

In a software development we will be creating different ADT's (OR) classes (or) user-defined data types in order to represent the blue-prints used for the creation of objects. While creating a ADT (or) class a s/w engineer identifies the properties.

He programmatically represents the properties using variables and he is also responsible for identifying the operations and expressing the operations using functions (or) methods. For ex: In the creation of piston, we have identified the properties like length, radius, material type and thickness of the piston as properties.

And we are also identifying the operations moveUp(), and moveDown() as methods (or) functions. And we have created the new data types called piston, Connecting rod, valves etc, as the code above shown. These new datatypes are called "classes".

Mechanical device	Software project
1. A device is made up of different parts objects.	1. A s/w project is made out of different objects.
2. These parts objects falls under different classes.	2. These objects are also falls under different classes categories.
3. In order to create classes you need a blue-prints.	3. Here also in order to create Classes, you need blue-prints.

An Object oriented programming language (OOP) allows the creation of s/w solutions by assembling different parts/objects created from the blue-prints/classes. When we develop a mechanical instrument, we will be developing it by assembling different parts using spanners, screws and so on. In developing a s/w project we apply our brains (to assemble logically) in assembling the objects. By writing the code that logically combines all the objects.

In the case of mechanical device at any point time we will be having only fixed no of parts. But in the case of s/w project at the run-time, some times you may having 10 objects and in sometimes it may be more or less than 10 objects.

In s/w everything is conceptual under language level.

When a mechanical engineer wants to design a part he has to express the design as a blue-print. A programmer who is using an OOP language has to represent the blue-print using the facilities provided by the language. In an OOP language, we can combine 'data & operations' together in order to represent the design.

At the s/w level we can create a class by identifying the properties and representing them as variables, and identifying the operations and representing them as methods.

A tube (bulb) can be used as a part in multiple devices. In the same way once the class is created, we can use the objects of that particular class in multiple projects. Even though most of the books says OOP languages says that a class that is created for a particular project can be used in any number of projects. It should be noted that different projects

may require different objects for the same class and in some cases that class used in project 1 should not satisfied for the requirements of project2.

In Java language we can create an object of type piston by using a statement as shown below:

```
public static void main (String s[])
{
    Piston p1;    //New piston object
    p1=new piston();    //Name of the class followed by new Keyword.
    p1.moveUp();        //operations
    p1.moveDown();
}
```

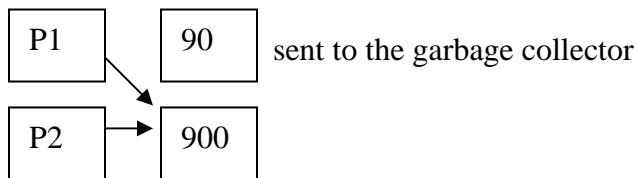
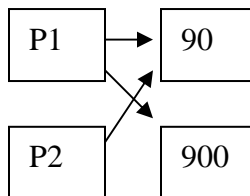
In JVM we have a component called 'garbage collector' which is responsible for removing unused objects, and cleaning up the memory. Garbage collector inside the JVM will be collecting the unused objects periodically (we don't know when it is actually occurs).

So a Java programmer, if required can invoke the garbage collector on his own, by writing the method.

```
int pistonno;
class Setpno(int pno)
{
    pistonno=pno;
}
```

Most of the JVM uses algorithm called mark & sweep algorithm for implementing the garbage collector.

```
piston p1,p2;
p1=new piston();
p1.setno(90);
p1.moveUp();
p2=p1;
p1=new piston();
p1.setPno(900);    //if(p2=p1) here
p1.moveDown();    Then p1&p2 are pointing to 900,and 90
p1.moveUp();    sent to the garbage collector
p2.moveDown();
```



In Java language there are

1. Primitive datatypes ex:int,float,char,byte...
2. Referenced data types. ex:piston p1,class c1....

When JVM starts executing it will create different memory areas they are

1. Method stack -where the local the variables are stored.
2. Operand stack -where all the operations can be performed.
3. Heap (huge amount of memory)-used to store the (newly created) objects. This is filled up from top to bottom.

JVM starts executing the methods first.

Ex: *Piston P1,P2;*

int i = 20;

P1 = New piston();

P2 = New piston();

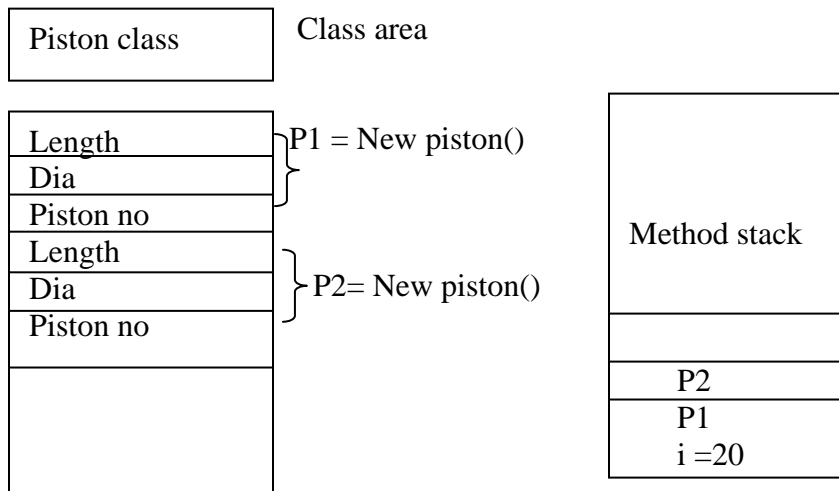
Piston(){

Int length, dia, pistonno;

moveup();

movedown();

}



JVM starts executing the methods first.

Steps:

1. It will store all the local variables declared in a method, on the 'method stack'.
2. In the method stack the memory space will be occupied by the variables. In general it is 32 bits, if the variable is not initialized, but it is not for all the cases, it will depend on the JVM.
3. It will check whether a class file is existing or not in a 'class area'(memory area)

If it exists, then it will store the 'piston class' in the memory area called 'class area'.(class file contains properties and methods details)

4. Then the JVM reserves the memory area required in order to store the 'piston object'. Heap is used to store the object.

5. If `p1=new piston();` not used, then p1 holds the memory area referred to the object of type piston, but it should not stores the object physically in the memory area called heap. Here two instances of p1 and p2 piston objects were created. And their properties are stored in a memory two times.

6. From p1 we can reach the memory area of new piston object that is referenced by p1. (p1 referencing the object i.e. created in the heap area)

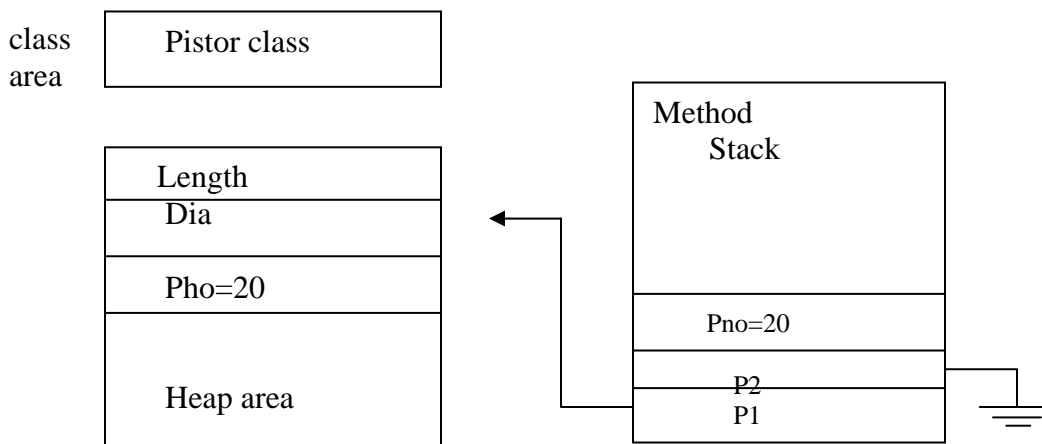
7. If `p2=NULL;` is assigned

Null is used to represent no object.

Null can be graphically represented to ground.

The space required by an object is always reserved on the heap. JVM implementer is responsible for allocating the space required for an object and re-allocating when the object is not referenced by a program.

How methods inside a class are executed:-



```
P1= new piston();
P2= new piston();
P2=null;
P1.setno(20);
{
Pistonno = pno;
}
```

Parameters are same as Local variables.

1. If we want to perform the operator on an object, which is already pointed by p1, use `p1.setno(20)`.

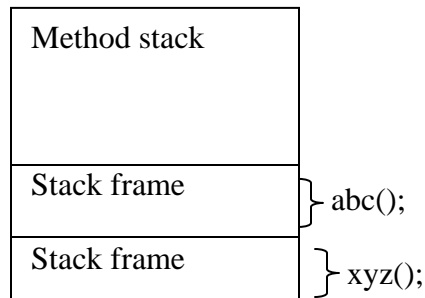
2. *p1.setno(20)*-> Stores the pno value on the method stack. After this *pistonno=pno* is executed, and pno=20 is stored in the class object which is referred by p1. Since p2=Null, it points to the ground.

In this piston example we have declared 3 different variables length, dia and pistonno. Every object is created based on the 'piston class', has its own copy of the three variables. Since every instance of the class has its own copies of the variables.

These variables are known as '**Instance variables**'.

Stack trace: Is used to find which method caused for an error or which method is currently executing.

In some books the memory area reserved for a particular method on the top of the stack is referred as '**Stack frame**'.



In general 'Method should have return type, name of the method and can take n number of parameters.

Ex:

```
void moveUp();  
void setNo(int);
```

In the same way constructor can have the name and can take n number of parameters but it should not contain return type.

Constructor: Constructing the code during the creation of an object by the java compiler is known as 'constructor'.

*Constructor should not be explicitly called by the programmer.

JVM performs the following tasks on the object:

1. When the JVM creates an object it is responsible for allocating the space for the object on the heap.
2. JVM is also responsible for the default initial values of the instance variables.
3. JVM is responsible for executing a constructor after assigning the object values to the instance variables.
4. Java compiler adds the constructor, which takes zero arguments after examining the source code written by the programmer.

While compiling a class, the java compiler checks whether a developer provided the constructor or not. If the java compiler identifies that, the developer has not provides a constructor it automatically creates its own constructor.

Irrespective of whether the compiler has provided or the developer has provided, the java compiler adds the code for the initialization statements that are written by the programmer when the variables are declared.

Ex: *class Piston()*

```
{
    int length = 10;      // These are executed first,
    int dia ;             // before the
    int pistonno = 90;    // constructor.
}
Piston(){ // constructor
    System.out.println(" In a constructor");
    length = 90;
}
// This will be executed after the class initializes.
```

When a constructor is compiled, java compiler adds the code written for the initializers

*A constructor cannot be invoked directly in the program. (Ex: P1.Piston)

Java language allows us to have a method with the same name as that of a class. (This will be done to improve the performance of the computer)

Create a class with the name color, with 3 instance variables Red, Green, and Blue, and provide the methods SetRed, SetGreen, SetBlue, GetRed, GetGreen, GetBlue and also provide a constructor which initializes Red, Green, Blue = 10.

*Constructor name should be as the Class name.

The purpose of the constructor is to construct (or) initialize the object. It is similar to a method. But technically it is not called as a method. Constructor is called whether the object is created.

```
class Color {
    int red = 255; // To create default
    int blue = 0;  // Red color object.
    int green = 0;

    void SetRed ( int r ) { Red = r; }
    void SetBlue ( int r ) { blue = r; }
    void SetGreen( int r ) { Green = r; }
    void pint() {
        System.out.println (red);
        S.o.p (Blue);
    }
}
```

```
S.o.p (Green); }
```

```
class Usecolor{  
    public static void main ( String a[]){  
        Color c1;  
        Color c2;  
        C1 = New Color(); /* Red colored objects  
        c2 = New color(); will be created */  
/* If you want a Brown color object now */  
// if Brown is the mix of Red = Blue = Green = 200  
        c2.SetRed(200);  
        c2.SetBlue(200);  
        c2.SetGreen(200);
```

/* To Improve the performance of the application / compiler simply write the construction and pass the no of arguments required */

```
        color(int r,int b,int g){ //constructor  
        System.out.println("In S parameters Constructor");  
        red=r;  
blue=b;  
green=g;  
}
```

```
//In this case the code to create a New object will be  
c1=New color();  
c2=New color(255,0,0);//To create Red object  
c2=New color(200,200,200);//To create Brown object
```

/* If you want to create n no. of red objects then you will write another Constructor, which is having zero arguments, as follows */

```
color() //another constructor  
{  
    System.out.println("In zero argument constructor");  
    red=255;  
    blue =0;  
    green=0;  
}
```

Note: Multiple constructors are used to create n no. of objects of same type and color, but the arguments passed to it should not be the same. Multiple constructors are allowed in classes.

*Compiler shouldn't create a 'Zero argument constructor'. Depending upon the project requirement, you can go for multiple constructors.

In any class we can provide multiple constructors. While creating the class it is always advisable to think about providing multiple constructors, to give flexibility for other developers who want to create the objects based on your class.

If an OOP Language doesn't provide the flexibility of creating multiple constructors, then the programmer can't provide enough flexibility to the user of the class.

The initialization statements that are written as a part of the declarations are always added to the first part of the constructors.

*Constructor invocation is first in a method, while calling one constructor in another.

We can invoke one constructor from another by simply using '*this*' followed by the parameters to the constructor. Whenever a constructor has been invoked from another, *this* has to be used as a first statement in a method, else error will occur.

Ex:

```
this(255,10,10);  
System.out.println("In zero Constructor");  
/* Red=255;    //Redundant coding  
Blue=10;
```

In most of the cases we will be providing multiple constructors, but we will be invoking one constructor from another constructor.

Overloading: Writing the multiple constructors in a class is known as "overloading constructors".

Multiple methods with the same name in a class is known as "Method overloading".

this keyword-> is a reference to the current object or which the method is invoked.

