www.questpond.com

The Software Interview Question Bank

-- Maintained by Shivprasad Koirala shiv_koirala@yahoo.com

How to buy the book

BPB has done a great job of making this book reach to places where i can hardly imagine. But just incase its not near to your place mail bpb@bol.net.in.
If you are from India you can contact one of the shops below:-
MUMBAI-22078296/97/022-22070989
KOLKATA-22826518/19
HYDERABAD-24756967,24756400
BANGALORE-25587923,25584641
AHMEDABAD-26421611
BHATINA(PUNJAB)-2237387,
CHENNAI-28410796,28550491
DELHI/NEW DELHI-23254990/91,23325760,26415092,24691288

Pakistan
M/s. Vanguard Books P Ltd, 45 The Mall, Lahore, Pakistan (Tel: 0092-42-7235767, 7243783 and 7243779 and Fax: 7245097)
E-mail: vbl@brain.net.pk

If you are not from india or pakistan :-
Ray McLennan, director,Motilal (UK) Books of India,367 High Street.
London Colney,
St.Albans, Hertfordshire,AL2 1EA, U.K.
Tel. +44 (0)1727 761 677,Fax.+44 (0)1727 761 357,info@mlbduk.com,www.mlbduk.com

If you want to purchase the book directly through BPB Publication's delhi , India :-

bpb@bol.net or bpb@vsnl.com

Contents

From the Author

First thing thanks to all those who have sent me complaints and also appreciation for what ever titles i have written till today. But interview question series is very near to my heart as i can understand the pain of searching a job. Thanks to my publishers (BPB) , readers and reviewers to always excuse all my stupid things which i always do.

So why is this PDF free ?. Well i always wanted to distribute things for free specially when its a interview question book which can fetch a job for a developer. But i am also bounded with publishers rules and regulations. And why not they have a whole team of editor, printing guys, designers, distributors, shopkeepers and including me. But again the other aspect, readers should know of what they are buying , the quality and is it really useful to buy this book. So here are sample free questions which i am giving out free to the readers to see the worth of the book.

I can be contacted at shiv_koirala@yahoo.com its bit difficult to answer all answers but as i get time i do it.

We have recently started a career counselling drive absolutely free for new comers and experienced guys. So i have enlisted the following guys on the panel. Thanks to all these guys to accept the panel job of consulting. Feel free to shoot them questions just put a title in the mail saying “Question about Career”. I have always turned up to them when i had some serious career decision to take.

Shivprasad Koirala :- Not a great guy but as i have done the complete book i have to take up one of the positions. You can contact me at shiv_koirala@yahoo.com for technical career aspect.

Tapan Das :- If you think you are aiming at becoming a project manager he is the right person to consult. He can answer all your questions regarding how to groom your career as a project manager tapand@vsnl.com.

Kapil Siddharth :- If you are thinking to grow as architect in a company then he is a guy. When it comes to role model as architect i rate this guy at the top. You can contact him at kapilsiddharth@hotmail.com

Second if you think you can help the developers mail me at shiv_koirala@yahoo.com and if i find you fitting in the panel i will display your mail address. Please note there are no financial rewards as such but i am sure you will be proud of the work you are doing and whos knows what can come up.

Lets make Software Industry a better place to work ..... Happy Job Hunting and Best of Luck

Career Path Institute

Author runs the “Softwar Career Path Insitute” personally in mumbai. If you are interested you can contact him regarding admissions at shiv_koirala@yahoo.com. Our courses are mainly targetting from how to get a job perspective.

Below are some of the courses offered :-

-- Interview preparation course two days (Saturday and Sunday Batch). (C# , SQL Server)
-- Full one year course for C# , SQL Server

(B) What is JVM (Java Virtual Machine)?

Twist: - What are Java Byte Codes?

JVM stands for Java Virtual Machine. It's an abstract computer or virtual computer which runs the compiled java programs. Actually JVM is a software implementation which stands on the top of the real hardware platform and operating system. It provides abstraction between the compiled java program and the hardware and operating system.

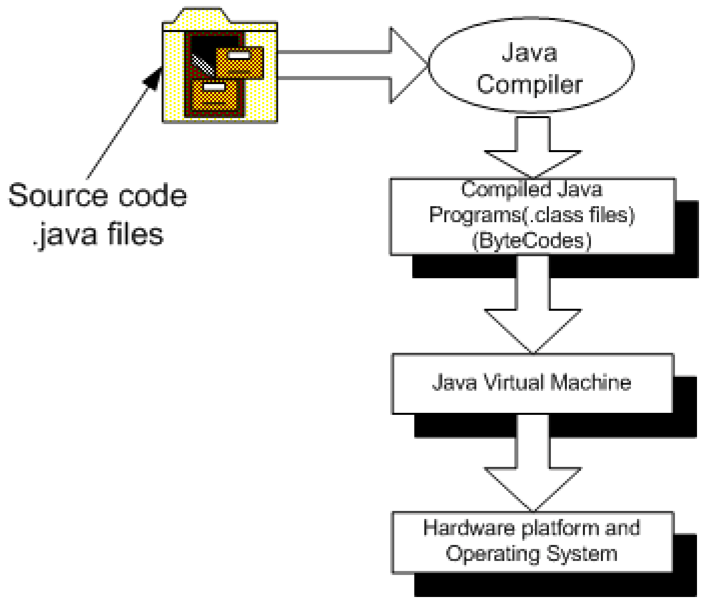

Java Compiler
Source code .java files
Compiled Java Programs(.class files) (ByteCodes)
Java Virtual Machine
Hardware platform and Operating System

Figure 1.1:- Java Virtual Machine

So the compiled program does not have to worry about what hardware and operating system he has to run in, it's all handled by the JVM and thus attaining portability. All Java programs are compiled in to bytecodes. JVM can only understand and execute Java bytecodes. You can visualize Java bytecodes as machine language for JVM. Java compiler takes the .java files and compiles it to a "bytecode" file with .class file extension. Compiler generates one class file for one source file.

(B) What is JIT (Just-in-Time) Compilation?

Note: - Before we can get in to what is JIT we need to get some basic compiler fundamentals.

There are two basic ways languages are compiled:-

√ Compiled Languages

√ Interpreted Languages

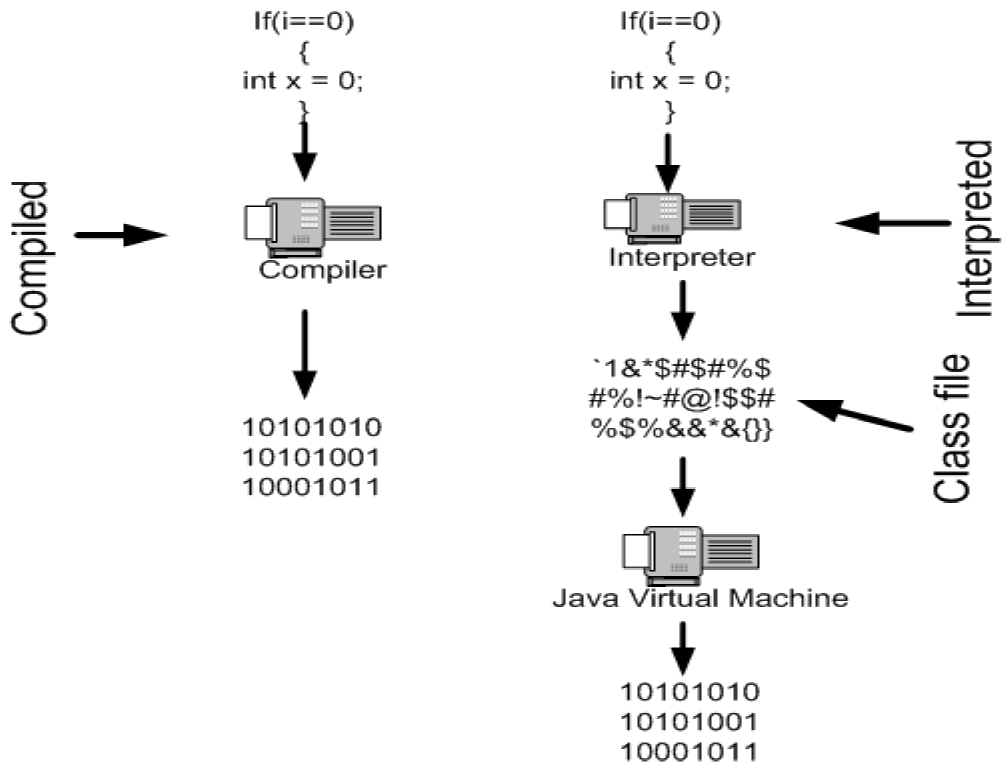


Figure 1.2 : - Compiled and Interpreted

Machine understands only binary language. So finally source code has to be compiled in binary format. Compiled and interpreter varies from the way they generate the binary files. You can see from the above figure on the left hand side we have the compiled way of generating the binary and on the right hand side we have the interpreted way. In the compiled way the compiler directly generates the binary file from the source code. While in the interpreted way it generates the class file which is then run by the virtual machine. That means the binary file is generated during runtime.

Now lets try to answer what is JIT?. When JVM compiles the class file he does not compile the full class file in one shot. Compilation is done on function basis or file basis. Advantage gained from this is that heavy parsing of original source code is avoided. Depending on need basis the compilation is done. This typ of compilation is termed as JIT or Just-in-Time compilation.

(B) What is Object Oriented Programming?

It is a problem solving technique to develop software systems. It's a technique to think real world in terms of objects. Object maps the software model to real world concept. These objects have responsibilities and provide services to application or other objects.

(B) What's a Class?

A class describes all the attributes of objects, as well as the methods that implement the behavior of member objects. It's a comprehensive data type which represents a blue print of objects. It's a template of object.

(B) What's an Object?

It's a basic unit of a system. An object is an entity that has attributes, behavior, and identity. Objects are members of a class. Attributes and behavior of an object are defined by the class definition.

(A) What's the relation between Classes and Objects?

They look very much same but are not same. Class is a definition, while object is instance of the class created. Class is a blue print while objects are actual objects existing in real world. Example we have class CAR which has attributes and methods like Speed, Brakes, Type of Car etc.Class CAR is just a prototype, now we can create real time objects which can be used to provide functionality. Example we can create a Maruti car object with 100 km speed and urgent brakes.

(B) What are different properties provided by Object-oriented systems?

Twist: - Can you explain different properties of Object Oriented Systems?

Note:- Difference between abstraction and encapsulation is one of the favorite interview question and quiet confusing as both the terminology look alike. Best is if you can brainstorm with your friends or do a little reading.

Following are characteristics of Object Oriented System's:-

Abstraction

It allows complex real world to be represented in simplified manner. Example color is abstracted to RGB.By just making the combination of these three colors we can achieve any color in world. It's a model of real world or concept.

Encapsulation

The process of hiding all the internal details of an object from the outside world.

Communication

Using messages when application wants to achieve certain task it can only be done using combination of objects. A single object can not do the entire task. Example if we want to make order processing form. We will use Customer object, Order object, Product object and Payment object to achieve this functionality. In short these objects should communicate with each other. This is achieved when objects send messages to each other.

Object lifetime

All objects have life time. Objects are created, initialized, necessary functionalities are done and later the object is destroyed. Every object have there own state and identity, which differ from instance to instance.

Class hierarchies (Inheritance and aggregation)

Twist: - What's difference between Association, Aggregation and Inheritance relationships?

In object oriented world objects have relation and hierarchies in between them. There are basically three kind of relationship in Object Oriented world:-

Association

This is the simplest relationship between objects.Example every customer has sales. So Customer object and sales object have a association relation between them.

Aggregation

This is also called as composition model. Example in order to make a "Accounts" class it has use other objects example "Voucher"," Journal" and "Cash" objects. So accounts class is aggregation of these three objects

Inheritance

Hierarchy is used to define more specialized classes based on a preexisting generalized class. Example we have VEHICLE class and we can inherit this class make more specialized class like CAR, which will add new attributes and use some existing qualities of the parent class. Its shows more of a parent-child relationship .This kind of hierarchy is called inheritance.

Polymorphism

When inheritance is used to extend a generalized class to a more specialized class, it includes behavior of the top class (Generalized class).The inheriting class often implement a behavior that can be somewhat different than the generalized class, but the name of the behavior can be same. It is important that a given instance of an object use the correct behavior, and the property of polymorphism allows this to happen automatically.

(B)How do you implement inheritance in Java?

Short answer from interview point of view inheritance is implemented by using "EXTEND" keyword. As discussed in the previous basic definitions of OOP's inheritance is used to define more specialized class below the hierarchy. To understand this better lets do a small project to get the actual feel of the same.

Note: - You can get the source code for the same implementation in the "Inheritance" project which is available in "SourceCode" folder.

Below is the project structure of the "Inheritance" project. It has the three classes:-

√ "ClsAdd.java"

This class has a basic subroutine which adds two numbers.

√ “ClsSpecialAdd.java”

This class inherits from “ClsAdd.java” and overrides the subroutine which does the basic addition. It changes the basic addition functionality by adding an extra adjustment value of “20”.

√ “ClsRun.java”

This is the class which has the “public static void main (String [] args)” method. It’s the main class which will run the whole show by creating the objects of “ClsAdd” and “ClsSpecialAdd” and executing the “Add” method.

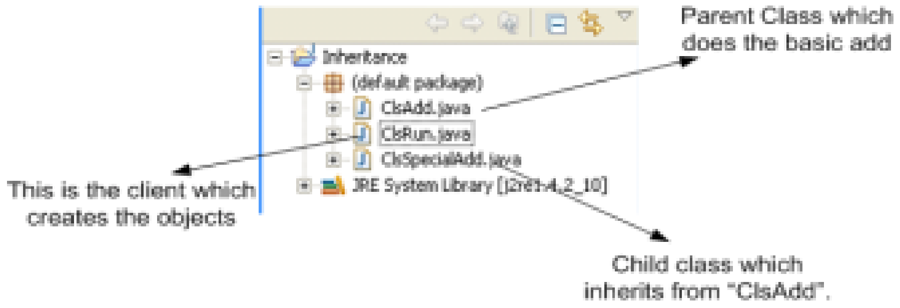
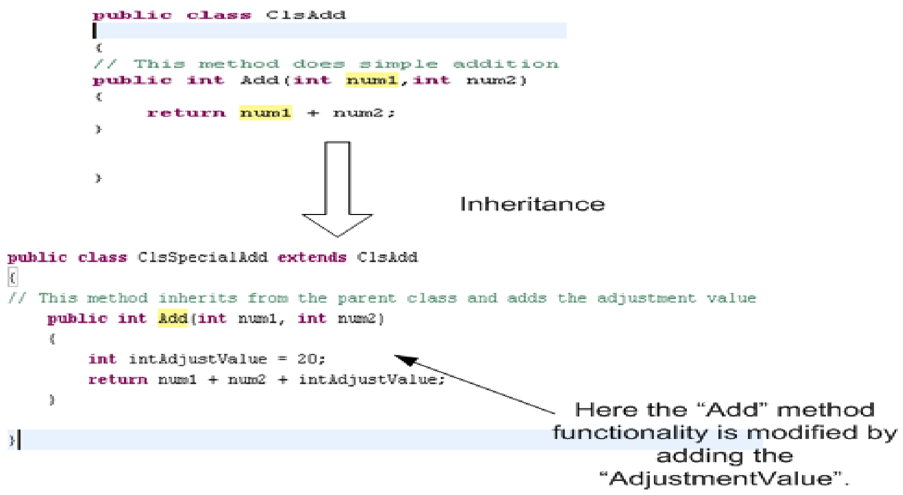

Inheritance
(default package)
ClsAdd.java
ClsRun.java
ClsSpecialAdd.java
JRE System Library [j2re1.4.2_10]
Parent Class which does the basic add
This is the client which creates the objects
Child class which inherits from "ClsAdd".

Figure 1.3 : - Project File structure of inheritance structure



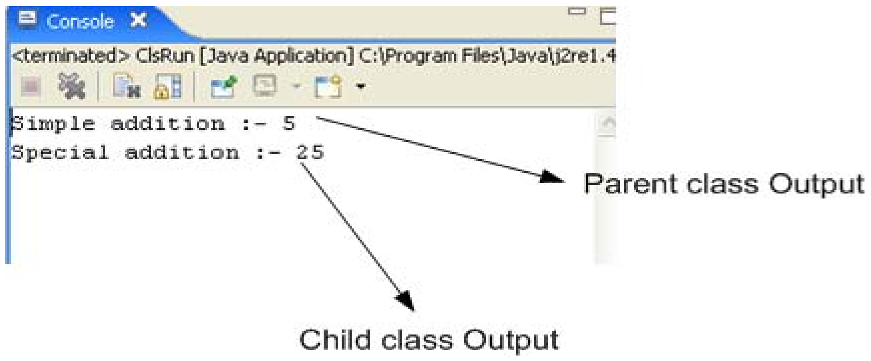
```
public class ClsAdd

{
// This method does simple addition
public int Add(int num1,int num2)
{
     return num1 + num2;
}


}
```

```
public class ClsSpecialAdd extends ClsAdd
{
// This method inherits from the parent class and adds the adjustment value
    public int Add(int num1, int num2)
    {
        int intAdjustValue = 20;
        return num1 + num2 + intAdjustValue;
    }

}
```

Here the “Add” method functionality is modified by adding the “AdjustmentValue”.

Figure 1.4 : - Inheritance in Action

When you run the project you can see two outputs one from the parent class and second from the child class with the adjustment value added up. So the child class has made the parent add class more specialized.



```
<terminated> ClsRun [Java Application] C:\Program Files\Java\j2re1.4
Simple addition :- 5
Special addition :- 25
```

Figure 1.5 : - Output from Inheritance Project

(B)How can we implement polymorphism in Java?

Twist: - How can we implement overloading in Java?

Polymorphism is the capability of an action or method to do different things based on the object that it is acting upon. There are two types of polymorphism:-

√ Method Polymorphism through overloading.

√ Object polymorphism by inheritance / interfaces.

Method Polymorphism through overloading

Let's try to understand this concept by a small sample. In CD you can find the "Polymorphism" folder which has two files "clsPolymorphism.java" and "clsTestRun.java".

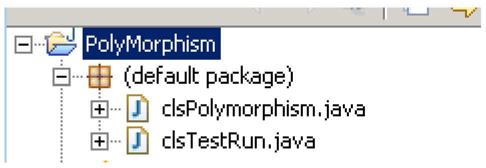
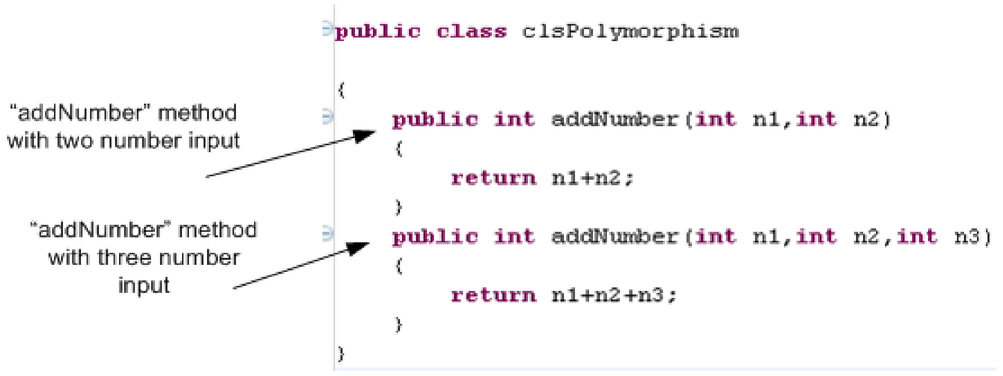


Figure 1.6 : - Polymorphism Explorer

“ClsTestRun.java” is the driver which has the main method, while “ClsPolymorphism” class has the actual stuff which demonstrates polymorphism. Below is the snippet of what “Clspolymorphism.java” looks like.

