connection, initializing the parameters, etc.
2. **Public void start ():** After calling the init method, the next method which is from second request to sub-sequent requests the start method only will be called i.e., short method will be called each and every time. In this method we write the block of statement which provides business logic.
3. **Public void stop ():** This id the method which is called by the browser when we minimize the window. In this method we write the block of statements which will temporarily releases the resources which are obtained in init method.
4. **Public void destroy ():** This is the method which will be called by the browser when we close the window button or when we terminate the applet application. In this method we write same block of statements which will releases the resources permanently which are obtained in init method.

Another method which is not a life cycle method is **public void paint ()**. This is the method which will be called by the browser after completion of start method. This method is used for displaying the data on to the browser. Paint method is internally calling the method called drawString whose prototype is given below.

```
java.awt.Graphics
```
(Graphics => public void drawString (String, int row position, int column position))

An object of Graphics class will be created automatically after loading the applet into the browser.

<u>STEPS for developing APPLET PROGRAM</u>:

1. Import java.applet.Applet package.
2. Choose the user defined class that must extends java.applet.Applet class and ensure the modifier of the class must be public.
3. Overwrite the life cycle methods of the applet if require.
4. Save the program and compile.
5. Run the applet: To run the applet we have two ways. They are **using HTML program** and **using applet viewer tool**.

<u>Using HTML program</u>: In order to run the applet through HTML program we must use the following tag.

<u>Syntax</u>**:**
```
<applet code =".class file of the applet" height = height value width = width
value>
       </applet>
```

<u>For example</u>:
```
File name: MyApp.html
<HTML>
 <HEAD>
  <TITLE> My applet example </TITLE>
 </HEAD>

 <BODY>
  <APPLET code="MyApp" height=100 width=150>
  </APPLET>
 </BODY>
</HTML>
```

<u>Using appletviewer</u>:

Appletviewer is a tool supplied by SUN micro system to run the applet programs from the command prompt (in the case of browser is not supporting).

<u>Syntax</u>**:** `appletviewer filename.java`

<u>For example</u>:
```
appletviewer MyApp.java
```

When we use appletviewer to run the above applet, MyApp.java program must contain the following tag within the multi line comment.

```
/*<applet code= "MyApp" height=300 width=300> </applet>*/
```

Write an applet program which displays "AshaKrishna My Love"?

<u>Answer</u>:
```
/*<applet code= "MyApp" height=300 width=300> </applet>*/
import java.applet.Applet;
public class MyApp extends Applet
```

```
{
	public void paint(java.awt.Graphics g)
	{
		g.drawString ("AshaKrishna MyLove", 20, 15);
	}
};
```

Day - 41:

NOTE: To set the font we must use a class called Font.

Syntax: `Font f=new Font (“arial”, Font.BOLD, 40);`

In the Graphics class we have the following method which will set the font.

```
java.awt.Graphics
public void setFont (Font fobj)
```

For example:

```
g.setFont (f);
```

Write a JAVA program which illustrates the life cycle methods of applet?

Answer:

```
import java.applet.*;
import java.awt.*;
/*<applet code="MyApp1" height=300 width=300>
</applet>*/

public class MyApp1 extends Applet
{
	String s=" ";
	public void init ()
	{
		setBackground (Color.green);
		setForeground (Color.red);
		s=s+" INIT ";
	}
	public void start ()
	{
		s=s+" START ";
	}
	public void stop ()
	{
		s=s+" STOP ";
	}
	public void destroy ()
	{
		s=s+" DESTROY ";
	}
	public void paint (Graphics g)
	{
		Font f=new Font ("arial", Font.BOLD, 40);
		g.setFont (f);
		g.drawString (s,100,100);
	}
};
```

awt (abstract windowing toolkit):

In JAVA we can develop to types of GUI (Graphic User Interface) applications. They are **standalone GUI applications** and **distributed GUI applications**.

- A standalone GUI application is one which runs in the context of local disk and our class must extends a predefined class called **java.lang.Frame** class.
- A distributed GUI application is one which runs in the context of browser and our class must extend **java.applet.Applet** class.

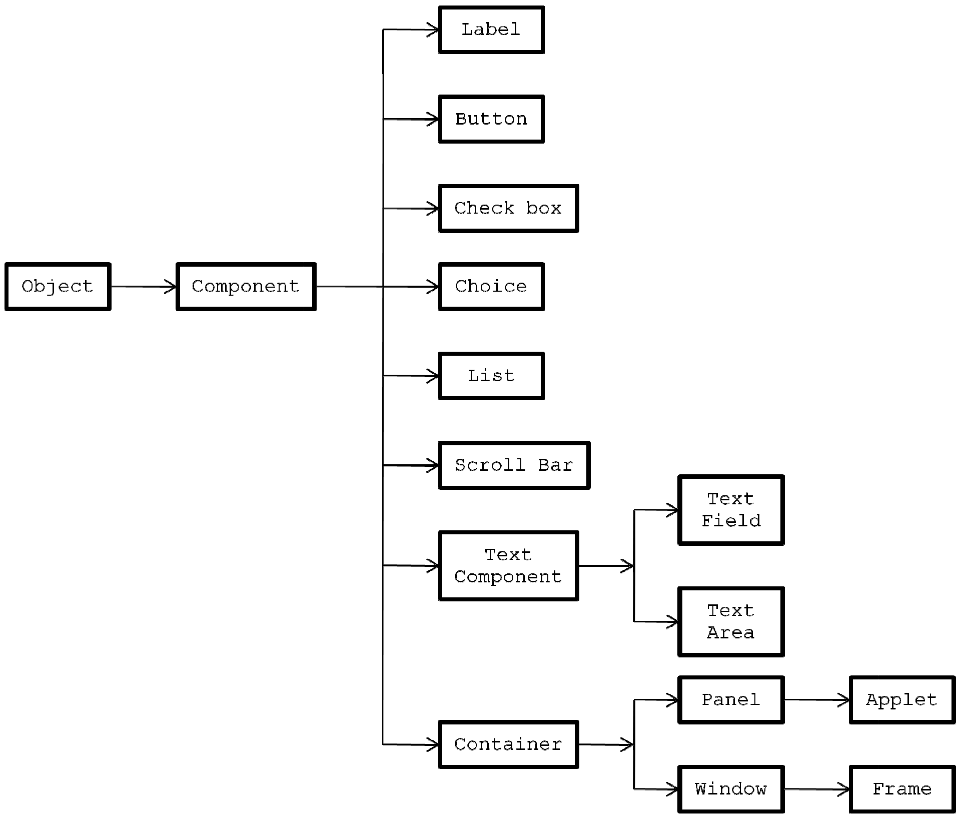
As a part of GUI applications we use to create two types of components. They are **passive components** and **active components**.

- A passive component is one where there is no interaction from the user. For example label.
- An active component is one where there is an interaction from the user. For example button, check box, scroll bar, etc.

The active and passive components in JAVA are available in terms of classes. In order to deal with any GUI applications we must import a package called java.awt.* (contains various classes and interfaces for creating GUI components) and java.awt.event.* (contains various classes and interfaces which will provide functionality to GUI components).

Day - 42:

AWT hierarchy chart:



Whenever we develop any GUI application we must have readily available window component and window component must contain frame component. Any GUI component which we want to create gives the corresponding class and adds all those components to the object of Container class.

A Container is a class whose object allows us to add 'n' number of similar or different GUI components to make a final application.

Except Object class and Applet class all the classes in the above hierarchy chart are belongs to java.awt.* package.

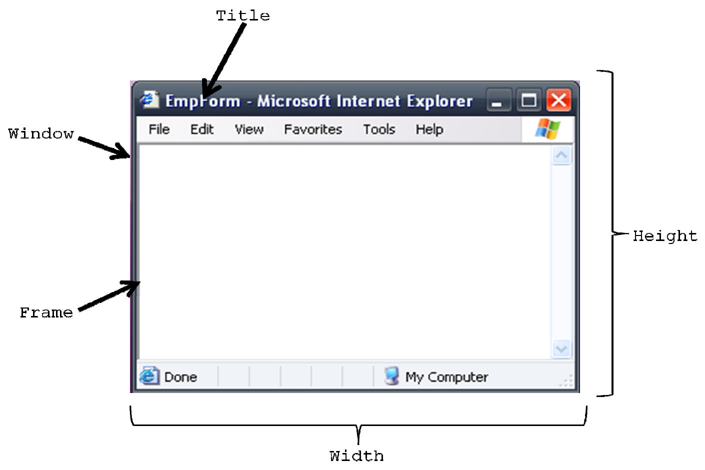
Write a JAVA program which creates Window and Frame?

<u>Answer</u>:

```
import java.awt.*;
class myf extends Frame
{
	Myf ()
	{
		setText ("AshaKrishna"); ---1
		setSize (100, 100);		---2
		setBackground (Color, cyan);	---3
		setForeground (Color, red);	---4
		setVisible (true);	---5
	}
};

class FDemo
{
	Public static void main (String [] args)
	{
		Myf mo=new myf ();
	}
};
```

The methods in line numbers 1 and 2 defined in java.awt.Window class. The methods in line numbers 3, 4 and 5 are defined in java.awt.Component class.


Title
Window
Frame
Height
Width
EmpForm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

<u>Label class</u>:

Label is a class which is used for creating label as a part of windows application. The component label comes under passive component. Labels always improve the functionality and readability of active components. Creating a label is nothing but creating an object of label components.

<u>Label class API</u>:

Data members:

`public static final int LEFT (0)` //by default the alignment is always left only

```
public static final int CENTER (1)
public static final int RIGHT (2)
```

The above three statements are called alignment parameters or modifiers.

Constructors:

```
Label ()
Label (String)
Label (String label name, int alignment modifier)
```

Instance methods:

```
public void setText (String);
public String getText ();
public void setAlignment (int);
public int getAlignment ();
```

For example we can write a program as follows:

```
Label l=new Label ();
l=setText (“ENTER EMPNO”);
l=setAlignment (Label.CENTER)
```

or

```
Label l1=new Label (“ENTER EMPNAME”);
String Label=l1.getText ();
l1.setAlignment (Label.RIGHT);
```

or

```
Label l=new Label (“ENTER EMPSAL”, Label.CENTER);
```

Day - 43:

Event Delegation Model:

Whenever we want to develop any windows applications one must deal with event delegation model. Event delegation model contains four properties. They are:

In order to process any active components, we must know either name or caption or label or reference of the component (object).

Whenever we interact any active component, the corresponding active component will have one predefined Event class, whose object will be created and that object contains two details:

1. Name of the component.
2. Reference of the component.

The general form of every Event class is xxx event.

For example:

Component name	**Event name**
print	java.awt.event.ActionEvent
choice	java.awt.event.ItemEvent
textField	java.awt.event.TextEvent
textArea	java.awt.event.TextEvent
scrollbar	java.awt.event.AdjustmentEvent

In order to perform any action or operation when an interactive component is interacted we must write some set of statements into the appropriate methods. These methods are not defined or developed by SUN micro system, but they have supplied those methods with no definition i.e., undefined methods.

In JAVA purely undefined methods are abstract methods present in interfaces. Interfaces in awt are called listeners. Hence, every interactive component must have a predefined listener whose general notation is xxx listener.

For example:

Component name	**Listener name**
`Button`	`java.awt.event.ActionListener`
`Choice`	`java.awt.event.ItemListener`
`TextField`	`java.awt.event.TextListener`
`TextArea`	`java.awt.event.TextListener`
`Scrollbar`	`java.awt.event.AdjustmentListener`

Identity what is the undefined method or abstract method in xxx Listener.

For example:

Component name	**Undefined method name**
`Button`	`public void actionPerformed (java.awt.event.ActionEvent)`
`Choice`	`public void actionPerformed (java.awt.event.ItemEvent)`
`TextField`	`public void actionPerformed (java.awt.event.TextEvent)`
`TextArea`	`public void actionPerformed (java.awt.event.TextEvent)`

Each and every interactive component must be registered and unregistered with particular event and Listener. The general form of registration and un-registration methods is as follows:

```
public void addxxxListener (xxxListener);
public void removexxxListener (xxxListener);
```

For example:

Component name	Registration method	Un-registration method
`Button`	`public void addActionListener (ActionListener);`	`public void removeActionListener (ActionListener);`
`Choice`	`public void addItemListener (ItemListener);`	`public void removeItemListener (ItemListener);`
`TextField`	`public void addTextListener (TextListener);`	`public void removeTextListener (TextListener);`
`TextArea`	`public void addTextListener (TextListener);`	`public void removeTextListener (TextListener);`

Day - 44:

BUTTON:

Button is an active component which is used to perform some operations such as saving the details into the database, deleting the details from the database, etc.

Creating a button is nothing but creating an object of Button class.

Button API:

Constructors:

```
Button ();
Button (String);
```

Instance methods:

```
public void add ActionListener (ActionListener);
public void remove ActionListener (ActionListener);
```

Write a java program to illustrate the concept of buttons?