Ex: *a1.setInt(10);*

```
Class x  
{  
    public static void main(String ar[])  
    {  
        p1=new xx();  
        p1.setInt(20);  
    }  
}  
class xx  
{  
    int i=30; //Instance variable  
    void setInt(int i)  
    {
```



```

        i=i; //[i=this.i]/[(this.i=i)]
        System.out.print(i);
    }

```

if *i=i*-->Local variable value is printed.
this.i=i-->Local variable value is printed.
i=this.i-->Instance variable value is printed.

this.i --> used to identify the Instance variables

Rule: If Local variable and Instance variable with same name then use '*this*' keyword to identify the instance variable explicitly.

this keyword is used in

1. Methods,
2. Constructors, and
3. To pass the reference to the same object from particular memory area.

When '*setInt*' method is executed, *i=i* refers to the local variable. Since the name of the local variable is same as the Instance variable. In order to refer the Instance variable in this method, we need to explicitly.

4. Write *this.i* . some of the programmers always refers to the instance variable by preceding this before the variable name.

Setters: is a method which sets a property.

Getters: is a method which gets a property.

Ex: *setInt(int x)* {-----}; //mutator

getInt(int x) {-----}; //Non-mutator

Mutator:- is a method which modify the object.

Non-Mutator:- The method which doesn't modify the object.

In general all the 'getters' are non-mutators. But in almost all the cases, all the 'setters' are mutators.

Mutable-class--> If you have single mutator in your class, then it is known as 'mutable class'

Immutable-class--> If there are no mutators in a class then it is known as "Immutable class" (Which does not allow to modify an object after the object is created).

```

class fan()
{
    boolean Running
    int speed;
    public static void main(Stirng args[])
    {

```

```

        fan f=new fan();
        f.setSpeed(5);
        f.setRunning(True);
    }
}
fan(){
    Running=false;
    Speed=0;
    void setSpeed(int s)
    {
        speed=s;
        System.out.println(s);
    }
    void setRunning(int r)
    {
        Running=r;
        System.out.println(r);
    }
}

```

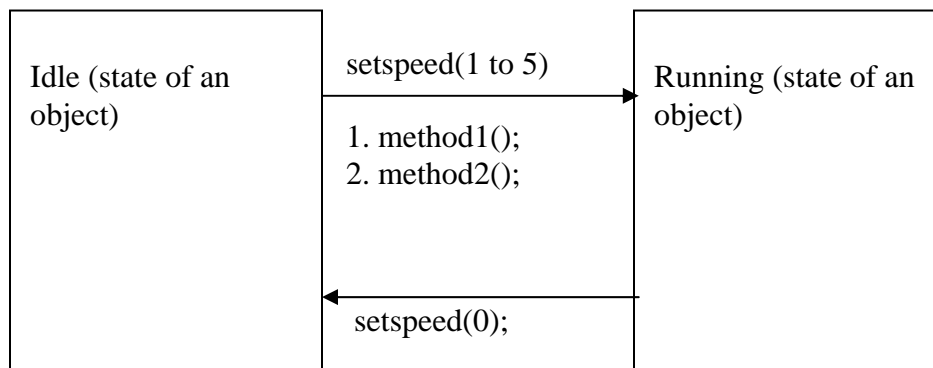
*The state of the object is stored inside the instance variables

State Transission diagram:- To identify the different sates of an object.

The mutator can change the state of an object.

The state of the non-mutator object cannot be modified,once the object is created.

state Transission diagram for fan:-



idle and running are properties of a fan

Ex:

```
class xx{
    public static void main(String args[])
    {
        int x=20;
        int y=30;
        p1=new object()
        p1.setInt(50)
    }
}
```

stack

logical
nos

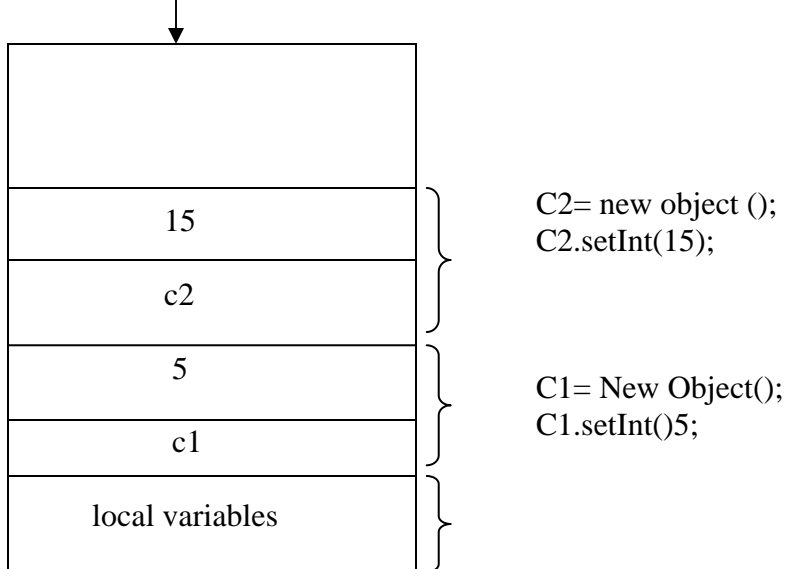
{ 2
1
0

P1	
Local variable	y=30
Local variable	x=20
String args[]	

If the Example is taken then the space allocated in the memory area is as shown in the figure.

```
class object{
    System.out.println(i);}
}
```

Make the experiment by changing the Local variables by placing them after the p1.setInt(50) and so on, and observe the byte codes.



Always when the method is executed on the stack, the reference of the object is passed even though you haven't mention it.

Access Specifiers: 1. public 2. private

Specifies who can access methods, variables, constructors, and instance variables.

public : it can be accessed outside the class also.

private: it can be accessed only inside the class only.

public is used to improve the performance of our application.

Calling a method is expensive rather than assigning a value to a variable. In this case we declare the variables as public.

while designing, the designer will frame like this:

class xyz
variables declaration int a; int b;
meth1(); meth2(); meth3();

If the code is shared by other methods, then write the common method named as 'helper() method' and call this method whenever required. These helper() methods should not be accessed outside the class. So declare the Helper() as private.

Depending upon the project requirements, and to improve the performance of an application use 'public' access specified.

We can use the 'private' access specified for the constructors also. But you can't use this constructor to create the objects outside the class.

I want to have restriction of creating of our object from my constructor. According to the project requirement. Only one object should be created, but not multiple objects.

To do this static or class method is used to the constructor, so that the objects should not created. C++ and Java languages provides this.

note: static is a class method can be called without thru creation of the object, use static followed by public or private access specifiers.

ex:

```
public static color stm()  
{  
    return new color();  
}
```

calling:

```
classname.methodname;
```

Create the object inside this method and use reference variable.static method name.

A static method can be called without the creation of object.

```
c1 = color.stm();      // factory method to create an object
```

In color class to create our object in a constructor.

single term design: only one object can be created using constructor.

While designing projects, in some cases creation of the objects we will be writing the methods exclusively for the creation of the objects. Such a method is called as a "factory method".

For creation of classes ---- factory classes.

Static method cannot access instance variables. why ?

ans: static method can be invoked before the object is created. Instance variables are created only when new object is created. Since, there is no possibility to the static method to access the instance variables.

Instance variables are also known as non-static variables.

static methods are called by the static methods only. An ordinary method can call the static methods. But the static methods cannot call the ordinary methods.

Static methods should not be used, without creating an object.

Eg:

```
class color{
private static color singleobj = null; // should be considered as a static or class variable,
                                // but not as an instance variable

private int red;
private int blue;
private int green;
private static int sa;
static
{
System.out.println("in static block: ");
}
private static color stm()
{
if (singleobj == null)
singleobj = new color();
return singleobj;
}
// this class will creates an object when loads this class
class usecolor
{
public static void main(String args[])
```

```

{
int i = 22;
color c1,c2;
c1=color.stm();
c2=color.stm();
System.out.println(c1);
System.out.println(c2); // both the objects are pointing to the same object
}
}

```

This means only one object is created, even though we will try to create two objects.

If 'static methods are not available then it is not possible to write this type of constructors to create only one object.

The information about the 'class variables' is stored in the class area, and these class variables created when the class is loaded in the JVM. The class will be loaded in to the JVM will depends upon the JVM implementation. Irrespective of number of objects created only one copy of class variables are created.

Static Blocks: are useful to set the initial values of static variables. It is a set of statements written between curly braces inside a class by using the keyword static. The JVM has to execute the static block whenever it loads the class.

In most of the cases static blocks are useful in initializing the class variables or static variables. There is no chance of accessing the instance variable from a static block.

ex:

```

static {
System.out.println("it is a static bloc");
}

```

Developer writes the code, to execute the static block and the JVM loads the class file, and executes the static block.

How programmatically loads the class file:

```

class proj
{
public static void main (String args[]) throws Exception
{
piston p1;
Class c = Class.forName("proj"); // proj is name of class
System.out.println("the class is : " + c);
}
}
// in the above program static is used to execute the method without creating an object.

```

At the back it is creating an object.

Programmer can also load the class file using
Class.forName("classname"); ---> Used to load the class file.

If the class file is not loaded, then only it will load the class file (When it is first referenced, JVM loads the class file).

JVM is responsible to load the class file in these classes:-

- 1) When the object is created
- 2) *ClassName.staticmethod()*.
- 3) *p1=new piston();*
- 4) *Class.forName("className")*.

Lazy Loading: Loading the class file whenever it is required.

In almost all the class file, whenever it is required. In almost all the virtual Machine implementations, JVM implementators use lazy loading technique in order to load a class file. (JVM loads a class file only when it is required).

When we use the command *java* followed by class name the JVM first loads the class file and executes the static block, after that it searches for main method. If the main method is not found, it simply throws the error message and terminates.

In most of the Core libraries provided by the JavaSoft the methods are implemented in native languages i.e C or C++.

Ex:- *private native void open(String name)* *FileNotFoundException* throws *Exception*

These core libraries in Unix, Linux are called as shared objects, in Windows DLL's (Dynamic Link Libraries).

JavaSoft provided the system class which has a static method *loadLibrary*. Using this method we can load the libraries on different platforms.

Most of the developers use the static blocks to load the libraries implemented in C, C++ kind of languages in the static blocks.

Ex:-

```
class Xyz
{
    static
    {
        System.out.println("xyz.dll");
    }
    native void abc();
    native void xxx();
}
```

```

xyz
{
    System.out.println("Hello Raj");
}
}

```

A single copy of static variable is shared between multiple copies of objects. This will save the memory space.

Ex:-

```

class Emp
{
    static int noofemp;
    int empno;
}

```

Try to re-implement the Color class without using static Variable?

Calling a method is passing the message to the object in object oriented Technology.

Signature of a Method: Which is used to uniquely identifies a particular method.

Ex:-

- 1) `int abc(int a,int b){.....}`
`abc int,int ----> Signatue of abc method.`
- 2) `int abc(float f){.....}`
`abc float ----> Signatue of abc method.`

Name of the method, followed by parameters.

Signatures are different for different methods, these should not be the same.

Ex:-

```

class OverLoad
{
    public void message()
    {
        System.out.println("No arguments");
    }
    public void message(int a)
    {
        System.out.println("One int argumet");
    }
    public float message(float a, float b)
    {
        System.out.println("Two arguments");
        return 9.9f;
    }
}

```



```

        return 9;
    }
}

```

Run this and observe the result, add the following one

```

    public void message(int a)
    {
        System.out.println("\nOne int argument");
    }
/* This is invalid, because already method name with the same parameter exists*/
class UseOverLoading
{
    public static void main(String args[])
    {
        OverLoad ol=new OverLoad();
        ol.message(); //method name.
        ol.message(23);//we are calling
        ol.message(); //same method in all the classes.
    }
}
/* Observe the results.

```

Polymorphism:-

In general, polymorphism is a single object taking different shapes or forms.

Overloading the methods is also known as polymorphism. Most of the objects oriented programmers used the term polymorphism for method overloading. When a method is overloaded, we can carryout the different tasks using the same message (method). Since our programs takes different forms by using overloaded methods, most of the object oriented programmers apply the same polymorphism for overloading.

In this case first it will check whether there is a direct match, if not it will pick-up the nearest match.

```

class UseOverLoading
{
    public static void main(String args[])
    {
        short s=80;
        char c='A';
        OverLoad ol=new OverLoad();
        ol.message(s);
        ol.message(c);
        ol.message(i,i);
    } }

```

In this case error will occur, like ambiguous calling of method. In this case developer is responsible to resolve the compiler from the overloading methods. To do this check the parameters passed to the method from left to right (or) right to left then this type of problems will be resolved.

Create a class with 2 method which takes a short, int. Try to create an object call the method by passing the character arguments.

Inheritance: Inherit the properties (or) behaviors from their parents.

```
1)    class Raj
        {
            int height;
            void eat(int qty)
            {
                System.out.println("Raj is eating "+i);
            }
            void drink(int qty)
            {
                System.out.println("Raj is drinking "+i);
            }
        }
2)    class Shiva
        {
            void eat(int qty)
            {
                System.out.println("Shive is eating "+i);
            }
            void sleep(int hours)
            {
                System.out.println("Shiv is sleeping "+i);
            }
        }
    class Movie
    {
        public static void main(String args[])
        {
            Raj r=new Raj();
            Shiv s=new Shiv();
            r.eat(10);
            r.drink(100);
            s.eat(30);
            s.sleep(8);
        }
    }
```

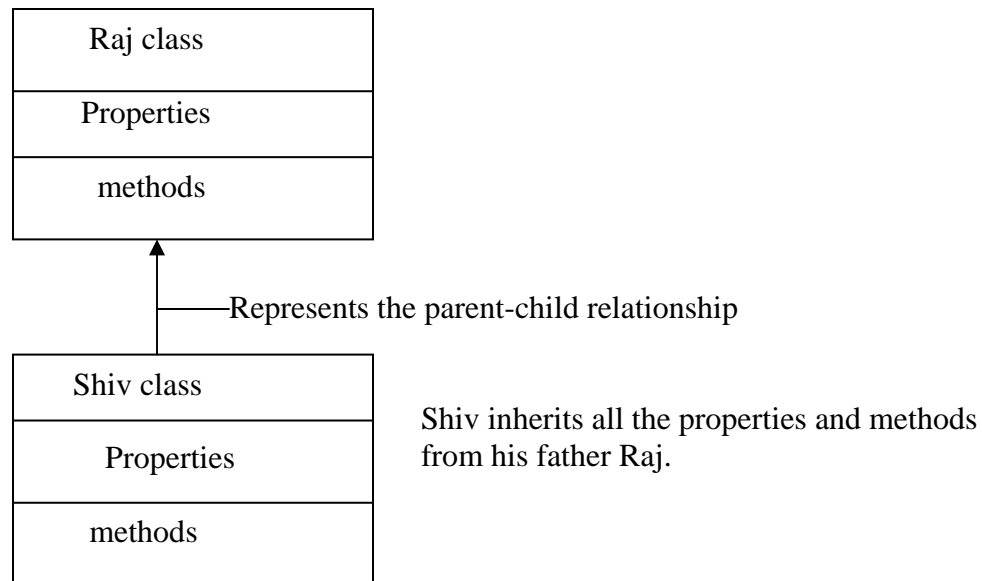
compile and observe the results.

```
javac ---classpath
//to search for the class path within the current directory (both raj & siv class files in the dir)
```

If the result is
Raj is eating
Raj is eating

This is correct because both will eat in the same way.