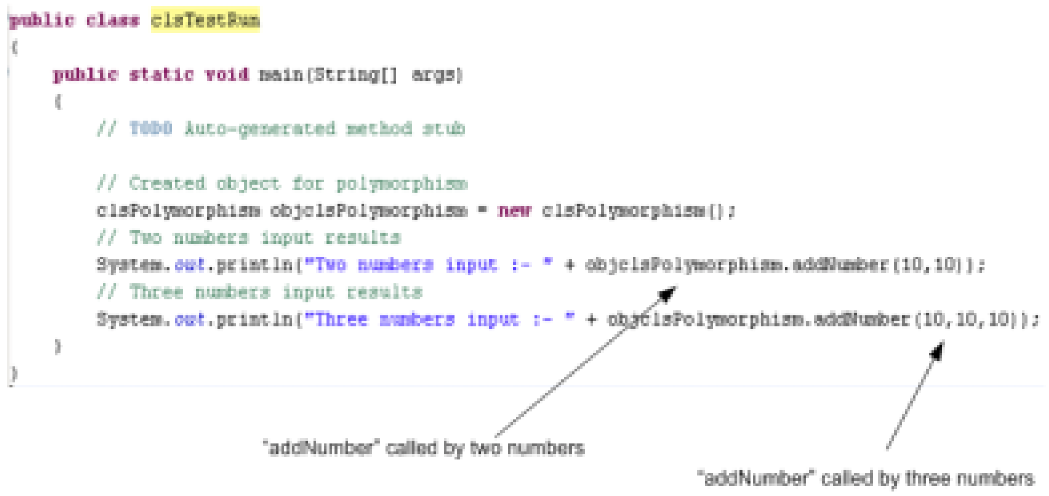


```
public class clsPolymorphism

{
    public int addNumber(int n1,int n2)
    {
        return n1+n2;
    }
    public int addNumber(int n1,int n2,int n3)
    {
        return n1+n2+n3;
    }
}
```

“addNumber” method with two number input
“addNumber” method with three number input

Figure 1.7 : - Snippet of “Clspolymorphism.java”

It has “addNumber” function which is overloaded with two types of signatures. One takes two number (n1 and n2) and other takes three (n1, n2 and n3).

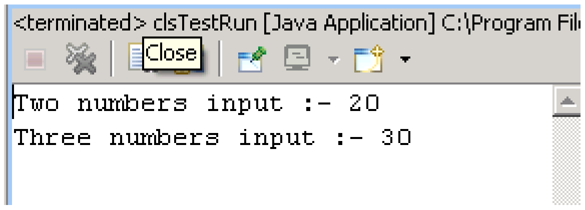


```
public class clsTestRun
{
    public static void main(String[] args)
    {
        // TODO Auto-generated method stub

        // Created object for polymorphism
        clsPolymorphism objclsPolymorphism = new clsPolymorphism();
        // Two numbers input results
        System.out.println("Two numbers input :- " + objclsPolymorphism.addNumber(10,10));
        // Three numbers input results
        System.out.println("Three numbers input :- " + objclsPolymorphism.addNumber(10,10,10));
    }
}
```

"addNumber" called by two numbers
"addNumber" called by three numbers

Figure 1.8 : - “addNumber” method called by different signatures

Below is the output when we run the program.



```
Two numbers input :- 20
Three numbers input :- 30
```

Figure 1.9 : - Output of polymorphism

You can see "addNumber" method performed differently in different inputs. "Poly" means multi and "morph" means change. So "addNumber" method is having multiple outputs depending on different inputs.

Object polymorphism by inheritance / interfaces

The above polymorphism sample was on the method level, but real worth and implementation of polymorphism concept is by objects. Object polymorphism is possible by inheritance or interfaces.

So let's make a quick walk through of an "Objectpolymorphism" sample. The below sample which I will be walking through is also provided in CD in "ObjectPolymorphism" folder.

There are four java files in "Objectpolymorphism" sample project:-

√ ClsAdd.java

√ ClsSpecialAdd.java

√ ClsSpecialAddandDivide.java

√ ClsTestRun.java

Below is how the "objectPlymorphism" project looks like with four files.

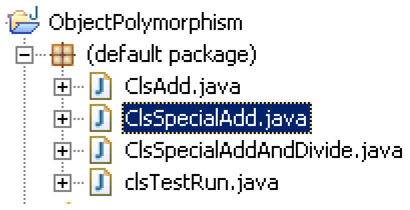


Figure 1.10 : - Object polymorphism explorer

"clsTestRun.java" is class which will have the "main" method which will be executed on the first instance. Below is the relationship between the other three classes. "ClsAdd" is the main parent class and "clsSpecialAdd" and "clsSpecialAddandDivide" derives from "ClsAdd" class.

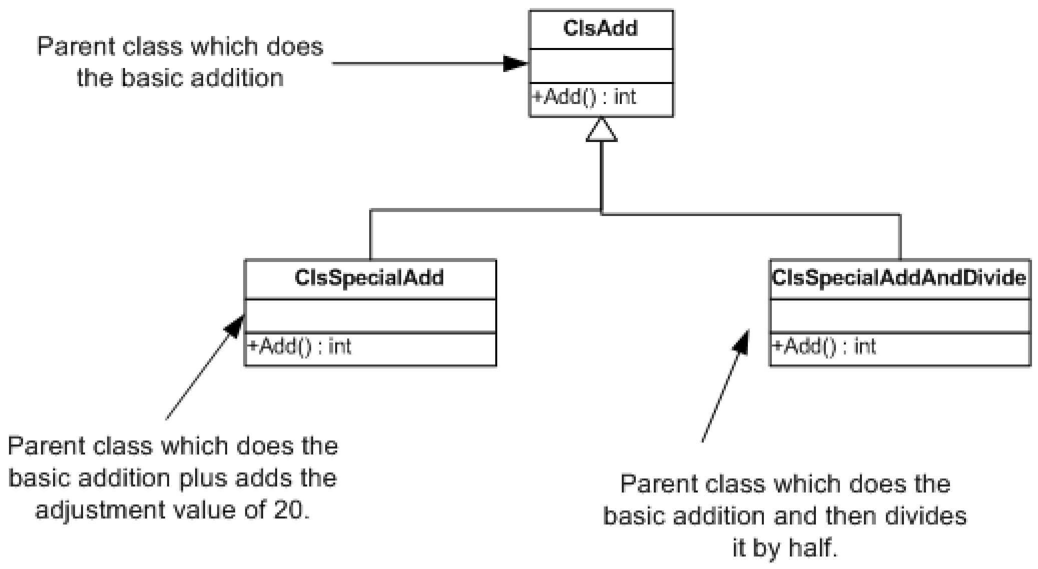

Parent class which does the basic addition
ClsAdd
+Add() : int
ClsSpecialAdd
+Add() : int
ClsSpecialAddAndDivide
+Add() : int
Parent class which does the basic addition plus adds the adjustment value of 20.
Parent class which does the basic addition and then divides it by half.

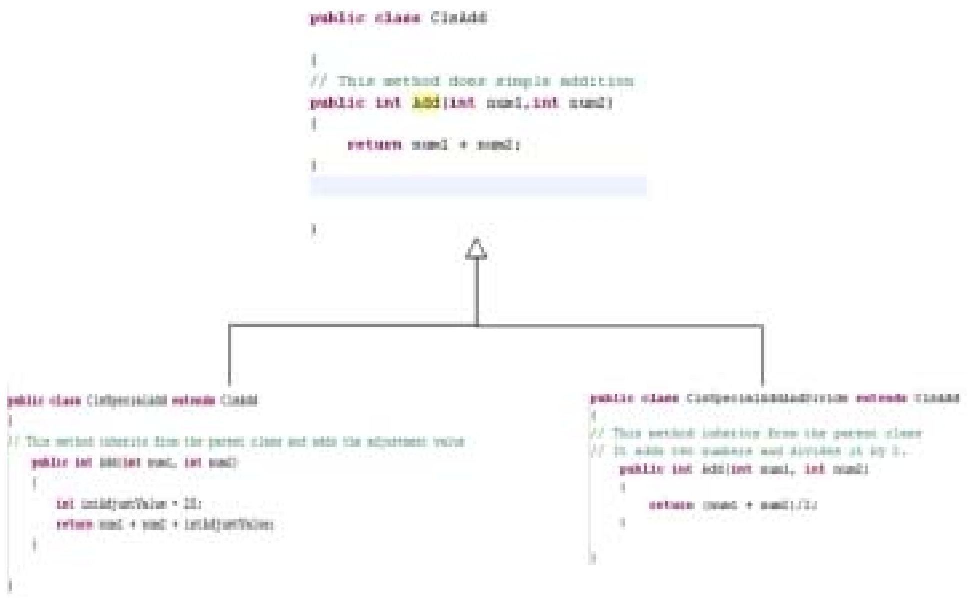
Figure 1.11 : - Class diagram for object polymorphism project.

Below is the source code explanation.

√ "ClsAdd" only adds the two numbers and returns the results.

√ "ClsSpecialAdd" adds the numbers and also adds adjustment value of 20.

√ “ClsSpecialAddandDivide” adds the numbers and divides it by 2.



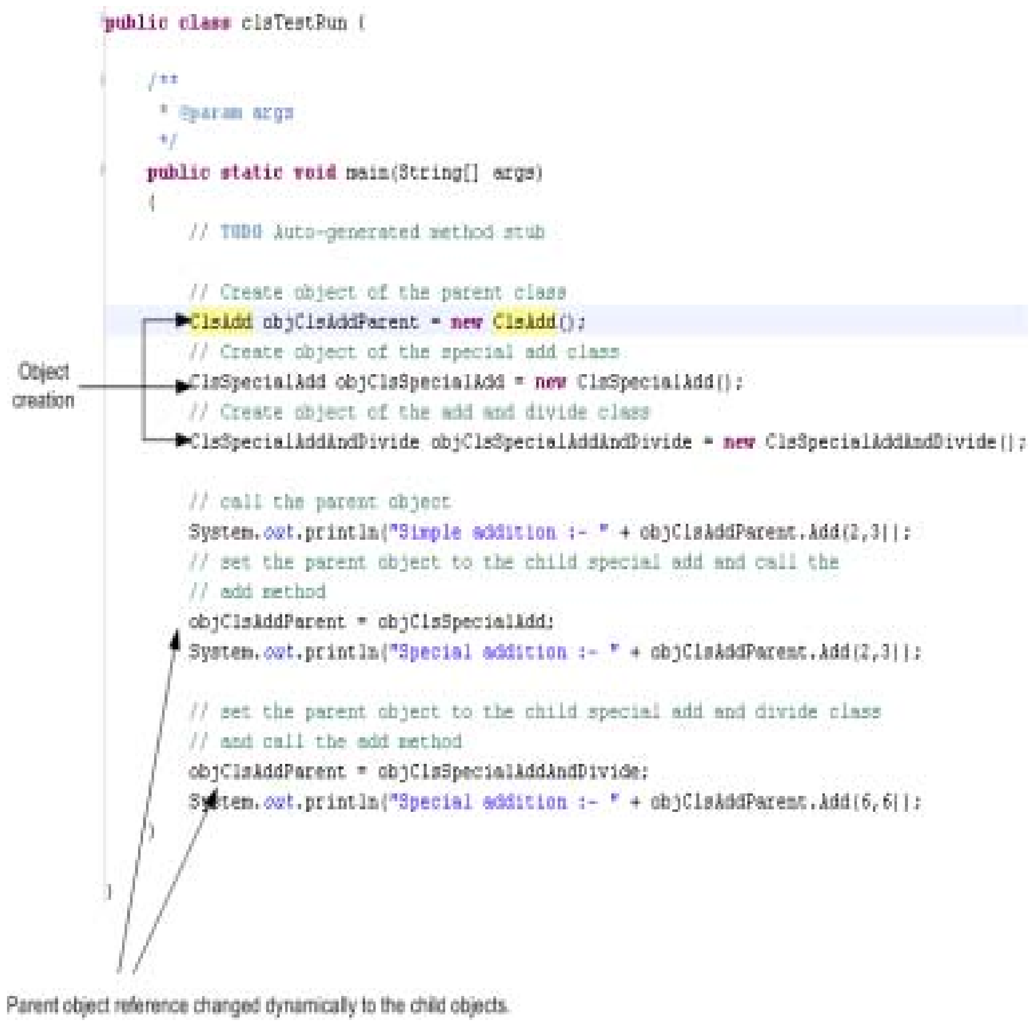
```
public class ClsAdd

{
// This method does simple addition
public int Add(int num1,int num2)
{
    return num1 + num2;
}

}
```

```
public class ClsSpecialAdd extends ClsAdd
{
// This method inherits from the parent class and adds the adjustment value
   public int Add(int num1, int num2)
   {
      int intAdjustValue = 20;
      return num1 + num2 + intAdjustValue;
   }

}
```

```
public class ClsSpecialAddandDivide extends ClsAdd
{
// This method inherits from the parent class
// It adds two numbers and divides it by 2.
   public int Add(int num1, int num2)
   {
      return (num1 + num2)/2;
   }

}
```

Figure 1.12 : - “Objectpolymorphism” project source code.

Finally we will walk through “clsTestRun” class which will demonstrate polymorphism using the above three classes. Below is the code snippet. “ClsTestRun” first creates object of all the three classes.

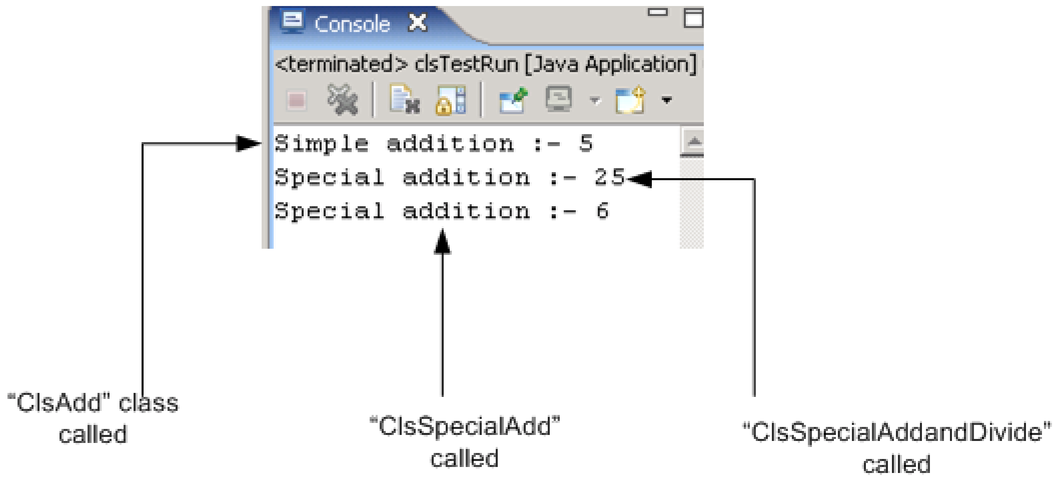

Object
creation
Parent object reference changed dynamically to the child objects.

```
public class clsTestRun {

    /**
     * @param args
     */
    public static void main(String[] args)
    {
        // TODO Auto-generated method stub

        // Create object of the parent class
        ClsAdd objClsAddParent = new ClsAdd();
        // Create object of the special add class
        ClsSpecialAdd objClsSpecialAdd = new ClsSpecialAdd();
        // Create object of the add and divide class
        ClsSpecialAddAndDivide objClsSpecialAddAndDivide = new ClsSpecialAddAndDivide();

        // call the parent object
        System.out.println("Simple addition :- " + objClsAddParent.Add(2,3));
        // set the parent object to the child special add and call the
        // add method
        objClsAddParent = objClsSpecialAdd;
        System.out.println("Special addition :- " + objClsAddParent.Add(2,3));

        // set the parent object to the child special add and divide class
        // and call the add method
        objClsAddParent = objClsSpecialAddAndDivide;
        System.out.println("Special addition :- " + objClsAddParent.Add(6,6));
    }
}
```

Figure 1.13 : - "clsTestRun" for "objectpolymorphism" project

You can see in the code we have set both the child object references to parent object and called the add method respectively. Depending on which child is referenced by the parent object the child object method is called. For instance when we referred "ClsSpeicalAdd" object it added the two numbers adding the adjustment value. So depending on what object the parent object is referring the actions is taking place which is nothing but polymorphism.

Below is the output you can see both the child classes are called by the parent object.


Console
<terminated> clsTestRun [Java Application]
"ClsAdd" class called
"ClsSpecialAdd" called
"ClsSpecialAddandDivide" called

```
Simple addition :- 5
Special addition :- 25
Special addition :- 6
```

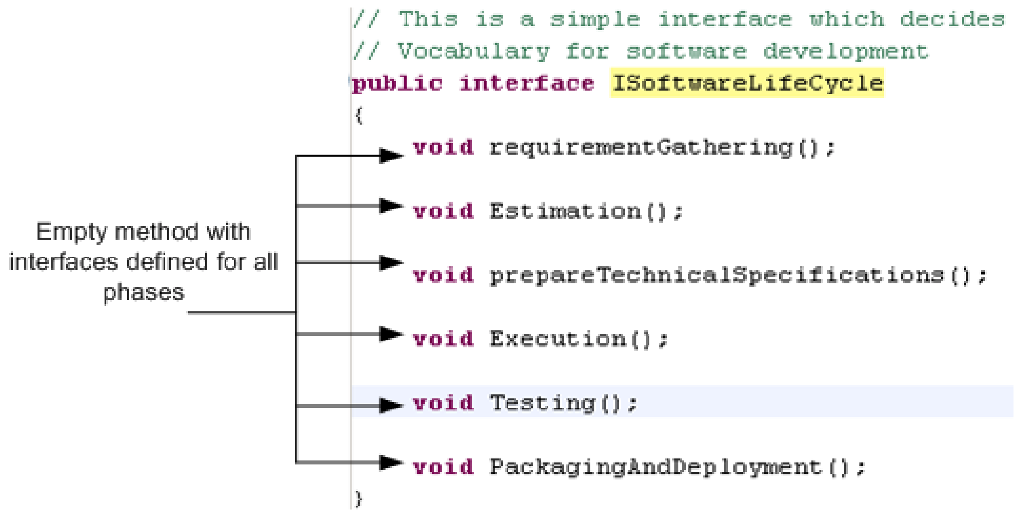
Figure 1.14 : - Output results of "Objectpolymorphism"

(B)What's an interface and how will you go about implementing an interface?

An interface is a named collection of method definitions, without implementations. Interface defines a protocol of behavior for a class. So if "Class1" is adhering to an interface and "Class2" wants to communicate with "Class1" then it should follow the same interface which "Class1" is following". In short it defines a contract to the external consumer of the class that if you want use me then you need to follow these interfaces.

So let's run through a sample in order to understand the interface concept.

In the below sample we want to develop a class which will make us run through software development phases in a proper order. So first let's decide an interface which will have all the necessary vocabulary so that we do not miss anything.

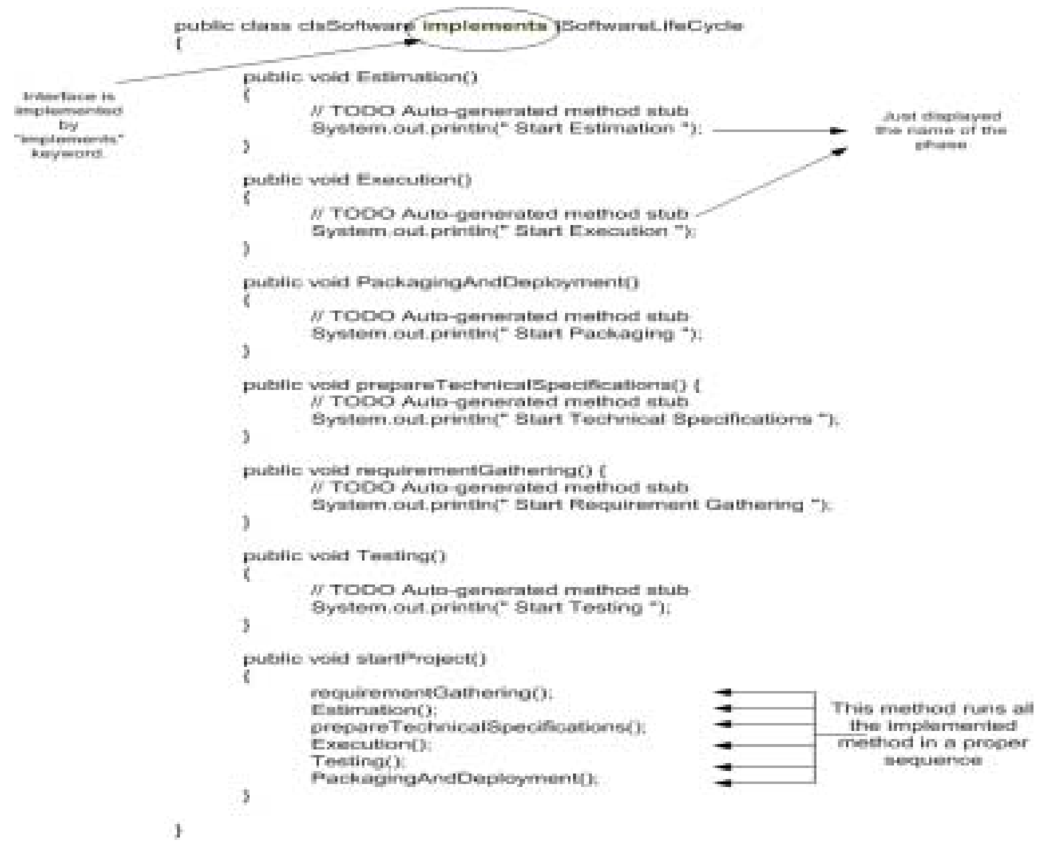


```
// This is a simple interface which decides
// Vocabulary for software development
public interface ISoftwareLifeCycle
{
    void requirementGathering();

    void Estimation();

    void prepareTechnicalSpecifications();

    void Execution();

    void Testing();

    void PackagingAndDeployment();
}
```

Figure 1.15 : - "ISoftwareLifeCycle" interface

So the above is the interface with six phases of software life cycle. Preparing this interface is a one time exercise. Now any time we want to do a project regarding software development life cycle we just have to implement this interface. So we can say interfaces define vocabulary and contract.