Answer:

```
import java.awt.*;
import java.awt.event.*;
class BDemo extends Frame implements ActionListener
{
	Button b1,b2,b3,b4;
	Label l;
	BDemo ()
	{
		super ("BUTTON EXAMPLE...");// set title
		setSize (200,200);
		b1=new Button ("NORTH");
		b2=new Button ("SOUTH");
		b3=new Button ("EAST/EXIT");
		b4=new Button ("WEST");
		l=new Label ();
		l.setAlignment (Label.CENTER);
		add (b1,"North");
		add (b2,"South");
		add (b3,"East");
		add (b4,"West");
		add (l);// add at center of border layout by default
		b1.addActionListener (this);
		b2.addActionListener (this);
		b3.addActionListener (this);
		b4.addActionListener (this);
		setVisible (true);
	}
	public void actionPerformed (ActionEvent ae)
	{
		if (ae.getSource ()==b1)
		{
			System.out.println ("U HAVE CLICKED "+b1.getLabel ());
//			String s1=b1.getLabel ();
//			l.setText (s1);
			l.setText (b1.getLabel ());
		}
		if (ae.getSource ()==b2)
		{
			System.out.println ("U HAVE CLICKED "+b2.getLabel ());
			l.setText (b2.getLabel ());
		}
		if (ae.getSource ()==b3)
		{
			System.out.println ("U HAVE CLICKED "+b3.getLabel ());
			l.setText (b3.getLabel ());
//			System.exit (0);
		}
		if (ae.getSource ()==b4)
		{
			System.out.println ("U HAVE CLICKED "+b4.getLabel ());
			l.setText (b4.getLabel ());
		}
	}// actionPerformed
};// BDemo
class BDemo1
{
	public static void main (String [] args)
	{
		new BDemo ();
	}
};
```

Output:
```
U HAVE CLICKED WEST
U HAVE CLICKED SOUTH
U HAVE CLICKED EAST/EXIT
U HAVE CLICKED NORTH
```

<u>Steps for developing awt program</u>:

1. Import the appropriate packages.
2. Choose the appropriate class and it must extend java.awt.Frame and implements appropriate Listener if required.
3. Identify which components we want and declare them as data members in the class.
4. Set the title for the window.
5. Set the size of the window.
6. Create the components which are identified in step 3.
7. Add the created components to container.
8. Register the events of the appropriate interactive component with appropriate Listener.
9. Make the components to be visible.
10. Define the undefined methods in the current class which is coming from appropriate Listener. This method provides functionality to GUI component.

Rewrite the above program using applets?

<u>Answer</u>:
```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
/*<applet code="ButtonApp" height=200 width=200>
</applet>*/
public class ButtonApp extends Applet implements ActionListener
{
	Button b1, b2, b3, b4;
	Label l;
	public void init ()
	{
		setLayout (new BorderLayout ());
		b1=new Button ("North");
		b2=new Button ("East");
		b3=new Button ("West");
		b4=new Button ("South");
		l=new Label ();
		l.setAlignment (Label.CENTER);
		add (b1, "North");
		add (b2, "East");
		add (b3, "West");
		add (b4, "South");
		add (l);
	}
	public void Start ()
	{
		b1.addActionListener (this);
		b2.addActionListener (this);
		b3.addActionListener (this);
		b4.addActionListener (this);
	}
	public void actionPerformed (ActionEvent ae)
	{
		if (ae.getSource ()==b1)
		{
			l.setLabel (b1.getLabel ());
		}
		if (ae.getSource ()==b2)
		{
			l.setLabel (b2.getLabel ());
		}
		if (ae.getSource ()==b3)
		{
			l.setLabel (b3.getLabel ());
		}
		if (ae.getSource ()==b4)
		{
			l.setLabel (b4.getLabel ());
		}
	}
};
```

Steps for developing a FRAME application or APPLET application:

1. Import appropriate packages for GUI components (java.awt.*) providing functionality to GUI components (java.awt.event.*) and for applet development (java.applet.Applet).
2. Every user defined class must extend either Frame or Applet and it must implement appropriate Listener if required.
3. Identify which components are required to develop a GUI application.
4. Use life cycle methods (init, start, destroy) in the case of applet, use default Constructor in the case of Frame for creating the components, adding the components, registering the components, etc.
5. Set the title of the window.
6. Set the size of the window.
7. Set the layout if required.
8. Create those components which are identified.
9. Add the created components to container.
10. Every interactive component must be registered with appropriate Listener.
11. Make the components to be visible in the case of Frame only.
12. Implement or define the abstract method which is coming from appropriate Listener.

CHOICE:

Choice is the GUI component which allows us to add 'n' number of items. This component allows selecting a single item at a time.

Creating a choice component is nothing but creating an object of Choice class.

Choice API:

Constructors:

```
choice ();
```

Instance methods:

```
public void add (String); →1
public void addItem (String); → 2
public void add (int pos, String); →3
public void addItem (int pos, String); →4
public String getSelectedItem (); →5
public int getSelectedIndex (); →6
public void remove (String); →7
public void remove (int); →8
public void removeAll (); →9
public void addItemListener (ItemListener); →10
public void removeItemListener (ItemListener); →11
```

- Methods 1 and 2 are used for adding items to the choice component at end.
- Methods 3 and 4 are used for adding the items at the specified position.

- Methods 5 and 6 are used for obtaining the items either based on its name or on its index.
- Methods 7 and 8 are used for removing the item of the choice component either based on its name or on its index (position).
- Method 9 is used for removing all the items of choice components.
- Methods 10 and 11 are used for registering and unregistering the events of choice components.

Day - 45:

Write a java program which illustrates the concept of Choice?

Answer:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="ChoApp" height=300 width=300>
</applet>*/
public class ChoApp extends Applet
{
	Choice ch1, ch2;
	Label l1,l2;
	public void init ()
	{
		setBackground (Color.yellow);
		setForeground (Color.red);
		l1=new Label ("AVAILABLE FONTS: ");
		l2=new Label ("REMOVED FONTS: ");
		ch1=new Choice ();
		ch2=new Choice ();
		add (l1);add (ch1);
		add (l2); add (ch2);
		GraphicsEnvironment ge=GraphicsEnvironment.getLocalGraphicsEnvironment ();
		String f[]=ge.getAvailableFontFamilyNames ();
		for (int i=0; i<f.length; i++)
		{
			ch1.add (f [i]);
		}
	}// init ()
	public void start ()
	{
		ch1.addItemListener (new ch1 ());
	}
	class ch1 implements ItemListener
	{
		public void itemStateChanged (ItemEvent ie);
		{
			if (ie.getSource ()==ch1)
			{
				String s1=ch1.getSelectedItem ();
				ch2.add (s1);
				ch1.remove (s2);
			}
		}
	}// ch1-inner class
}// ChoApp class
```

CHECKBOX:

Checkbox is basically used for maintaining checklist purpose checkbox is an interactive component which contains a square symbol and a label.

Creating a checkbox is nothing but creating an object of checkbox class

Checkbox API:

Constructors:

```
checkbox ();
checkbox (String);
checkbox (String, boolean);
```

Instance methods:

```
public void setLabel (String);
public String getLabel ();
public void setState (boolean);
public boolean getState ();
public void addItemListener (ItemListener);
public void removeItemListener (ItemListener);
```

Write a java program which illustrates the concept of Checkbox?

Answer:

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
/*<applet code="CheckboxApp" height=200 width=200>
</applet>*/
public class CheckboxApp extends Applet
{
	Checkbox cb;
	Label l,l1;
	public void init ()
	{
		cb1=new Checkbox ("Java");
		l=new Label ("Course: ");
		l1=new Label ();
		add (cb);
		add (l);
		add (l1);
	}
	public void Start ()
	{
		cb1.addItemListener (new itl ());
	}
	class itl implements ItemListener
	{
		public void itemStateChanged (ItemEvent ie)
		{
			Object obj=ie.getItemSelectable ();
			Checkbox cb1= (Checkbox) obj;
			if (cb1.getState ())
			{
				l1.setText ("U HAVE SELECTED "+cb1.getLabel ());
			}
			else
			{
				l1.setText ("U HAVE SELECTED NONE");
			}
		}
	}
};
```

RADIO BUTTONS:

In java.awt we are not having a special class for radio buttons but we can create radio button from Checkbox class.

In java.awt we have a predefined class called CheckboxGroup through which we can add all the checkboxes to the object of CheckboxGroup class.

Steps for converting checkboxes into radio button:

1. Create objects of CheckboxGroup class.

   For example:
   ```
   CheckboxGroup cbg1=new CheckboxGroup ();
   CheckboxGroup cbg2=new CheckboxGroup ();
   CheckboxGroup cbg3=new CheckboxGroup ();
   CheckboxGroup cbg4=new CheckboxGroup ();
   ```
   CheckboxGroup object allows to select a single checkbox among 'n' number of checkboxes.

2. Create 'n' number of objects of Checkbox class.

   For example:

   Checkbox b1, b2, b3, b4;
   ```
   b1=new Checkbox ("C");
   b2=new Checkbox ("Cpp");
   b3=new Checkbox ("Java");
   b4=new Checkbox ("Oracle9i");
   ```

3. Decide which checkboxes are adding to which CheckboxGroup object. In order to add the checkboxes to the CheckboxGroup object, in Checkbox class we have the following method:
   ```
   public void setCheckboxGroup (CheckboxGroup);
   ```
   For example:
   ```
   cb1.setCheckboxGroup (cbg1);
   cb2.setCheckboxGroup (cbg2);
   cb3.setCheckboxGroup (cbg3);
   cb4.setCheckboxGroup (cbg4);
   ```

4. Register the events of checkbox with ItemListener by using the following method:
   ```
   public void addItemListener (ItemListener);
   ```

   NOTE: Instead of using a setCheckboxGroup method we can also used the following Constructor which is present in Checkbox class.
   ```
   Checkbox (String, CheckboxGroup, boolean);
   Checkbox (String, boolean, CheckboxGroup);
   ```
   For example:
   ```
   Checkbox cb1=new Checkbox ("C", cbg1, false);
   Checkbox cb2=new Checkbox ("Cpp", cbg2, false);
   Checkbox cb3=new Checkbox ("Java", cbg3, false);
   Checkbox cb4=new Checkbox ("Oracle9i", cbg4, false);
   ```

Write a java program which illustrates the concept of Radio Button?

Answer:

```
import java.awt.*;
import java.awt.event.*;
class RadioApp extends Frame
{
	Checkbox cb1, cb2, cb3, cb4, cb5;
	Label l, l1;
	RadioApp ()
	{
		setSize (200, 200);
		setLayout (new FlowLayout ());
		CheckboxGroup cbg1=new CheckboxGroup();
		CheckboxGroup cbg2=new CheckboxGroup();
		cb1=new Checkbox ("C", cbg1, false);
		cb2=new Checkbox ("Cpp", cbg1, false);
		cb3=new Checkbox ("Java", cbg2, false);
		cb4=new Checkbox ("Oracle9i", cbg2, false);
		cb5=new Checkbox ("exit");
		l=new Label ("Course : ");
		l1=new Label ();
		add (l);
		add (cb1);
		add (cb2);
		add (cb3);
		add (cb4);
		add (cb5);
		add (l1);
		cb1.addItemListener (new itl ());
		cb2.addItemListener (new itl ());
		cb3.addItemListener (new itl ());
		cb4.addItemListener (new itl ());
		cb5.addItemListener (new itl ());
		setVisible (true);
	}
	class itl implements ItemListener
	{
		public void itemStateChanged (ItemEvent ie)
		{
			Object obj=ie.getItemSelectable ();
			Checkbox cb=(Checkbox) obj;
			if (cb.getState ())
			{
				l1.setText ("U HAVE SELECTED : "+cb.getLabel ());
				String lab=cb.getLabel ();
				if (lab.equalsIgnoreCase ("exit"))
				{
					System.exit (0);
				}
			}
			else
			{
				l1.setText ("U HAVE SELECTED NONE");
			}
		}
	}
};
class RadioAppDemo
{
	public static void main (String [] args)
	{
		new RadioApp ();
	}
};
```

Day -46:

TEXTFIELD:

TextField is GUI interactive component which allows entering the data in single line. Whatever the data, we enter to text field, by default that data will be treated as string.

If we are able to read the data of text field then that text field is known as editable or non-echo or normal text field.

If we are unable to read the data of the text field then that text field is known as echo text field and the character which is available is known as echo character.

Creating a text box is nothing but creating an object of TextField class.

TextField API:

Constructors:

```
TextField ();
TextField (int);// int represents size of the text field
TextField (String);// data in the text field
TextField (String, int);
```

Instance methods:

```
public void setSize (int);
public int getSize ();
public void setText (String);
public String getText ();
public void setEchoChar (char);
public char getEchoChar ();
public void setEditable (boolean);
public boolean getEditable();
public boolean isEditable ();
public void addTextListener (TextListener);
public void removeTextListener (TextListener);
```

Write a java program which illustrates the concept of TextField?

Answer:

```
import java.awt.*;
import java.awt.event.*;
class Textfield extends Frame
{
	Label l1, l2, l3;
	TextField tf1, tf2, tf3;
	Button b1, b2, b3, b4;
	Textfield ()
	{
		setTitle ("Operations");
		setSize (200, 200);
		setLayout (new FlowLayout ());
		l1=new Label ("ENTER FIRST NUMBER");
		l2=new Label ("ENTER SECOND NUMBER");
		l3=new Label ("ENTER THIRD NUMBER");
		tf1=new TextField (15);
		tf2=new TextField (15);
		tf3=new TextField (15);
		b1=new Button ("Sum");
		b2=new Button ("Sub");
		b3=new Button ("Mul");
		b4=new Button ("Exit");
		add (l1);
		add (l2);
		add (l3);
		add (tf1);
		add (tf2);
		add (tf3);
		add (b1);
		add (b2);
		add (b3);
		add (b4);
		b1.addActionListener (new al());
		b2.addActionListener (new al());
		b3.addActionListener (new al());
		b4.addActionListener (new al());
		setVisible (true);
	}
	class al implements ActionListener
	{
		public void actionPerformed (ActionEvent ae)
		{
			String cap=ae.getActionCommand ();
			if (cap.equalsIgnoreCase ("Sum"))
			{
				String s1=tf1.getText ();
				String s2=tf2.getText ();
				int n1=Integer.parseInt (s1);
				int n2=Integer.parseInt (s2);
				int n3=n1+n2;
				String s=String.valueOf (n3);
				tf3.setText (s);
			}
			if (cap.equalsIgnoreCase ("Sub"))
			{
				String s1=tf1.getText ();
				String s2=tf2.getText ();
				int n1=Integer.parseInt (s1);
				int n2=Integer.parseInt (s2);
				int n3=n1-n2;
				String s=String.valueOf (n3);
				tf3.setText (s);
			}
			if (cap.equalsIgnoreCase ("Mul"))
			{
				String s1=tf1.getText ();
				String s2=tf2.getText ();
```

```
                        int n1=Integer.parseInt (s1);
                        int n2=Integer.parseInt (s2);
                        int n3=n1*n2;
                        String s=String.valueOf (n3);
                        tf3.setText (s);
                  }
                  if (cap.equalsIgnoreCase ("Exit"))
                  {
                        System.exit (0);
                  }
            }
      }
};
class TextfieldDemo
{
      public static void main (String [] args)
      {
            new Textfield ();
      }
};
```

Write a java program to close the window of awt?