Class Diagram: Class in object oriented programming language is represented by a rectangle.



In java language parent -child relationship can be depicted by the keyword extends

Ex:-

```
class Shiv extends Raj{.....}
```

/* In this case instead of copying and paste the methods, shiv will extend all the properties and methods from Raj class. Compile and execute this you will get the same result as the previous case.

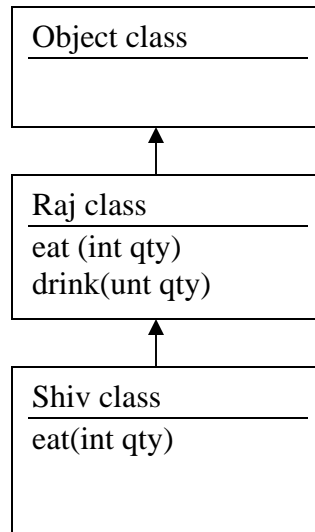
Implementation of Raj class is different from Shiv but drinking is the same. In this case re-implement the code in the Shiv class (Overriding)

```
class Shiv extends Raj
{
    void eat(int qty)
    {
        System.out.println("Shiv style of eating");    } }
```

Output:-

*Raj is eating
Shiv is eating.*

Class Diagram:-



By seeing the class diagram we can say that eating method is different for Raj and Shiv but the drinking method is same for the both.

For ex, if we write the code as

```
Shive s=new Shiv();  
s.drink(30);
```

First it searched for this method in shiv class or Sub class, if this was not found then it will searched in the other classes ie in the Raj class (super class) or parent class. If it is found it prints that particular message. If that is not found in that parent class also known as object class and prints the message that is there in that class.

To verify the code , check

```
javap --classpath. shiv  
this gives how JVM internally performing this.
```

```
class Shiv extends Raj  
{  
    shiv();  
    void eat(int);  
    void sleep(int);  
}
```

```
javap ...classpath Raj
```

```
class Raj  
{  
    Raj();  
    void eat(int);
```

```

        void drink(int);
    }

```

Re-implementing the method in a child (sub) class is known as "over-riding" the method. According to the diagram both Raj & Shiv classes consist of two interfaces, like drinking and eating. Sub class inherits from super class.

3 cases:-

- 1) Behaviors exhibited are the same
- 2) The way in which one exhibiting the behavior is different from other.
- 3) Having more behaviors than your parent (more behaviors than your child. This is not implemented in java language)

In this case Raj class is end of the class chart. But in java language above this Raj class one object class is available. This is known as the root class and this is the end of the classes.

In java language object class is the super class of all the classes.

Add the following code to the above example and observe the results:

Ex:

```

System.out.println("sm"+s.getClass());
System.out.println(("sm"+s.hashCode());
System.out.println(("sm"+r.getClass());
System.out.println(s);

```

This code represents:

classname followed by @symbol followed by hascode

If hashCode method is available in the object class it should not give any error. Otherwise it will give an error. This hashCode will be shared by all the classes (or) objects.

While designing java language, JavaSoft has identified that all the classes (objects) need some basic services like giving the information about the class on which the object is created and so on. Instead of asking the developers to provide the implementation of all these classes in their own, in all these classes JavaSoft provided a class with the name 'object with all these methods', and designed that the language in such a way that all the classes will become the subclasses of an object.

toString() method: is used to get the default method that is currently in use.

getClass(): on the object class return an object of type class, which represents the class based on which an object is created.

```

class c=s.getClass();
class c1=r.getClass();
System.out.println(c.getName());
System.out.println(c1.getName());
System.out.println(toString());

```

```
System.out.println(toString());
```

Try to override a method hashCode, toString in one of your class. In the toString method return null, and in the hashCode method return 200.

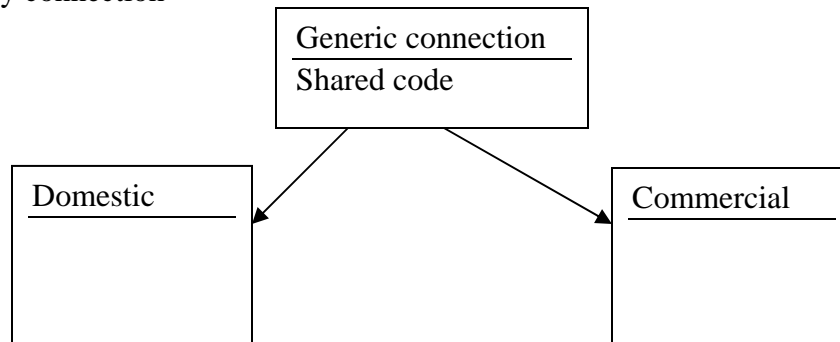
After examining this, try to provide getClass(), inside this return null.

```
public class getClass()  
{  
    return null;  
}
```

In most of the classes we will extend the capability of super-class.

To share the code between multiple sub-classes we will go for sub and super classes.

Ex:- Electricity connection

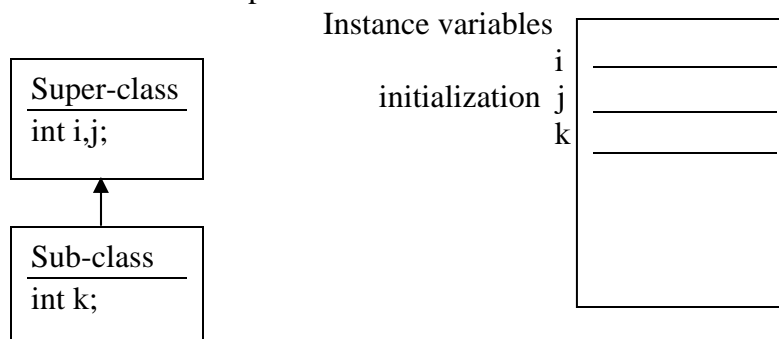


Generic connection will be shared by the both domestic and commercial (methods) classes. Any no of classes can be share the code in a practical project.

Fragile base class problem: In c++ language this problem arises, but java language doesn't have this problem. This problem occurs if there is any change in super-class. Then the sub-class requires re-compiling or repair. Because in c++ if these are not recompiled the sub-classes will broken. This known as fragile.

Implementation: Writing the code at the back to perform some operations is known as implementation.

Implementation Inheritance: Super class constructor executes first & then sub class constructor.



Sub s=new Sub(); // 3 instance variables are created.

Super class constructor is responsible to initializing the instance variables. Here the initialization is share by both super and the sub-classes constructors. How this is ensured by the compiler? When a class is executed two lines of code (in the byte code) will be added. In this code super-class constructor invocation is in the first line and this will be executed first and invoked in the sub-class. There if any instance variables that are in the sub-class are get initialized. In this way initialization at instance variables are shared by both.

Ex: Add zero constructor to Raj and Shiv class in the above example and observe the results.

```
public Raj()
{
    System.out.println("Raj Kumar");
}
public Shiv()
{
    System.out.println("Shiv Raj");
}
```

In byte code the first two lines in shiv class are

```
0 load-0
1 invokespecial #7<method raj()>
```

Super class constructor is executed first and then sub-class constructor is executed, but technically it is wrong. Because as a part of sub-class constructor, the compiler adds the two lines of code such that executed the super-class before executing the sub-class.

If we write super() in the shiv class and observe the code. There is no difference in the byte code developed.

Constructor invocation is very first line of the code, otherwise it will gives an error. If you want to retrieve any super-class variables or things, simply use super...

Ex: *System.out.println(super.);*

If we pass any arguments to the super-class constructor Raj, compile and execute this. And again re-compile the Shiv class, Shiv class (sub-class) will gives an error saying that 'No matching method found'. In this case modify the Shiv constructor by passing arguments, then it will works well and good.

Ex:-

```
public Raj(int i)
{
    System.out.println("Raj Constructor");
}
public Shiv(float f)
{
    super(89);
}
```

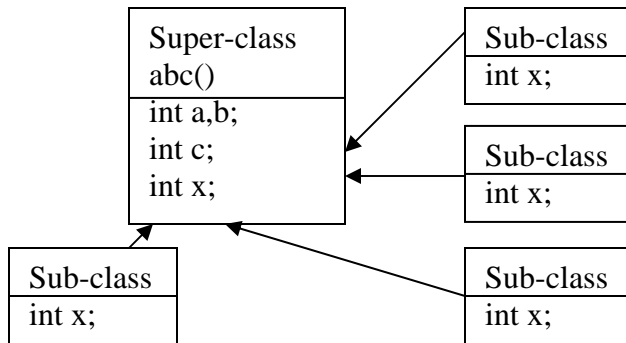
```

        //this(67.7f);
        System.out.println("Shiv Constructor");
    }
//observe the result in both the constructors.

```

The compiler of a java language places a restriction for using this() and super() keywords inside the constructor to ensure that the super-class initializing code is executed before the execution of the sub-class initialization code. These are in the very first lines of code in the constructor.

Why java Language allows some variable names both in the super and sub-classes?



If we buy the abc() super-class by a third party vendor. This is of abc() version1 according to them. We are using this class and this has been shared by 10 sub-classes. After some time due to the competition in the market, he modified the super-class abc() by using other variable int x, and released as abc() Version-II. But we are already using the int x variable in allmost all our sub-classes for some purpose. In this case, if java language doesn't supports the variables with same name, all our sub-classes will be broken. To avoid this type of probles java language designers allowed this in their language.

Note: Even though Java language allows the same variable names in both super and sub-classes, but it is not required.

Dynamically creating the Object in Java:-

At the compilation time, we don't know which type of object is going to be created only the objects are created at the run-time. By using the reference to the object we can do this Class.forName(clasname);

Ex:-

```

class Dyna
{
public static void main(String a[])
{
    for(int i=0;i<a.length;i++)
    {
        System.out.println(a[i]);
    }
}
}

```



```
}
```

output: is empty (Blank Line)

In java language arrays are treated as objects. In the above example, we are not passing any command-line arguments. If we will pass the command-line arguments they are available in an array. Every string is treated as one command-line argument.

a[0] in c-language is then name of the program a[0]=0 in java language.

Compile the above program, and give the command-line arguments. These arguments are taken by the array, and they are displayed. In the next step, give the command-line arguments

java Dyna this is a test

Output:-

```
This
is
a
test
```

if `System.out.println("a["+i+"]:"+ a[i]);` is used then the result is

```
a[0]: this
a[1]: is
a[2]: a
a[3]: test
```

Java Dyna "This is a test"

Output:-

```
This is a test.
```

Limitation: The class should have a zero argument constructor, on top of your object to perform this.

```
c.newInstance();
```

Ex:-

```
class Dyna
{
    public static void main(String a[])
    {
        System.out.println(a[0]);
        Class c=Class.forName(a[0]);
        System.out.println("Object created"+c.newInstance());
    }
}
```

Advantags:-

- 1) There is no need to touch the source code.
- 2) Extensions can be done without touching the source code.

In java Language we can dynamically create an objects by using NewInstance method, on the top of the class. New Instance method uses zero argument constructor in order to create an object.

//Solution for the 8th Agu..2k Problem

```
final class Dynacreation
{
    public void x()
    {
        .....
    }
    /* public class getclass(){
        return null;
    }
    public int hashCode(){
        return 200;
    }
    /* public string toString()
    {
        String s=new String("From you own to string method");
        return s;//this will overrides the hashCode() & toString()
    }
}
class UseDynacreation
{
    public static void main(String args[])
    {
        Dyanacreation d=new Dynacreation();
        System.out.println(d.hascode());
        System.out.println(d);//executes the method
    }
}
```

Output:-200

from our toString method

A final method cannot be overridden by the sub-class. In the case of getClass of an object method, this method is declared as a final method, this-method is declared as final method. so that a developer cannot over-ride the final method.

In this case if javasoft cannot declared getClass method as final, any programmer can re-impliment the getClass method and return some in-appropriate value.

final ensures that we can't create sub-classes in a methods.

Arrays:-

class Arrya1

```

{
    public static void main(String args[])
    {
        int i=55,j;
        int[] a;//declaration of an array.
        a=new int[15];//New object created.
        System.out.println(a.length);
        System.out.println(a);//it has used toString()
        a[0]=90;
    }
}

```

Output:- 12

[I@779e638e

```

class array2
{

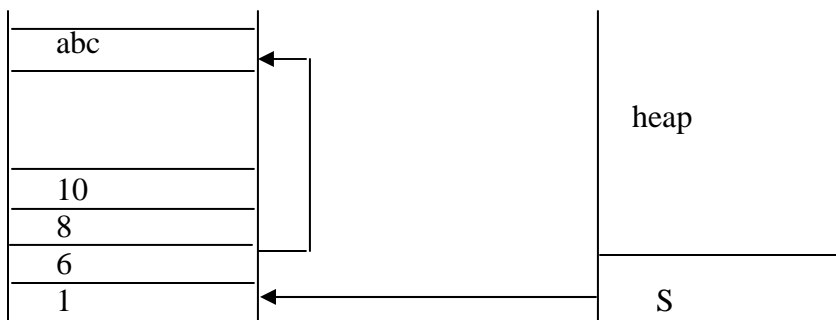
```

```

    public static void main(String args[])
    {
        String[] a; //declaration of an array.
        a=new String[12]; //New object created.
        System.out.println(a[0]);
        a[0]=true;    //return false
        System.out.println(a[0]);
        a[0]=new String("abc");    //return abc
        String[] s;
        s=new String[12];
    }
}

```

/* It will create an object of type string array s[0]. contains null value, but string object.
s[0]=new String{12};



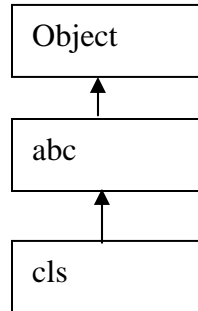
Note:-

In case of primitive data type arrays, the value of the element is stored directly in an array.