Now let's make a class which will implement this interface and provide us with necessary functionalities. This class does not do anything special but will just display the name of the phase.

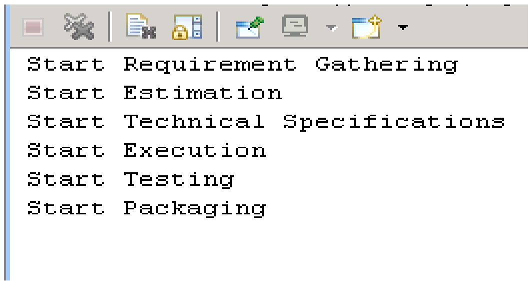

Interface is implemented by "implements" keyword.
Just displayed the name of the phase
This method runs all the implemented method in a proper sequence

```
public class clsSoftware implements ISoftwareLifeCycle
{
    public void Estimation()
    {
            // TODO Auto-generated method stub
            System.out.println(" Start Estimation ");
    }
    public void Execution()
    {
            // TODO Auto-generated method stub
            System.out.println(" Start Execution ");
    }
    public void PackagingAndDeployment()
    {
            // TODO Auto-generated method stub
            System.out.println(" Start Packaging ");
    }
    public void prepareTechnicalSpecifications() {
            // TODO Auto-generated method stub
            System.out.println(" Start Technical Specifications ");
    }
    public void requirementGathering() {
            // TODO Auto-generated method stub
            System.out.println(" Start Requirement Gathering ");
    }
    public void Testing()
    {
            // TODO Auto-generated method stub
            System.out.println(" Start Testing ");
    }
    public void startProject()
    {
            requirementGathering();
            Estimation();
            prepareTechnicalSpecifications();
            Execution();
            Testing();
            PackagingAndDeployment();
    }
}
```

Figure 1.16 : - Class "ClsSoftware" implementation

Interfaces are implemented using "implements" keyword. You can see the best thing about using interfaces is that the shell is predefined (in this case it's the "ISoftwareLifeCycle"). Programmer who is coding "clsSoftware" class does not have to think about what different phases exists for software but rather just implement the "ISoftwareLifeCycle" interface. In big projects where you have a huge programmer group coding it's much possible that every one will not follow same naming conventions and method names. So the best way is to define an interface and let the programmers implement the interface and write the implementation. Due to this approach because every programmer is implementing the same interface it brings in consistency of method and naming conventions through out the project.

In “ClsSoftware” class we also have a “StartProject” method which calls all the phases in a proper order. If you run the project using eclipse you will get the following output.



```
Start Requirement Gathering
Start Estimation
Start Technical Specifications
Start Execution
Start Testing
Start Packaging
```

Figure 1.17 :- Output from the interface project.

If you want to apply interface to class you can automate it by clicking on “Extract Interface” menu.

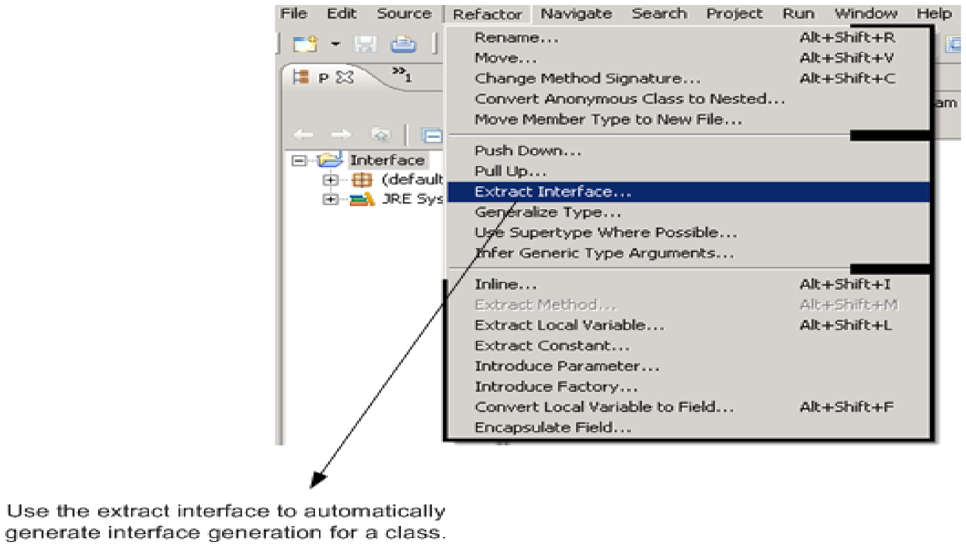

File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
Interface
(default
JRE Sys
Rename... Alt+Shift+R
Move... Alt+Shift+V
Change Method Signature... Alt+Shift+C
Convert Anonymous Class to Nested...
Move Member Type to New File...
Push Down...
Pull Up...
Extract Interface...
Generalize Type...
Use Supertype Where Possible...
Infer Generic Type Arguments...
Inline... Alt+Shift+I
Extract Method... Alt+Shift+M
Extract Local Variable... Alt+Shift+L
Extract Constant...
Introduce Parameter...
Introduce Factory...
Convert Local Variable to Field... Alt+Shift+F
Encapsulate Field...
Use the extract interface to automatically generate interface generation for a class.

Figure 1.18 :- Click on “Extract Interface” for automating process

(B)What is an Abstract class?

Abstract class defines an abstract concept which can not be instantiated and comparing to interface it can have some implementation while interfaces can not. Below are some points for abstract class:-

√ We can not create object of abstract class it can only be inherited in a below class.

√ Normally abstract classes have base implementation and then child classes derive from the abstract class to make the class concrete.

Ok let's try to get the abstract class fundamentals just to make sure that you do not stumble in interview. Abstract classes normally represent concept. Just to make it precise it represents half implemented concepts. What does that mean fundamentally?. You can have concept like food but you can not have an instance of the same. You can have instance of cheese, milk etc. So food represents an abstract concept while cheese, milk etc represent the actual instance. In your project you can have a general concept called as "Transaction" but then further your project can make the "Transaction" class more concrete by saying "Bank Transaction" or "Ledger Transaction" class. So you would like to model the "Transaction" class but not create objects of the same. It makes more sense to create "Bank Transaction" class object or "Ledger Transaction" class objects.

In order to understand abstract class fundamentals lets do a sample project. You can get the sample project in CD in "\Sourcecode\Abstract" folder. Below is the class diagram of the project. "clsAbstractPerson" class represents a person. It has three methods "Numberoflegs", "NumberofEyes" and "Sex". Now conceptually all humans have two legs and two eyes, but sex is something which can vary from person to person. So what we do is we write implementation for "numberofEyes" method and "numberOfLegs" method and leave the "Sex" method implementation to subclasses.

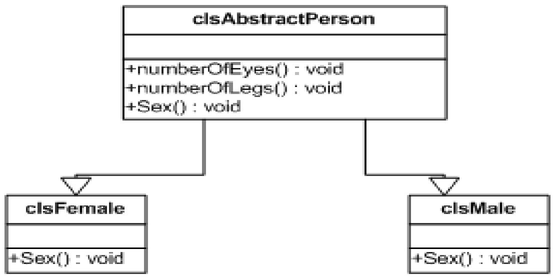
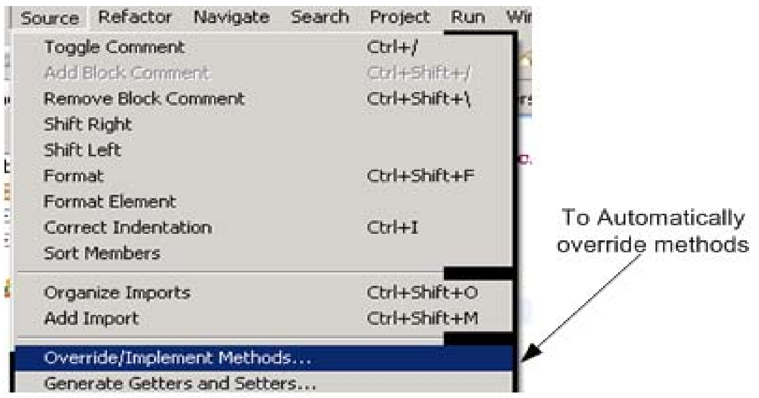


Figure 1.19 : - Class diagram for abstract project

We have made two subclasses “clsFemale” and “clsMale” which will provide us concrete implementation for “Sex” method. So let’s make a code walk through for all of the classes defined in the class diagram.

> *Note: - If you want the method names to included automatically in the subclasses you click on “Override/Implement methods” menu which is present in the main “Source” menu as shown in figure below. You will then be prompted with a dialog box as to which method from the parent should be overridden. For the current “Abstract” project we only need to check the “Sex” method as that’s the only method we will need to override.*


Source
Refactor
Navigate
Search
Project
Run
Toggle Comment Ctrl+/
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Shift Right
Shift Left
Format Ctrl+Shift+F
Format Element
Correct Indentation Ctrl+I
Sort Members
Organize Imports Ctrl+Shift+O
Add Import Ctrl+Shift+M
Override/Implement Methods...
Generate Getters and Setters...
To Automatically override methods

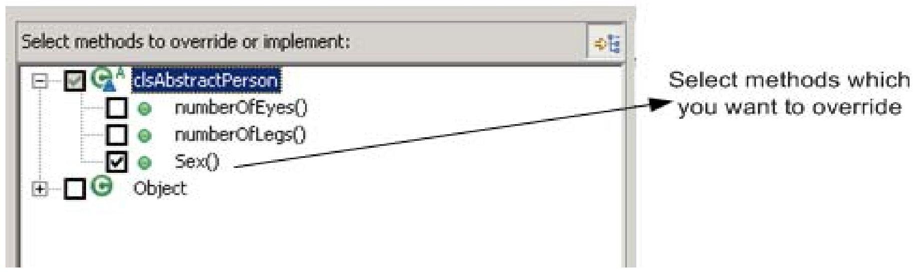
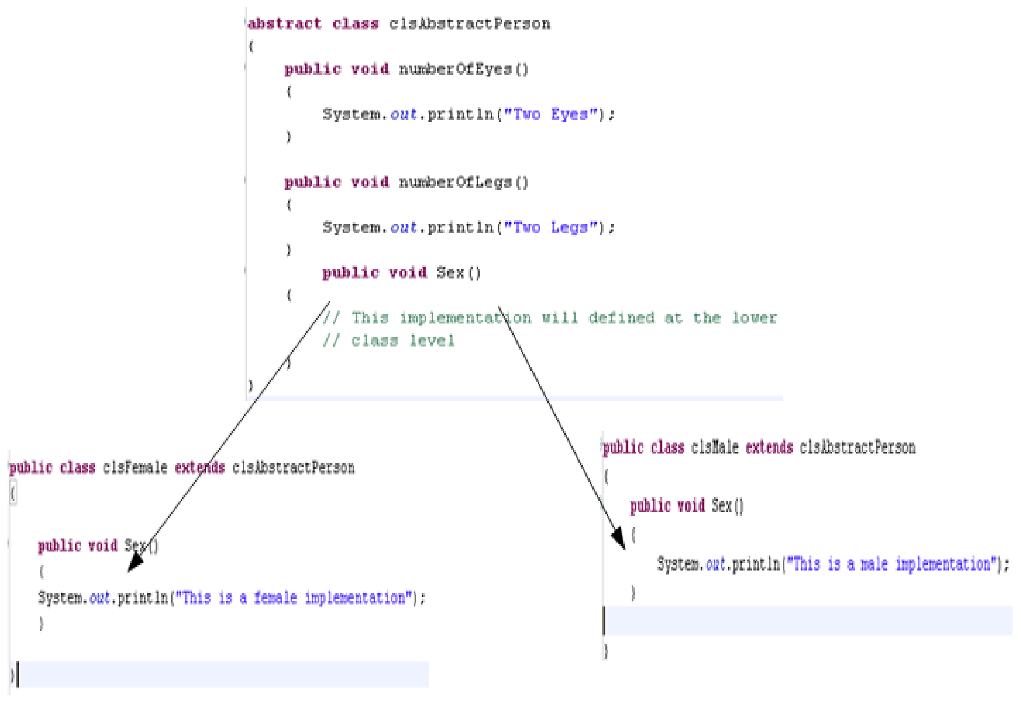

Select methods to override or implement:
clsAbstractPerson
numberOfEyes()
numberOfLegs()
Sex()
Object
Select methods which you want to override

Figure 1.20 : - Automating implementation using Eclipse

Below is the code walk through for all the three class parent class i.e. “clsAbstractPerson” and the two child classes i.e. “clsMale” and “clsFemale”.



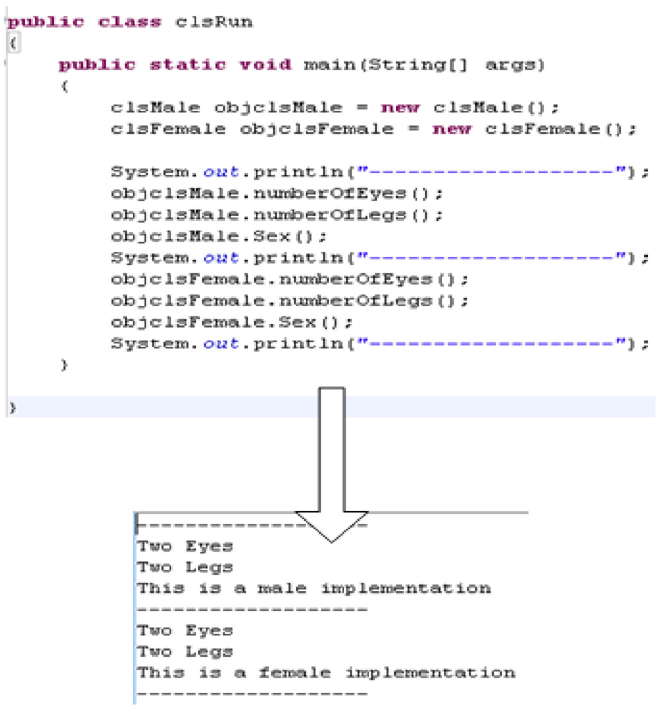
```
abstract class clsAbstractPerson
{
    public void numberOfEyes()
    {
        System.out.println("Two Eyes");
    }

    public void numberOfLegs()
    {
        System.out.println("Two Legs");
    }
        public void Sex()
    {
        // This implementation will defined at the lower
        // class level
    }
}
```

```
public class clsFemale extends clsAbstractPerson
{

    public void Sex()
    {
    System.out.println("This is a female implementation");
    }

}
```

```
public class clsMale extends clsAbstractPerson
{
    public void Sex()
    {
        System.out.println("This is a male implementation");
    }

}
```

Figure 1.21 : - Code walk through for abstract class

Nothing special has been done in the “Sex” method implementation other than just displaying the whether its “Female” or “Male” implementation according to the class. You can now visualize the concept of using “Abstract” class. In this situation “Person” is a concept but it can not have instance as there is no point in creating an object of “Person”. It’s a half made class which does not have “Sex” implementation. Conceptually “Person” can be either female or male. But later when subclasses inherit the “ClsAbstractPerson” class they provide a true implementation by providing functionality for the “Sex” method. It sounds sense for creating objects for “clsFemale” or “clsMale” class.



```
public class clsRun
{
    public static void main(String[] args)
    {
        clsMale objclsMale = new clsMale();
        clsFemale objclsFemale = new clsFemale();

        System.out.println("--------------------");
        objclsMale.numberOfEyes();
        objclsMale.numberOfLegs();
        objclsMale.Sex();
        System.out.println("--------------------");
        objclsFemale.numberOfEyes();
        objclsFemale.numberOfLegs();
        objclsFemale.Sex();
        System.out.println("--------------------");
    }

}
```

```
--------------------
Two Eyes
Two Legs
This is a male implementation
--------------------
Two Eyes
Two Legs
This is a female implementation
--------------------
```

Figure 1.22 :- Output of “Abstract” project

“ClsRun” class is the driver which creates objects for both classes “clsMale” and “clsFemale” and calls all the three methods. You can see the output “Eyes” and “Legs” implementation remain same for both the classes but “Sex” implementation varies. So basically abstract class “clsAbstractPerson” provides conceptual implementation but later the concrete classes give more detail implementation and complete meaning to the class.

(B) What are Abstract methods?

Abstract class can contain abstract methods. Abstract methods do not have implementation. Abstract methods should be implemented in the subclasses which inherit them. So if an abstract class has an abstract method class inheriting the abstract class should implement the method or else java compiler will through an error. In this way, an abstract class can define a complete programming interface thereby providing its subclasses with the method declarations for all of the methods necessary to implement that programming interface.

Abstract methods are defined using “abstract” keyword. Below is a sample code snippet.

```
abstract class myAbstractGraphics
{
abstract void draw();
}
```

Any class inheriting from “myAbstractGrpahics” class should implement the “draw” method or else the java compiler will throw an error.

Note: - One way the interviewer would like to confuse you is if we do not implement a abstract method will the program compile.

(B)What’s the difference between “Abstract” classes and “Interfaces”?

Difference between Abstract class and Interface is as follows:-

- √ Abstract class can only be inherited while interfaces can not be it has to be implemented.
- √ Interface cannot implement any methods, whereas an abstract class can have implementation.
- √ Class can implement many interfaces but can have only one super class.
- √ Interface is not part of the class hierarchy while Abstract class comes in through inheritance.
- √ Unrelated classes can implement the same interface.

(I)What’s difference between Static and Non-Static fields of a class?

Non-Static values are also called as instance variables. Each object of the class has its own copy of Non-Static instance variables. So when a new object is created of the same class it will have completely its own copy of instance variables.

While Static values have only one copy of instance variables and will be shared among all the objects of the class.

Here’s a small practical stuff we will do to understand the concept. Let’s create a small sample project to demonstrate the static and non-static concepts. In CD in the source code folder there is a sample provided for the same in “StaticNonStatic” folder. Below is the file structure. Basically the project has two main classes “clsStaticNonStatic.java” and “ClsTestRun.java”. We will be explaining the above two classes in the coming sections.

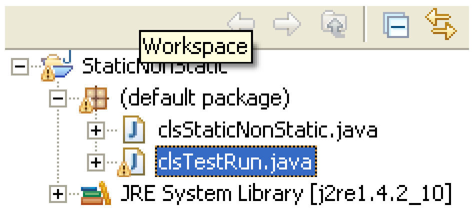
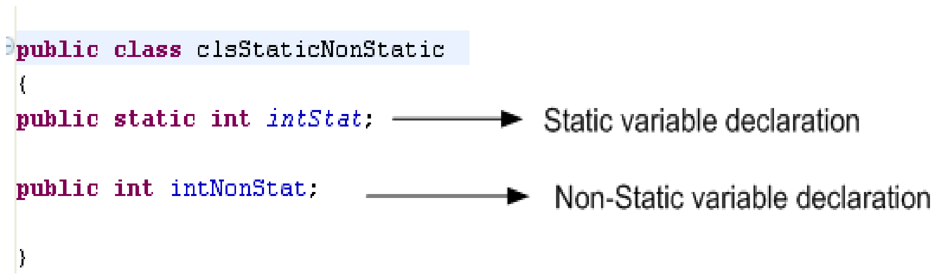


Figure 1.23 : - Static and Non-Static Project Files

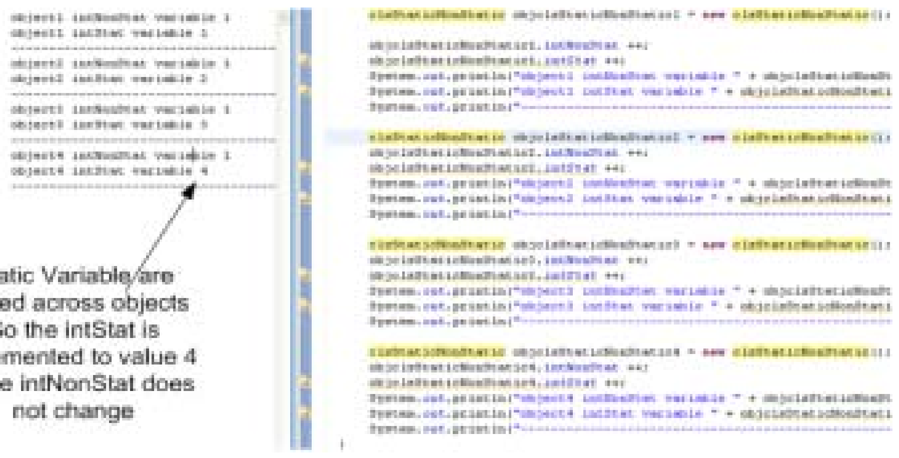
“clsStaticNonStatic.java” is the core class it has two variable declarations as shown in figure below.



```
public class clsStaticNonStatic
{
public static int intStat;        Static variable declaration

public int intNonStat;            Non-Static variable declaration

}
```

Figure 1.24 : - clsStaticNonStatic java class.