<u>Answer</u>:

```
import java.awt.*;
import java.awt.event.*;
class CloseWin extends Frame
{
      CloseWin ()
      {
            setTitle ("Java Example");
            setBackground (Color.green);
            setSize (200, 200);
            this.addWindowListener (new WinAdap ());
            setVisible (true);
      }
      class WinAdap extends WindowAdapter
      {
            public void windowClosing (WindowEvent we)
            {
                  System.exit (0);
            }
      }
};
class CloseWinDemo
{
      public static void main (String [] args)
      {
            new CloseWin ();
      }
};
```

<u>Adapter class</u>:

It is one which contains null body definition for those methods which are inheriting from appropriate Listener.

In java.awt.event.* we have Listener interface called WidowListener which contains seven abstract methods. In the derived class implements WindowListener interface; it is mandatory for derived class to define all the methods even though the derived class is not required. If the derived class wants to define the required methods, it has to extend its corresponding adapter class called java.awt.event.WindowAdapter and this class contains null body definition for WindowListener interface methods.

Therefore which our Listener interface contains more than one undefined method for that Listener interfaces we have the appropriate adapter class whose general notation is XXXAdapter.

For example:

Listener	**Adapter**
WindowListener	WindowAdapter
ActionListener	WindowAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter

TEXTAREA:

TextArea is the GUI interactive component in which we can enter multiple lines of information such as addresses, remarks, achievements, etc.

Creating a TextArea is nothing but creating an object of TextArea class. Whenever we create a TextArea component by default Scrollbars will be in invisible mode. Whenever number of rows are exceeding automatically we get vertical scrollbar whereas when number of columns are exceeding we get horizontal scrollbar.

TextArea API:

Data members:

```
public static final int SCROLLBARS_NONE;
public static final int SCROLLBARS_BOTH;
public static final int SCROLLBARS_VERTICAL_ONLY;
public static final int SCROLLBARS_HORIZONTAL_ONLY;
```

Constructors:

```
TextArea ();
TextArea (int rows, int cols);
TextArea (int rows, int cols, int scroll visibility modifier);
```

Instance methods:

```
public void setRows (int rows);
public void setColums (int cols);
public int getRows ();
public int getCols ();
public void setScrollbarVisibility (int);
public int getScrollbarVisibility ();
public void setText (String);
public String getText ();
public void append (String);
public void replace (int start pos, int end pos, String);
public void addTextListener (TextListener); →registered
public void removeTextListener (TextListener); →unregistered
```

Write a java program which illustrates the concept of TextArea?

Answer:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet.*;
/*<applet code="TextareaApp" height=200 width=200>
</applet>*/
public class TextareaApp extends Applet
{
	Label l1, l2;
	TextField tf;
	TextArea ta;
	public void init ()
	{
		setBackground (Color.yellow);
		l1=new Label ("Enter a text");
		l2=new Label ("Copied text");
		tf=new TextField (20);
		ta=new TextArea ();
		add (l1);add (l2);add (tf);add (ta);
	}
	public void start ()
	{
		tf.addTextListener (new tl ());
	}
	class tl implements TextListener
	{
		public void textValueChanged (TextEvent te)
		{
			ta.setText ("");
			String s=tf.getText ();
			ta.append (s+"\n");
		}
	}
};
```

LIST:

List is the GUI interactive component which allows to select either single or multiple items at a time.

Creating List component is nothing but creating an object of List class.

List API:

Constructors:

```
List ();
List (int size);
List (int size, boolean mode);
```

Instance methods:

```
public void setSize (int size);
public int getSize ();
public void add (String);
public void addItem (String);
public void add (int, String);
public void addItem (int, String);
public String getSelectedItem ();
public String [] getSelectedItems ();
public int getSelectedIndex ();
public int [] getSelectedIndexes ();
public void remove (int index);
public void remove (String item);
public void removeAll ();
public void addItemListener (ItemListener); →single click of List item
public void removeItemListener (ItemListener); →single click of List item
public void addActionListener (ActionListener); →double click of List item
public void removeActionListener (ActionListener); →double click of List item
```

Day -47:

NOTE:

When we want to select the items of the list with single click then we must register with ItemListener to select items of the list with double click then we must register with ActionListener.

Write a java program which illustrates the concept of List?

Answer:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/*<applet code="ListApp" height=200 width=200>
</applet>*/
public class ListApp extends Applet
{
	Label l1, l2;
	List li;
	TextArea ta;
	public void init ()
	{
		setBackground (Color.yellow);
		l1=new Label ("Available fonts");
		l2=new Label ("Selected fonts");
		ta=new TextArea ();
		li=new List ();
		li.setMultipleMode (true);
		GraphicsEnvironment ge=GraphicsEnvironment.getLocalGraphicsEnvironment ();
		String s []=ge.getAvailableFontFamilyNames ();
		for (int i=0; i<s.length; i++)
		{
			li.add (s [i]);
		}
		add (l1);add (li);add (l2);add (ta);
	}
	public void start ()
	{
		li.addItemListener (new itl ());
		li.addActionListener (new atl ());
	}
```

```
      class itl implements ItemListener
      {
            public void itemStateChanged (ItemEvent ie)
            {
                  if (ie.getSource ()==li)
                  {
                        String s1 []=li.getSelectedItems ();
                        for (int j=0; j<s1.length; j++)
                        {
                              ta.append (s1 [j]+"\n");
                        }
                  }
            }
      }
      class atl implements ActionListener
      {
            public void actionPerformed (ActionEvent ae)
            {
                  if (ae.getSource ()==li)
                  {
                        String s2 []=li.getSelectedItems ();
                        for (int l=0; l<s2.length; l++)
                        {
                              ta.append (s2 [l]+"\n");
                        }
                  }
            }
      }
};
```

Mouse operation:

Mouse is the hardware component which always deals with user interactive with various active components. On mouse we can perform two types of operations, they are **basic operations** and **advanced operations**.

- To deal with *basic operations* of mouse we must take an interface called MouseListener. This interface contains the following undefined methods:

  ```
  public void mousePressed (MouseEvent);
  public void mouseClicked (MouseEvent);
  public void mouseExited (MouseEvent);
  public void mouseReleased (MouseEvent);
  public void mouseEntered (MouseEvent);
  ```

  In order to perform the basic operations of the mouse we must have the following registration method which are present in component class.

  ```
  public void addMouseListener (MouseListener);
  public void removeMouseListener (MouseListener);
  ```

- To perform *advanced operations* of mouse we must use a predefined interface called MouseMotionListener and it contains the following methods:

  ```
  public void mouseDragged (MouseEvent);
  public void mouseMoved (MouseEvent);
  ```

  In order to deal with advanced operations we must have the following registration methods:

  ```
  public void addMouseMotionListener (MouseMotionListener);
  public void removeMouseMotionListener (MouseMotionListener);
  ```

Write a java program which illustrates the basic and advanced operations of mouse?

Answer:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/*<applet code="mouse" width=300 height=300>
</applet>*/
public class mouse extends Applet
{
	String msg;
	int x,y;
	public void init (0
	{
		x=100;
		y=100;
		msg=null;
		setBackground (Color.yellow);
		setForeground (Color.green);
	}
	public void start ()
	{
		addMouseListener (new ml (this));
		addMouseMotionListener (new mml (this));
	}
	public void paint (Graphics g)
	{
		Font f=new Font ("verdana", Font.BOLD, 60);
		g.getFont (f);
		g.drawString (msg+"("+x+","+y+")",x,y);
	}
	class ml implements MouseListener
	{
		mouse m;
		ml (mouse m)
		{
			this.m=m;
		}
		public void mouseEntered (MouseEvent me)
		{
			m.showStatus ("MOUSE ENTERED IN THE WINDOW");
		}
		public void mouseExited (MouseEvent me)
		{
			m.showStatus ("MOUSE EXITED FROM WINDOW");
		}
		public void mousePressed (MouseEvent me)
		{
			m.msg="Mouse Pressed";
			m.x=me.getX ();
			m.y=me.getY ();
			m.repaint ();
		}
		public void mouseReleased (MouseEvent me)
		{
			m.msg="Mouse Released";
			m.x=me.getX ();
```

```
                        m.y=me.getY ();
                        m.repaint ();
                }
                public void mouseClicked (MouseEvent me)
                {
                        m.msg="Mouse Clicked";
                        m.x=me.getX ();
                        m.y=me.getY ();
                        m.repaint ();
                }
                class mml extends MouseMotionAdapter
                {
                        mouse m;
                        mml (mouse m)
                        {
                                this.m=m;
                        }
                        public void mouseMoved (MouseEvent me)
                        {
                                m.msg="Mouse Moved";
                                m.x=me.getX ();
                                m.y=me.getY ();
                                m.repaint ();
                        }
                        public void mouseDragged (MouseEvent me)
                        {
                                m.msg="Mouse Dragged";
                                m.x=me.getX ();
                                m.y=me.getY ();
                                m.repaint ();
                        }
                }
        }
};
```

<u>NOTE</u>:

```
java.applet.Applet;

public void showStatus (String); →1
public void repaint (); →2
```

Method 1 is used for setting the status of applet window and Method 2 is used for calling the paint method automatically for number of times.

<u>SCROLLBAR</u>:

Scrollbar is the GUI component which allows us to see invisible number of rows and invisible number of columns with vertical and horizontal scrollbars.

Write a java program which illustrates the concept of Scrollbar?

<u>Answer</u>:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="Scroll" width=300 height=300>
```

```
</applet>*/
public class Scroll extends Applet implements AdjustmentListener
{
	Scrollbar hsb, vsb;
	int hr, vr;
	public void init ()
	{
		hsb=new Scrollbar (Scrollbar.HORIZONTAL, 10, 100, 10, 1000);
		vsb=new Scrollbar (Scrollbar.VERTICAL, 10, 100, 10, 1000);
		add (hsb);
		add (vsb);
		hsb.addAdjustmentListener (this);
		vsb.addAdjustmentListener (this);
		setVisible (true);
	}
	public void adjustmentValueChanged (AdjustmentEvent ae)
	{
		hr=hsb.getValue ();
		vr=vsb.getValue ();
		repaint ();
	}
	public void paint (Graphics g)
	{
		g.setColor (Color.cyan);
		g.fillOval (20, 20, hr, vr);
	}
};
```

Day - 48:

IO STREAMS:

Generally, in JAVA programming we write two types of applications or programs. They are **volatile or non-persistent program** and **non-volatile or persistent program**.

- A *volatile program* is one whose result is always stored in main memory of the computer i.e., RAM. Whatever the data which is stored in the main memory of the computer that data will be temporary i.e., the data which is available in main memory is volatile.
- A *non-volatile program* is one whose result is stored in secondary storage devices i.e., hard disk, magnetic tapes, etc. the data which is stored in secondary storage devices is permanent. To store the data we have two approaches. They are **using files** and **using database**.

If we store the data permanently in the form of a file, then the data of that file can be modified by any unauthorized user. Hence, industry always recommends not storing the data in the files. Since, files do not provide any security. Hence, industry always recommends storing the data in the form of database. Since, most of the popular databases provide security in the form of user names and passwords.

In order to store the data permanently in the form of files we must use or import a package called java.io.*

- Collection of **records** is known as **file**. A **record** is a collection of **field values**.
- A **stream** is a flow of data between primary memory and secondary memory either locally (within the system) or globally (across the network) or [A **stream** is a flow of bytes between primary memory and secondary memory either locally (within the system) or globally (across the

network)] or [A **stream** is a flow of bits between primary memory and secondary memory either locally (within the system) or globally (across the network)].

Types of operations or modes on files:

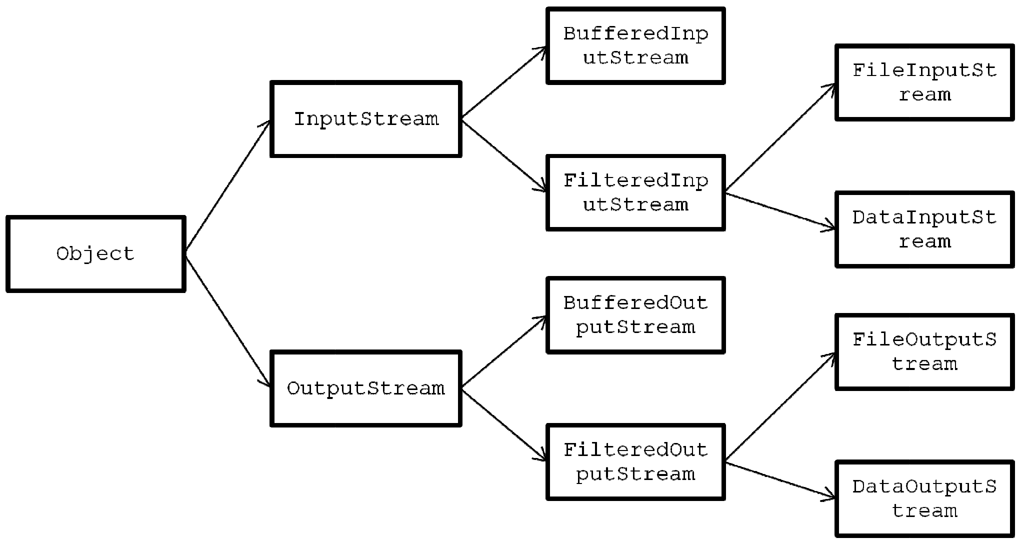
On files we perform two types of operations they are **read mode** and **write mode**. The following diagram will illustrate how to open the file either in read mode or in write mode or in both modes:

Types of streams in JAVA:

Based on transferring the data between primary memory to secondary memory and secondary memory to primary memory. In JAVA we have two types of streams they are byte streams and char streams.