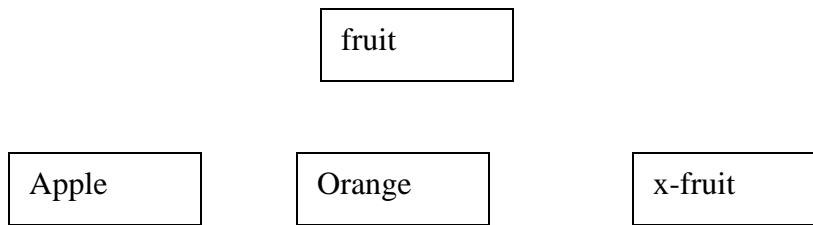
In case of reference data type arrays the reference of the object will be stored inside an array.

Q:- `cls d=new cls();`
`System.out.println(d);` What is the result?

It will depends upon the toString() method. we will ask for class hierarchy whether it has over-rider the toString method or not. If not it will searches in abc() class. If it is not found, it will searches in the object class.



Overriding is also known as binding (or) shadowing.



Observe the figure by moving from top to bottom we can get the specialized classes (categories) moving from bottom to top we can get the generalized classes.

In java Language object class is the super class of all object.

In the above figure: Not all the fruits are considered as an apple or the oranges.

```
fruit f=new fruit();  
fruit f=new Apple();  
fruit o=new Orange();
```

/* In an object oriented language, sub-class object can be pointed by super-class reference*/

```
Apple a=new Apple();  
f=0;
```

Note: Every sub class is converted into super-class very easily and it is always possible and is known as safe conversion (or) upcasting (or) widening.

Sub class reference to super-class object if we are moving from bottom to top, then this is known as 'up-casting'.

```
fruit f1=new Fruit();  
Apple a1=new Applet();  
Orange o1=new Orange();
```

```

f=o;//valid
o1=f;//valid, but not always;
//o1=(orange)f;

```

at run time this is possible and this is always not possible. So we must use explicit type casting operator. Here super-class object referenced by sub-class.

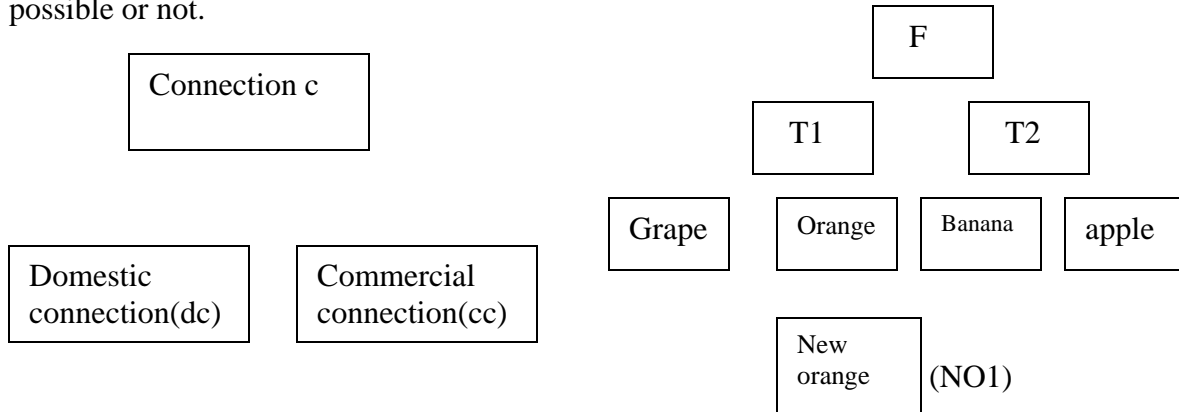
```

a1=f; //This is not possible

```

Down-casting:-

The conversion of super-class reference to the sub-class, known as narrowing (or) un-safe conversion (or) down-casting. We must use type-casting even though at run-time it is possible or not.



o-is an object

```

o=new cc;
c=(c)o;
dc=(dc)o//Not valid at run time
at compile time it will checks whether
type-casting is possible or not
cc= new cc();
o = (connection) cc;
//valid

```

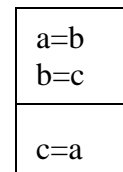
```

T1=new t1();
f=T1;
o1=(orange)f; //not possible at runtime
T1 = new orange()
f = T1
o1= (orange)f; // valid
o1= new T1();
o1= (orange)new T1();//not valid(possible)
o1= new orange();
NO1=(orange)o1;//possible at the compile
time

```

Limitation at compile time:-

- 1) All sub class reference can be converted to super-class
- 2) Super-class converted to sub-class but not all the time.



Method operated on an object:-

```

c=new Connection();
c.m1();
c.m2();
c.mx();//Not valid
c.m3();//Not valid

```

These two method are not on the interface of car object.
Because of this these two methods are not accessed
by the variable c. Even though m3(); in the
sub-class xxxxxxxxxxxx

```
d=new Dc();
d.m3();//valid
d.m1();//valid
d.m2();//Not Valid
```

```
Con c=new Dc();
c.m1();// IN DC m1() method will be executed.
c.m2();//Not possible
```

Connection
m1(); m2();

DC
m3(); m4();

ABC
m2(); m4();

Even though an object has several methods, you can invoke the methods that are exposed
by the class (con).

```
Co c=ABC();
c.m1();//valid
c.m2();//valid
DC(DC)c;//always not possible at run-time
DC.m3();//valid
DC.M4();//Not Valid
ABC=(ABC)DC //valid at the compile-time and also at the run time.
ABC.m1();//valid
ABC.m2();//valid
```

super-class=sub-class ----→ Always possible

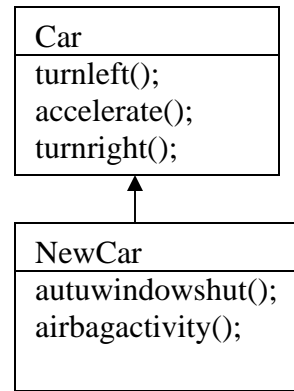
sub-class=(typecast)super-class --→ At compile time it is acceptable if explicit type-
casting is used. But (run-time) not possible.

class Cmdargs

```
{
    public static void main(String args[])
    {
        int i,j;
        System.out.println("\nLength of args:" + args.length);
        for(i=args.length-1;i>=0;i--)
            for(j=args[i].length()-1;j>=0;j--)
                System.out.println(args[i].charAt(j));
    }
}
```

You can able to drive the New car even though you don't know the special features (methods) of a new car because you knows the common features(methos) exposed by super-class car. But you cannot able to access the methods that are exposed by sub-class new car when the super-class is referenced by su-class.

```
Car c=new Car();
c.autoindowshut();//In valid
c.accelerate();//valid
c.turnleft();//valid
```



Different behaviors of objects are known by the methods.
 Different shapes (or) forms of objects are known by the properties.

Here there is no need to develop /create original object of car class. This is used to re-use the code. If you want to prohibit the developer to create an object of any class, then use the keyword 'Abstract' before the class. In this case nobody is allowed to create that particular class.

Abstract method: is a method which has no implementation (method without implementation).

Advgs:-

- 1) You can't create an objects.
- 2) You can write polymorphic code

Polymorphism: One(interface) object that can acts as a different forms/shapes.

Advgs:-

- 1) We can write very flexible code.
- 2) We can write the code that operates on different objects.

Concreate Method: is a method which has implementation (or) is a method is not an abstract method then it can be called as 'concrete-method'

Concrete-class: The class by which a developer can creates an object

Abstract-class: The class by which a developer cannot creates an object.

Q:- ABC();& XYZ(); are the two methods. xyz();is an abstract method inside ABC(); If you are trying to create an object of ABC(); what will be happen?

Ans:- C.ABC();

Here, you are prohibiting the developer not to create an object of abstract type. So it will cause an error.

Ex: */*Polymorphism example */*

```
class Conn
{
    public int connid;
    public String custname;
    conn(int connid,String custname)
    {
        this.connid=connid;
        this.custname=custname;
    }
    /* code to write getid() and getname().*/
    int getid()
    {
        return connid;
    }
    String getname()
    {
        return custname;
    }
}
```

This code is used to write the polymorphic effect.

The above code gives the method on the super-class object reference. calcbill() doesn't have the implementation in the super-class. This is known as polymorphism. Using this we can write the generic code.

//Code to create com object and calculating the bill.

```
class com extends conn
{
    comm(int connid,String custname)
    {
        super(connid,custname);//to invoke the super-class instance variables.
    }
    float calcbill(int nounits)
    {
        System.out.println("\nYour bill is calculated according to the commercial connection ploicy");
        return(nounits*3.5f);
    }
}
```

//Code to create dom object and calculating the bill

```
class dom extends conn
{
    dom(int connid,String custname)
    {
```



```

        super(connid,custname);
    }
    float calcbill(int nounits)
    {
        System.out.println("You bill is calculated according to the Demestic
connection policy");
        return(nounits*2.5f);
    }
}

```

//Code to print the connid, custname and calcbill of com and dom connections

```

class poly
{
    public static void main(String args[])
    {
        Conn c1=new Conn(1,"Sumanth");
        Conn c2=new Dom(2,"Anil");
        System.out.println(c1.getid());
        System.out.println(c1.getname());
        System.out.println(c1.calcbill(20));
        System.out.println(c2.getid());
        System.out.println(c2.getname());
        System.out.println(c2.calcbill(20));
    }
}
/* Insted of writing the redundant code like this, we can reduce the code by writing as
follows*/

```

//Method to create co, dom objects and sendbill()

```

class poly
{
    public static void main(String args[])
    {
        com c1=new com(1,"Sumanth");
        dom d1=new dom(2,"Anil");
        sendbill(c1,200);
        sendbill(d1,250);
    }
}
//method to print the details
public static void sendbill(conn c,int nounits)
{
    System.out.println('-----');
    System.out.println("Dear Mr"+c.custname);
}

```

```

        System.out.println("your connection id is"+c.connid);
        System.out.println("Total no of units consumed:"+nounits);
        System.out.println("your total bill is"+calcbill(nounits));
        System.out.println('-----');
    }

```

Output:

*Drear Mr. Sumanth
 your bill is calculated according to the commerical connection policy.
 Total no of units consumed: 200
 Your total bill is : 700/-*

In class poly{.....} we can effectively reduced the redundant coding. And here we can easily make the future modification without touching the source code. We have defined the calcbill() dummy implementation in super-class; and that is over-ridden in the sub-classes of com and dom

If a method is declared as an abstract method then the class defined in that method must be an abstract class

Abstract method is better than that of the dummy implementation in the super-class. This allows the sub-class creator to provide the implementation.

For an abstract class you can have the concrete -class. Concrete class should have the implementations of all the classes.

You can able to create the variables to reference to the super-class object but you cannot create the object using 'abstract-classes /methods'.

Q) I bought the abc(); from one company and in that xyz(); is an abstract method,for my requirement i defined subabc();as the subclass,in that can i use abstract xyz();?

abc() abstract xyz();	sub abc() axyz(); abstract xyz();
--------------------------	---

If java allows this type ,in which case you will use this types?

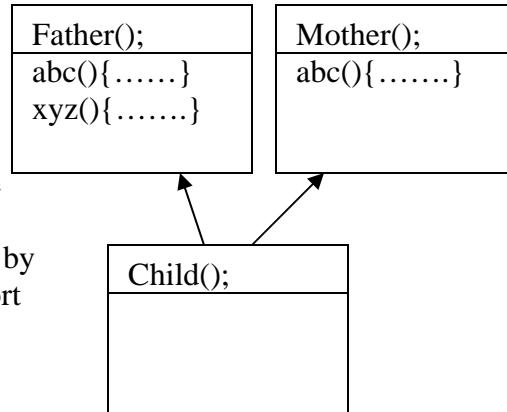
Ans: If you are going to create several sub-classes of abc(); but none of these have the same name. In this case we will use this types.

Using this we will write the polymorphic code.

NOTE: observe this whether java language supports this or not.

Multiple Implementation Inheritance:

In this figure child class inherits the concrete methods, both from father and mother super-classes. But here `abc(){.....}` method is available in both the super-classes. In this case an ambiguous situation will arise. This is known as 'Multiple-Inheritance', and this is supported by C++ language, but Java language doesn't support this 'multiple inheritance'.



Q) Why "Multiple-Inheritance" is not supported by Java language?

Ans: JavaSoft engineers eliminated this facility because this will create more problems and in some cases this will break the classes.

At maximum you can have only one direct super-class

To overcome this:

- 1: provide some sort of work around the sub-class.
- 2: Remove this kind of facility (concept) from the language.

Note: MFC (Microsoft Foundation Classes) are the most widely used classes in the world. There are more than 250 classes. Even though they are developed in C++ language, they have not used "multiple inheritance" facility any of these classes. Because even though it solves many problems, but it creates much more problems.

This type of situations can be solved in Java Language using "Multiple Implementation Inheritance" (MII).

MII is a set of methods without having implementations .i.e. a set of abstract methods.

In this case Java allows the developer to declare new variables reference to the super-class but not the implementation.

Ex:

```
abstract class abc(){
    abstract int x();
    abstract int y();
}
```

/ if in some cases when the developer by mistake writes the code(statement) as `abstract int x(){...}`, then this will cause a problem. Because it treats as the definition of a class. Here abstract class does not have the definition/implementation. */*

Instead of declaring all the classes as an abstract classes, better to go for interface. Using an interface we can write the above example as follows:

Ex:

```
interface abc{ //Interface is a keyword
    int x(); //like an abstract
    int y();
}
```

Advantages of an Interface:

1. You need not to say explicitly the method as an abstract method.
2. There is no chance of providing the implementation

Note: All the interface classes by default is public and abstract.

Try the following example whether java supports or not:

```
class xyz implements abc(){.....}

Interface abc(){
    int x();
    int y();
}
```

In Java language, we can inherit the methods implementation.
we can not provide implementation for an abstract class.