“ClsTestRun.java” is the class who will run create objects of “clsStaticNonStatic.java” and run it. Below is the figure which has “clsTestRun.java” in action. It does nothing special but creates four objects and increments the static and non-static properties. In the below figure at the left hand side you can see the static variables values are preserved as they are. So the “intStat” value goes up to “4” and “intNonStat” has value “1”.


```
object1 intNonStat variable 1
object1 intStat variable 1
--------------------------------------
object2 intNonStat variable 1
object2 intStat variable 2
--------------------------------------
object3 intNonStat variable 1
object3 intStat variable 3
--------------------------------------
object4 intNonStat variable 1
object4 intStat variable 4
--------------------------------------
```

```
clsStaticNonStatic objclsStaticNonStatic1 = new clsStaticNonStatic();

objclsStaticNonStatic1.intNonStat ++;
objclsStaticNonStatic1.intStat ++;
System.out.println("object1 intNonStat variable " + objclsStaticNonSt
System.out.println("object1 intStat variable " + objclsStaticNonStati
System.out.println("----------------------------------------------------

clsStaticNonStatic objclsStaticNonStatic2 = new clsStaticNonStatic();
objclsStaticNonStatic2.intNonStat ++;
objclsStaticNonStatic2.intStat ++;
System.out.println("object2 intNonStat variable " + objclsStaticNonSt
System.out.println("object2 intStat variable " + objclsStaticNonStati
System.out.println("----------------------------------------------------

clsStaticNonStatic objclsStaticNonStatic3 = new clsStaticNonStatic();
objclsStaticNonStatic3.intNonStat ++;
objclsStaticNonStatic3.intStat ++;
System.out.println("object3 intNonStat variable " + objclsStaticNonSt
System.out.println("object3 intStat variable " + objclsStaticNonStati
System.out.println("----------------------------------------------------

clsStaticNonStatic objclsStaticNonStatic4 = new clsStaticNonStatic();
objclsStaticNonStatic4.intNonStat ++;
objclsStaticNonStatic4.intStat ++;
System.out.println("object4 intNonStat variable " + objclsStaticNonSt
System.out.println("object4 intStat variable " + objclsStaticNonStati
System.out.println("----------------------------------------------------
}
```

Static Variable are stored across objects So the intStat is incremented to value 4 While intNonStat does not change

Figure 1.25 : - clsTestrun.java class

Note: - You can find the sample project in "StaticNonStatic" folder.

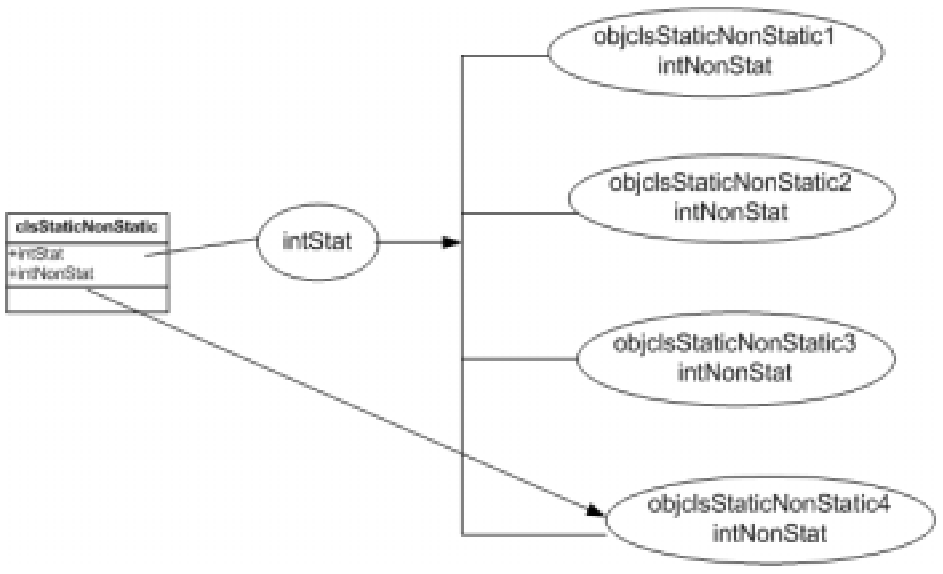


Figure 1.26 : - Static and Non-Static Values

The above figure shows how “Static” and “NonStatic” properties are shared across object. You can see “intStat” property is shared across objects and “intNonStat” have there own local copy.

(I) What are inner classes and what’s the practical implementation of inner classes?

Inner classes are nested inside other class. They have access to outer class fields and methods even if the fields of outer class are defined as private.

```
public class Person
{
  class clsName
  {
  // inner class defines the required structure
    String first;
    String last;
  }
  // array of  name objects
  clsName personArray[] = {new clsName(), new clsName(), new clsName()};
}
```

Normally inner classes are used for data structures like one shown above or some kind of helper classes.

(B)What are packages?

Packages group related classes and interfaces together and thus avoiding any name conflicts. From OOP’s point of view packages are useful for grouping related classes together. Classes are group together in a package using “package” keyword.

Below is a sample snippet for a package which groups “Class1” and “Class2” in one package “Package1”.

```
package Package1;
public class Class1
{
        public void displayClassName()
        {
          System.out.println(" I am class1 from Package1");
        }
}
public class Class2
{
        public void displayClassName()
        {
          System.out.println(" I am class2 from Package1");
        }
}
```

When we want to use the above “package1” in project we need to use the import keyword.

```
import Package1.*;
public class clsRun
{
        public static void main(String[] args)
        {
          Package1.Class1 objp1Class1 = new Package1.Class1();
          objp1Class1.displayClassName();
        }}
```

Note: - You can add a package using eclipse using the wizard as shown below.

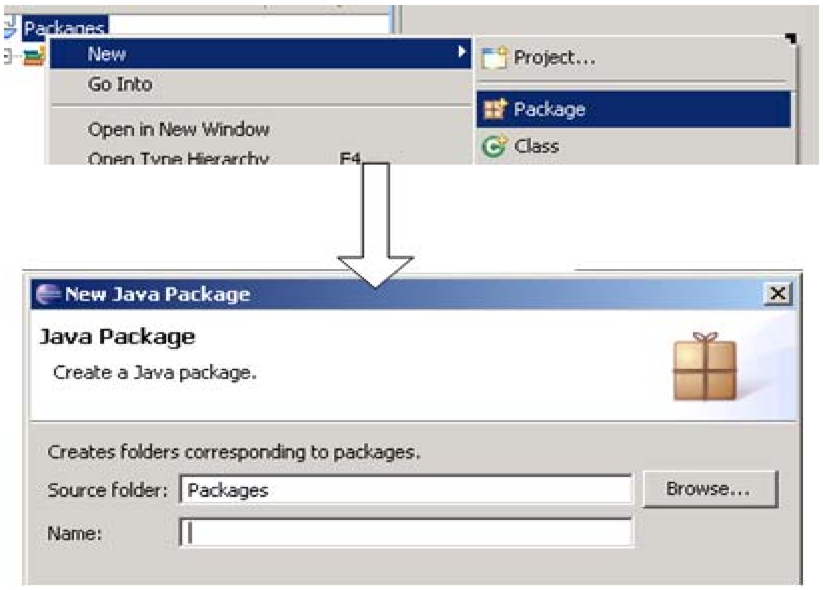


Figure 1.27 : - Adding package using eclipse wizard

In order to make concepts more clear there is a sample package project shipped in CD in "package" folder. Below is the solution of the package project. There two packages "Package1" which has "Class1" class and "Package2" which has again a class by name "Class1".

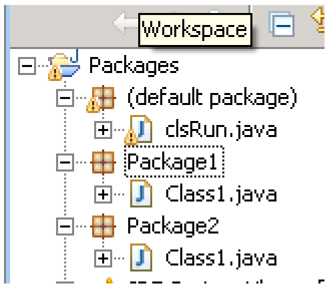
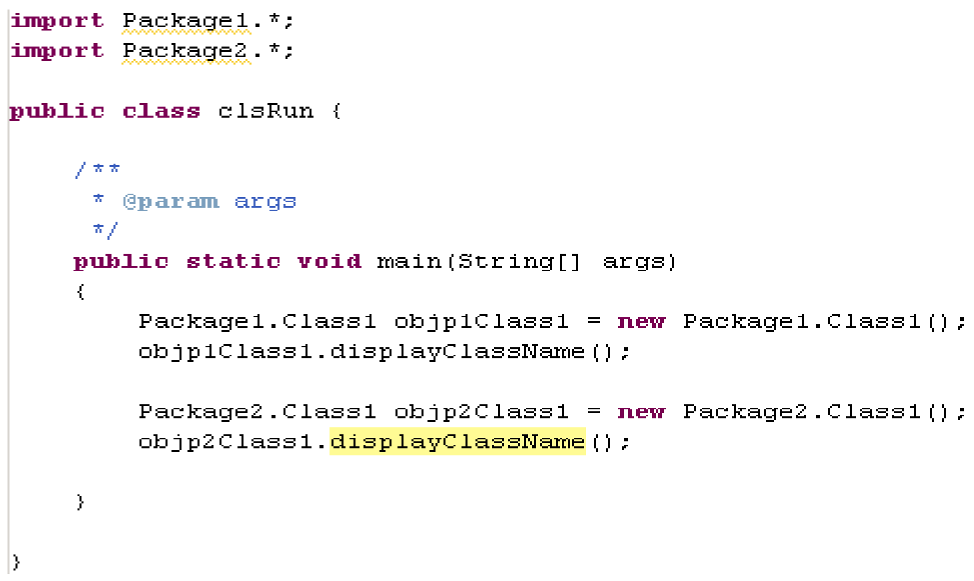


Figure 1.28 : - Solution for package project



```
import Package1.*;
import Package2.*;

public class clsRun {

    /**
     * @param args
     */
    public static void main(String[] args)
    {
        Package1.Class1 objp1Class1 = new Package1.Class1();
        objp1Class1.displayClassName();

        Package2.Class1 objp2Class1 = new Package2.Class1();
        objp2Class1.displayClassName();

    }

}
```

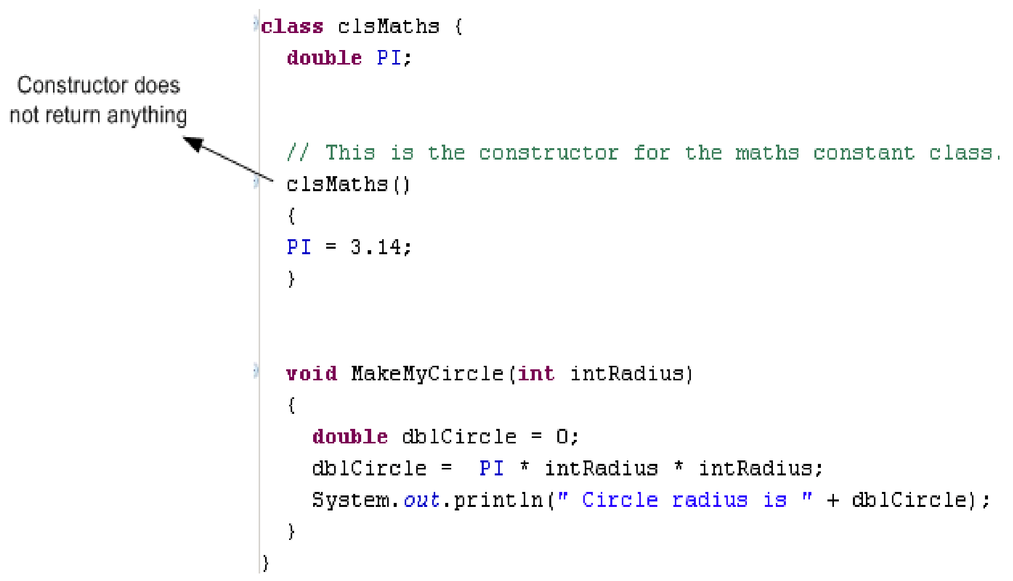
Figure 1.29 : - “clsRun” class for package project

In “clsRun” class we have created objects of “Class1” class which exist in “package1” as well as “package2”. There are two things which is important to be noted one is the “import” keyword at the top of the class and second is both objects are created using “PackageName.Classname” . For instance “objp1class1” object is created using “Package1.Class1”. You can see that we can have class with same name but they lie in different packages. For instance in the above example “Class1” name is give two files but they lie in different packages.

Normally in big projects where we have huge amount of classes its better to group them logically together so that can be manageable.

(B)What is a constructor in class?

Constructor has the same name as the class in which it resides and looks from syntax point of view it looks similiar to a method. Constructor is automatically called immediately after the object is created, before the new operator completes. Constructors have no return type, not even void. This is because the implicit return type of a class' constructor is the class type itself. It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately. Below is a snippet of constructor code:-



```
class clsMaths {
  double PI;

  // This is the constructor for the maths constant class.
  clsMaths()
  {
  PI = 3.14;
  }

  void MakeMyCircle(int intRadius)
  {
    double dblCircle = 0;
    dblCircle =  PI * intRadius * intRadius;
    System.out.println(" Circle radius is " + dblCircle);
  }
}
```

Figure 1.30 : - Constructor in action

Above snippet shows declaration for constructor in “clsMaths” class. We have initialized the PI value to “3.14”.

Note: - Same sample has been provided in “Constructor” folder in CD.

(I)Can constructors be parameterized?

Yes we can have parameterized constructor which can also be termed as constructor overloading. Below is a code snippet which shows two constructors for “clsMaths” class one with parameter and one with out.

```
class clsMaths

{

  double PI;

  // This is the constructor for the maths constant class.
clsMaths()
{PI = 3.14;}
clsMaths(int pi)
{
      PI = pi;
 } }
```

(I)What is the use if "instanceof" keyword?

"instanceof" keyword is used to check what is the type of object. For instance in the below code snippet we are trying to check is "object1" of the same type as "object2".

```
if(object1 instanceof object2)
```

(B)What are Native methods in Java?

There may be times when you want to call subroutines which are written in some other language other than Java like C++, VB6 etc.

(B)How do refer to a current instance of object?

You can refer the current instance of object using "this" keyword. For instance if you have class which has "color" property you can refer the current object instance inside any of the method using "this.color".

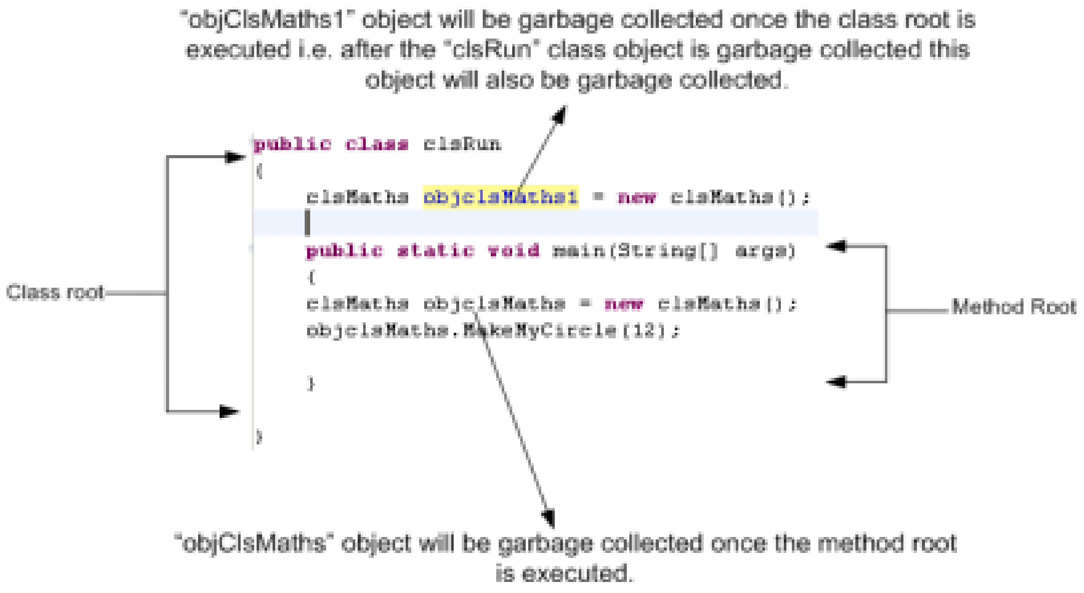
(I) Explain in depth Garbage collector?

Garbage collection is the process of automatically freeing objects that are no longer referenced by the program. This frees the programmer from having to keep track of when to free allocated memory, thereby preventing many potential bugs. Thus making programmers more productive as they can now put more effort in coding rather than worrying about memory management.

The only disadvantage of garbage collector is it adds overheads. Because the JVM (Java virtual machine) has to keep a constant track of the objects which are not referenced and then free these unreferenced objects on fly. This whole process has a slight impact on the application performance. But garbage collector has a good algorithm and it runs in its own thread thus having a least impact on the application performance but still it has some impact.

(I)How does the garbage collector determine that the object has to be marked for deletion?

GC has defined set of "roots" and "object reach from the roots". Ok let's try to understand this statement. "roots" can be a method, class etc. So if the "method" or "class" root has access to the objects then the object can not be garbage collected. Lets try to understand root and reach from the below code snippet.


"objClsMaths1" object will be garbage collected once the class root is executed i.e. after the "clsRun" class object is garbage collected this object will also be garbage collected.
Class root
Method Root
"objClsMaths" object will be garbage collected once the method root is executed.

```
public class clsRun
{
    clsMaths objclsMaths1 = new clsMaths();

    public static void main(String[] args)
    {
    clsMaths objclsMaths = new clsMaths();
    objclsMaths.MakeMyCircle(12);

    }

}
```

Figure 1.31 : - "root" and "reach" in garbage collection

In the above snippet we have two roots one is the class root i.e. "clsRun" class itself and second method root "main" method. We have two objects at both levels of roots "objClsMaths1" object is in class root while "objClsMaths" object is on method root. One the "main" method execution is completed object "objClsMaths" object will be marked for garbage collection. Once object of "clsRun" class is destroyed then "objClsMaths1" object will marked for garbage collection.

"Mark" and "Sweep" algorithm is done for removing unused objects from memory. "Mark" and "Sweep" refer to two phases before an unused object is garbage collected. In "Mark" phase the garbage collector runs through all objects and marks object as unused depending on the roots reach to the object. In the "Sweep" phase unmarked objects are freed, and the resulting memory is made available to the executing program.

(B)Can you explain "finalize()" method?

Sometimes the object needs to perform some action before the object is destroyed. For instance if an object is holding some non-Java resource such as a file handle which you might want to make sure are released before the object is destroyed. For such conditions Java provides a mechanism called finalization. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

You can add a finalizer to a class by simply defining the "finalize()" method. JVM calls that method whenever it is about to recycle an object of that class. Inside the "finalize()" method you can specify those actions that must be performed before an object is destroyed. GC runs periodically checking for objects that are no longer referenced. Right before an asset is freed, the Java run time calls the "finalize( )" method on the object.

Below is the snippet for finalize method

```
protected void finalize()
{
// Your non java resource deletion code goes here
}
```

Protected access prevents access to "finalize()" by code defined outside its class.

(I)How can we force the garbage collector to run?

Garbage collector can be run forcibly using "System.gc()" or "Runtime.gc()"

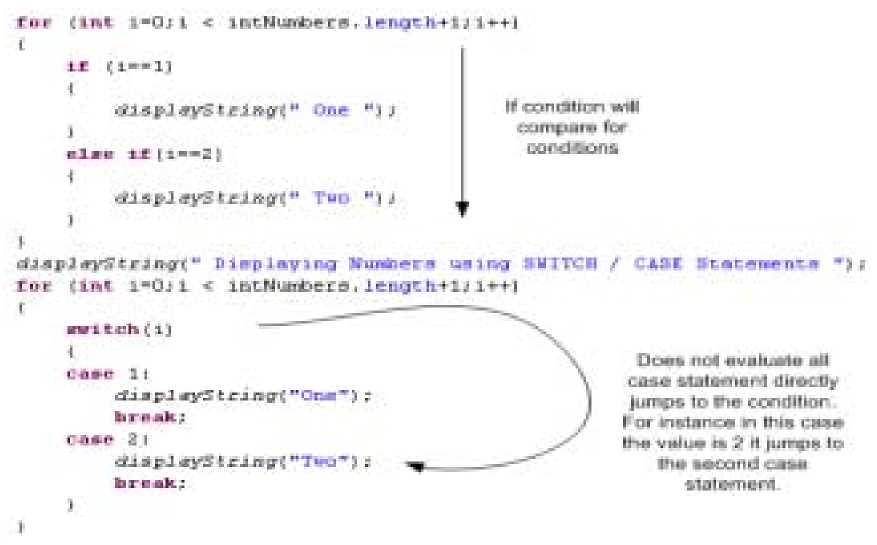
(B)What's the main difference between "Switch" and "If" comparison?