Byte streams are those in which the data will be transferred one byte at a time between primary memory to secondary memory and secondary memory to primary memory either locally or globally. java.io.* package contains some set of classes and interfaces which will transfer one byte at a time.

Hierarchy chart for byte streams:



Day -49:

InputStream class:

This is an abstract class; hence we cannot create an object of this class directly. This class is basically used to open the file in *read* mode. In this class we have the following methods:

```
1. public int read ();
2. public int length (); // total size of the file
3. public int available (); // available number of bytes only
4. public void close ();
```

In JAVA end of file (EOF) is indicated by -1.

OutputStream class:

This is also an abstract class; hence we cannot create an object of this class directly. This class is used for opening the file in *write* mode. In this class we have the following methods:

```
1. public void write (int);
2. public int length ();
```

```
3. public void available ();
4. public void close ();
```

<u>FileInputStream class</u>:

This is the concrete (which we can create an object or it contains all defined methods) sub-class of all InputStream class. This class is always used to open the file in *read* mode. Opening the file in read mode is nothing but creating an object of FileInputStream class.

Constructor:

```
FileInputStream (String fname) throws FileNotFoundException
```

<u>For example</u>:

```
FileInputStream fis=new FileInputStream ("abc.txt");
```

If the file name abc.txt does not exist then an object of FileInputStream fis is **null** and hence we get FileNotFoundException. If the file name abc.txt is existing then the file abc.txt will be opened successfully in read mode and an object fis will point to beginning of the file.

<u>FileOutputStream class</u>:

This is the concrete sub-class of all OutputStream classes. This class is always used for opening the file in write mode is nothing but creating an object of FileOutputStream class.

Constructors:

```
1. FileOutputStream (String fname);
2. FileOutputStream (String fname, boolean flag);
```

If the flag is true the data will be appended to the existing file else if flag is false the data will be overlapped with the existing file data.

If the file is opened in write mode then the object of FileOutputStream will point to that file which is opened in write mode. If the file is unable to open in write mode an object of FileOutputStream contains null.

<u>Day -50</u>:

Write the following JAVA programs: 1) create a file that should contain 1 to 10 numbers. 2) read the data from the file which is created above.

<u>Answer</u>:

1)

```
import java.io.*;
class Fos
{
	public static void main (String [] args)
	{
		try
		{
			String fname=args [0];
			FileOutputStream fos=new FileOutputStream (fname, true);// mean append mode
			for (int i=1; i<=10; i++)
			{
				fos.write (i);
			}
			fos.close ();
		}
		catch (IOException ioe)
		{
			System.out.println (ioe);
		}
		catch (ArrayIndexOutOfBoundsException aiobe)
		{
			System.out.println ("PLEASE PASS THE FILE NAME..!");
		}
		System.out.println ("DATA WRITTEN..!");
	}
};
```

2)

```
import java.io.*;
class  Fis
{
	public static void main(String[] args)
	{
		try
		{
			String fname=args [0];
			FileInputStream fis=new FileInputStream (fname);
			int i;
			while ((i=fis.read ())!=-1)
			{
				System.out.println (i);
			}
			fis.close ();
		}
		catch (FileNotFoundException fnfe)
		{
			System.out.println ("FILE DOES NOT EXIST..!");
		}
		catch (IOException ioe)
		{
			System.out.println (ioe);
		}
		catch (ArrayIndexOutOfBoundsException aiobe)
		{
			System.out.println ("PLEASE PASS THE FILE NAME..!");
		}
	}
};
```

DataInputStream class:

This is used for two purposes. They are **reading the data from input device** like keyboard and **reading the data from remote machine** like server.

In order to perform the above operations we must create an object of DataInputStream class.

Constructors:

```
DataInputStream (InputStream);
```

For example:

```
DataInputStream dis=new DataInputStream (System.in);
```

An object of InputStream called '**in**' is created as a static data member in System class.

Instance methods:

```
1. public byte readByte ();
2. public char readChar ();
3. public short readShort ();
4. public int readInt ();
5. public long readLong ();
```

```
6. public float readFloat ();
7. public double readDouble ();
8. public boolean readBoolean ();
9. public String readLine ();
```

DataOutputStream class:

This is used for displaying the data onto the console and also used for writing the data to remote machine.

Constructor:

```
DataOutputStream (OutputStream);
```

For example:

```
DataOutputStream dos=new DataOutputStream (System.out);
```

Instance methods:

Old methods:

```
1. public void writeByte (byte);
2. public void writeChar (char);
3. public void writeShort (short);
4. public void writeInt (int);
5. public void writeLong (long);
6. public void writeFloat (float);
7. public void writeDouble (double);
8. public void writeBoolean (boolean);
9. public void writeLine (String);
```

New methods:

```
1. public void write (byte);
2. public void write (char);
3. public void write (short);
4. public void write (int);
5. public void write (long);
6. public void write (float);
7. public void write (double);
8. public void write (boolean);
9. public void write (String);
```

Write a JAVA program to read two numbers from keyboard and display their product (using older methods)?

Answer:

```
import java.io.*;
class DataRead
{
	public static void main (String [] args)
	{
		try
		{
			DataInputStream dis=new DataInputStream (System.in);
			System.out.println ("Enter first number : ");
			String s1=dis.readLine ();
			System.out.println ("Enter second number : ");
			String s2=dis.readLine ();
			int n1=Integer.parseInt (s1);
			int n2=Integer.parseInt (s2);
			int n3=n1*n2;
			System.out.println ("Product = "+n3);
		}
		catch (FileNotFoundException fnfe)
		{
			System.out.println ("FILE DOES NOT EXISTS");
```

```
        }
        catch (IOException ioe)
        {
              System.out.println (ioe);
        }
        catch (NumberFormatException nfe)
        {
              System.out.println ("PASS INTEGER VALUES ONLY");
        }
  }
};
```

Day - 51:

Implement copy command of DOS?

Answer:

```
import java.io.*;
class  Xcopy
{
      public static void main(String[] args)
      {
            FileInputStream fis=null;
            FileOutputStream fos=null;
            if (args.length!=2)
            {
                  System.out.println ("INVALID ARGUMENTS..!");
            }
            else
            {
                  try
                  {
                        fis=new FileInputStream (args [0]);
                        fos=new FileOutputStream (args [1], true);
                        int i;
                        do
                        {
                              i=fis.read ();
                              char ch=(char)i;// type casting int value into char value
                              fos.write (ch);
                        }
                        while (i!=-1);
                  }
                  catch (FileNotFoundException fnfe)
                  {
                        System.out.println (args [0]+"DOES NOT EXIST..!");
                  }
                  catch (IOException ioe)
                  {
                        System.out.println (ioe);
                  }
                  catch (Exception e)
                  {
                        e.printStackTrace ();
                  }
                  finally
                  {
                        try
                        {
                              if (fis!=null)// file is closed when it is opened
                              {
                                    fis.close ();
```

```
                                }
                                if (fos!=null)
                                {
                                      fos.close ();
                                }
                          }
                          catch (IOException ioe)
                          {
                                ioe.printStackTrace ();
                          }
                          catch (Exception e)
                          {
                                e.printStackTrace ();
                          }
                    }// finally
              }// else
        }//main
};
```

Implement type command of DOS?

<u>Answer</u>:

```
import java.io.*;
class DosType
{
      public static void main (String [] args)
      {
            FileInputStream fis=null;
            if (args.length!=2)
            {
                  System.out.println ("INVALID ARGS");
            }
            else
            {
                  try
                  {
                        fis=new FileInputStream (args [0]);
                        int i;
                        do
                        {
                              i=fis.read ();
                              char ch=(char) i;
                              System.out.println (ch);
                        }
                        while (i!=-1);
                  }
                  catch (FileNotFoundException fnfe)
                  {
                        System.out.println (args [0]+" DOES NOT EXIT");
                  }
                  catch (Exception e)
                  {
                        e.printStackTrace ();
                  }
                  finally
                  {
                        try
                        {
                              if (fis!=null)
                              {
                                    fis.close ();
                              }
                        }
                        catch (Exception e)
                        {
```

```
                                        e.printStackTrace ();
                                  }
                            }// finally
                      }// else
                }// main
};// DosType
```

<u>Serialization</u>: *Serialization is the process of saving the state of the object permanently in the form of a file*.

<u>Steps for developing serialization sub-class</u>:

1. Choose the appropriate package name.
2. Choose the appropriate class name whose object is participating in serialization.
3. Whichever class we have chosen in step 2 that must implements a predefined interface called java.io.Serializable (this interface does not contain any abstract methods and such type of interface is known as **marked** or **tagged interface**).
4. Choose set of properties or data members of the class.
5. Define set of setter methods (these are called **mutators** or **modifiers**).
6. Define set of getter methods (these are also known as **inspectors**).

Write a JAVA program for student serializable sub-class?

<u>Answer</u>:

```
package sp;
import java.io.*;
public class Student implements Serializable
{
      int stno;
      String sname;
      float marks;
      public void setSno (int stno)
      {
            this.stno=stno;
      }
      public void setSno (String sname)
      {
            this.sname=sname;
      }
      public void setSno (float marks)
      {
            this.marks=marks;
      }
      public int getStno ()
      {
            return (stno);
      }
      public String getSname ()
      {
            return (sname);
      }
      public float getMarks ()
      {
            return (marks);
      }
};
```

Day - 52:

SERIALIZATION PROCESS:

1. Create an object of serializable sub-class.

   For example:
   ```
   Student so=new Student ();
   ```

2. Accept the data either from keyboard or from command prompt.
3. Call set of set methods to set the values for the serializable sub-class (Student) object.

   For example:
   ```
   so.setSno (sno);
   ```

4. Choose the file name and open it into write mode with the help of FileOutputStream class.
5. Since an object of FileOutputStream class cannot write the entire object at a line to the file. In order to write the entire object at a time to the file we must create an object of ObjectOutputStream class and it contains the following Constructor:
   ```
   ObjectOutputStream (FileOutputStream);
   ```
   For example:
   ```
   ObjectOutputStream oos=new ObjectOutputStream (fos);
   ```
   The object oos is pointing to object fos, hence such type of streams are called chained or sequenced stream.

6. ObjectOutputStream class contains the following instance method, which will write the entire object at a time to the file.

   For example:
   ```
   ObjectOutpurStream.writeObject (so);
   ```

Write a JAVA program to save or serialize student data?

Answer:
```
import java.io.*;
class StudentData
{
	public static void main (String [] args)
	{
		try
		{
			sp.Student so=new sp.Student ();
			DataInputStream dis=new DataInputStream (System.in);
			System.out.println ("ENTER STUDENT NUMBER : ");
			int stno=Integer.parseInt (dis.readLine ());
			System.out.println ("ENTER STUDENT NAME : ");
			String sname=dis.readLine ();
			System.out.println ("ENTER STUDENT MARKS : ");
			float marks=Float.parseFloat (dis.readLine ());
			so.setStno (stno);
			so.setSname (sname);
			so.setMarks (marks);
			System.out.println ("ENTER THE FILE NAME TO WRITE THE DATA");
			String fname=dis.readLine ();
			FileOutputStream fos=new FileOutputStream (fname);
			ObjectOutputStream oos=new ObjectOutputStream (fos);
			oos.writeObject (so);
			System.out.println ("STUDENT DATA IS SERIALIZED");
			fos.close ();
			oos.close ();
		}
		catch (IOException ioe)
		{
```

```
                        System.out.println ("PROBLEM IN CREATING THE FILE");
                }
                catch (Exception e)
                {
                        System.out.println (e);
                }
        }
};
```

Day – 53:

DESERIALIZATION PROCESS: *De-serialization is a process of reducing the data from the file in the form of object.*

Steps for developing deserialization process:

1. Create an object of that class which was serialized.
   For example:
   ```
   Student so=new Student ();
   ```

2. Choose the file name and open it into read mode with the help of FileInputStream class.
   For example:
   ```
   FileInputStream fis=new FileInputStream (“Student”);
   ```

3. Create an object of ObjectInputStream class. The Constructor of ObjectInputStream class is taking an object of FileInputStream class.
   For example:
   ```
   ObjectInputStream ois=new ObjectInputStream (fis);
   ```

4. ObjectInputStream class contains the following method which will read the entire object or record.
   ```
          public Object readObject ();
   ```
   For example:
   ```
   Object obj=ois.readObject ();
   ```

5. Typecast an object of java.lang.Object class into appropriate Serializable sub-class object for calling get methods which are specially defined in Serializable sub-class.
   For example:
   ```
   So= (Student) obj;
   ```

6. Apply set of get methods for printing the data of Serializable sub-class object.
   For example:
   ```
   int stno=so.getStno ();
   String sname=so.getSname ();
   flaot marks=so.getMarks ();
   ```

7. Close the chained stream.
   For example:
   ```
   fis.close ();
   ois.close ();
   ```

Write a JAVA program to retrieve or de-serialize student data?