While designing a product/project, you should consider the following 3 rules in mind:

1. Product vendors who develop and who sells the product-->Vendor 1(v1)
2. User of the product -----> user/customer (c1)
3. Third-party vendor who adds the enhancements to our product without touching the source code. (TPV1)

Interface: is a set of methods that can be implemented

```
-->interface IAddressbook{
    String getName(int id); //Interface
    String getAddress(int id);
}
```

Any no of classes can have the implementation.

```
>class Netscape implements IAddressBook{.....}
/* This line of the code implements all the methods that are declared in the
```

IAddressBook should be implemented in the class NetscapeImpl. If you want to add more classes to this, you can also do this.

If you don't want the implementation of any of these classes, simply use the 'abstract' keyword in front of those classes/methods.

Ex:

```
Interface x{
    int abc();
}
class ximpl implements X{
    public int abc(){.....} //implemented from x
    public int xyz(){.....} //one more method is added here
    abstract public int cby(){.....}
}
```

An interface variable can be printed to the implements the interface.

```
X a=(X)new XImpl();
a.xyz();//not valid
a.cby();//not valid
```

because X-represents the abc(); method only.

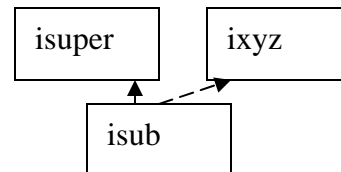
```
a.abc();//valid
```

I don't want the implementation of cby(); method that's why I used the 'abstract' keyword there.

Implements --> we are implementing the super-interface.

```
isub implements isup{ ...}
```

here we are implementing the interface of isuper to the isub.



```
isub implements isuper, ixyz{....}
```

In this case we are implementing the interfaces of isuper and ixyz to isub. Here this shows that the java language supports "multiple implementation inheritance".

Advantages:- A child can be viewed as father, and a child can be viewed as mother. child inherits the properties from both father & mother.

A class can implement any number of interfaces. This is identified by the dotted line as shown in fig.,

Ex: Intimpl.java --> Interface implementation application.

```
interface IAddressBook{
    String getName(int id); // This is an interface
    String getAddress(int id);
}
//code to get the addresses from outlook express.
class outlookimpl implements IAddressBook{
```

```

    public String getName(int id){
    switch(id){
        case 1:
            return new String("Murthy M");
        case 2:
            return new String("Anil Kumar A");
        default:
            return new String("no name matched, try again");
            null;
        }
    }
}
public String getAddress(int id){
    switch(id){
        case 1:
            return new String("murthy_vsm@yahoo.com");
        case 2:
            return new String("aniladusumalli@hotmail.com");
        default:
            return null;
    }
}
}

```

//code to get the address from the netscape navigator

```

class NetscapeImpl implements IAddressBook{
    public String getName(int id){
        switch(id){
            case 1:
                return new String("Murthy M");
            case 2:
                return new String("Anil kumar A");
            default:
                return new String("no name matched");
        }
    }
    public String getAddress(int id){
        switch(id){
            case 1:
                return new String("murthy_vsm@yahoo.com");
            case 2:
                return new String("aniladusumalli@hotmail.com");
            default:
                return new String("no address matched");
        }
    }
}

```

```
}
```

/ here, you are using different classes, but the Interface is same. So we are getting the polymorphic effect here. */*

// main class to run an application.

```
class IntImpl{
    public static void main(String args[]){
        System.out.println(args[0]);
        class c=class.forName(args[0]);//created an object at run-time
        IAddressBook ia=((IAddressBook) c.newInstance());
        //creates new object by taking the class name
        //from the command-line arguments i.e., it is
        //dynamically creating an object
        System.out.println(ia.getAddress(1);
    }
}
```

compile this, and at the run-time pass the classname from which you want to pull the addresses.

```
intImpl java outlookImpl
//NetscapeImpl
//NetscapeImpl murthy_vsm@yahoo.com
output:- outlookImpl
murthy_vsm@yahoo.com
```

In java language, arrays are treated as objects. That's why when we pass the class name as command line argument; it takes that class and treats this class as an object dynamically at run-time.

If a third-party vendor wants to develop a new-class to pull the addresses from the IAddressBook. He need not touch the source code develop by us he can do this by using the interface of the 'addressBook'

In UNIX o/s --> '0'(zero) represents No error condition. Why because 0 for success, and other than zero represents different failures.(Error messages)

Modify the IAddressBook example by declaring IAddressBook as a static variable and provide the implementation of the method setImpl and make this as a static method. Inside this method create an object of type IAddressBook and observe the output.

```

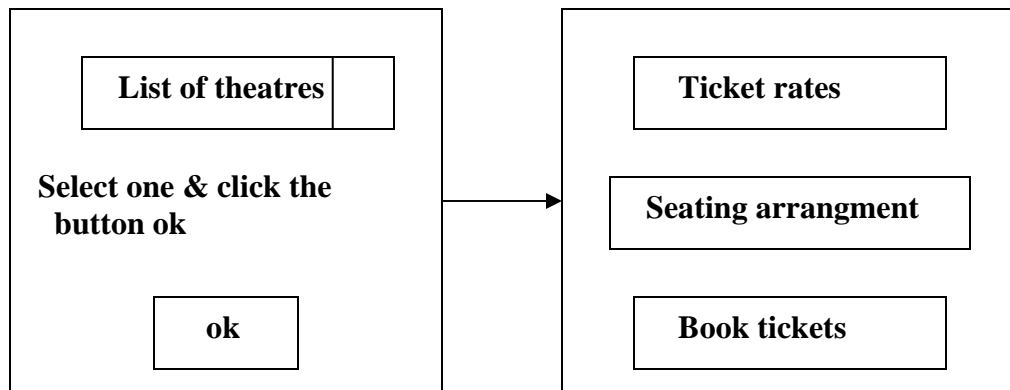
/* static method for IAddressBook Example */

class IntImpl {
    public static void main(String args[]) throws Exception {
        for(int i=0;i<args.lenth;i++) {
            setImpl(args[i]);
        }
        public static void setImpl(String R) throws Exception {
            class c=class.forName(k);
            IAddressBook ia=((IAddress Book) c.newInstance());
            int id=5; //no., of times pritns the values
            for(int i=0;i<=id;i++) //To simplify the code
            {
                System.out.println(ia.getName(i));
                System.out.println(ia.getAddress(i));
            }
        }
    }
}

```

GUI application Example:-

Write an application to communicate the theatres in Hyderabad though the network and that should be as shown in figure:



Application should be like this and it should behave as shown. To simplify the application first if one clicked book tickets number of tickets irrespective of the seating arrangement should be booked according to one's requirement.

How can you design the flexible application and if a new theatre manager wants to join in this program then you should not allow the base code to be modified?

You must consider the following things to design the flexible application.

1. Seating arrangements from theatre to theatre is different.
2. Business policies are thoroughly observed, and every theatre has its own ticket rates.

You cannot write the generic code to contact all the theatre servers, because ticket rates are different.

Operations of all the theatres are same, but the implementations are different.

Write one Interface as:

```
ITheatre{  
    int[] getTicketRates();  
    int[] getSeatingArrangement();  
    boolean (day,showno,class);  
}
```

Write one file with 10 classes (suppose if we are communicating the 10 theatres), with

TheatreName and class of the theatre, display the list of the theatres in a GUI. Theatre name & class of that particular theatre is dynamically taken from the file.

Get the file from the theatre, select one theatre and click ok. In this case the class which loads the file, create one instance (new) with that class and perform the operations that are required.

If any other theatre owners want this facility, simply add one more class to the file and perform the same operations. In this you are not touching the source code.

Methodologies: Standard ways of finding the designs to solve a particular problem.

Types:-

1. Booch
2. Object oriented programming and
3. Object oriented programming software engineering methodologies

But no one is following the specific methodologies for designing

UML(Unified Modeling Language): is used to represent the notations or conventions for your design.

Callback mechanism:- calling function by passing the address as parameter in another function.

address of one function is sending as parameter to another function.

In 'c'-language:- If you want a flexible design or a c-language, you must use callback mechanism this can be done by using the pointers in c.

Ex:- Designing a sales tax calculation for the particular item.
(using c-language)
c

```
CalcTax(int itemcode){  
    CT= code for calculating Central Tax  
    ST= code for calculating state of andhrapradesh  
    (this is different for difficult states)  
    Total Tax = Central Tax + State government Tax  
    return;  
}
```

This is not a flexible design, if we want to calculate the State government Tax for other states, then we should modify the code. The design is such a way that we should not touch the base code, even though we are calculating the tax for other states also.

This is done through pointers and this design is known as "callback mechanism"

Assume that you have a function that takes the address of a function dynamically, which is used to calculate the state-tax(S.T)

```
calcTax(int itemcode, address of a function);  
  
calcTax(int itemcode, addr){  
    StategovernmentTax=addr; //calling function is parameter  
    TotalTax= CentralTax + Stategovernmenttax;  
    return;  
}
```

This is flexible design, and using this you are able to calculate the taxes for other states also, but here you are not touching the source code.

This is widely used to develop the products, using c-language

Callback function in java language: We don't have the direct method to get the pointers in java. But using Interface we are do this.

```
interface IST{  
    int calcST();  
}  
class STAPP{  
    public static void main(String a[]){
```

```

        Ap a= new Ap();
        calcTax(1,a); //Thsi is a caller function
    }
    public static int calcTax(ing itemcode, IST is    ){
        System.out.println("here is the code to calculate the CT");
        ct=11; //assume that 'ct' is calculated according to the procudere
        st=is.clacst(); ..calling function, passed by the caller
        retunr ct+st;
    }
}

```

/* Any class that prevides an interface is used to create an object, and this object is passed as a paramers to the other function */

```

class AP implements IST{
    public int calcst(){
        System.out.println("according to the ap government vales");
        return 11; //code to perform the logic.
    }
}
class kar implements IST{
    public int calcst(){
        System.out.println("According to karnataka government");
        return 11;
    }
}

```

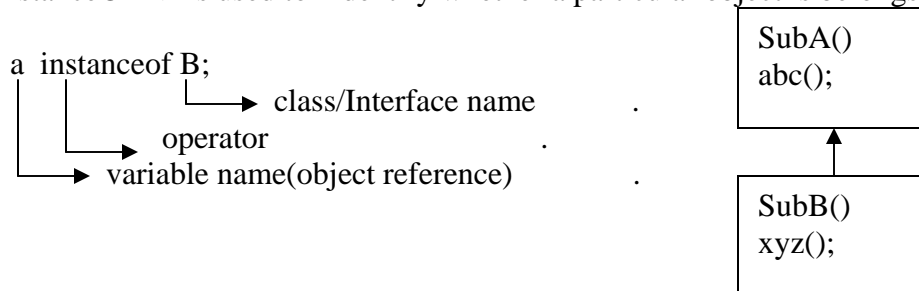
InstanceOf:-

```

A a=new A();
a=b;

```

instanceOf --> is used to indentify whether a particular object is belongs to a or b class.



In order to dynamically find out whether an object is created based on which class can also be found using the 'instance operator'.

The above returns 'true', if the class based on which the object is created provides the implementation of the interface. If it returns 'false' then a particular object is not belongs to that class.

In most of the cases while performing down-casting, it is always advisable to 'instanceof operator' and to check whether the conversion is possible or not.

Marked/Tagged Interface: Interface which has zero interface methods, then it is known as marked/Tagged interface.

Interface without any methods.

Ex:-

```
interface T1{  
    }  
class A implements T1{.....}
```

This is a very convenient mechanism to the developer to put a marker/tag on an object to identify the objects of particular type.

JavaSoft used this technique in identifying the:

Ex:-

1. Remote or local objects, the interface is declared without any methods.
2. Whether the object is serializable or non-serializable, (tagged interface as serializable interface)
3. To identify an Enterprise Java Beans or not.

Ex:- If you have n no. of classes to identify that some classes belong to marketing, and some belong to production, and some belong to the financial. In such a case we can use marker/tagged interface

```
/*Marked/Tagged Interface Example */  
interface marketing { }  
interface production{ }  
interface financial { }  
  
class abc implements marketing { }  
class def implements production { }  
class xyz implements financial { }  
  
class tagged{  
    public static void main(String ar[])  
    {  
        abc a=new abc();  
        def b=new def();  
        xyz c=new xyz();  
  
        System.out.println(a instanceof marketing);-->True  
        System.out.println(a instanceof production);-->False  
        System.out.println(a instanceof Financial);-->False  
  
        System.out.println(b instanceof marketing);-->False  
        System.out.println(b instanceof production);-->True  
        System.out.println(b instanceof Financial);-->False
```

```

        System.out.println(c instanceof marketing);-->False
        System.out.println(c instanceof production);-->False
        System.out.println(c instanceof Financial);-->True
    }
}

```

Try with

```

class def extends abc implements production{
    System.out.println(a instanceof marketing);-->True
    System.out.println(b instanceof marketing);-->True
    System.out.println(b instanceof production);-->True
    System.out.println(c instanceof Financial);-->True
}

```

Exceptions:-

fp=fopen(".....");//Code in c-language,
fread(".....");//To open and read a file.

If a file is not found, c-compiler doesn't give any errors. The c-compiler should not restrict the developer to impose any care to handle the possible errors.

Programmer has not checked for the errors:

1. Absorb the function and
2. Returning the error code as return value.

Most of the library functions are designed to perform the above two strategies.

Handling the exceptions in Java language is nothing but handling the possible errors in c-language.

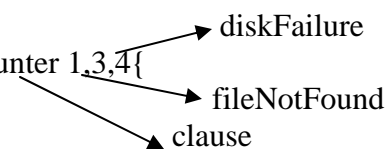
In c-language compiler there is no direct way of expressing that myfunction may encounter the errors.

In Java:-

```

int function1() mayencounter 1,3,4{
    i=function1();
}

```



The language should be designed such that function should have a clause like the above example so that anybody used the function, by seeing this he should know that there may be a chance of encounter the possible errors of 1,3,4.

If possible programmer should mention in the program that:

1. Programmer can handle the errors.
2. Programmer should explicitly mention that, he is not able to handle the errors.

in c-language this type of facility is not available.

In java language all the errors/exceptions are represented as an objects. Handling the errors in java language is know as "Exception handling".