Switch differs from if statements in the following way:-

- √ Switch can only test for equality, whereas if can evaluate any type of Boolean expression. That is, the switch looks only for a match between the value of the expression and one of its case constants.
- √ Switch statements are more efficient that if statements.

When Java compiler compiles a switch statement it looks at the case constants and creates a jump table that it will use for selecting the path of execution depending on the value of the expression. So if you are comparing large group of values switch will be faster than sequence of if-elses. Compiler knows that the case constants are all the same type and simply must be compared for equality with the switch expression. The compiler has no such knowledge of a long list of if expressions.

In the below code snippet we are displaying the number in words. If its "1" then display "one" and if its "2" display "two. For "if " condition it scans through the entire if " conditions but in switch statement it directly jumps to the logic after looking at the lookup table.



```
for (int i=0;i < intNumbers.length+1;i++)
{
    if (i==1)
    {
        displayString(" One ");
    }
    else if(i==2)
    {
        displayString(" Two ");
    }
}
displayString(" Displaying Numbers using SWITCH / CASE Statements ");
for (int i=0;i < intNumbers.length+1;i++)
{
    switch(i)
    {
    case 1:
        displayString("One");
        break;
    case 2:
        displayString("Two");
        break;
    }
}
```

If condition will compare for conditions
Does not evaluate all case statement directly jumps to the condition. For instance in this case the value is 2 it jumps to the second case statement.

Figure 1.32 : - Difference between SWITCH and IF statements

Note: Sample for "Switch and If" condition is provided in "SwitchAndIf" folder in CD.

(I)What's the use of JAVAP tool?

"javap" disassembles compiled Java files and spits out representation of the Java program. This is a useful option when the original source code is not available.

(B)What are applets?

Applets are small applications that are accessed from web server automatically installed, and run from the browser. Once an applet arrives on the client it has limited access to resources thus ensuring security for the end user. An applet is controlled by the software that runs it. Usually, the underlying software is a browser, but it can also be applet viewer. If you run the applet source code from eclipse it runs inside an applet viewer. All applets should inherit from applet class.

Below are sequences of events which occur in applet:-

The init Method: The init method is called when the applet is first loaded. Init method can be used to initialize color, fonts or any type of one type operation needed for the applet.

The start Method: The start method is called when user visits a browser with an applet on it. In start method applet spawns a thread in which it runs the paint method.

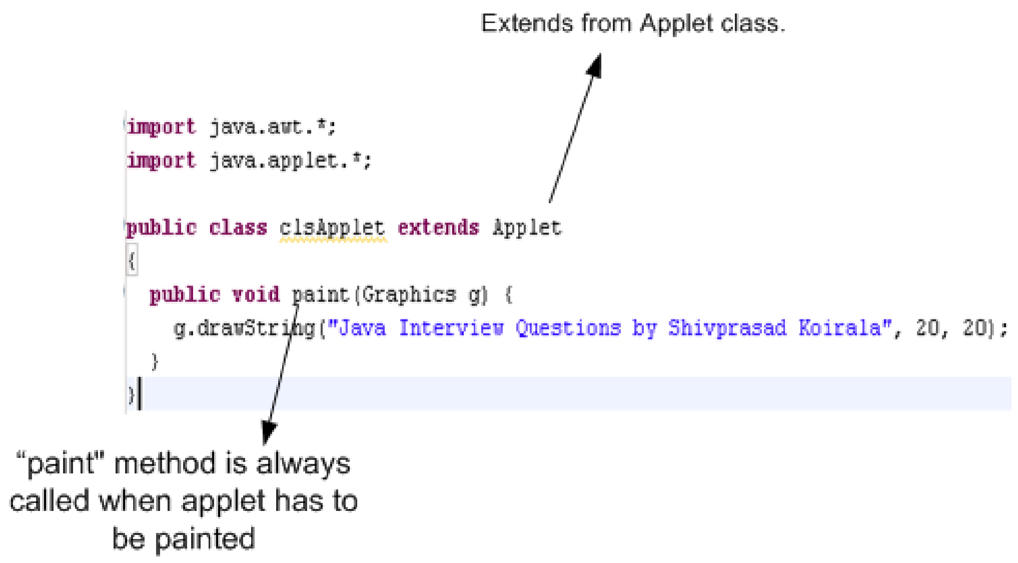
paint() is called every time when applet has to re-display everything. paint() event can occur due to various reasons some of the reasons are :-.

- √ When browser running the applet is covered or uncovered by some other window.
- √ When browser running the applet is minimized and maximized.
- √ paint() is also called when the applet begins execution.

Applet must always redraw its output when paint() method is called. paint() accepts graphics object as a input parameter you can see the same in the figure below. To just add something about this object it actually contains the graphics context. Graphic context determines the graphics environment in which the applet will be running and applet will use the same context to display output on the browser.

The stop and destroy Methods: The stop method stops the applet user goes to a different web page.

The destroy method is called when the browser exits. This is the place we can put in cleaning up of threads or any resource which is not a java resource.


Extends from Applet class.
"paint" method is always called when applet has to be painted

```
import java.awt.*;
import java.applet.*;

public class clsApplet extends Applet
{
  public void paint(Graphics g) {
    g.drawString("Java Interview Questions by Shivprasad Koirala", 20, 20);
  }
}
```

Figure 1.33 : - applet in action.

In order to make your fundamental stronger there is a small program on applet which displays my name of the book. You can get the above code of applet in CD in "applet" folder.

(B)In which package is the applet class located?

Applet classes are located in "java.applet" package.

(I)What are native interfaces in Java?

Note :- This can be answered in one liner to just make sure that you get enough grasp of native interfaces we have one topic which is completely dedicated to JNI.

(I) what are Class loader's?

JAVA was designed thinking platform independency in mind. So from loading libraries point of view it should not rely only on only file systems. In short it's not necessary that you should be able to load libraries which are in only file systems but also from other sources like FTP, HTTP etc. So JAVA went one step further you can load classes from across network or from any other source like FTP, HTTP etc. In order to attain this Class loaders came in to picture. Class loader is the class responsible for finding and loading classes at runtime. You can either use different class loaders or you can make a custom one.

(I) what is Bootstrap, Extension and System Class loader?

Twist: - Can you explain primordial class loader?

There three types of class loaders:-

√ BootStrap Class loader also called as primordial class loader.

√ Extension Class loader.

√ System Class loader.

Let's now try to get the fundamentals of these class loaders.

Bootstrap Class loader

Bootstrap class loader loads those classes those which are essential for JVM to function properly. Bootstrap class loader is responsible for loading all core java classes for instance java.lang.*, java.io.* etc. Bootstrap class loader finds these necessary classes from "jdk/jre/lib/rt.jar". Bootstrap class loader can not be instantiated from JAVA code and is implemented natively inside JVM.

Extension Class loader

The extension class loader also termed as the standard extensions class loader is a child of the bootstrap class loader. Its primary responsibility is to load classes from the extension directories, normally located the "jre/lib/ext" directory. This provides the ability to simply drop in new extensions, such as various security extensions, without requiring modification to the user's class path.

System Class loader

The system class loader also termed application class loader is the class loader responsible for loading code from the path specified by the CLASSPATH environment variable. It is also used to load an application's entry point class that is the "static void main ()" method in a class.

(I) Can you explain the flow between bootstrap, extension and system class loader?

Ok below is a simple class lets get in to details of how the class will be loaded.

Import Koirala.Interview.Java;

```
public class mySimpleClass
{
  public static void main(String[] args)
{
    String myStr = "I am going to get a job";
     System.out.println(myStr);
}}
```

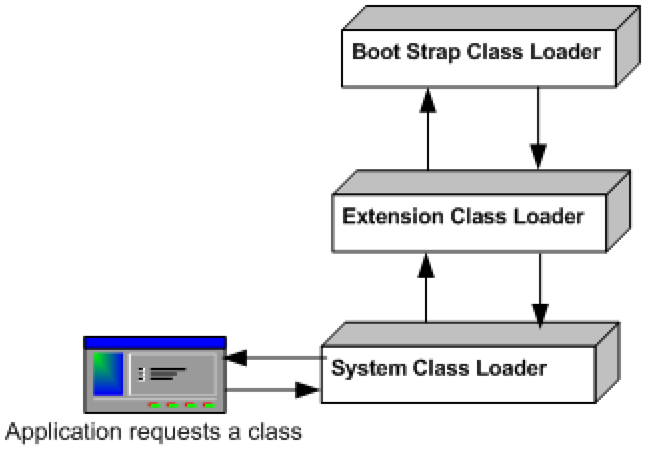


Figure 1.34 : - Flow between class loaders

The above class uses "String" class that means it has reference to "java.lang.String". JVM will request the system class loader to load "java.lang.String". But before he tries to load it will delegate to extension class loader. Extension class loader will pass it to Boot strap class loader. Now Boot Strap class loader does not have any parent so it will try to load "java.lang.String" using "rt.jar". Now the Boot strap will return the class back using the same chain to the application.

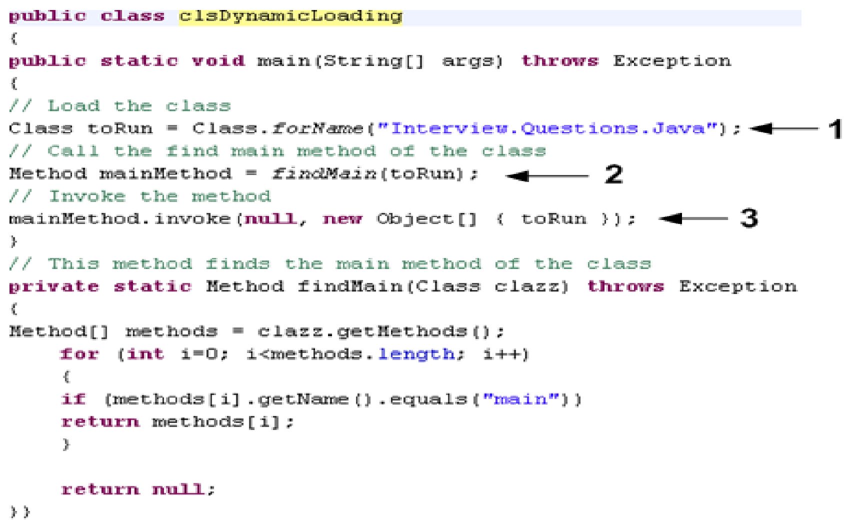
Now lets see how "Import Koirala.Interview.Java;" is loaded using the class loaders. For the import statement JVM will make a call to the system class loader who will delegate the same to the extension class loader which will delegate the same to the boot strap class loader. Boot strap loader will not find "Koirala.Interview.Java" and return nothing to the extension class loader. Extension class loader will also check the same in its path and will not find anything thus returning nothing to the system class loader. System class loader will use its class path and load the class and return the same to the JVM who will then return it to the application.

Note :- In the coming question we will be answering three question in on shot.

(I) Can you explain how can you practically do dynamic loading?

(I) What is Reflection API in Java ?

(I) What's the difference between static and dynamic class loading ?



```
public class clsDynamicLoading
{
public static void main(String[] args) throws Exception
{
// Load the class
Class toRun = Class.forName("Interview.Questions.Java");
// Call the find main method of the class
Method mainMethod = findMain(toRun);
// Invoke the method
mainMethod.invoke(null, new Object[] { toRun });
}
// This method finds the main method of the class
private static Method findMain(Class clazz) throws Exception
{
Method[] methods = clazz.getMethods();
    for (int i=0; i<methods.length; i++)
    {
    if (methods[i].getName().equals("main"))
    return methods[i];
    }

    return null;
}}
```

Figure 1.35 : - Dynamic class loading in action

Above is the code snippet to implement dynamic class loading. In the above code snippet we also have marked the steps in order. First thing we need to do is load the class which is done by using "Class.forName". In the second step we just try to find out the main method and get the method reference using "getMethods()" collection. Finally invoke the method using "invoke".

In static loading we use the "new" keyword to create object. "myObj obj = new myObj();". In short we need to know from the start what type of object we will be creating. While in dynamic class loading we need to know the string name of the class and then everything else will be dynamically loaded.

Dynamic class loading is only possible due to Reflection API Interface. Reflection API is a member of core java.lang package. The methods provided by the Reflection API obtain information about a class, such as the fields, constructors, methods, and super classes of the class. In addition, you can obtain the interfaces implemented by the class. You can notice in the above sample how we inspected the methods in the class.

Note: - You can find dynamic class loading sample project in the "\Source" folder.

(B) How can you copy one array in to a different array?

System.arraycopy(myOldArray, 0, myNewArray, 0, length);+

(B) Can you explain the core collection interfaces?

There are six interfaces and come under two different inheritance group one which comes under the collection interface root and the other in the map interface root.

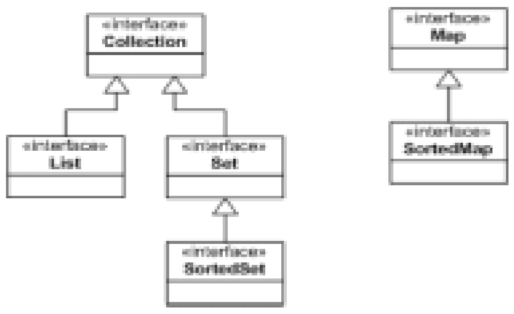


Figure 1.36 : - Collection Interface hierarchy

Below are the details of core collection interface.

Collection

It's the base of all collection classes. It provides a unified way to manipulate collection objects. Collection has group of object called as elements. These elements can be accessed and manipulated using Iterator.

List

In List interface the elements are arranged sequentially. Elements can be inserted in to any location and you can also insert duplicate elements. In order to access the elements you need to use the "ListIterator". Using "ListIterator" you can move back and forth which makes it unique as compared to other iterators.

Set

It represents a collection but no duplicates are allowed in this case.

SortedSet

It extends the Set interface and sorts the set in ascending order.

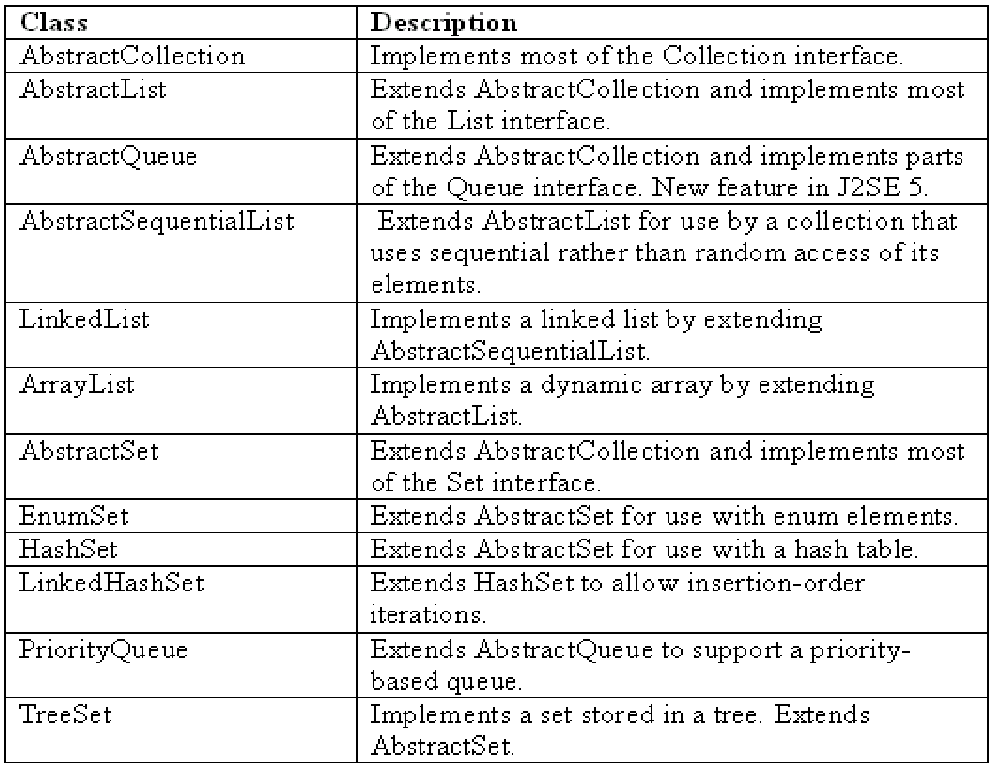
Map

Map stores association between keys and value pairs. So given a key you can easily find the value. One thing important to note is they do not implement iterable interface. But yes you can obtain a collection view of the map which allows you loop using for loop.

SortedMap

It Extends Map so that the keys are maintained in ascending order.

(I)Can you explain in brief the collection classes which implement the collection interfaces?



Class	Description
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of the List interface.
AbstractQueue	Extends AbstractCollection and implements parts of the Queue interface. New feature in J2SE 5.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList.
ArrayList	Implements a dynamic array by extending AbstractList.
AbstractSet	Extends AbstractCollection and implements most of the Set interface.
EnumSet	Extends AbstractSet for use with enum elements.
HashSet	Extends AbstractSet for use with a hash table.
LinkedHashSet	Extends HashSet to allow insertion-order iterations.
PriorityQueue	Extends AbstractQueue to support a priority-based queue.
TreeSet	Implements a set stored in a tree. Extends AbstractSet.

Figure 1.37: - Standard collection classes

(B) What's the difference between standard JAVA array and ArrayList class?

ArrayList support dynamic arrays that can grow depending on demand. But traditional JAVA arrays are of fixed length once they are created.

(B) What's the use of "ensureCapacity" in ArrayList class?

ArrayList are designed so that they can grow depending on demand. But there is a cost attached to the same. The cost of allocating and reallocating memory. You can prevent this reallocation by increasing the capacity at once. This can be attained by "ensureCapacity" method. Below is the syntax:-

```
void ensureCapacity(int cap)
```

(I) How can we obtain an array from an ArrayList class?

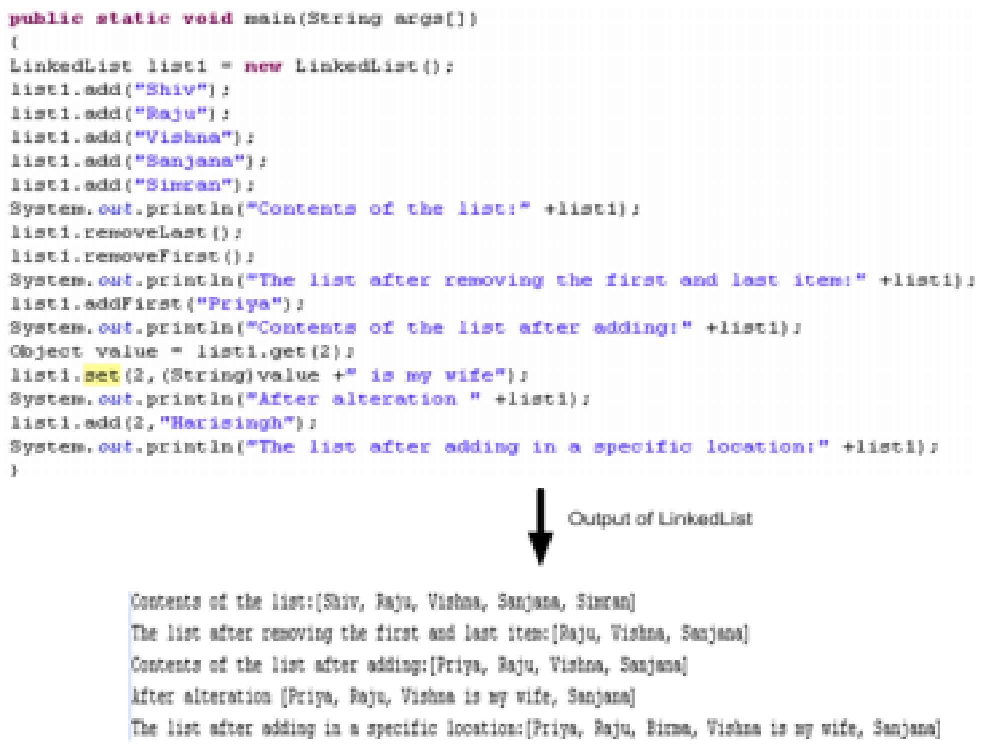
You can convert from arraylist to traditional arrays using "toArray()" below is the code snippet for the same.

```
ArrayList<Integer> myArrayList = new ArrayList<Integer>();

// add in to the arraylist collection

myArrayList.add(1);

myArrayList.add(2);

// Get the array.

Integer tempArray[] = new Integer[myArrayList.size()];

tempArray = myArrayList.toArray(tempArray);
```

Note: - This is a small home work to the readers write a simple program and sees how you can use "ArrayList". As this is an interview question book I have not got in to sample code for arraylist.

(B)What is "LinkedList" class for?