Answer:
```
import java.io.*;
import sp.Student;
class DeSerial
{
```

```
      public static void main (String [] args)
      {
            try
            {
                  Student so=new Student ();
                  DataInputStream dis=new DataInputStream (System.in);
                  System.out.println ("ENTER FILE NAME TO READ");
                  String fname=dis.readLine ();
                  FileInputStream fis=new FileInputStream (fname);
                  ObjectInputStream ois=new ObjectInputStream (fis);
                  Object obj=dis.readObject ();
                  so=(Object) obj;
                  System.out.println ("STUDENT NUMBER "+so.getStno ());
                  System.out.println ("STUDENT NAME "+so.getSname ());
                  System.out.println ("STUDENT MARKS "+so.getMarks ());
                  fis.close ();
                  ois.close ();
            }
            catch (FileNotFoundException fnfe)
            {
                  System.out.println ("FILE DOES NOT EXISTS");
            }
            catch (Exception e)
            {
                  System.out.println (e);
            }
      }
};
```

Types of serialization:

In java we have four types of serialization; they are **complete serialization, selective serialization, manual serialization** and **automatic serialization**.

- *Complete serialization* is one in which all the data members of the class participates in serialization process.
- *Selective serialization* is one in which few data members of the class or selected members of the class are participating in serialization process.

  In order to avoid the variable from the serialization process, make that variable declaration as transient i.e., transient variables never participate in serialization process.
- *Manual serialization* is one in which the user defined classes always implements java.io.Serializable interface.
- *Automatic serialization* is one in which object of sub-class of Serializable sub-class participates in serialization process.

```
class RegStudent extends Student
{
      ………………..
      ………………..
};
```

Buffered Streams:

Buffered streams are basically used to reduce physical number of read and write operation i.e., buffered streams always increases the performance of ordinary streams.

In byte streams we have two buffered streams, they are **BufferedInputStream** and **BufferedOutputStream**.

BufferedInputStream:

BufferedInputStream class is used for reducing number of physical read operation. When we create an object of BufferedInputStream, we get a temporary peace of memory space whose default size is 1024 bytes and it can be increased by multiples of 2.

Constructor:
```
public BufferedInputStream (FileInputStream);
```
For example:
```
FileInputStream fis=new FileInputStream ("abc.dat");
BufferedInputStream bis=new BufferedInputStream (fis);
```

BufferedOutputStream:

BufferedOutputStream class is used for reducing number of physical write operation when we create an object of BufferedOutputStream, we get a temporary peace of memory space whose default size is 1024 bytes and it can be increased by multiple of 2.

Constructor:
```
public BufferedOutputStream (FileOutputStream);
```
For example:
```
FileOutputStream fos=new FileOutputStream ("abc.dat");
BufferedOutputStream bos=new BufferedOutputStream (fos);
```

Day - 54:

Byte streams will transfer one byte of data and they will be implemented in system level applications such as flow of data in electronic circuits, development of ftp protocol etc.

Character streams are those in which 2 bytes of data will be transferred and it can be implemented in higher level applications such as internet applications, development of protocols etc., like http etc.

MULTI THREADING:

Thread:

1. A flow of control is known as thread.
2. If a program contains multiple flow of controls for achieving concurrent execution then that program is known as multi threaded program.
3. A program is said to be a multi threaded program if and only if in which there exist 'n' number of sub-programs there exist a separate flow of control. All such flow of controls are executing concurrently such flow of controls are known as threads and such type of applications or programs is called multi threaded programs.

The languages like C, C++ comes under single threaded modeling languages, since there exist single flow of controls where as the languages like JAVA, DOT NET are treated as multi threaded modeling languages, since there is a possibility of creating multiple flow of controls.

When we write any JAVA program there exist two threads they are fore ground thread and back ground thread.

Fore ground threads are those which are executing user defined sub-programs where as back ground threads are those which are monitoring the status of fore ground thread. There is a possibility of creating 'n' number of fore ground threads and always there exist single back ground thread.

Multi threading is the specialized form of multi tasking of operating system.

Day - 55:

In information technology we can develop two types of applications. They are **process based applications** and **thread based applications**.

Process Based Applications	**Thread Based Applications**
1. It is the one in which there exist single flow of control.	1. It is the one in which there exist multiple flow of controls.
2. All C, C++ applications comes under it.	2. All JAVA, DOT NET applications comes under it.
3. Context switch is more (context switch is the concept of operating system and it says switching the control from one address page to another address page).	3. Context switch is very less.
4. For each and every sub-program there exist separate address pages.	4. Irrespective of 'n' number of sub-programs there exist single address page.
5. These are treated as heavy weight components.	5. These are treated as light weight components.
6. In this we can achieve only sequential execution and they are not recommending for developing internet applications.	6. In thread based applications we can achieve both sequential and concurrent execution and they are always recommended for developing interact applications.

States of a thread:

When we write any multi threading applications, there exist 'n' numbers of threads. All the threads will under go different types of states. In JAVA for a thread we have five states. They are **new, ready, running, waiting** and **halted** or **dead** state.

New state: It is one in which the thread about to enter into main memory.
Ready state: It is one in which the thread is entered into memory space allocated and it is waiting for CPU for executing.
Running state: A state is said to be a running state if and only if the thread is under the control of CPU.
Waiting state: It is one in which the thread is waiting because of the following factors:

a) For the repeating CPU burst time of the thread (CPU burst time is an amount of the required by the thread by the CPU).
b) Make the thread to sleep for sleep for some specified amount of time.
c) Make the thread to suspend.
d) Make the thread to wait for a period of time.
e) Make the thread to wait without specifying waiting time.

Halted state: It is one in which the thread has completed its total execution.

As long as the thread is in new and halted states whose execution status is false where as when the thread is in ready, running and waiting states that execution states is true.

Creating a thread:

In order to create a thread in JAVA we have two ways. They are **by using java.lang.Thread class** and **by using java.lang.Runnable interface**.

In multi threading we get only one exception known as **java.lang.InterruptedException**.

Day - 56:

Using java.lang.Thread:

Creating a flow of control in JAVA is nothing but creating an object of java.lang.Thread class. An object of Thread class can be created in three ways. They are:

i) **Directly** `Thread t=new Thread ();`
ii) **Using factory method** `Thread t1=Thread.currentThread ();`
iii) **Using sub-class of Thread class**

```
class C1 extends Thread
{
        ……………………;
        ……………………;
};
C1 o1=new C1 ();
Thread t1=new C1 ();
```

Here, C1 is the sub-class of Thread class.

Thread API:

```
public static final int MAX_PRIORITY (10);
public static final int MIN_PRIORITY (1);
public static final int NORM_PRIORITY (5);
```

The above data members are used for setting the priority to threads are created. By default, whenever a thread is created whose default priority NORM_PRIORITY.

Constructors:

i) **Thread ():** With this Constructor we can create an object of the Thread class whose default thread name is Thread-0.
   For example:
   ```
   Thread t=new Thread ();
   System.out.println (t.getName ());// Thread-0
   ```

ii) **Thread (String):** This Constructor is used for creating a thread and we can give the user specified thread name.
   For example:
   ```
   Thread t=new Thread (“JAVA”);
   t.setName (“JAVA”);
   t.setPriority (Thread.MAX_PRIORITY);
   ```

iii) **Thread (Runnable):** This Constructor is used for converting Runnable object into Thread object for entering into run method of Runnable interface by making use of start method of Thread class without giving thread name.

iv) **Thread (Runnable, String):** This Constructor is similar to above Constructor but we give thread name through this Constructor.

Instance methods:

```
public final void setName (String);
public final String getName ();
```

The above two methods are used for setting the name of the thread and getting the name from the thread respectively.

For example:
```
Thread t1=new Thread ();
T1.setName ("JAVA");
String tp=t1.getName ();
System.out.println (tp);// JAVA

public final void setPriority (int);
public final int getPriority ();
```

The above two methods are used for setting the priority to the thread and getting the priority of the thread respectively.

For example:
```
Thread t1=new Thread ();
Int pri=t1.getPriority ();
System.out.println (pri);// 5 → by default
t1.setPriority (Thread.MAX_PRIORITY);
pri=t1.getPriority ();
System.out.println (pri); // 10
```

public void run ():

Any JAVA programmer want to define a logic for the thread that logic must be defined only run () method. When the thread is started, the JVM looks for the appropriate run () method for executing the logic of the thread. Thread class is a concrete class and it contains all defined methods and all these methods are being to final except run () method. run () method is by default contains a definition with null body. Since we are providing the logic for the thread in run () method. Hence it must be overridden by extending Thread class into our own class.

For example:
```
class C1 extends Thread
{
      public void run ()
      {
      ………………………;
      ………………………;
      }
};
```

public final void start ():

This is the method which is used for making the Thread to start to execute the thread logic. The method start is internally calling the method run ().

For example:
```
Thread t1=new Thread ();
t1.start ();
Thread t2=Thread.currentThread ();
t2.start ();
```

public final void suspend ():

This method is used for suspending the thread from current execution of thread. When the thread is suspended, it sends to waiting state by keeping the temporary results in **process control block** (PCB) or **job control block** (JCB).

public <u>final void resume ()</u>:

This method is used for bringing the suspended thread from waiting state to ready state. When the thread is resumed to start executing from where it left out previously by retrieving the previous result from PCB.

public <u>final void stop ()</u>:

This method is used to stop the execution of the current thread and the thread goes to halted state from running state. When the thread is restarted it starts executing from the beginning only.

public <u>final void wait (long msec)</u>:

This method is used for making the currently executing thread into waiting state for a period of time. Once this period of time is over, automatically the waiting thread will enter into ready state from waiting state.

<u>Day - 57</u>:

<u>Static methods</u>:

i) **public static void sleep (long msec) throws InterruptedException** method is used (waiting state). If the sleep time is over automatically thread will come from waiting state to ready state.

<u>For example</u>:
```
Thread.sleep (1000);
```

ii) **public static Thread currentThread ()** is used for obtaining the threads which are running in the main memory of the computer.

<u>For example</u>:
```
Thread t=Thread.currentThread ();
System.out.println (t);// Thread [main (fat), 5, main (bat)]
```

Write a JAVA program to find the threads which are running internally and print priority values.

Answer:
```
class ThDemo
{
	public static void main (String [] args)
	{
		Thread t=Thread.currentThread ();
		System.out.println (t);
		t.setName ("ABC");
		System.out.println (t);
		System.out.println ("IS IT ALIVE..?"+t.isAlive ());// true
		Thread t1=new Thread ();// new state
		System.out.println ("IS IT ALIVE..?"+t.isAlive ());// false
		System.out.println ("DEFAULT NAME OF THREAD = "+t1.getName ());// Thread-0
		System.out.println ("MAXIMUM PRIORITY VALUE = "+Thread.MAX_PRIORITY);// 10
		System.out.println ("MINIMUM PRIORITY VALUE = "+Thread.MIN_PRIORITY);// 1
		System.out.println ("NORMAL PRIORITY VALUE = "+Thread.NORM_PRIORITY);// 5
	}
};
```

Output:
```
Thread[main,5,main]
Thread[ABC,5,main]
IS IT ALIVE..?true
IS IT ALIVE..?true
DEFAULT NAME OF THREAD = Thread-0
MAXIMUM PRIORITY VALUE = 10
```

```
MINIMUM PRIORITY VALUE = 1
NORMAL PRIORITY VALUE = 5
```

<u>Instance methods</u>:

public boolean isAlive () method is used for checking whether the thread is executing or not. It returns 'true' as long as the thread is in ready running and waiting states. It returns 'false' as long as the thread is in new and halted state.

Write a thread program which displays 1 to 10 numbers after each and every 1 second.

Answer:

```
class Th1 extends Thread
{
	public void run ()
	{
		try
		{
			for (int i=1; i<=10; i++)
			{
				System.out.println ("VALUE OF I = "+i);
				Thread.sleep (1000);
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
	}
};
class ThDemo1
{
	public static void main (String [] args)
	{
		Th1 t1=new Th1 ();
		System.out.println ("IS T1 ALIVE BEFORE START = "+t1.isAlive ());
		t1.start ();
		System.out.println ("IS T1 ALIVE AFTER START = "+t1.isAlive ());
	}
};
```

Output:

```
IS T1 ALIVE BEFORE START = false
IS T1 ALIVE AFTER START = true
VALUE OF I = 1
VALUE OF I = 2
VALUE OF I = 3
VALUE OF I = 4
VALUE OF I = 5
VALUE OF I = 6
VALUE OF I = 7
VALUE OF I = 8
VALUE OF I = 9
VALUE OF I = 10
```

<u>Day - 58</u>:

Re-write the above program using runnable interface.

Answer:

```
class Th1 implements Runnable
{
	public void run ()
	{
		try
		{
			for (int i=1; i<=10; i++)
			{
				System.out.println ("VALUE OF I = "+i);
				Thread.sleep (1000);
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
	}
};
class ThDemo2
{
	public static void main (String [] args)
	{
		Runnable t=new Th1 ();
		Thread t1=new Thread (t, "ABC");
		System.out.println ("THREAD NAME = "+t1.getName ());
		System.out.println ("IS T1 ALIVE BEFORE START = "+t1.isAlive ());
		t1.start ();
		System.out.println ("IS T1 ALIVE AFTER START = "+t1.isAlive ());
	}
};
```

Output:

```
THREAD NAME = ABC
IS T1 ALIVE BEFORE START = false
IS T1 ALIVE AFTER START = true
VALUE OF I = 1
VALUE OF I = 2
VALUE OF I = 3
VALUE OF I = 4
VALUE OF I = 5
VALUE OF I = 6
VALUE OF I = 7
VALUE OF I = 8
VALUE OF I = 9
VALUE OF I = 10
```

Write a JAVA program which produces 1 to 10 numbers in which even numbers are produced by one thread and odd numbers are produced by another thread.