Ex:- *public class excep*

```
{
    public void Error() throws ClassNotFoundException
    {
    }
    public static void main(String ar[]) throws Exception
    {
        class.forName("abc");
        System.out.println("After method call");
    }
}
```

Compile the program without throws exception first, and observe the result. It will gives an error such that:

ClassNotFoundException must be caught or must be declared in the throws clause of this method. Modify the program, and run with the classname "abc", then there will be no errors, and the output will be displayed as "after the method call".

While in the case when an error occurs the execution will be stopped where the error occurred. The next statements after the error should not be executed, if there is no such class name, it will goes to the caller.

Here JVM is the caller, which handles the errors occurred during the program execution. JVM will display the stacktrace (Which is having all the details of an error i.e where an error is occurred, error message etc) and terminates the program, when we are not going to catch the exceptions, at run-time we will get an error and the stack-trace will be displayed.

Stack-trace: In order to detect the errors stack trace is very useful, to tell developer where an error is occurred.

Ex:-

```
class Excep
{
    public static void Error() throws Exception
    {
        System.out.println("In method call");
        Class.forName("Excep");
        System.out.println("In method call");
    }
    public static void main(String ar[]) throws Exception
    {
        Error();
        System.out.println("After method call");    } }
```

By changing the program observe the results.

Try & Catch: Try block is used to catch the exceptions in java language. Try block is a set of statements that are enclosed in a curly braces.

Ex:-

```
try
{
    Class.forName("ex");
    System.out.println("in an exception");
}
catch(ClassNotFoundException e)
{
    /--->Exception object pointed by this variable.
    System.out.println("Inexception");
}
```

When try is failed, then catch will be executed.

If try is executed without fail, then the catch block will be skipped, and the statements followed by the catch block will be executed.

If there is any error in try block or try block is failed, the statements in that block should also be not executed.

If there are multiple statements and multiple errors in a try block, if possible try to write the statements to catch all the errors in a catch block or catch some of the errors and throws the remaining errors.

Single try block can have any no. of catch blocks.

Q) *fopen();*

----- *if error occurs before the fclose(); then what can be done to handle the error?*
fclose();

A). In this case write *fclose();* method in a catch block, But this is not recommended. In that case use finally block.

Finally block: This blocks main purpose is to perform some clean up operations, and this will be executed irrespective of the success or failure of the try block.

It is guaranteed to be executed in all most all the cases.

```
/* Exception Example */
class ex
{
    public static void Error()
    {
        System.out.println("In Error method");
    }
    try
```

```

    {
        //class c=class.forName("ex");
        System.out.println("In a try block");
        //object o=c.newInstance();
    }
    catch(Exception e)
    {
        System.out.println("In a Exception Handler");
    }
    finally
    {
        System.out.println("In Finally block");
    }
}
public static void main(String ar[])
{
    System.out.println("In main method");
    Error();
    //class c=class.forName("xxx");
    System.out.println("In a method");
}
}

```

Output:-

In main method.

In Error method.

In a try block.

In finally block.

In a method.

Try with different options and observe the results.

String a[i] = new String();

In this case n number of objects(Strings) should be created, and an error out of memory space or memory is not available will be occurred.

JavaSoft find out three types of exceptions that are occurred during compile & run-time execution.

1. Checked exceptions: These exceptions are checked by the compiler itself. Programmer can handle or throws these exceptions.
2. Run-time exceptions: These are occurred during the run-time of the program, and a programmer should take care about these errors. These are not checked by the compiler.
Ex:- divide by zero. Array out of bound.
3. Catastrophic exceptions: These errors are not handled by the user, and these are occurred due to the insufficient memory space and due to the errors in JVM. JVM can't

do and a programmer also can't do anything when these exceptions are occurred. In such a case JVM terminates the execution or a programmer should shutdown the machine.

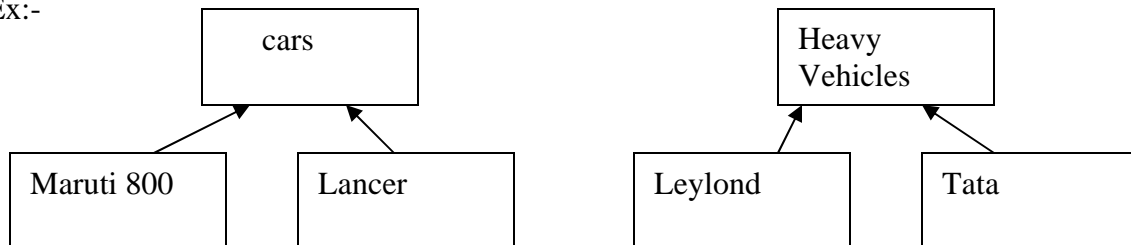
ex:- out of memory space.
stack is not available.
Error in JVM.

Runtime exceptions are designed by JavaSoft engineers, not to put much burden on compiler to do all the compilations (even though it is possible to design a compiler such that it can do any time of compilation checkings), it will take much more to execute even a simple application. To avoid this and to improve the performance of an application, the run-time exceptions are designed. It is the developer response to take care and to handle these exceptions.

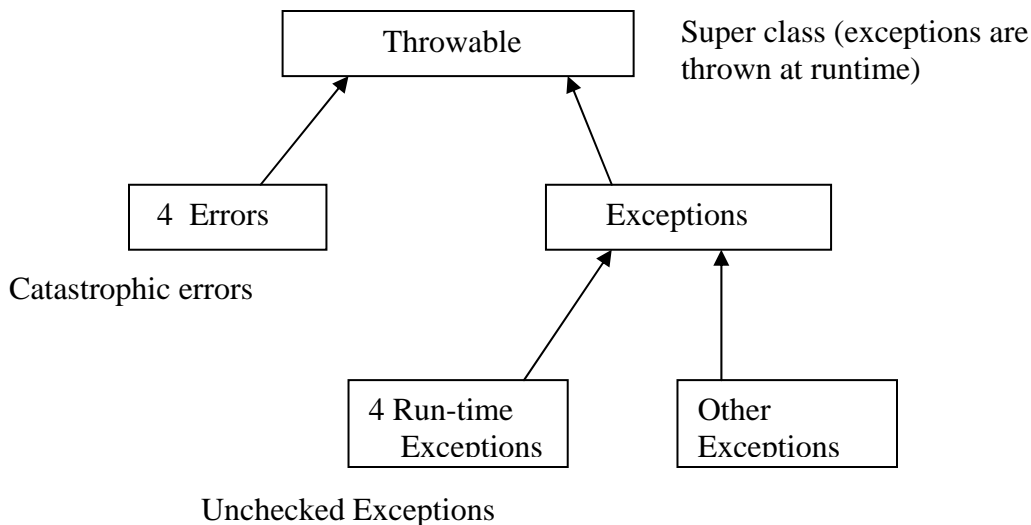
In java language every exception is an object, so each and every exception should have one separate class.

To find out whether a particular exception falls under particular category, check whether the sub-class belongs (or) falls under the super-class or not. If it falls under the super-class, we can simply say that this exception belongs to that super-class, otherwise it is not belongs to that category.

Ex:-



Exception Diagram:-



By seeing this diagram, one can easily say that this exception should fall under this category and so on.

Checked Exceptions: Checked Exceptions are sub-classes of Exceptions but not of runtime exceptions.

ex:-//Runtime Exception example.

```
class excep1
{
    public static void main(String ar[])
    {
        System.out.println("After method call.....");
        int i,j,g,o;
        i=i/j;
    }
}
```

Output:-

At compile time there is no error.

At runtime ArithmeticException : / by zero -- error occurs

Ex:-//out of memory space error example

```
//import java.lang.*;
class excep{
public static void main(String arg[])throws Exception
{
String[] s={" "," "," "," "};
try{
    System.out.println("After method call...");
    for(int i=0;i<99999;i++)
    {
        System.out.println(i+" ");
        s[i]=new String("RajRajRajRaj.....");
        System.out.println(s[i]);
    }
}
catch(Exception e)
{
    System.out.println("Error Handler.....");
}
}
}
```

Note:-First try without try & catch and try with try & catch and observe the results. In the first case programmer should not be able to do anything he has to Shutdown the system.

In the Second case we are handling the error, so "Error handler....."message will be displayed.

And a programmer can move forward to the next step.

```
catch(Error e)
{
e.printStackTrace();
}
```

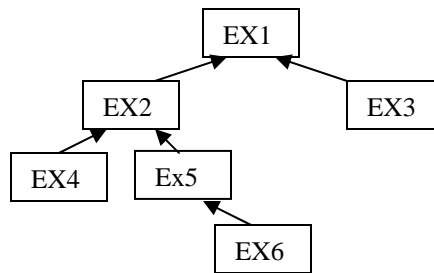
All the exceptions are the subclasses of an error object/class. So, we can handle the errors/exceptions using "Error class" in the above catch.

Even though run-time exceptions and Errors are not checked by the compiler, a programmer is responsible to write an exception handler for both of them.

Note:-Observe the class/Exception hierarchy diagram carefully before catching an Exception.

In this case EX3 is subclass of EX1 and EX5 &Ex6 are the sub-classes of Ex2 and Ex2 is a sub-class of Ex1.

```
catch(Ex3 e){
s.o.p("3rd exception");
}
catch(Ex1 e) {
s.o.p("5th and 6th exception");
}
```



Super-class object reference to the sub-class object. write a simple program and find out whether we can use catch and throwable in a single class?

If you are catching two exceptions and throwing one then write the catch block to handle the two exceptions and then write a catch block to throw the one.

Otherwise you will get an error saying that : "catch not reached" at the compile-time.

Ex:

```
import java.lang.*;
public class excep{
public int Error() throws InstantiationException{
int Status=0;
try{
Object o=Class.forName("excep").newInstance();
Status =1;
}
catch(Exception e)
{
System.out.println("Please check the config file2");
}
```

```

}
try{
Object o=Class.forName("excep").newInstance();
//Status 1;
}

```

/(or) Instead of writing all these try & catch blocks, we can simply write the code as a sub-class of Exception*/*

```

catch(Exception e)
{
System.out.println("Please check the config files"+e);}
//return 0;
return Status;
}

```

```

//main class
public static void main(String args[]) throws InstantiationException
{
excep e=new excep();
System.out.println("Before method call");
System.out.println("After method call"+e.Error());
}
}

```

While developing a project/program, a java developer may use a pre-existing java exception classes or he may create his own exception classes. All the exceptions classes created by the developer should be checked exceptions. The compiler should check it, whether a caller is handled or not.

/ We can derive the class from any of the sub-classes of throwable class*/*

```

class mychkException extends Throwable{
mychkException(){//Zero argument constructor.
super();
}
mychkException(String s){//Single argument constructor.
super(s);
}
/*public string toString(){
return new string("RajRajRajRaj.....");
}*/
}
class excep3{
public void Error() throws mychkException{
throw (new mychkException("Exception mye mychkException"));}
public static void main(String arg[])
{

```

```

excep3 e=new excep3();
System.out.println("Before the method call");
try{
e.Error();
};//student
catch(mychkException mye){
System.out.println(mye);/*object class passed as a constructor during the costruction of
a class,to override the toString method */
}
finally{
System.out.println("This part of the code exeutes irrespective of the errors in try
block");
}
}
}
}
output:

```

There should be atleast two constructors in a class, even though one constructor works good. But this is according to our requirement.

Zero argument constructor return the toString method.toString method will give name of the class:name of the caller.

one argument constructor overrides this and returns the appropriate exception message/class name.

If you want to create your own exception class, create a sub-class of Exception, but not the subclass of a runtime exception/catastrophic exception.

```
-->class myException extends Exception{.....}
```

This creates myException class which is a subclass of an Exception.

While creating your own Exception class, it is very important to give an appropriate name to your exception class. Because, the other developers who are using your class should knows that the type of error that will reflect during the execution of that particular method.

toString method returns:

Name of the class: String passed as a parameter. Throwable class provides the implementation of 'toString' method which will display the name of the class followed by the string passed as a one argument constructor.

In general we will not be providing the overriding method toString in our own exception class.(the result is according to our own implementation,if we have override toString by writing our own toString method.)

why you are not creating your own Exception class as a sub-class of run-time exception?

Ans: If we create our own Exception class as a sub-class of a run-time/catastrophic exception class, then the compiler should not check the error that are occurred and the compiler will not give any error messages that handles or throws the Exception .while at run-time the caller JVM receives this and simply terminates the program.

This is the main problem with the run-time and catastrophic exception classes. That's why we are creating our own Exception classes as a sub-class of an Exception classes. In most of the Exception classes, we may not providing much functionality apart from deriving the class from Exception classes.

Environment Variables:-(E.Vs)

E.V is a variable with name and value.

If we give any command at DOS prompt in DOS or WINDOWS operating system, it searches for the file with the given name. If exists it displays appropriate message, otherwise it will gives an error such that command with the given name is not available. Generally in DOS or WINDOWS operating systems where searches for the files. In windows operating system it will searches in c:\winnt\iview. All the commands should be checked in this directory.

E.V's are used to search for the commands given at the command prompt. Whether they are in a particular directory of an operating system or not.

c:\>echo %PATH% in DOS operating system.

\$ echo \$PATH\$ in UNIX operating system.

* In general it is better to use capital letters for E.V's

IN DOS O/S TO SET OUR OWN E.V.'s:-

c:\> set MYNAME= Value..

c:\>echo %MYNAME% To see the value

Value -> displayed here.

c:\>

Once you set the E.V's in DOS on winNt flatform, if you closed the DOS window and again open the DOS window then the E.V's that you have set should not be available. Because, they are not saved to the disk. Try with the above Commands and observe the result.

HOW TO MAKE IT PERMANENT THIS E.V's:-

Goto the Start -> Control panels -> select Environment Tab from the window.

In the Administrator you will find two text boxes:

Variable

Value

Enter the variable name & value, press **set** and **apply** Buttons. Then open the New DOS window, and check whether the given E.V value is stored permanently or not.

```
c:\>echo %xyz%...
```

xyzvalue -> will be displayed.

Note: You must close the old DOS window and open the new window to reflect the E.V's that are set in the control panel.

E.V's are used by the o/s or these can be used by the program. for Ex:windows o/s,unix o/s uses the environment variable 'PATH' in order to search for the executable file.