It provides link-list data structure. Below is the code snippet which shows LinkedList class in action.



```
public static void main(String args[])
{
LinkedList list1 = new LinkedList();
list1.add("Shiv");
list1.add("Raju");
list1.add("Vishna");
list1.add("Sanjana");
list1.add("Simran");
System.out.println("Contents of the list:" +list1);
list1.removeLast();
list1.removeFirst();
System.out.println("The list after removing the first and last item:" +list1);
list1.addFirst("Priya");
System.out.println("Contents of the list after adding:" +list1);
Object value = list1.get(2);
list1.set(2,(String)value +" is my wife");
System.out.println("After alteration " +list1);
list1.add(2,"Harisingh");
System.out.println("The list after adding in a specific location:" +list1);
}
```

```
Contents of the list:[Shiv, Raju, Vishna, Sanjana, Simran]
The list after removing the first and last item:[Raju, Vishna, Sanjana]
Contents of the list after adding:[Priya, Raju, Vishna, Sanjana]
After alteration [Priya, Raju, Vishna is my wife, Sanjana]
The list after adding in a specific location:[Priya, Raju, Birma, Vishna is my wife, Sanjana]
```

Figure 1.38 : - LinkedList sample code and output

We can add elements to the beginning or the end of the list using the below code snippet:-

void addFirst(Object objlst)

void addLast(Object objlst)

objlst is the element added to the list.

In order to get the first and the last element of the linked list we can use the getFirst() and getLast() methods respectively. Below is the code snippet for the same:-

Object getFirst()

Object getLast()

In order to delete the first and the last element of the list we can use the removeFirst() and removeLast() methods, respectively. Below is the code snippet for the same:-

Object removeFirst()

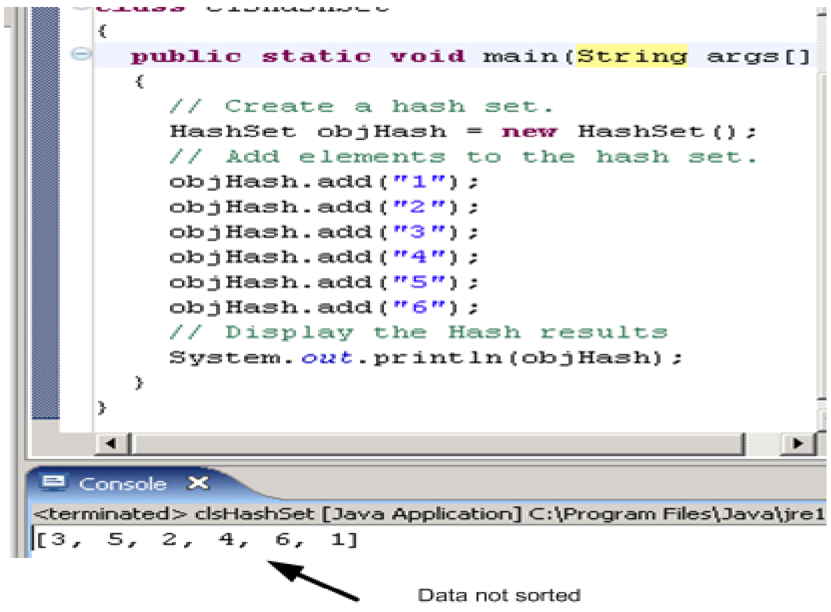
Object removeLast()

Note: - You can find the above code for LinkedList in "\Source Code\LinkedLIst" folder.

(B)Can you explain HashSet class in collections?

HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a hash table for storage. Hash table stores information by using a mechanism called hashing. Information of the key is used to determine a unique value termed hash code. This hash code is then used as an index for data associated with the key is stored. This happens automatically and you will never need to know the hash code value. Also, your code can't directly index the hash table.

The biggest advantage of hashing is that execution time of add( ), contains( ), remove( ), and size( ) to remain same even when the collection are large in size. Hash set does not guarantee sorting. You can see from the below code snippet that data is not sorted.



```
{
    public static void main(String args[]
    {
        // Create a hash set.
        HashSet objHash = new HashSet();
        // Add elements to the hash set.
        objHash.add("1");
        objHash.add("2");
        objHash.add("3");
        objHash.add("4");
        objHash.add("5");
        objHash.add("6");
        // Display the Hash results
        System.out.println(objHash);
    }
}
```

```
<terminated> clsHashSet [Java Application] C:\Program Files\Java\jre1
[3, 5, 2, 4, 6, 1]
```

Figure 1.39 : - HashSet Sample code

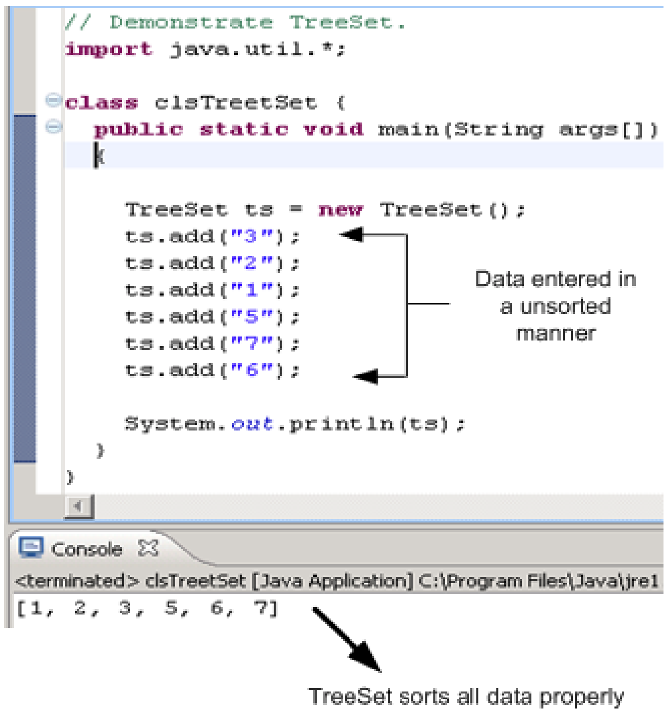
Note: - You can fine the source code for the same in "\SourceCode\Hashset" folder in CD.

(I)what is LinkedHashSet class?

LinkedHashSet extends HashSet class but it does not add any extra members as such. LinkedHashSet maintains list of entries in the order in which they are inserted. So when you iterate through a LinkedHashSet class elements will be returned in the order they are inserted. In order to just see how LinkedHashSet works you can try the HashSet code with LinkedHashset you should get the order as "[1, 2, 3, 4, 5, 6]".

(I) what is a TreeSet class?

TreeSet creates a collection that uses a tree for storage. Items in collection are stored in ascending order. TreeSet are good choice for storing large amount of data. Below is a small snippet which shows "TreeSet" in action.



```
// Demonstrate TreeSet.
import java.util.*;

class clsTreetSet {
  public static void main(String args[])
  {

    TreeSet ts = new TreeSet();
    ts.add("3");
    ts.add("2");
    ts.add("1");
    ts.add("5");
    ts.add("7");
    ts.add("6");

    System.out.println(ts);
  }
}
```

```
<terminated> clsTreetSet [Java Application] C:\Program Files\Java\jre1
[1, 2, 3, 5, 6, 7]
```

Data entered in a unsorted manner
Console
TreeSet sorts all data properly

Figure 1.40 : - TreeSet collection in action

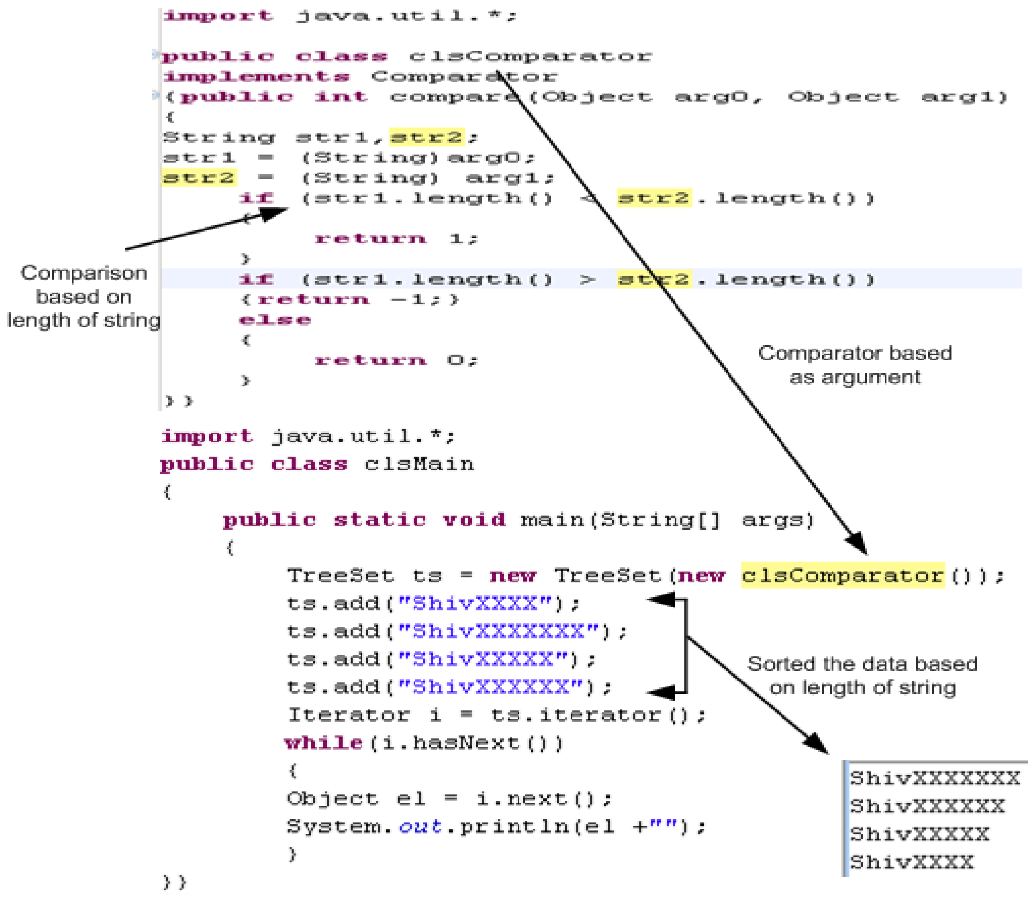
Note :- You can find "TreeSet" code in "TreeSet" directory.

(I) what's the use of Comparator Interface?

By default JAVA sorts collection in ascending order. We can use the Comparator Interface to specify a sort order. Comparator interface provides compare() method which compares two elements. Below is the syntax snippet for the same:-

int compare(Object o1, Object o2)

Below is a sample code which shows how to use the comparator interface. It sorts using the string length. In order to achieve this we have created two classes clsComparator and clsMain. clsComparator implements the Comparator interface and implements the compare method with out custom comparison. clsMain is the entry point and has the void main method. We can see object of clsComparator is passed to the TreeSet object.



```
import java.util.*;
public class clsComparator
implements Comparator
{public int compare(Object arg0, Object arg1)
{
String str1,str2;
str1 = (String)arg0;
str2 = (String) arg1;
    if (str1.length() < str2.length())
    {
        return 1;
    }
    if (str1.length() > str2.length())
    {return -1;}
    else
    {
        return 0;
    }
}}
```

```
import java.util.*;
public class clsMain
{
    public static void main(String[] args)
    {
        TreeSet ts = new TreeSet(new clsComparator());
        ts.add("ShivXXXX");
        ts.add("ShivXXXXXXX");
        ts.add("ShivXXXXX");
        ts.add("ShivXXXXXX");
        Iterator i = ts.iterator();
        while(i.hasNext())
        {
        Object el = i.next();
        System.out.println(el +"");
        }
}}
```

```
ShivXXXXXXX
ShivXXXXXX
ShivXXXXX
ShivXXXX
```

Comparison based on length of string
Comparator based as argument
Sorted the data based on length of string

Figure 1.41 : - Comparator in action

Note: - You can find the above code snippet in “Comparator” folder.

(B) How can we access elements of a collection?

You can access the elements using Iterator or ListIterator. For sample you can look at the above comparator sample which uses iterator to display the details. You can also use for-each to loop.

(B) What is Map and SortedMap Interface?

Map is an object that stores associations between keys and values, or key/value pairs. Given a key, you can find its value. Keys and values are stored as objects. Keys always have to be unique but the values may be duplicated. Some maps can accept null key and null values. Maps don't implement the Iterable interface. So that means you cannot iterate through a map using a for-each. But collection-view of a map allows getting around the sort coming of iteration.

Sorted Map ensures that the entries are maintained in ascending order based on the keys. You can also use the comparator specify some custom sort order.

(B) Have you used any collection algorithm?

Note: - During interview you only need to mention few of them. But for the sake of just grabbing some in your mind I have listed all. Just see which you have used and say the same during interview.

static <T> boolean addAll(Collection <? super T> c, T ... elements)

Inserts the elements specified by elements into the collection specified by c. Returns true if the elements were added and false otherwise.

static <T> int binarySearch(List<? extends T> list, T value, Comparator<? super T> c)

Searches for value in list ordered according to c. Returns the position of value in list, or a negative value if value is not found.

static <T> int binarySearch(List<? extends Comparable<? super T>> list, T value)

Searches for value in list. The list must be sorted. Returns the position of value in list, or a negative value if value is not found.

static <E> Collection<E> checkedCollection(Collection<E> c, Class<E> t)

Returns a run-time type-safe view of a collection. An attempt to insert an incompatible element will cause a ClassCastException.

static <E> List<E> checkedList(List<E> c, Class<E> t)

Returns a run-time type-safe view of a List. An attempt to insert an incompatible element will cause a ClassCastException.

static <K, V> Map<K, V> checkedMap(Map<K, V> c, Class<K> keyT, Class<V> valueT)

Returns a run-time type-safe view of a Map. An attempt to insert an incompatible element will cause a ClassCastException.

static <E> List<E> checkedSet(Set<E> c, Class<E> t)

Returns a run-time type-safe view of a Set. An attempt to insert an incompatible element will cause a ClassCastException.

static <K, V> SortedMap<K, V> checkedSortedMap(SortedMap<K, V> c, Class<K> keyT, Class<V> valueT)

Returns a run-time type-safe view of a SortedMap. An attempt to insert an incompatible element will cause a ClassCastException.

static <E> SortedSet<E> checkedSortedSet(SortedSet<E> c, Class<E> t)

Returns a run-time type-safe view of a SortedSet. An attempt to insert an incompatible element will cause a ClassCastException.

static <T> void copy(List<? super T> list1, List<? extends T> list2)

Copies the elements of list2 to list1.

static boolean disjoint(Collection<?> a, Collection<?> b)

Compares the elements in a to elements in b. Returns true if the two collections contain no common elements (i.e., the collections contain disjoint sets of elements). Otherwise, returns true.

static <T> List<T> emptyList( )

Returns an immutable, empty List object of the inferred type.

static <K, V> Map<K, V> emptyMap( )

Returns an immutable, empty Map object of the inferred type.

static <T> Set<T> emptySet( )

Returns an immutable, empty Set object of the inferred type.

static <T> Enumeration<T> enumeration(Collection<T> c)

Returns an enumeration over c. (See "The Enumeration Interface," later in this chapter.)

static <T> void fill(List<? super T> list, T obj)

Assigns obj to each element of list.

static int frequency(Collection<?> c, Object obj)

Counts the number of occurrences of obj in c and returns the result.

static int indexOfSubList(List<?> list, List<?> subList)

Searches list for the first occurrence of subList. Returns the index of the first match, or –1 if no match is found.

static int lastIndexOfSubList(List<?> list, List<?> subList)

Searches list for the last occurrence of subList. Returns the index of the last match, or –1 if no match is found.

static <T> ArrayList<T> list(Enumeration<T> enum)

Returns an ArrayList that contains the elements of enum.

static <T> T max(Collection<? extends T> c, Comparator<? super T> comp)

Returns the maximum element in c as determined by comp.

static <T extends Object & Comparable<? super T>> T max(Collection<?

extends T> c)

Returns the maximum element in c as determined by natural ordering. The collection need not be sorted.

static <T> T min(Collection<? extends T> c, Comparator<? super T> comp)

Returns the minimum element in c as determined by comp. The collection need not be sorted.

static <T extends Object & Comparable<? superT>> T min(Collection<? extends T> c)

Returns the minimum element in c as determined by natural ordering.

static <T> List<T> nCopies(int num, T obj)

Returns num copies of obj contained in an immutable list. num must be greater than or equal to zero.

static <T> boolean replaceAll(List<T> list, T old, T new)

Replaces all occurrences of old with new in list. Returns true if at least one replacement occurred. Returns false, otherwise.

static void reverse(List<T> list)

Reverses the sequence in list.

static <T> Comparator<T> reverseOrder(Comparator<T> comp)

Returns a reverse comparator based on the one passed in comp. That is, the returned comparator reverses the outcome of a comparison that uses comp.

static <T> Comparator<T> reverseOrder( )

Returns a reverse comparator, which is a comparator that reverses the outcome of a comparison between two elements.

static void rotate(List<T> list, int n)

Rotates list by n places to the right. To rotate left, use a negative value for n.

static void shuffle(List<T> list, Random r)

Shuffles (i.e., randomizes) the elements in list by using r as a source of random numbers.

static void shuffle(List<T> list)

Shuffles (i.e., randomizes) the elements in list.

static <T> Set<T> singleton(T obj)

Returns obj as an immutable set. This is an easy way to convert a single object into a set.

static <T> List<T> singletonList(T obj)

Returns obj as an immutable list. This is an easy way to convert a single object into a list.

static <K, V> Map<K, V> singletonMap(K k, V v)

Returns the key/value pair k/v as an immutable map. This is an easy way to convert a single key/value pair into a map.

static <T> void sort(List<T> list,Comparator<? super T> comp)

Sorts the elements of list as determined by comp.

static <T extends Comparable<? super T>> void sort(List<T> list)

Sorts the elements of list as determined by their natural ordering.

static <T> void swap(List<T> list, int idx1, int idx2)

Exchanges the elements in list at the indices specified by idx1 and idx2.

static <T> Collection<T>synchronizedCollection(Collection<T> c)

Returns a thread-safe collection backed by c.

static <T> List<T> synchronizedList(List<T> list)

Returns a thread-safe list backed by list.

static <K, V> Map<K, V> synchronizedMap(Map<K, V> m)

Returns a thread-safe map backed by m.

static <T> Set<T> synchronizedSet(Set<T> s)

Returns a thread-safe set backed by s.

static <K, V> SortedMap<K, V> synchronizedSortedMap(SortedMap<K, V> sm)

Returns a thread-safe sorted map backed by sm.

static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> ss)

Returns a thread-safe set backed by ss.

static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c)

Returns an unmodifiable collection backed by c.

static <T> List<T> unmodifiableList(List<? extends T> list)

Returns an unmodifiable list backed by list.

static <K, V> Map<K, V> unmodifiableMap(Map<? extends K, ? extends V> m)

Returns an unmodifiable map backed by m.

static <T> Set<T> unmodifiableSet(Set<? extends T> s)

Returns an unmodifiable set backed by s.

static <K, V> SortedMap<K, V> unmodifiableSortedMap(SortedMap<K, ?

extends V> sm)

Returns an unmodifiable sorted map backed by sm.

static <T> SortedSet<T> unmodifiableSortedSet(SortedSet<T> ss)

Returns an unmodifiable sorted set backed by ss.

(B) Why do we use collections when we had traditional ways for collection?

Before the collection framework JAVA provided ad hoc classes such as Dictionary, Vector, Stack and Properties to store and manipulate group of objects. Following are the main reason why collection framework is more desirable than the traditional JAVA collection objects:-

√ Traditional objects did not have the unifying theme. The way you access VECTOR is different from Properties. Due to this ADHOC approach software is not easily extendable and adaptable. One of the unifying themes in the new JAVA collection is the iterator interface. It gives a unifying way for looping through JAVA collections.

√ Ready made algorithms are one of the important features of the JAVA collection. Algorithms operate on collections and are defined as static methods within the Collections class. Thus, they are available for all collections. Each collection class need not implement its own versions. The algorithms provide a standard means of manipulating collections.

√ Implementation of dynamic arrays, linked lists, trees, and hash tables are done in efficient manner. In case of traditional JAVA collection you will need to code all these implementation by yourself.

(B) Can you name the legacy classes and interface for collections?

Below are the legacy classes and interfaces for JAVA:-

Dictionary

Dictionary is represents a key/value storage repository and operates in very similiar fashion like Map. Given a key and value, you can store the value in a Dictionary object. Once the value is stored, you can retrieve it by using its key. Thus, like a map, a dictionary can be thought of as a list of key/value pairs. Although not currently deprecated, Dictionary is classified as obsolete, because it is fully replaced by Map.

Hashtable

Hashtable stores key/value pairs in a hash table. When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table. You can visual HashMap as replacement for Hashtable.

Properties

Properties is a child class of Hashtable. It is specially used to maintain lists of values in which the key and the value is String. The Properties class is used by many Java classes. For instance type of object returned by System.getProperties( ) when obtaining environmental values is type of Properties class. One of the most useful aspects of Properties is that the information contained in a Properties object can be easily stored to or loaded from disk with the store( ) and load( ) methods. At any time, you can write a Properties object to a stream or read it back.