Answer:

```
class Th1 extends Thread
{
	public void run ()
	{
		try
		{
			for (int i=1; i<=10; i+=2)
			{
				System.out.println ("VALUE OF ODD : "+i);
				Thread.sleep (1000);
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
```

```
      }
};
class Th2 implements Runnable
{
      public void run ()
      {
            try
            {
                  for (int j=2; j<=10; j+=2)
                  {
                        System.out.println ("VALUE OF EVEN : "+j);
                        Thread.sleep (1000);
                  }
            }
            catch (InterruptedException ie)
            {
                  System.out.println (ie);
            }
      }
};
class ThDemo6
{
      public static void main (String [] args)
      {
            Th1 t1=new Th1 ();// object of Thread class
            Th2 t2=new Th2 ();// object of Runnable class
            Thread t=new Thread (t2);// Runnable is converted into Thread object
            System.out.println ("BEFORE START T1 IS : "+t1.isAlive ());
            System.out.println ("BEFORE START T2 IS : "+t.isAlive ());
            t1.start ();
            t.start ();
            System.out.println ("AFTER START T1 IS : "+t1.isAlive ());
            System.out.println ("AFTER START T2 IS : "+t.isAlive ());
            try
            {
                  t1.join ();// to make thread to join together for getting performance
                  t.join ();
            }
            catch (InterruptedException ie)
            {
                  System.out.println (ie);
            }
            System.out.println ("AFTER JOINING T1 IS : "+t1.isAlive ());
            System.out.println ("AFTER JOINING T2 IS : "+t.isAlive ());
      }
};
```

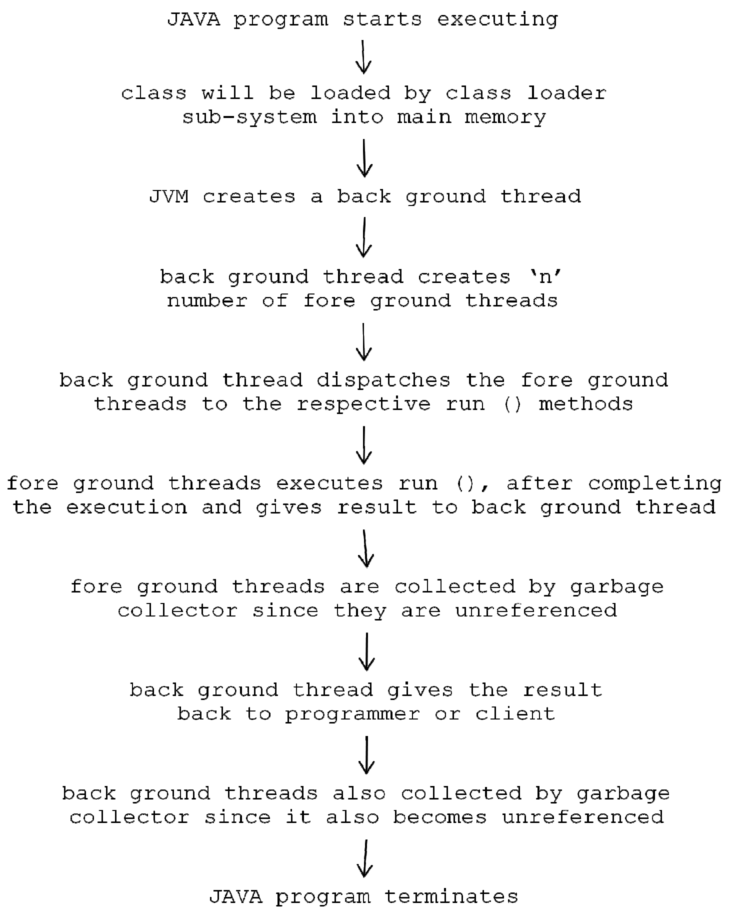
Output:

```
BEFORE START T1 IS : false
BEFORE START T2 IS : false
AFTER START T1 IS : true
AFTER START T2 IS : true
VALUE OF ODD : 1
VALUE OF EVEN : 2
VALUE OF ODD : 3
VALUE OF EVEN : 4
VALUE OF ODD : 5
VALUE OF EVEN : 6
VALUE OF ODD : 7
VALUE OF EVEN : 8
VALUE OF ODD : 9
VALUE OF EVEN : 10
AFTER JOINING T1 IS : false
AFTER JOINING T2 IS : false
```

public <u>void join () Throws InterruptedException</u>:

This method is used for making the fore ground threads to join together so that JVM can call the garbage collector only one time for collecting all of them instead of collecting individually.

How a thread executes internally?
Answer:


JAVA program starts executing
class will be loaded by class loader sub-system into main memory
JVM creates a back ground thread
back ground thread creates 'n' number of fore ground threads
back ground thread dispatches the fore ground threads to the respective run () methods
fore ground threads executes run (), after completing the execution and gives result to back ground thread
fore ground threads are collected by garbage collector since they are unreferenced
back ground thread gives the result back to programmer or client
back ground threads also collected by garbage collector since it also becomes unreferenced
JAVA program terminates

<u>Day -59</u>:

<u>SYNCHRONIZATION</u>:

It is the process of allowing only one thread at a time among 'n' number of threads into that area which is sharable to perform write operation.

If anything (either data members or methods or objects or classes) is sharable then we must apply the concept of synchronization.

Let us assume that there is a sharable variable called balance whose initial value is zero. There are two threads t1 and t2 respectively. t1 and t2 want to update the balance variable with their respective values i.e., 10 and 20 at the same time. After completion of these two threads the final value in the balance is either 10 or 20 but not 30 which is the inconsistent result. To achieve the consistent result we must apply the concept of synchronization.

When we apply synchronization concept on the above scenario, when two threads are started at same time, the first thread which is started is given a chance to update balance variable with its respective value. When second thread is trying to access the balance variable value JVM makes the second thread to wait until first thread completes its execution by locking balance variable value. After completion of first thread the value in the balance is 10 and second thread will be allowed to update balance variable value. After completion of second thread the value of the balance is 30, which is the consistent result. Hence, in synchronization locking and unlocking is taking place until all the threads are completed their execution.

Synchronization Techniques:

In JAVA we have two types of synchronization techniques. They are **synchronized methods** and **synchronized blocks**.

1. Synchronized methods:

If any method is sharable for 'n' number of threads then make the method as synchronized by using a keyword **synchronized**.

In JAVA we have two types of synchronized methods. They are **synchronized Instance methods** and **synchronized static methods**.

Synchronized Instance methods: If the ordinary instance method is made it as synchronized then the object of the corresponding class will be locked.

Syntax:
```
synchronized <return type> method name (method parameters if any)
{
	Block of statements;
}
```

For example:
```
class Account
{
	int bal=0;
	synchronized void deposit (int amt)
	{
		bal=bal+amt;
		System.out.println ("CURRENT BALANCE = "+bal);
	}
};
```

Synchronized static method: If an ordinary static method is made it as synchronized then the corresponding class will be locked.

Syntax:
```
synchronized static <return type> method name (method parameters if any)
{
	Block of statements;
}
```

For example:
```
class Account
{
	static int bal=0;
	synchronized static void deposit (static int amt)
	{
		bal=bal+amt;
		System.out.println ("CURRENT BALANCE = "+bal);
	}
};
```

Day -60:

2. Synchronized block:

This is an alternative technique for obtaining the concept of synchronization instead of synchronized methods.

When we inherit non-synchronized methods from either base class or interface into the derived class, we cannot make the inherited method as synchronized. Hence, we must use synchronized blocks.

Syntax:
```
synchronized (object of current class)
{
       Block of statement(s);
}
```

For example:
```
class BankOp
{
       void deposit (int amt);// p a instance
};
class Account implements BankOp
{
       int bal=0;
       public void deposit (int amt)
       {
              synchronized (this);
              {
                     bal=bal+amt;
                     System.out.println ("current value="+bal);
              }
       }
};
```

Write a synchronization program in which there exist an account, there exist 'n' number of customers and all the customers want to deposit 10 rupees to the existing balance of account.

Answer:
```
class Account
{
       private int bal=0;
       synchronized void deposit (int amt)
       {
              bal=bal+amt;
              System.out.println ("CURRENT BALANCE="+bal);
       }
       int getBal ()
       {
              return (bal);
       }
};
class cust extends Thread
{
       Account ac;// has-a relationship
       cust (Account ac)
       {
              this.ac=ac;
       }
       public void run ()
       {
              ac.deposit (10);
       }
};
class SyncDemo
{
	public static final int noc=5;
	public static void main (String [] args)
	{
		Account ac=new Account ();
		cust cu []=new cust [noc];
		for (int i=0; i<noc; i++)
		{
			cu [i]=new cust (ac);// giving account object 'ac' to each & every customer object
		}
		for (int i=0; i<noc; i++)
		{
			cu [i].start ();
		}
		for (int i=0; i<noc; i++)
		{
			System.out.println ("IS ALIVE..? "+cu [i].isAlive ());
		}
		try
		{
			for (int i=0; i<noc; i++)
			{
				cu [i].join ();
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
		for (int i=0; i<noc; i++)
		{
			System.out.println ("IS ALIVE..? "+cu [i].isAlive ());
		}
		System.out.println ("TOTAL BALANCE="+ac.getBal ());
	}
};
```

Output:

```
IS ALIVE..? true
IS ALIVE..? true
IS ALIVE..? true
IS ALIVE..? true
CURRENT BALANCE=10
CURRENT BALANCE=20
CURRENT BALANCE=30
CURRENT BALANCE=40
CURRENT BALANCE=50
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
TOTAL BALANCE=50
```

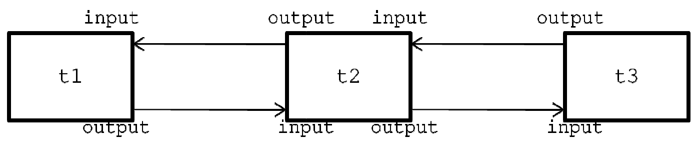
NOTE:

If we execute the above program on single user operating systems it is mandatory for the JAVA programmer to write synchronized keyword since user operating system cannot take care about synchronization by default. Whereas if we run the above program in multi user or threaded operating systems we need not to use a keyword synchronized (optional).

It is always recommended to write synchronized keyword irrespective which ever operating system we use.

Inter thread communication:

If two or more threads are exchanging the data then that communication is known as inter thread communication. In inter thread communication; an output of one thread is given as an input to another thread. In order to develop inter thread communication applications we must make use of a class called java.lang.Object.

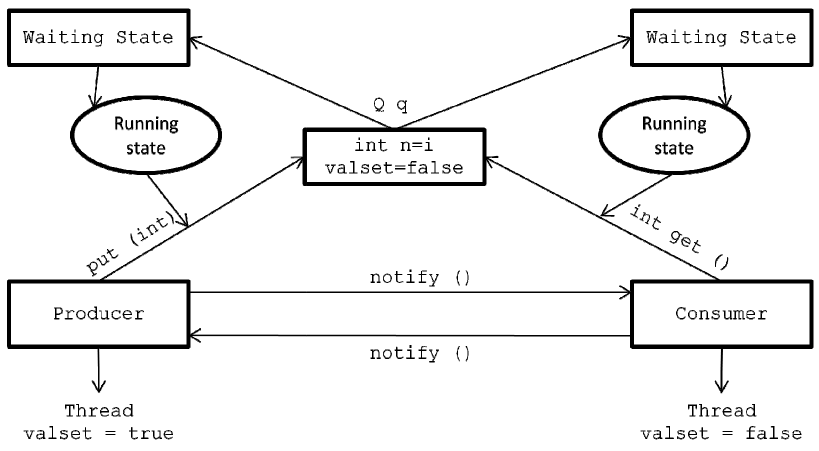


Methods in java.lang.Object:

```
1. public void wait (long msec)
2. public void wait (long msec, int nanosec)
3. public void wait ()
4. public void notify ()
5. public void notifyAll ()
```

Methods 1 and 2 are used for making the thread to wait for some period of time. Once the waiting time is over automatically the thread will come from waiting state to ready state. Method 3 is used for making the thread to wait without specifying waiting time. Method 4 is used for bringing a single waiting thread into ready state. Method 5 is used for bringing all the threads from waiting state to ready state.

Day - 61:



Develop producer consumer program by using inter thread communication?