After creating an E.V. provide the documentation to know the other users how to use this in the programs.

If you store all the E.V's in the following directories:

c:\abc, and

c:\abc\xyz

To retrieveing, you must search through the PATH environment variable as:

```
PATH=c:\abc\xyz; c:\abc;
```

To identify in which folder the o/s to search for the files.

Ex:- *x=geten("UN"); //UN is an E.V.developed*

```
printf("%S",x); // Created by some one
```

As per the user/your requirement,user/you can set the E.V as:

```
set UN="Hai Raj";
```

Whenever you call this E.V UN. This will display "Hai Raj" message.

Whenever an E.V.'s are used in the program, the document of the program will explain about the set of E.V's that are used by the program.

Ex: The tools in the JDK uses the E.V. called 'CLASS PATH' -> This is known according to the document.

While you are writing an Environment variables in your program,you have to decide the name for the E.V's . Any number of E.V's can be created, in some M/C's this is limited according to the ..M/C. configuration.

In windows95/98 all the E.V's are saved in the 'c:\AutoExec.bat' file.If the E.V's are not available in c:\ folder, then ask the user to store all the E.V's in the output directory in c:\.

```
getEnv("outdir");
```

```
getEnv("classpath");
```

CLASSPATH E.V should not be used by any other tools, but Sunsoft\Sun Microsystems are the only one user who are using this, in their tools.

Classpath can also be set with a list of directories. but they are seperated by a ;.

Setting the Classpath:-

```
c:\> set CLASSPATH=c:\;d:\...
```

.(dot) represents the current working directory from where you are issuing the commands.

Why you need to set the Classpath:-

Java tools to search for the class files in the directories mentioned(javac,Java,Javap) into the CLASSPATH E.V that are supplied by the Sun Microsystems in order to search for the classes.we can set the classpath to a list of directories separated by a semicolon(;).

```
* c:\>set CLASSPATH =. ....
```

This will set the CLASSPATH not related to any of the folders.

```
* c:\>set CLASSPATH =C:\ . ....
```

This will search for the classes in the c:\folder only, but not the sub-folders of c:\.

```
* =c:\:. ....
```

searches in the c:\ folder and in the current working directory.

```
* c:\> set CLASSPATH=%CLASSPATH% .....
```

This will set the previous classpath, instead of typing the whole path.

```
* to see the current path use
```

```
c:\>echo %CLASSPATH%...
```

\$pwd... In Linux (present working directory) to know the current working directory.

If the two users created the two class files, and one is placed his file in c:\abc, and the other one is placed his file in c:\xyz. If you set the as

```
c:\>set CLASSPATH = c:\abc; c:\xyz ...
```

At the run-time you will get an error, so to avoid this, set classpath to i.e. to the current working directory and you store all the class files in that directory (or) create only one version of class file.

```
c:\>PROMPT=x... -> To change the command prompt
```

x-> will be displayed, but this is not a command prompt.

To set this again to c:\> prompt

```
x set PROMPT=$P$G.....
```

```
c:\> -> will be displayed.
```

\$ P-> represents path to the current directory.

\$G-> represents the greater than symbol.

```
c:\> Javac except5.java ....
```

except5.java will be picked up from the current directory and this will search for the class file in the directories mentioned in the list of directories in the CLASSPATH.

The Java compiler of Sun's JDK, while compiling a particular class, searches for the class file in the CLASSPATH. If the class file is not found in the CLASSPATH, then the java

compiler recompile the java file available in the java classpath and creates a class file for its use.

We can set the classpath by setting the E.V or by passing as a command line option classpath. These are the two methods that are used by the Java tools supplied by SunSoft.

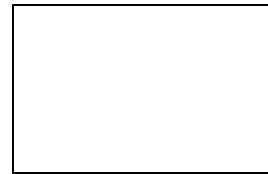
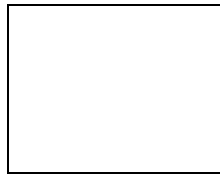
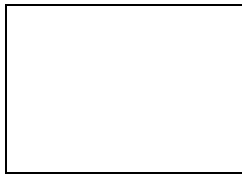
PACKAGES

Packages are purely logical. Package is a very simple mechanism provided by JavaSoft to logically group a set of classes and interfaces.

ADVANTAGES:

1. It is the developer responsibility to give an appropriate name to the packages. According to the project requirement, you are going to logically group the classes/packages.

Ex.



Oracle.pack1();
Database.oracle.pack5();
Database.Sybase.pack6();

(By seeing the package name, we can able to identify what should be done by that package.) and why it is created.

2. It is very easy to distribute classes as packages. (management of classes should be very easy)

3. Two classes /interfaces to have same names in different packages.

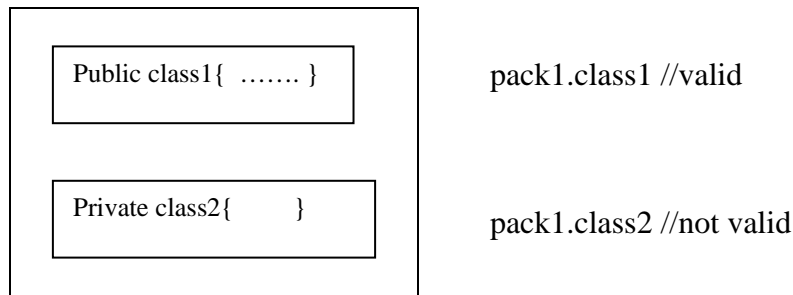
If both the package and class name of ours and the third party is same, then there is a solution by SunSoft is, we have to change any one of the package names. If the package names are different, then there is no problem even if the class names are same.

4. It forces to place the classes under different directories.

5. When we develop a class we can use two different access specifiers. one, is public and other one is Default.

public class-> is accessible outside the package.

private class-> is accessed only inside the package.



Package Name: Package Name is the reverse of your domain name. This is for better hierarchical structures. JavaSoft recommended this because there are no conflicts between the package names. But the compiler doesn't force you to follow the specific rule to write the package names.

Ex: package com.inetsolv.utilpack;

```

public class Pack
{
    public void method()
    {
        System.out.println("method is called");
    }
}
  
```

compile this class and set the calsspath to (.)current directory

C:\> set CLASSPATH=.

If there is an ambiguity in the class names, then use the fully qualified class name to identify the particular class. In our example, the fully qualified name is com.inetsolv.utilpack.classname → fully qualified package name.
com.inetsolv.utilpack.pack → fully qualified classname

```

class usepack
{
    public static void main(String args[])
    {
        com.inetsolv.utilpack.Pack p1=new com.inetsolv.utilpack();
    }
}
  
```

compile and run this, we will get an Error saying that:

com.inetsolv.utilpack.pack not found

com/inetsolv/utilpack-> Forlder structure should be unbder the search path, here it is

Folder sturcture which will reflects the package structure.

To over come the above Error, you should have com->inetsolv->utilpack folder structure under the current search path where we have saved the package. Now, again compile and run this you should not get any error message.

Distribute the classes under different directories and set the classpath for the class and the package as shown below:

```
c:\>set CLASSPATH=%classpath%; c:\path
path=where you saved the package.
```

To compile a pack as a part of package

1. c:\>javac -d c:\>pack.java
- or
2. c:\>javac -d . pack.java

In order to put class/interface as a part of package identify the output directory.
Then the output is stored in the output directory.

Whenever a class under package has to be compiled, it is better to use -d option by specifying the output directory. When -d option is specified the java compiler places the class file under the folder structure that reflects the package structure in the output directory.

un_named/Default package: is a package which has no name. When we have created a class/interface without specifying the name of the package, then it will be placed under un_named/default package. The fully qualified name of the class under the un_named/default package is the name of the class itself.

Very simple mechanism developed by JavaSoft instead of writing the fully qualified name of the class as many times as you require is

```
import com.inetsolv.utilpack.*;
(*--> indicates all classnames) at the very first line in your program.
```

In this case, it identifies/knows that the developer is need to import all the class names form the fully qualified classes, from the package.

Here, class files should not be included.

If you want only one single class, then use:

```
import com.inetsolv.utilpack.pack;
```

In some cases the compiler forces you to write the fully qualified class names, instead of importing the classes.

If you have two separate package names with the same classes in the packages with the same package structure, then the ambiguous situation occurs. In this case compiler forces you to write the fully qualified names of the classes, in the source file.

```
com.inetsolv.utilpack.pack1
com.inetsolv.utilpack.pack1 this is very rare case.
```

Because, compiler may not be able to resolve which class has to be used.

```
import com.wipro.utilpack.*;
import com.inetsolv.utilpack.*;
```

Very IMP Instructions to avoid Errors in packages:

1. Choose a particular working directory(create one new directory on your own)
Ex: c:\mydir
2. Do not create source file under this directory. Decide name and Interface of the class.
3. Create a directory structure under c:\mydir which will reflect the package structure.
ex. c:\mydir\com\inetsolv\utilpack
4. Create the source file in the above directory, i.e., in the package structure.
5. If you are compiling from c:\mydir, then give the command line argument as
c:\mydir>javac -d . \com\inetsolv\utilpack\source.java
otherwise, goto c:\mydir\com\inetsolv\utilpack and compile this from here
javac -d . source.java

Whenever we want to use the classes from a package created by others, we need to copy the classes with the same directory structure under specific directory say c:\raj. So later on to use these classes, include c:/raj in the classpath.

HOW TO SHIP THE JAVA CORE LIBRARIES:

All the core libraries that are used in the java language or used by the javasoft are combined together in a file called rt.jar

rt->RUNTIME

jar->JAVA ARCHIEV

JRE->JVM+Set of core Libraries

When ever we want to ship the set of classes instead o shipping the set of classes separately, we can create one or more jar files and give those files to customers.

Earlier with java1.0 version java soft has used a file with the name
"CLASSES.ZIP" to store all the core classes.

ADVANTAGES OF JAR FILE FORMAT:-

1. The size of the file should be reduced by compressing the combined files as in Zip files.
2. Distribution of the jar files is very convenient than distributing the individual files

CREATING THE JAR FILE

In unix o/s tar tool is used

tar-> TAPE Archeive

In windows o/s jar tool is used to create rt files

Step1: Create a working directory. Let us say work. and copy all the classes files by using an appropriate directory structure, that reflects the package structure under the work directory

c:\work>jar cvf app.jar

or

c:\work>jar csf app.jar

c-> create;

v->verbose;
f->name of the file;
s-silent

Do not silent during the run time, and create the output as app.jar file and store it in a work directory, file name as app.jar

Step2: use the command `jar cvf` followed by `app.jar` followed by a dot(`.`), which creates a jar file with the name `app` using all the files from the current directory as its sub directories.

`c:\work>jar cvf app.jar.`

WHEN YOU WANT TO USE THE CLASS FILES FROM A JAR FILE:-

`c:\work>java - classpath C:\work\app.jar usepack`

Simply include jar file in the classpath, then all the packages that are available within that jar file will be displayed as an output.

from our example, the output will be:

`com.wipro`
`com.inetsolv.utilpack` are displayed

After creating a jar file open up the jar file by using the zip method and observe that there is one default (unnamed) class without having any path name.

`usepack.class` -> is a default class and shouldn't belongs to any path in our example.

and we will find out one extra file "MANIFEST " is also created .

manifest -> is used to store some information as part of the manifest.(i.e. some additional important information about the jar file will be stored in the manifest file). This information which can be used by the tools later.

Extension to the manifest file is `.mf`

When we are compiling sources and if one of the source file is depends upon one of the class that is placed in a jar file. Then we can simply add the jar file as part of class path used by the compiler.

`c:\>jar cvf app.jar work`

It will creates a `app.jar` file, and that will be stored in `c:\directory`. when you use this in class path, it will gives an error message : `class usepack not found` .

In order to find out the table contents of a particular jar file.we can use `jar` command with 't' option

`c:\work> jar tvf app.jar`

`c:\work>jar xf app.jar`

No output will be diplayed, but the output is added to the `c:\work` directory.

`c:\work>jar xvf app.jar`
output will be displayed
v-> is used to idntify that

Mostly of the unix commands as a 'v' option ,which can be used to run the command in verbose mode.

When this is executed, the command emits messages which allows us to know what exactly is being done by the tool currently.

c:\>jar

gives the list of options that are available with the jar command
option cv0f -> Stores only,use no zip.

stores the file without compressing /zip them.

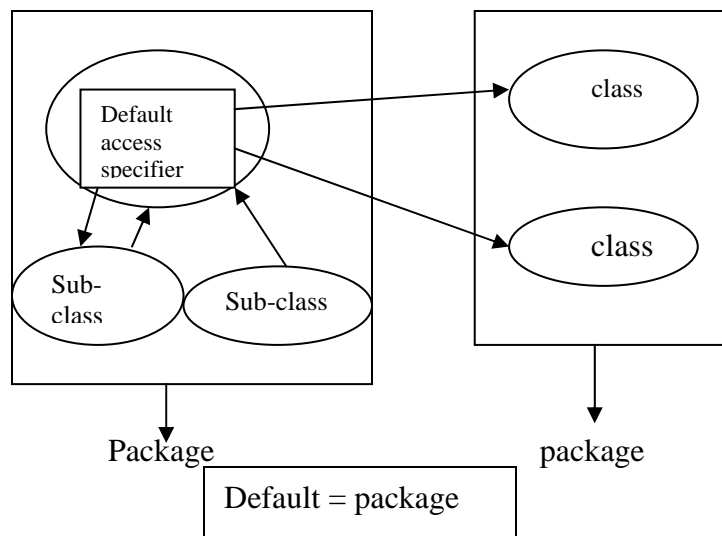
In a package:

If we have a public variable , and any one accessing this variable accessed improperly, then my object state will be in an inconsistent state.


so to overcome this type of problems we have default access specifier, and protect access specifiers available in java language.

DEFAULT ACCESS SPECIFIERS:-

Writing nothing or no access specifier before your method /constructor /class/variable.