Stack

Stack is a subclass of Vector that implements a standard last-in, first-out stack.

Vector

Vector implements a dynamic array. Though similar to ArrayList it has one main differences Vector is synchronized. Which mean any thread using Vector is thread safe.

(B)What is Enumeration Interface?

Enumeration interface defines the methods by which you can enumerate the elements in a collection of objects. In JAVA collection it's now replaced by Iterator.

(I) what's the main difference between ArrayList / HashMap and Vector / Hashtable?

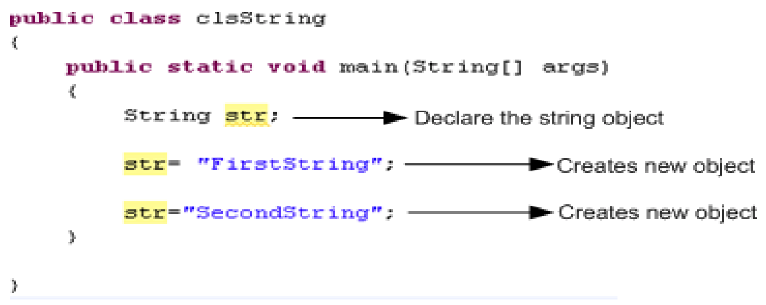
Vector / HashTable are synchronized which means they are thread safe. Cost of thread safe is performance degradation. So if you are sure that you are not dealing with huge number of threads then you should use ArrayList / HashMap.But yes you can still synchronize List and Map's using Collections provided methods :-

List OurList = Collections.synchronizedList (OurList);

Map OurMap = Collections.synchronizedMap (OurMap);

(I) Are String object Immutable, Can you explain the concept?

When we create a string object that means you are creating a string which can not be changed. So once you have created a string object you can not change the value of string. You must be thinking but I can always change the value. Yes but when you assign a value it creates a new object and the first object is still in memory. Lets try to understand the same by the below code snippet.



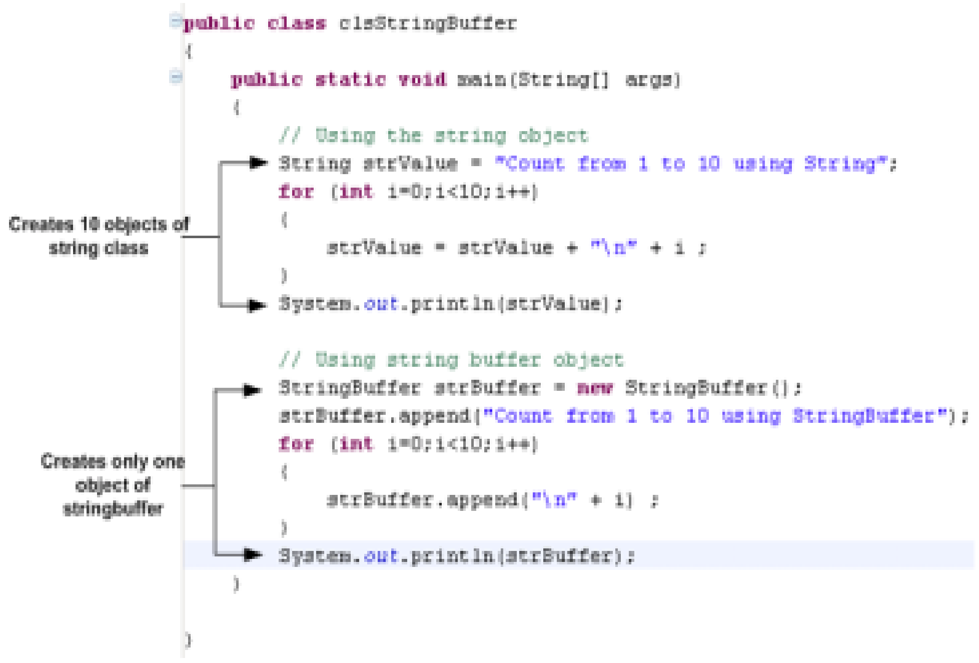
```
public class clsString
{
    public static void main(String[] args)
    {
        String str;
        str= "FirstString";
        str="SecondString";
    }

}
```

Figure 1.42 : - Strings in Action

In the above code we have assigned the string with two values. So there are two objects of string instances created in the above statement. First statement declares the string object at this point no object is created. As soon as you assign a value to the str a object is created and assigned to it. That means in the second step when you assign value “FirstString” value it creates a new instance. In the third step when you assign “SecondString” value it creates a new instance and assigns it to str. But the first instance is still in memory. It's only that str is now referring to the new instance of string object.

(I) what is a StringBuffer class and how does it differs from String class?

StringBuffer is a peer class of String that provides almost all functionality of strings. String represents fixed-length, immutable character sequences. Comparatively StringBuffer represents mutable, growable and writeable character sequences. But StringBuffer does not create new instances as string so it's more efficient when it comes to intensive concatenation operation. Below is a simple code which shows clearly the efficiency of StringBuffer object.


Creates 10 objects of string class
Creates only one object of stringbuffer

```
public class clsStringBuffer
{
    public static void main(String[] args)
    {
        // Using the string object
        String strValue = "Count from 1 to 10 using String";
        for (int i=0;i<10;i++)
        {
            strValue = strValue + "\n" + i ;
        }
        System.out.println(strValue);

        // Using string buffer object
        StringBuffer strBuffer = new StringBuffer();
        strBuffer.append("Count from 1 to 10 using StringBuffer");
        for (int i=0;i<10;i++)
        {
            strBuffer.append("\n" + i) ;
        }
        System.out.println(strBuffer);
    }

}
```

Figure 1.43 : - String and StringBuffer Comparison

When we loop using String class we are creating 10 objects. But with StringBuffer only one object is created.

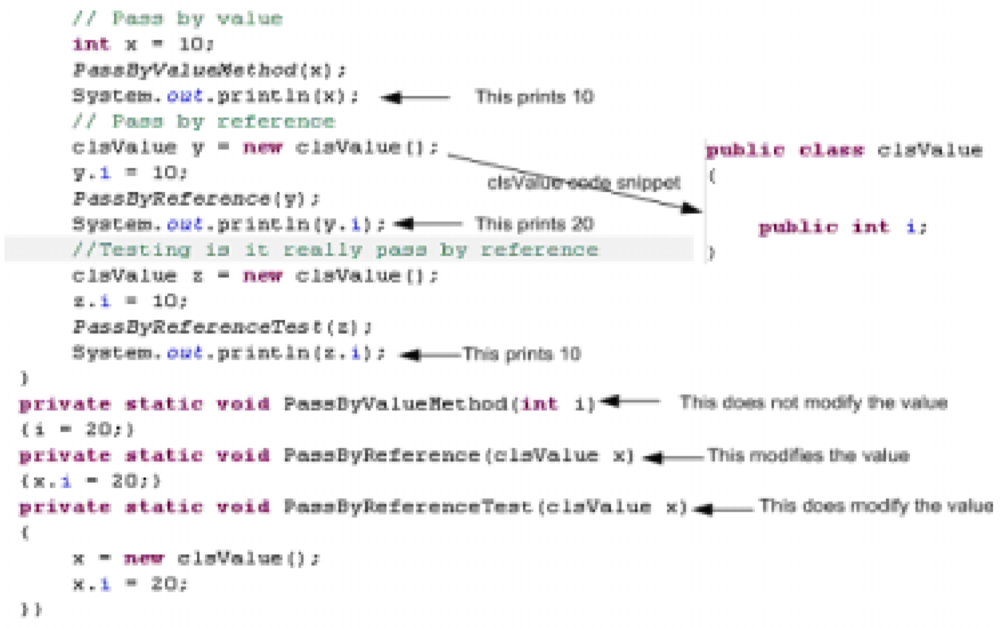
(I) what is the difference between StringBuilder and StringBuffer class?

It is very much similar to StringBuffer except for one difference: it is not synchronized, which means that it is not thread-safe. The advantage of StringBuilder is good performance. In case of multithreading, you must use StringBuffer rather than StringBuilder.

(B) What is Pass by Value and Pass by reference? How does JAVA handle the same?

In Pass by Value the function or subroutine receives a copy of variable. The function or the method can not change variable values.

In pass By ref the function or subroutine receives a pointer of the variable. And any changes to variable value are also visible outside the function or subroutine. Lets try to understand the concept from the below snippet code.



```
    // Pass by value
    int x = 10;
    PassByValueMethod(x);
    System.out.println(x);
    // Pass by reference
    clsValue y = new clsValue();
    y.i = 10;
    PassByReference(y);
    System.out.println(y.i);
    //Testing is it really pass by reference
    clsValue z = new clsValue();
    z.i = 10;
    PassByReferenceTest(z);
    System.out.println(z.i);
}
private static void PassByValueMethod(int i)
{i = 20;}
private static void PassByReference(clsValue x)
{x.i = 20;}
private static void PassByReferenceTest(clsValue x)
{
    x = new clsValue();
    x.i = 20;
}}
```

```
public class clsValue
{

    public int i;
}
```

Figure 1.44 : - Pass by Value and reference in action

Note: - You can find the above source code in "PassByValRef" folder in the CD.

There are two methods "PassByValueMethod" and "PassByReference" method. When we pass value to "PassByValueMethod" it does not return a changed value for the variable. When we pass value to "PassByReference" method (here we have passed object of "clsValue") it changes the value of property "i". Now that we know the fundamentals of "Pass by Value" and "Pass by Reference" lets try to see the same from JAVA perspective.

"JAVA is strictly Pass by Value. So if it's either primitive data type or an object JAVA always passes by value"

Hmmm i know what's going on in your mind if JAVA is pass by value why did "PassByReference" method change the value. Have a look at "PassByReferenceTest" its prints "10" and not "20". Even when we have changed the object reference it still reflects the old object value. In short object references are passed by value. So put it in your head in JAVA everything is "Pass By Value".

(I) What are access modifiers?

Access modifiers decide whether a method or a data variable can be accessed by another method in another class or subclass.

Java provides four types of access modifiers:

Public: - Can be accessed by any other class anywhere.

Protected: - Can be accessed by classes inside the package or by subclasses (that means classes who inherit from this class).

Private - Can be accessed only within the class. Even methods in subclasses in the same package do not have access.

Default - (Its private access by default) accessible to classes in the same package but not by classes in other packages, even if these are subclasses.

(I) what is Assertion?

Assertion includes a Boolean expression. If the Boolean expression evaluates to false AssertionError is thrown. Assertion helps a programmer to debug the code effectively. Assertion code is removed when the program is deployed. Below is a code snippet which checks if inventory object is null.

```
Inventory objInv = null;

// ...

// Get your inventory object reference here

// ...

// Use assert to check if the object is null

// by any chance

assert objInv != null;
```

(I) Can you explain the fundamentals of deep and shallow Cloning?

Many times in project you need to create exact copy of the object and operate on them. To do this there two ways of doing it Shallow clone and Deep clone. When an object is Shallow cloned the parent object or the top level object and its members are duplicated. But any lower level objects are not duplicated. Rather references of these objects are copied. So when an object is shallow cloned and you modify any of it child classes it will affect the original copy. When an object is deep cloned the entire object and its aggregated objects are also duplicated.

Below is the diagram which explains things in a clear and pictorial manner.

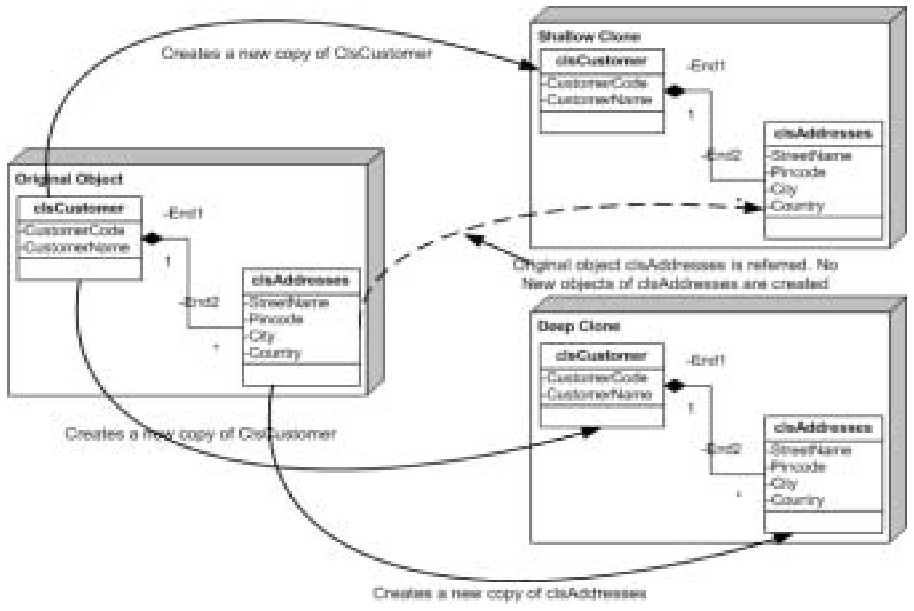
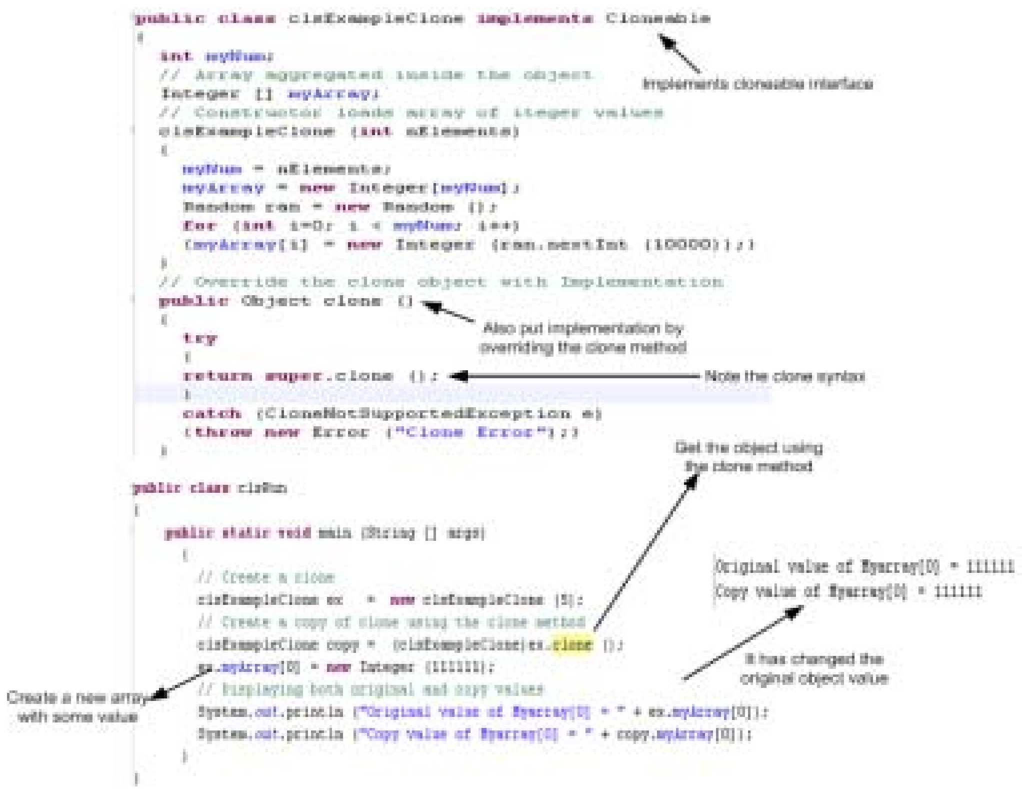


Figure 1.45 : - Deep clone and Shallow clone in action

In the above diagram there are three blocks the first block is the original object, second block is the Shallow clone and the third is Deep clone block. Here the object to be cloned is a simple Customer class which has multiple addresses. Now when you shallow clone it you can see the top class clsCustomer is duplicated but clsAddresses still refers to the original object. But in case of Deep clone complete new copy is created of the whole structure.

(I) How do we implement Shallow cloning?

In order to implement shallow cloning we need to implement cloneable interface and override the clone method with implementation. You can look in to the below code snippet.



Implements cloneable interface
Also put implementation by overriding the clone method
Note the clone syntax
Get the object using the clone method
It has changed the original object value
Create a new array with some value

```
public class clsExampleClone implements Cloneable
{
  int myNum;
  // Array aggregated inside the object
  Integer [] myArray;
  // Constructor loads array of integer values
  clsExampleClone (int nElements)
  {
    myNum = nElements;
    myArray = new Integer[myNum];
    Random ran = new Random ();
    for (int i=0; i < myNum; i++)
    {myArray[i] = new Integer (ran.nextInt (10000));}
  }
  // Override the clone object with Implementation
  public Object clone ()
  {
    try
    {
    return super.clone ();
    }
    catch (CloneNotSupportedException e)
    {throw new Error ("Clone Error");}
  }

public class clsRun
{
   public static void main (String [] args)
    {
     // Create a clone
     clsExampleClone ex   =  new clsExampleClone (5);
     // Create a copy of clone using the clone method
     clsExampleClone copy =  (clsExampleClone)ex.clone ();
     ex.myArray[0] = new Integer (111111);
     // Displaying both original and copy values
     System.out.println ("Original value of Myarray[0] = " + ex.myArray[0]);
     System.out.println ("Copy value of Myarray[0] = " + copy.myArray[0]);
    }
}
```

```
Original value of Myarray[0] = 111111
Copy value of Myarray[0] = 111111
```

Figure 1.46 : - Shallow cloning in action

In the above code snippet we have defined a class clsExampleClone which implements the Cloneable interface. We have also provided implementation for the clone method. It just returns the parents clone object. Once we have defined this object we created the same in void main and you can see even though we have cloned the object it still refers to original array and modifies the original object values.

Note: - You can find the above code in "ShallowClone" folder.

(I) How do we implement deep cloning?

Deep cloning can be done by two ways:-

All objects implement a clone method

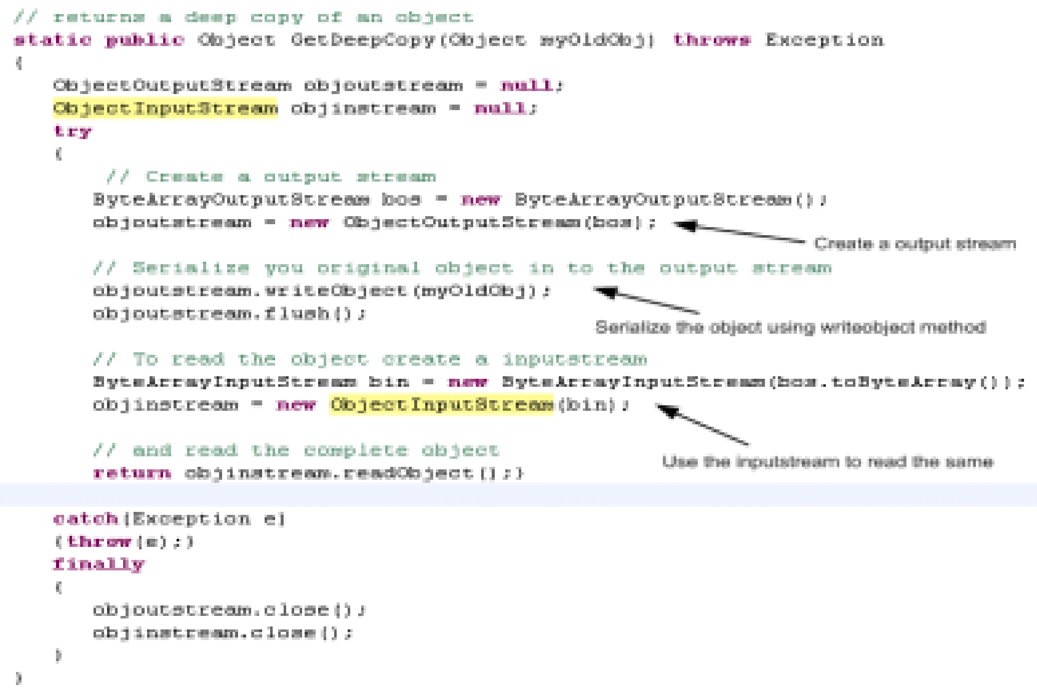
Every object is responsible for cloning itself via its clone () method. So when the parent is called it makes calls to all the referenced objects inside the class and calls its clone method.

Serialization

This is the best way to deep cloning and not to mention best answer in interview of how to implement deep cloning. There are three steps to do deep cloning using serialization:-

- √ Ensure that all classes in the object are serializable
- √ Create output stream for writing the new object and input stream for reading the same.
- √ Pass the object you want to copy to the output stream.
- √ And finally read the object using the input stream.