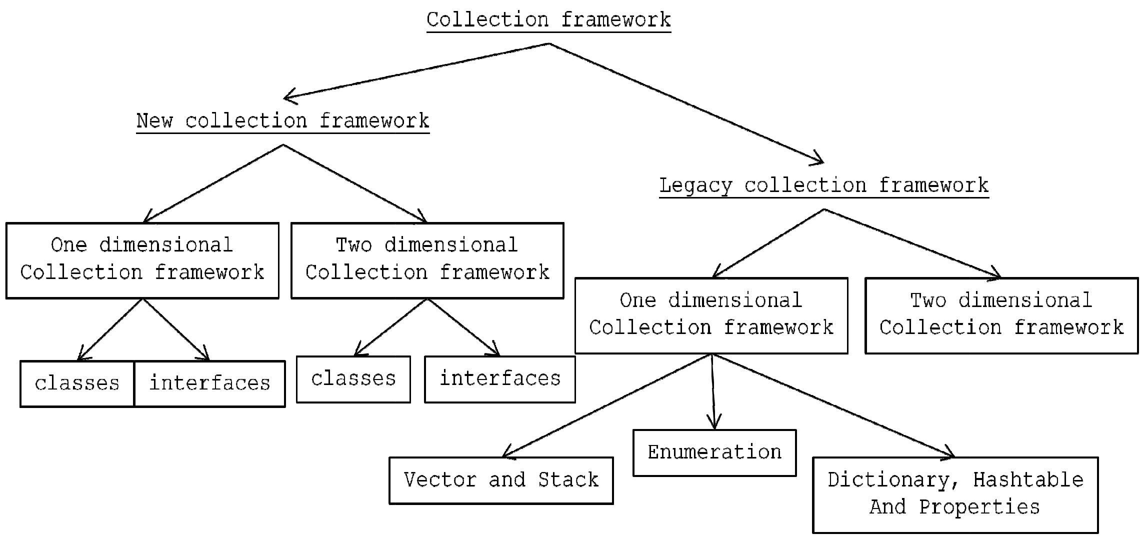
Answer:

```
class Q
{
	int n;
	boolean valset;
	synchronized void put (int i)
	{
		try
		{
			if (valset)
			{
				wait ();
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
		n=i;
		System.out.println ("PUT="+i);
		valset=true;
		notify ();
	}// put
	synchronized int get ()
	{
		try
		{
			if (!valset)
			{
				wait ();
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
		System.out.println ("GET="+n);
		valset=false;
		notify ();
		return (n);
	}// get
};// Q
class Producer implements Runnable
{
	Q q;
	Thread t;
	Producer (Q q)
	{
		this.q=q;
		t=new Thread (this, "Producer");
		t.start ();
	}
	public void run ()
	{
		int i=0;
		System.out.println ("NAME OF THE THREAD = "+t.getName ());
		while (true)
		{
			q.put (++i);
		}
	}
};// Producer
class Consumer implements Runnable
{
	Q q;
	Thread t;
	Consumer (Q q)
	{
		this.q=q;
		t=new Thread (this, "Consumer");
		t.start ();
	}
	public void run ()
	{
		System.out.println ("NAME OF THE THREAD = "+t.getName ());
		while (true)
		{
			int i=q.get ();
		}
	}
};// Consumer
class PCDemo
{
	public static void main (String [] args)
	{
		Q q=new Q ();
		Producer p=new Producer (q);
		Consumer c=new Consumer (q);
	}
};// PCDemo
```

Day - 62:

COLLECTION FRAMEWORK:

Collection framework is the standardized mechanism of grouping of similar or dissimilar type of objects into a single object. This single object is known as collection framework object.


Collection framework
New collection framework
Legacy collection framework
One dimensional Collection framework
Two dimensional Collection framework
classes
interfaces
classes
interfaces
One dimensional Collection framework
Two dimensional Collection framework
Vector and Stack
Enumeration
Dictionary, Hashtable And Properties

Goals of collection frameworks:

1. Collection framework improves the **performance** of JAVA, J2EE projects (when we want to transfer the bulk amount of data from client to server and server to client, using collection framework we can transfer that entire data at a time).
2. Collection framework allows us to prove **similar or dissimilar** type of objects.
3. Collection framework is dynamic in nature i.e., they are **extends** (arrays contains the size which is fixed in nature and they allows similar type of data).
4. Collection framework contains **adaptability** feature (adaptability is the process of adding one collection object at the end of another collection object).
5. Collection framework is **algorithmic oriented** (collection framework contains various sorting and searching techniques of data structures as a predefined concepts).

6. In order to deal with collection framework we must import a package called **java.util.***

Collection framework is divided into two categories. They are **new collection framework** and **legacy** (**old**) **collection framework**.

NEW COLLECTION FRAMEWORK:

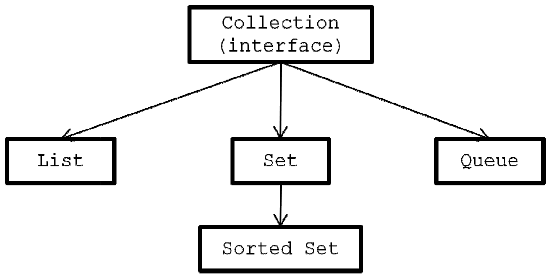
New collection framework is again broadly divided into two categories. They are **one dimensional collection framework** and **two dimensional collection framework**.

One dimensional collection framework:

A one dimensional collection framework is one in which the data is organized either in the form of row or in the form of column by containing similar or dissimilar categories of objects into a single object. This single object is known as one dimensional collection framework object. One dimensional collection framework contains **one dimensional collection framework interfaces** and **one dimensional collection framework classes**.

a) **One dimensional collection framework interfaces:**

As a part of one dimensional collection framework interfaces in JAVA we have five interfaces. They are **collection, list, set, sorted set** and **queue**.



Day - 63:

java.util.Collection:

Collection is an interface whose object allows us to organize similar or different type of objects into single object. The Collection interface is having the following features:

i) It is available at the top of the hierarchy of all the interfaces which are available in the collection framework.
ii) An object of Collection allows us to add duplicate elements.
iii) Collection object always displays the data in forward direction only.
iv) Collection object will print the data on the console in random order.
v) Collection object always allows us to insert an element at end only i.e., we cannot insert an element at the specific position.

java.util.List:

i) List is the sub-interface of **java.util.Collection** interface.
ii) List object also allows us to add duplicates.
iii) List object automatically displays the data in sorted order.
iv) List object allows us to add an element either at the ending position or at specific position.
v) List object allows us to retrieve the data in forward direction, backward direction and random retrieval.

java.util.Set:

i) Set is the sub-interface of **java.util.Collection** interface.
ii) An object of Set does not allows duplicates i.e., all the elements in the set must be distinct (unique).
iii) Set object always displays the data in random order.
iv) Set object allows us to add the elements only at ending position.
v) Set object allows us to retrieve the data only in forward direction.

java.util.SortedSet:

i) SortedSet is the sub-interface of **java.util.Set** interface.
ii) SortedSet object does not allows duplicates.
iii) SortedSet object will displays the data automatically in sorted order.
iv) SortedSet object allows us to add the elements only at ending position.
v) SortedSet object allows us to retrieve only in forward direction.

java.util.Queue

Methods in Collection interface:

1. **public int size ():** This method is used for determining the number of elements in Collection interface object.
2. **public boolean add (java.lang.Object):** This method is used for adding an object to Collection object. When we use this method to add the object to Collection objects and List object, this method always returns true. Since, Collection object and List object allows duplicates. When we apply the same method with Set and SortedSet methods and when we are trying to add duplicates, this method returns false.

NOTE:

Whatever the data we want to add through collection framework object that fundamental data must be converted into the corresponding wrapper class object and all the objects of wrapper classes are treated as objects of **java.lang.Object** class.

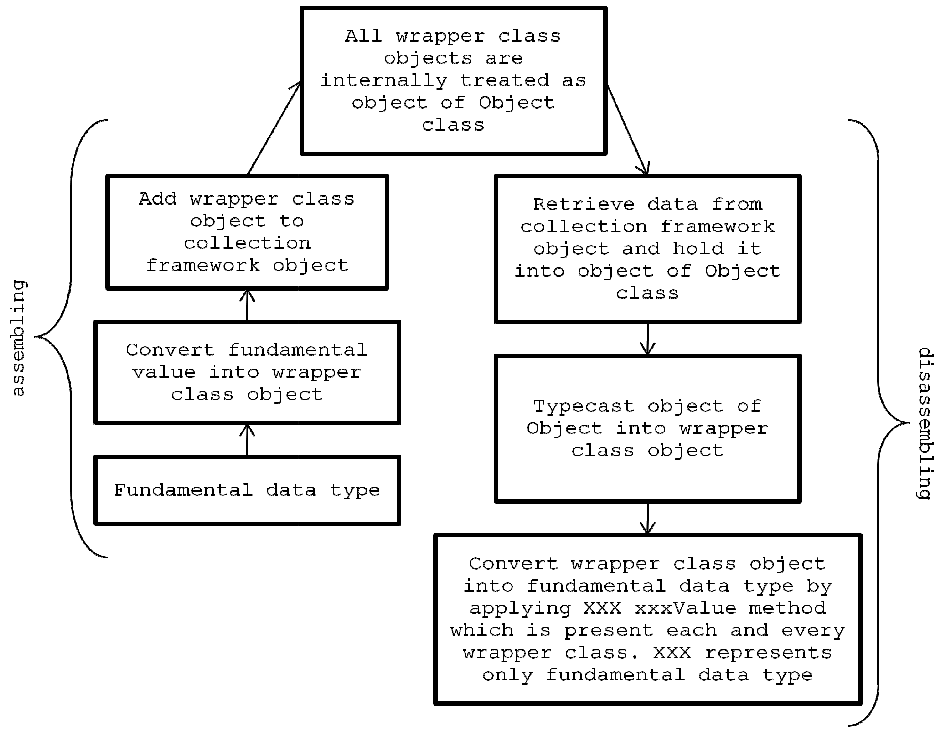
Day - 64:

3. public boolean addAll (Collection):

This method is used for adding the entire collection object at the end of another Collection object. As long as we apply this method with List and Collection interfaces, it returns true. Since, they allow duplicates. When we apply this method with Set and SortedSet, this method may return false. Since, duplicates are not allowed.

NOTE:

Collection framework is nothing but assembling different or similar type of objects and dissembling similar or different type of objects and it is shown in the following diagrammatic representation.


All wrapper class objects are internally treated as object of Object class
Add wrapper class object to collection framework object
Convert fundamental value into wrapper class object
Fundamental data type
assembling
Retrieve data from collection framework object and hold it into object of Object class
Typecast object of Object into wrapper class object
Convert wrapper class object into fundamental data type by applying XXX xxxValue method which is present each and every wrapper class. XXX represents only fundamental data type
disassembling

4. public <u>boolean isEmpty ()</u>:

Returns true when there are no object found in Collection interface object otherwise return false.

5. public <u>Object [] toArray ()</u>:

This method is used for extracting the data from Collection object in the form of array of objects of java.lang.Object class.

<u>For example</u>:

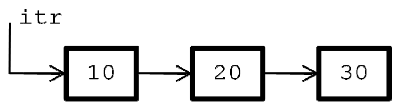
```
int s=0;
Object obj []=s.toArray ();
for (int i=0; i<obj.length; i++)
{
       Interger io= (Integer) obj [i];
       int x=io.intValue ();
       s=s+x;
}
System.out.println ("Sum = "+s);
```

6. public <u>Iterator iterator ()</u>:

This method is used for extracting the data from Collection framework object.

<u>For example</u>:

```
Iterator itr=co.iterator ();
Int s=0;
```



```
While (itr.hasNext ())
```

```
{
	Object obj=itr.next ();
	Integer io= (Integer) obj;
	int x=io.intValue ();
	s=s+x;
}
```

Day -65:

Iterator interface: Iterator is an interface which always uses the extract the data from any Collection object.

Methods in Iterator interface:

```
1. public boolean hasNext ()
2. public Object next ()
3. public Object remove ()
```

Method 1 is used for checking whether we have next element or not. If next element available in Collection object it returns true otherwise it returns false. Method 2 is used for obtaining the next element in the Collection object. Method 3 is used for removing the element from Collection object. Methods 2 and 3 can be used as long as method 1 returns true.

List: List is an interface which extends Collection.

Methods in List interface:

1. **public Object get (int):** This method is used for obtaining that element from the specified position. If the get method is not returning any value because of invalid position the value of object of object is NULL.
2. **public Object remove (int):** This method is used for removing the objects from List object based on position.
3. **public Object remove (Object):** This method is used for removing the objects from List object based on content.
4. **Public void add (int pos, Object):** This method is used for adding an object at the specified position.
5. **public void addAll (int pos, Collection):** This method is used for adding one Collection object to another Collection object at the specified position.
6. **public List headList (Object obj):** This method is used for obtaining those objects Xi's which are less than or equal to target object ($Xi \leq obj$).
7. **public List tailList (Object obj):** This method is used for obtaining those objects Xi's which are greater than target object ($Xi > obj$ or $Xi \geq (obj-1)$).
8. **public List subList (Object obj1, Object obj2):** This method is used for obtaining those values which are a part of List i.e., range of values [or] In subList method is select those values Xi's which are less than or equal to object 1 and greater than object 2 ($obj1 \leq Xi < obj2$).
9. **public ListIterator listIterator ():** This method is used for extracting the data from List object either in forward or in backward or in both the directions. ListIterator is an interface which extends Iterator interface. This interface contains the following methods:
   ```
   public boolean hasPrevious (); →1
   public Object previous (); →2
   public void set (Object); →3
   ```
   All the methods of iterator are also coming. Method-1 is used for checking weather we have previous element or not, this method returns true as long as we have previous elements otherwise

false. Method-2 is used for obtaining previous element. Method-3 is used for modifying the existing Collection object returns true. Methods 2 and 3 can be used as long as method-1 returns true.

Day - 66:

One dimentional collection framework classes:

One dimensional collection framework classes are defining all the methods of collection framework interfaces. The following table gives the collection framework classes and its hierarchy.

Interfaces	Classes
Collection	AbstractCollection implements Collection
List	AbstractList extends AbstractCollection implements List
Set	AbstractSet extends AbstractCollection implements Set
SortedSet	AbstractSequentialList extends AbstractList implements SortedSet
	LinkedList extends AbstractSequential
	ArrayList extends AbstractSequential
	HashSet extends AbstractSet
	TreeSet extends AbstractSet

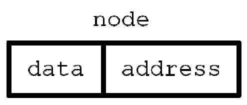
Collection framework classes contains the definition for those methods which are coming from Collection interfaces i.e., all Collection interface methods defined in collection classes AbstractCollection implements Collection, AbstractList extends AbstractCollection implements List, AbstractSet extends AbstractCollection implements Set, AbstractSequentialList extends AbstractList implements SortedSet, LinkedList extends AbstractSequential, ArrayList extends AbstractSequential, HashSet extends AbstractSet and TreeSet extends AbstractSet.