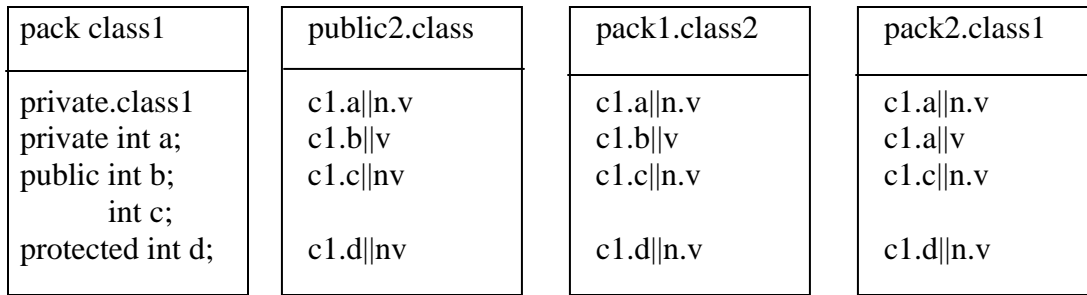
allows any of the classes ,sub classes in that package, whether it is a constructor /class/variable.

 this is a standard notation for a package according to UML.

SCOPE OF THE ACCESS SPECIFIERS :-

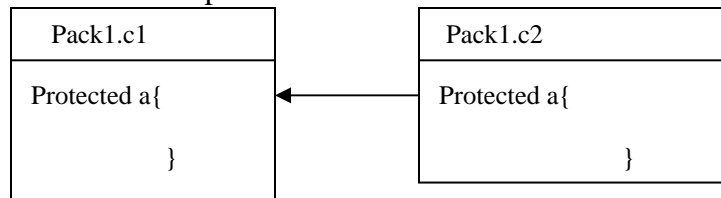
- | | |
|-------------|---|
| 1.Private | only one |
| 2.default | only inside a package |
| 3.Protected | only inside the package +subclasses outside the package |
| 4.public | every one |

CLASS DIAGRAMS



having a overridden method with lesser access specifier.

```
c1=new pack1 c2();
c1.a(); //not valid
```

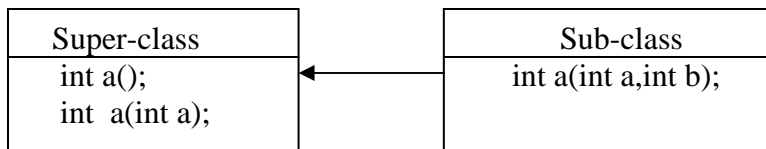


Whenever a method is overridden in the subclass method should have a wider scope or same scope as that of the method in the super class.

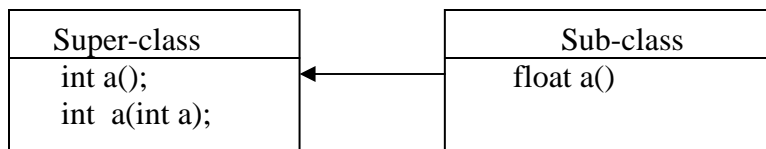
Java does not allow this type of things because here we are violating the fundamental rules

Q:can we overload a method in the same class and sub class ?

A: definitely it is possible



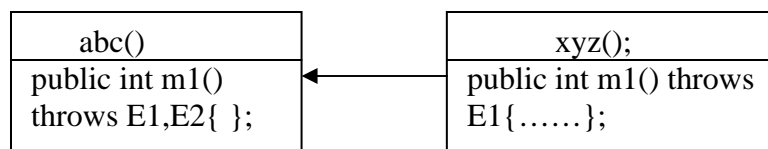
Q:



Is this possible ?

A: This is not possible and gives compilation error. for an overloaded method, signature and return type must be the same. this is not an overridden as well overloaded method

Q:



Is possible ?

A: Throws E1 & E2 in subclass method, then it is valid.

If subclass extends super class, then you should throw the exceptions every where you extends the super class what type of errors that you got.

Q) Check whether we can create interfaces with public and default scope?

In order to check to find out whether the interface can be declared as a default scope .Use javap to examine the class generated by compiling an interface.

We can have only one public class in a java source file. In the same source file we can have any no of classes with default scope. Whenever a public class is return in the java source file. Java source file should have the same name that of a public class name.

Q. xyz.java --->source file

```
public class xyz{---}  
public class abc{---}  
class def{---}
```

is this possible?

A. No, there should not be two public classes in a single java source file

```
xyz.java --->source file  
public class xyz{---}  
class abc{---}  
class def{----}  
Above is valid
```

Even though the java compiler enforces the above rules it is better to have separate source file for every class file with same name.

String and StringBuffer classes provide similar methods .but the operation is different.

You should not explicitly mention the classes that of java.lang. package .Because of compiler identifies that these imported classes from the java.lang package.

For every primitive data types in java language we have an equivalent wrapper classes.

Eg:	primitive datatype	wrapper classes
	int	Integer
	str	String
	boolean	Boolean

String class -----Immutable class

StringBuffer class -----Mutable class

Once you created the class you can not be modify this class
StringBuffer class once created,it can be changed

Eg:

```
String s1;  
String s="abc";
```


` *String s=new String("abc");/*you have a constructor as a string and you are passing a string argument to this string constructor*/*

```
System.out.println(s);
s.replace('a','r');
s.replace('b','a');
s.replace('c','j');
System.out.println(s);//abc
s1=s.replace('a','r');
System.out.println(s1); //rbc
s=new String("abc");
System.out.println(s);
s1=s;
s=s.replace('a','r');
Sytem.out.println(s);//rbc
System.out.println(s1);//abc
```

Here s and s1 are the two seperate objects.And in s=s.replace('a','r') we are creating a new string object.Previous object is not at all changed here.Because String is an immutable class

```
String s="raj";
StringBuffer="raj";
String s1="raj";
String s2="XYZ";
String s3="xyz";
```

JavaSoft has provided this mutable classes because for the code optimization .This will saves the memory.

Hashcode of an object can not be overridden.Every object has its own hashcode

```
class str{
public static void main(String args[])
{
String s1="string1";
s1=new String("string1");
String s2="string1";
s2=new String("string1");
System.out.println((s1==s2));//true
/* this will check whether the two strings s1 and s2 are pointing to the same object or not*/
String s3="string s2";
s3=new String("string s2");
System.out.println((s1==s3));//false
System.out.println(s1.equals(s2));//true
}
}
```

*/*compile writer knows that the String object is immutable class.It is possible for the code optimization in such a way that the String object references the same object*/*
int i = 20; *//This is an object of type StringBuffer.Because this can be changed during the program execution.(i = i+1;)*

System.out.println("This is :"+ i + "Value");
+→is transformed as a method at the compile time

see the byte code of this :

```
invokespecial<method java.lang.StringBuffer(java.lang.string)>  
invokespecial<method java.lang.StringBuffer append(int)>  
System.out.println("Alert");  
//Here all the string buffers are converted to strings while we used the string literal in a  
system.out.println  
Object o = new Object();  
System.out.println("This is an :"+ o);  
/* object appends to the StingBuffer.Internally object.toString() method is executed,and  
this will be passed to append to the StringBuffer. */
```

output: .classname @ hashCode

```
boolean equals(object o){  
    return(this == o);  
    }  
this is the implementation in the object class.
```

Ex: *Emp e1 = new Emp();*
Emp e2 = new Emp();
Emp e3 = e1;
e1.equals(e2); //false
/ e1&e2 are two separate objects.Under the option,we are not overriding the equals.*/*
e1.euals(e3); // true

Q:- *String s1 = new String("RAJ");*
String s2 = new String("RAJ");
s1.equals(s2); **what is the result?**

A:- equals method is re-implemented in the string class.

The equals method in the object class checks whether both the references are pointing to the same object or not. If they are pointing to the same object then it returns true, otherwise it returns false.

This method has been overridden in the string class which returns true,if the contents of both the strings are same even though the string references pointing to the different objects otherwise it returns false.

When you override an equals method:-

You have created two objects that are equal.

```
Emp e1 = new Emp(10);
Emp e2 = new Emp(10);
e1.equals(e2); //true
```

Even though they are pointing to the different/separate references, both the objects representing the same content. Here, we are overriding the equals method inside an Emp class.

```
Class Emp{ // This is according to your project requirement.
    int empno;
    boolean equals(object e){
        Emp e1 = (Emp) e;
        return(this.empno == e1.empno);
    }
}
```

Note: Whether the equals method is overridden we need to override the hashCode method by implementing an algorithm which returns the same hashCode for two different objects which are considered to be the same or equals.

e1 & e2 objects hashcodes should be the same, even though they have different hashcodes.

this is required since the hashtable class depends on the hashCode and equals method

```
/*Object class and String class Example */
class Test{
    Test(int a){} //constructor.
    public static void main(String args[]){
        Test t1=new Test(10); //Object-class
        Test t2=new Test(10); //Object-class
        t2=t1; //false.
        System.out.println("Test For Equality"+t2.equals(t1));
        String s1="hello";
        String s2="hello";
        s2=s1.replace('h','H'); // Test For s2=s1 & s1=s2
        //
        //      hello      Hello
        //      hello      Hello

        System.out.println(s1); //hello
        System.out.println(s2); //Hello

        String S1="RAJ";
        String S="RAJ";
```

```

S1=S.replace('A','O'); //false
System.out.println("Test For Equality"+S.equals(s1));
s1=new String("abc");
s2=new String("aj");
String s3=new String("abc");
s3=s1;
String s4=s3; //==>s4=s3=s1.
System.out.println(s1.equals(s2)); //false
System.out.println(s1.equals(s3)); //true
System.out.println(s4.equals(s1)); //true

```

```

StringBuffer l1=new StringBuffer("ABC");
StringBuffer l2=new StringBuffer("XYZ");

```

```

System.out.println("Test For Equality"+l2.equals(l1)); //false
System.out.println(l1.toString()); //OUTPUT---->ABC

```

```

}
}

```

DATA STRUCTURES: is a way in which you store the data in your computer program.
Standard data structures that are generally used :

- 1.HashCode
- 2.Tree
- 3.BinaryTrees etc.

```

Ex:  arr[10000];/*arr[0]="JOHN",arr[1]="RAj"*/
      count=0;
      store(String,word,meaning)
{//it is a function to store elements in an array//
arr[count]=word meaning;
count++;
}

```

TO SEARCH FOR A MEANING OF A SPECIFIC WORD:-

```

search(String){ //function
for(int i=0;i<count;i++){
.....
.....
.....
return(String);
}
return("word is not found");
}

```

MAJOR ADVANTAGES WITH THIS SET UP:-

If the data is not sorted, then the retrieval of data will take much time to search the particular word and to retrieve it.

In the best case this algorithm will take 1 ms to search for the word and the worst case is 10,000 ms .the average time is 5000 ms to search for a word.

In order to search the words in a faster way, we have several cases. Standard data Structure used is 'HashTable'. This gives you how the data is organized.

Linear Table:-Here we will take search for the data in a linear manner .i.e. from start to last word.

HashTable:-requires index(HashKey)&HashCode.

Function :-

```
store(String,word,meaning){  
  hc=hashCode();  
  index=hc%10000;/if hc=10,500.  
  //then index =500.
```

```
/*Take the ASSCIII code of each and every letter from the String (word) and add them.  
You will get a number ,this is known as 'hashcode'    */
```

```
arr[index]=word;
```

```
hashCode(){
```

```
/*hashCode is used here to 10,500 hc compute index into the hash Table */}
```

Assume that word=RAJ

R=8000

A=500

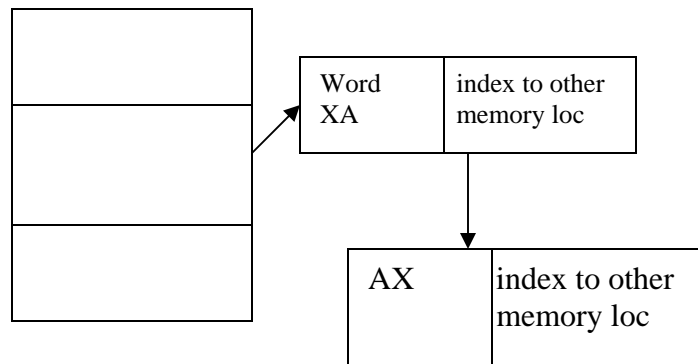
J=2000 are the ASCII values.

Search Function: Here we will use the hashcode to search a particular word directly and compares the String whether this is equals or not i.e., same or not.

Hashing with chaining:

Algorithm used to avoid the collision between the new hashcode and the older hashcode .

```
hashCode()  
{  
  X+A=101  
}
```



If the one word is going to search, then the Index to other memory location will be grounded. If You want to search another word then the index in the first location is going to refer and this will stores the new word by pointing the pointer to another memory location and the first grounded will be erased and for the new memory location grounded will be added if You want to search more, the same process will be continued.

In a program we can store the data by using different types of data structures. one popular data structure that can be used to store the data &search for the data is 'hashTable'.

In this data structure, we will be computing the 'hashcode' to find the index in the hashTable. Again when we want to search for the elements, we again find the hashcode of the element to be searched and use this hashcode as an index into the table.

Whenever we use that hashTable data structure, we need to implement the function that computes the hashcode in such a way that the data will be equally distributed in the table.(There is no Standard way, according to your requirement you can calculate the hashcode .) .

Md4 and Md5 are the algorithms to compute the hashcode of a sequence of characters. These algorithms are designed by mathematicians in such a way that they have given different hashcodes for different Strings. These puts very burden on your Central Processing Unit(C.P.U).

Java.util hashtable: we have the class with the above name which provides the implementation of hashtable.(is to store collection of elements.).

Collection Frame-work: is a set of classes and interfaces, which are mainly meant for storing set of elements.
can be stored as a part of hashcode.

When we write a program, we need to store the collection of elements using different ways. JavaSoft identifies this and designed the collection frame-work In this they have implemented the different algorithms to store and retrieve data by java programmer.

Documentation for hashTable ,Vector,Dictionary & Property----> In JDK1.2 |
Note:-SEE THE DOCUMENTATION IT IS VERY IMPORTANT.

Vector:-is another class provided by Java.util package .which can be used to store a set of elements.(any java object is known as an element)This is widely used whenever we don't know the maximum no. of elements to be stored in an array. i.e. for dynamic memory allocation .

we can use addElement() method to add element provided by Vector class to the Vector .In side the Vector class an array will be used to store the elements.When an array inside a Vector is full,the Vector class implimentation craetes a new array object with the length greater than the previous array and copies the previous elements.

There are so many classes like this which allows us to store the elements ,the major difference between these classes is the way in which the elements are stored.