The classes AbstractCollection implements Collection, AbstractList extends AbstractCollection implements List, AbstractSet extends AbstractCollection implements Set and AbstractSequentialList extends AbstractList implements SortedSet are abstract classes we never make use of them as a part of real time applications but we make use of the classes LinkedList extends AbstractSequential, ArrayList extends AbstractSequential, HashSet extends AbstractSet and TreeSet extends AbstractSet.

java.util.LinkedList:

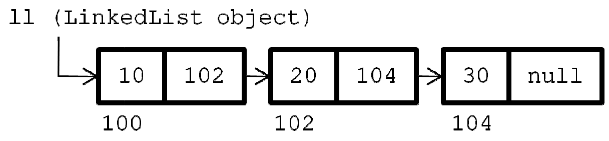
LinkedList is the concrete sub-class of collection classes. LinkedList object allows us to group similar or dissimilar type of objects. Creating a LinkedList is nothing but creating an object of java.util.LinkedList class.

The data is organized in LinkedList in the form of nodes. The node contains two parts, they are **data part**, **address part**.



Data part always gives the actual data which we want to represent. *Address part* represents address of the next node.

For the last node in the LinkedList the address part must be NULL which indicates end of the LinkedList.



<u>LinkedList API</u>:

Constructor<u>s:</u>

```
LinkedList ();
LinkedList (int size);
```

<u>Instance methods</u>:

```
Object getFirst (); →1
Object getLast (); →2
public void addFirst (Object obj); →3
public void addLast (Object obj); →4
public void removeFirst (); →5
public void removeLast (); →6
```

Methods 1 and 2 are used for obtaining first and last objects of LinkedList respectively. Methods 3 and 4 are used for adding similar or different objects to the LinkedList object respectively. Methods 5 and 6 are used for removing first and last objects from LinkedList respectively.

<u>Set</u>: Set does not contain any special method except Collection interface methods.

<u>SortedSet</u>:

SortedSet extends Set. The following are the methods which are specially available in SortedSet interface.

```
public Object first ();
public Object last ();
public SortedSet headSet (Object obj); xi <= obj
public SortedSet tailSet (Object obj1, Object obj2);
```

<u>Day - 67</u>:

Write a java program which implements the concept of LinkedList?

<u>Answer</u>:

```
import java.util.*;
class Linkedlist
{
	public static void main (String [] args)
	{
		LinkedList ll=new LinkedList ();
		System.out.println ("CONTENTS OF ll = "+ll);
		System.out.println ("SIZE = "+ll.size ());
		ll.add (new Integer (10));
		ll.add (new Integer (20));
		ll.add (new Integer (30));
		ll.add (new Integer (40));
```

```
            System.out.println ("CONTENTS OF ll = "+ll);
            System.out.println ("SIZE = "+ll.size ());
            // retrieving data of ll using toArray ()
            Object obj []=ll.toArray ();
            int s=0;
            for (int i=0; i<obj.length; i++)
            {
                  Integer io= (Integer) obj [i];
                  int x=io.intValue ();
                  s=s+x;
            }
            System.out.println ("SUM USING toArray () = "+s);
            ll.addFirst (new Integer (5));
            ll.addFirst (new Integer (6));
            System.out.println ("CONTENTS OF ll = "+ll);
            System.out.println ("SIZE = "+ll.size ());
            // retrieving data of ll using iterator ()
            Iterator itr=ll.iterator ();
            int s1=0;
            while (itr.hasNext ())
            {
                  Object obj1=itr.next ();
                  Integer io1= (Integer) obj1;
                  int x1=io1.intValue ();
                  s1=s1+x1;
            }
            System.out.println ("SUM USING iterator () = "+s1);
            // retrieving data of ll using ListIterator ()
            ListIterator litr=ll.listIterator ();
            while (litr.hasNext ())
            {
                  Object obj2=litr.next ();
                  System.out.print (obj2+",");
            }
            System.out.println ("\n");
            while (litr.hasPrevious ())
            {
                  Object obj3=litr.next ();
                  System.out.print (obj3+",");
            }
            System.out.println ("\n");
            Object obj4=ll.get (2);// random retrieval
            System.out.println (obj4);
      }
};
```

<u>Disadvantages of LinkedList</u>:

1. Additional memory space is created for address part of the node in heap memory.
2. Retrieval time is more.
3. Since, we are wasting most of the memory space for addresses, performance will be reduced.

java.util.ArrayList:

ArrayList is also concrete sub-class of collection framework classes whose object allows us to organize the data either in similar type or in different type.

Creating ArrayList is nothing but creating an object of ArrayList class.

ArrayList API:

```
ArrayList ();
ArrayList (int size);
```

Advantages of ArrayList over LinkedList:

1. No additional memory space is required for data of ArrayList.
2. Retrieval time is quite faster.
3. Performance is high. Since, there is no memory space is required for maintaining address of data of ArrayList.

Day - 68:

HashSet and TreeSet:

HashSet	TreeSet
1. It extends AbstractSet.	1. It extends AbstractSet and implements SortedSet.
2. It follows hashing mechanism to organize its data.	2. It follows binary trees (AVL trees) to organize the data.
3. We cannot determine in which order it displays its data.	3. It always displays the data in sorted order.
4. Retrieval time is more.	4. Retrieval time is less.
5. The operations like insertion, deletion and modifications takes more amount of time.	5. The operations like insertion, deletion and modifications take very less time.
6. Creating HashSet is nothing but creating an object of HashSet () class.	6. Creating TreeSet is nothing but creating an object of TreeSet () class.

Write a JAVA program which illustrates the concept of TreeSet?

Answer:

```
import java.util.*;
class tshs
{
      public static void main (String [] args)
      {
            TreeSet ts=new TreeSet ();
            System.out.println ("CONTENTS OF ts = "+ts);
            System.out.println ("SIZE OF ts = "+ts.size ());
            ts.add (new Integer (17));
            ts.add (new Integer (188));
            ts.add (new Integer (-17));
            ts.add (new Integer (20));
            ts.add (new Integer (200));
            ts.add (new Integer (177));
            System.out.println ("CONTENTS OF ts = "+ts);
            System.out.println ("SIZE OF ts = "+ts.size ());
            Iterator itr=ts.iterator ();
            while (itr.hasNext ())
            {
                  Object obj=itr.next ();
                  System.out.println (obj);
            }
      }
};
```

Write a JAVA program which illustrates the concept of HashSet?
Answer:

```
import java.util.*;
class hsts
{
      public static void main (String [] args)
      {
            HashSet hs=new HashSet ();
            System.out.println ("CONTENTS OF hs = "+hs);
            System.out.println ("SIZE OF hs = "+hs.size ());
            hs.add (new Integer (17));
            hs.add (new Integer (188));
            hs.add (new Integer (-17));
            hs.add (new Integer (20));
            hs.add (new Integer (200));
            hs.add (new Integer (177));
            System.out.println ("CONTENTS OF hs = "+hs);
            System.out.println ("SIZE OF hs = "+hs.size ());
            Iterator itr=hs.iterator ();
            while (itr.hasNext ())
            {
                  Object obj=itr.next ();
                  System.out.println (obj);
            }
      }
};
```

Two dimensional framework or maps:

Two dimensional framework organize the data in the form of (key,value) pair. The value of *key* is an object and they must be unique. The value of *value* is also an object which may or may not be unique. Two dimensional framework contains collection of interfaces and collection of classes which are also known as **map interfaces** and **map classes**.

Map interfaces:

In maps we have three essential interfaces; they are **java.util.Map**, **java.util.Map.Entry** and **java.util.SortedMap**

java.util.Map:

java.util.Map extends Collection. An object of Map allows to organize the data in the form of (key, value) pair. Here key and value must be objects. An object of Map allows displaying the data in that order in which order we have added the data.

Methods:

public boolean put (Object kobj, Object vobj):

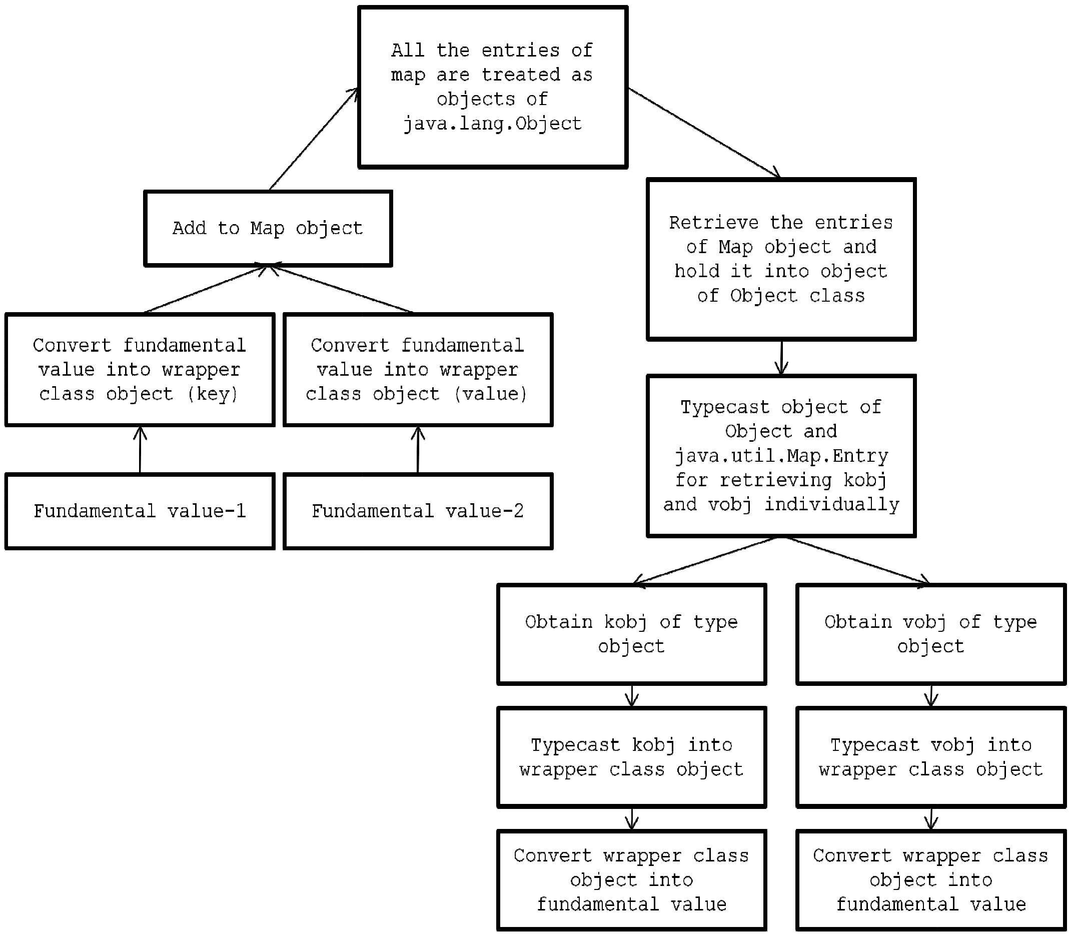
This method is used for adding the data in the form of (key, value). This method returns false when we are trying to add duplicate key and values. This method returns true as long as we enter unique key objects.

public boolean putAll (Map): This method is used for adding one Map object at the end of another Map object.

public Set entrySet (): This method is used for obtaining the data of the map in the form of Set object.

public Object get (Object vobj): This method is used for obtaining value of value by passing value of key object.

public void remove (Object kobj): This method is used for removing the entire map entry by passing the value of key object.


All the entries of map are treated as objects of java.lang.Object
Add to Map object
Convert fundamental value into wrapper class object (key)
Convert fundamental value into wrapper class object (value)
Fundamental value-1
Fundamental value-2
Retrieve the entries of Map object and hold it into object of Object class
Typecast object of Object and java.util.Map.Entry for retrieving kobj and vobj individually
Obtain kobj of type object
Obtain vobj of type object
Typecast kobj into wrapper class object
Typecast vobj into wrapper class object
Convert wrapper class object into fundamental value
Convert wrapper class object into fundamental value

NOTE:

The following diagram gives an idea about how to organize the data in the form of (key, value) and how to retrieve the data from Map object.

Day - 69:

java.util.Map.Entry:

Here Map is an interface and Entry is the class in Map interface. java.util.Map.Entry is used for retrieving the data separately in the form of key object and value object from the Map object.

Methods:

```
public Object getKey (); →1
public Object getValue (); →2
```

Method 1 and 2 is used for obtaining key object and value object separately.

java.util.SortedSet:

SortedMap extends Map. An object of SortedMap displays the data by default in sorted order.

Methods:

```
public Object first ();
public Object last ();
public SortedSet headMap (Object kobj); xi <= kobj
public SortedSet tailMap (Object kobj); xi > kobj
public SortedSet subMap (Object kobj, Object vobj); kobj1 <= xi < kobj2
```

Map classes:

Map classes contains all the definitions for the abstract methods of Map interface. In java.util.* package we have the following Map classes and whose hierarchy is given below:

1. AbstractMap implements Map
2. AbstractSortedMap extends AbstractMap implements SortedMap
3. HashMap extends AbstractMap
4. TreeMap extends AbstractSortedMap

HashMap	TreeMap
1. It extends AbstractMap.	1. It extends AbstractSortedMap.
2. It follows hashing mechanism.	2. It follows binary tree concept to obtain data in (k, v) form.
3. Retrieval time is more.	3. Retrieval time is less.
4. Insert, update and delete operations takes more time.	4. Insert, update and delete operations takes less time.
5. Creating an object of HashMap	5. Creating an object of TreeMap.
6. Random order.	6. Sorted order.

Creating Hashtable is nothing but creating an object of Hashtable class.