Below is the code snippet which implements all the above steps.



```
// returns a deep copy of an object
static public Object GetDeepCopy(Object myOldObj) throws Exception
{
    ObjectOutputStream objoutstream = null;
    ObjectInputStream objinstream = null;
    try
    {
        // Create a output stream
       ByteArrayOutputStream bos = new ByteArrayOutputStream();
       objoutstream = new ObjectOutputStream(bos);

       // Serialize you original object in to the output stream
       objoutstream.writeObject(myOldObj);
       objoutstream.flush();

       // To read the object create a inputstream
       ByteArrayInputStream bin = new ByteArrayInputStream(bos.toByteArray());
       objinstream = new ObjectInputStream(bin);

       // and read the complete object
       return objinstream.readObject();}

    catch(Exception e)
    {throw(e);}
    finally
    {
       objoutstream.close();
       objinstream.close();
    }
}
```

Create a output stream
Serialize the object using writeobject method
Use the inputstream to read the same

Figure 1.47 : - Deep cloning

In the above code snippet note the three important steps which have been marked for simplicity.

> *Note: - You can get the above code in "DeepCloning" folder. Define a complex object and try to create a deep cloned object from the function.*

(I) What's the impact of private constructor?

Private constructors prevent a class from being explicitly instantiated by callers.

(I) What are the situations you will need a constructor to be private?

Below are some of the situations when you will need a constructor to be private:-

- √ To implement singleton pattern. That means only one instance of the object need to be running.
- √ When classes contain static methods. It makes sense that no object of the class need to be created.
- √ If classes have only constants.

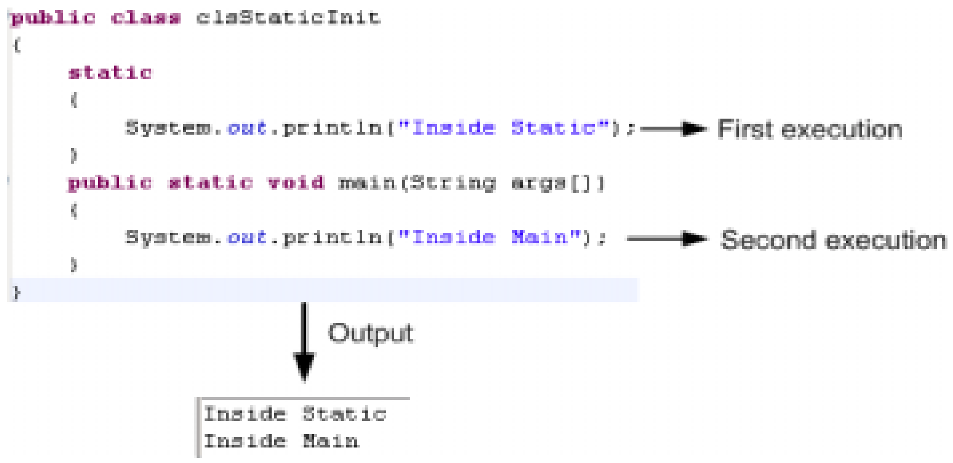
(I) Can you explain final modifier?

Below are some important points to be noted for final modifiers:-

- √ Final access modifiers apply to class, method and variables.
- √ So depending on what you are applying the final context changes.
- √ Final class can not be sub classed.
- √ A final variable cannot be modified once it has been assigned a value. They play the same role as constant variables.
- √ You can not change a final object reference variable. But you can change the object data.

(I) What are static Initializers?

Static initializer block is a method with no name (sounds funny hmm), no arguments, and no return type. It doesn't need a name, because there is no need to refer to it from outside the class definition. The code in a static initializer block is executed by the JVM when the class is loaded. You can use the static initializer block to initialize static variables or do some class level initialization. Static initializer block runs only once irrespective of how many instances you create of the object.



```
public class clsStaticInit
{
    static
    {
        System.out.println("Inside Static");
    }
    public static void main(String args[])
    {
        System.out.println("Inside Main");
    }
}
```

```
Inside Static
Inside Main
```

Figure 1.48 : - Static Initializers in action

Above is the code snippet which shows how static initializers work. The first thing which will execute is the static initializer block and then the void main method. One of the important point to keep in mind is it runs before the constructor.

Note :- You can find the above code sample in "StaticInitializers" folder. Feel free to test it using constructor.

(I) If we have multiple static initializer blocks how is the sequence handled?

You can have more than one static initializer block in your class definition, and they can be separated by other code such as method definitions and constructors. The static initializer blocks will be executed in the order in which they appear in the source code.

(B) Define casting? What are the different types of Casting?

Changing the type of the value from one type to other is termed as casting. For instance look at the below code snippet in which we are trying to cast integer value to double data type.

int i;

double d;

i = 10;

d = i; // we are trying to assign a int value to double

There are two types of casting explicit and implicit. To explicitly cast an expression prefix the expression with type name as shown in the code snippet below.

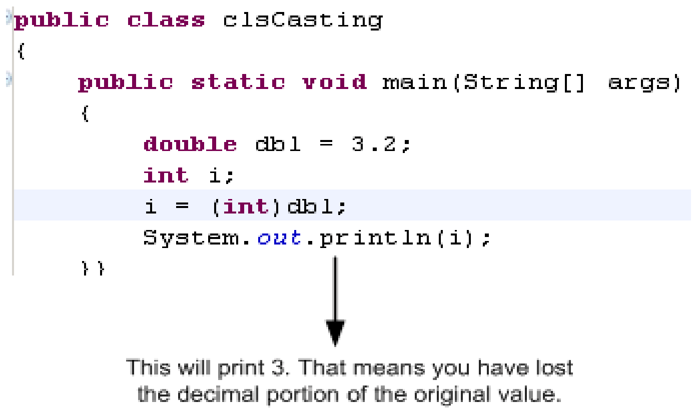
Button mybtn = (Button) (myVector.elementAt(9));

In some situations JAVA runtime changes the type of an expression with out performing a cast. For instance in the below code snippet my inventory object is type casted and stored as type Object.

myVectorSales.add(objInventory);

(B) Can you explain Widening conversion and Narrowing conversion?

When you try to move a big value data type to a small data type variable it leads to narrowing conversion. Below is the code snippet which explains in detail of how the conversion happens. In the below code we are trying to move double to an integer data type. Because integer data type is smaller than double data type it looses its decimal portion. So the below code will display the integer value as “3”.



```
public class clsCasting
{
    public static void main(String[] args)
    {
        double dbl = 3.2;
        int i;
        i = (int)dbl;
        System.out.println(i);
    }}
```

This will print 3. That means you have lost
the decimal portion of the original value.

Figure 1.49 : - Casting of narrowing conversion

Widening conversion is exactly vice versa of the same in which try to move small data type value to big one. In this case no conversion happens and neither have we loosed data. Below is the conversion tree which says when Widening conversion will happen and when narrowing conversion will be done.

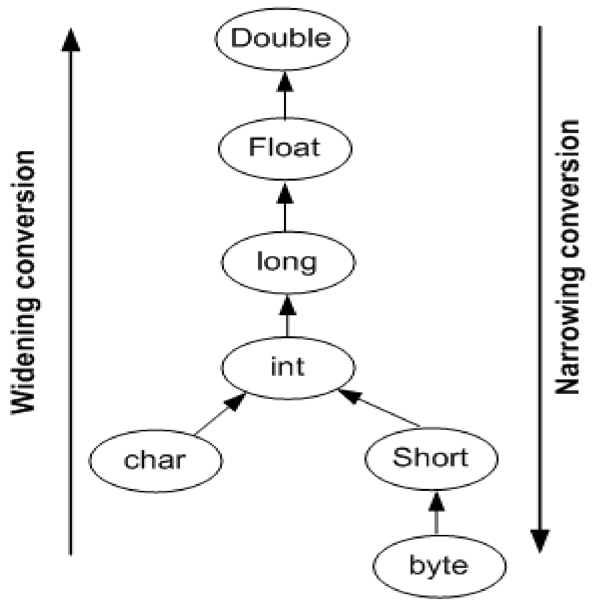


Figure 1.50 : - Conversion tree

So if we try to move up the tree that means if we try move char to int, int to long, float to double widening conversion will happen. If we try to move down the tree like int to char, double to int narrowing conversion will happen and probably also data loss. We need to specify explicit casting only if narrowing conversion is taking place. Doing that we say to the compiler that we are aware of the data loss which can happen.

(I) Can we assign parent object to child objects?

Let us first try to understand the question in a more precise manner. Glance the below class diagram which depicts a typical hierarchical relation ship between two wheelers. So can we move an object of Cycle to object of two wheeler?. Answer is yes we can do it by type casting the parent to child type. Again the law of conversion hierarchy applies here. In the below figure you can see the two arrows one which points from top to down and the other vice-versa.

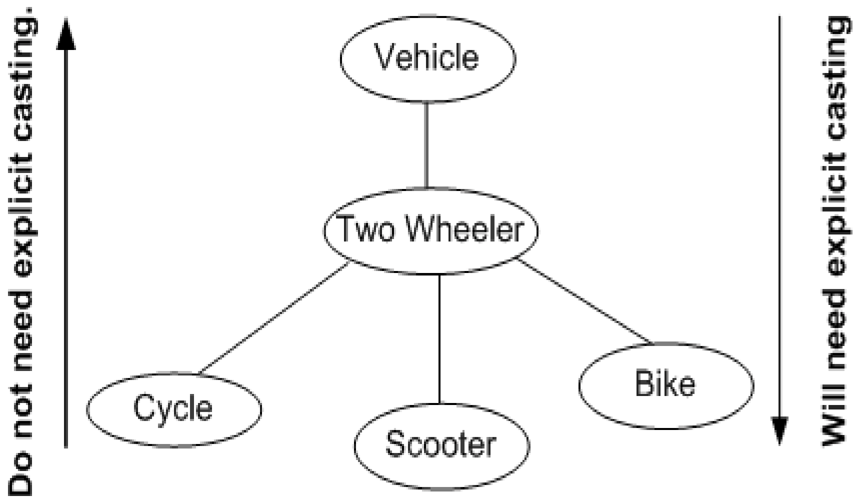


Figure 1.51 : - Object casting hierarchy

So if you are moving from down the hierarchy to the top you do not need to do explicit casting. That means if you move cycle object to two wheeler compiler will not throw error. But if you try to move two wheeler to cycle it will throw an error if you have not defined explicit casting of the object type. Below is the sample pseudo code which shows how things work.

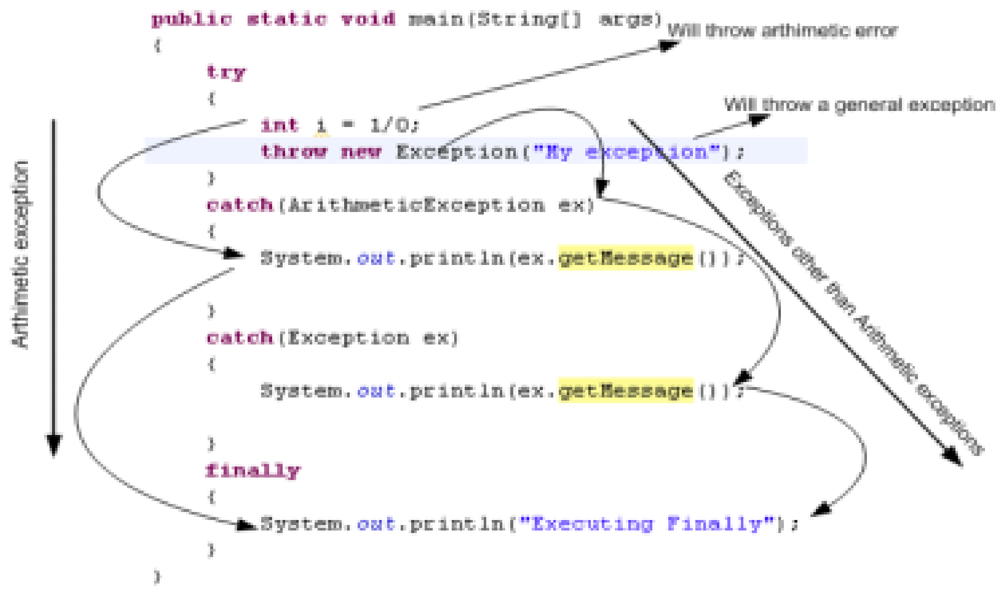
```
TwoWheeler T;
Cycle C = new Cycle();
// Moving child object to the parent
T = C; // Will work and compile fine
// Moving Parent to child
C = T // Will throw a error
// Moving Parent to child with type casting
C = (Cycle)T // Will work properly
```

(B) Define exceptions?

An exception is an abnormal condition that arises in a code sequence at run time.

(B) Can you explain in short how JAVA exception handling works?

Basically there are four important keywords which form the main pillars of exception handling: try, catch, throw and finally. Code which you want to monitor for exception is contained in the try block. If any exception occurs in the try block its sent to the catch block which can handle this error in a more rational manner. To throw an exception manually you need to call use the throw keyword. If you want to put any clean up code use the finally block. The finally block is executed irrespective if there is an error or not. Below is a small error code snippet which describes how the flow of error goes.



```
public static void main(String[] args)
{
    try
    {
        int i = 1/0;
        throw new Exception("My exception");
    }
    catch(ArithmeticException ex)
    {
        System.out.println(ex.getMessage());

    }
    catch(Exception ex)
    {
        System.out.println(ex.getMessage());

    }
    finally
    {
        System.out.println("Executing Finally");
    }
}
```

Will throw arthimetic error
Will throw a general exception
Arthimetic exception
Exceptions other than Arithmetic exceptions

Figure 1.52: - JAVA exception in action

In the try block we have two exceptions raised one (1/0) which will throw an exception error and the other will use the throw clause to throw a general exception error. Arithmetic exception will flow from try block to the arithmetic catch block and then execute the finally block. But the general exception will flow from the try block, then to the exception catch block and then execute the finally block. In short the finally block always execute. It’s the best place to put your clean up code in finally block as it executed irrespective that there is an error or not.

(B) Can you explain different exception types?

```
java.lang.Object
   |
   +--java.lang.Throwable
         |
         +--java.lang.Exception
         |     |
         |     +--java.lang.ClassNotFoundException
         |     |
         |     +--java.io.IOException
         |     |     |
         |     |     +--java.io.FileNotFoundException
         |     |
         |     +--java.lang.RuntimeException
         |           |
         |           +--java.lang.NullPointerException
         |           |
         |           +--java.lang.IndexOutOfBoundsException
         |                 |
         |                 +--java.lang.ArrayIndexOutOfBoundsException
         |
         +--java.lang.Error
               |
               +--java.lang.VirtualMachineError
                     |
                     +--java.lang.OutOfMemoryError
```

Figure 1.53 : - JAVA exception hierarchy

Above is the diagram which gives you a picture of how the exception classes’ hierarchy is. All exception types are subclasses of Throwable. Throwable is at the top of the exception class hierarchy. Immediately below Throwable are two subclasses that partition exceptions into two distinct branches. One branch is headed by Exception. This class is used for exceptional conditions that user programs should catch. This is also the class that you will subclass to create your own custom exception types. There is an important subclass of Exception, called RuntimeException. Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing.

The other branch is topped by Error, which defines exceptions that are not expected to be caught under normal circumstances by your program. Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.

(B) Can you explain checked and unchecked exceptions?

Exceptions generated from runtime are called unchecked exceptions, since it is not possible for the compiler to determine that your code will handle the exception. Exception classes that descend from RuntimeException and Error classes are unchecked exceptions. Examples for RuntimeException are illegal cast operation, inappropriate use of a null pointer, referencing an out of bounds array element. Error exception classes signal critical problems that typically cannot be handled by your application. Examples are out of memory error, stack overflow, failure of the Java VM.

Thrown exceptions are referred to as checked exceptions. The compiler will confirm at compile time that the method includes code that might throw an exception. Moreover the compiler requires the code that calls such a method to include this call within a try block, and provide an appropriate catch block to catch the exception.

Below are some unchecked exceptions:-

ArithmeticException:- Arithmetic error, such as divide-by-zero.

ArrayIndexOutOfBoundsException :- Array index is out-of-bounds.

ArrayStoreException :- Assignment to an array element of an incompatible type.

ClassCastException:- Invalid cast.

IllegalArgumentException :- Illegal argument used to invoke a method.

IllegalMonitorStateException :- Illegal monitor operation, such as waiting on an unlocked thread.

IllegalStateException :- Environment or application is in incorrect state.

IllegalThreadStateException :- Requested operation not compatible with current thread state.

IndexOutOfBoundsException :- Some type of index is out-of-bounds.

NegativeArraySizeException :- Array created with a negative size.

NullPointerException :- Invalid use of a null reference.

NumberFormatException :- Invalid conversion of a string to a numeric format.

SecurityException :- Attempt to violate security.

StringIndexOutOfBounds :- Attempt to access index outside the bounds of a string.

Below are some checked exceptions:-

ClassNotFoundException:- Class not found.

CloneNotSupportedException :- Attempt to clone an object that does not implement the Cloneable interface.

IllegalAccessException :- Access to a class is denied.

InstantiationException :- Attempt to create an object of an abstract class or interface.

InterruptedException :- One thread has been interrupted by another thread.

NoSuchFieldException :- A requested field does not exist.

NoSuchMethodException :- A requested method does not exist.

(I) Can we create our own exception class?

Just define a subclass of Exception and then you can override methods to provide custom functions.

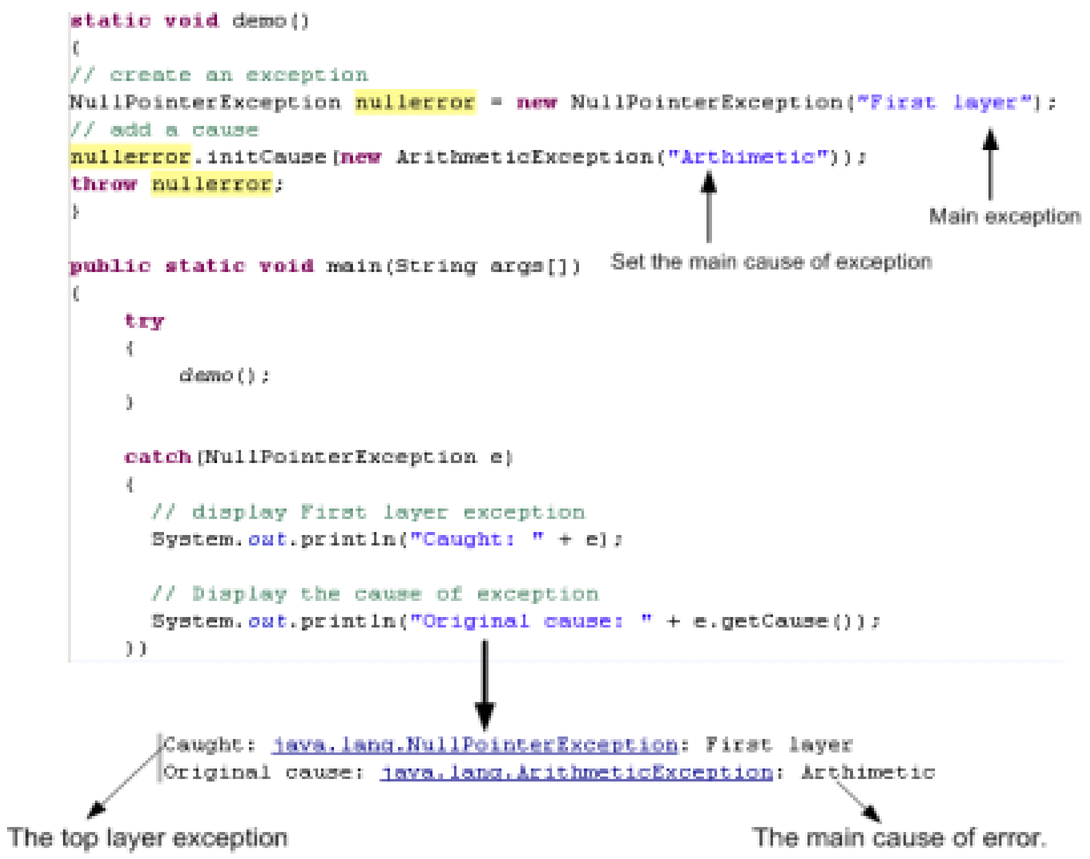
(A) What are chained exceptions?

The chained exception feature allows you to associate another exception with an exception. This second exception describes the cause of the first exception. Consider a situation in which you are getting null exception because of permission issues. You would like to know if this exception is associated with some other exception.For chained exceptions there are two constructors and two methods. The constructors are shown here:

Throwable(Throwable causeExc)

Throwable(String msg, Throwable causeExc)

In the first form, causeExc is the exception that causes the current exception. That is, causeExc is the underlying reason that an exception occurred. The second form allows you to specify a description at the same time that you specify a cause exception.



```
static void demo()
{
// create an exception
NullPointerException nullerror = new NullPointerException("First layer");
// add a cause
nullerror.initCause(new ArithmeticException("Arthimetic"));
throw nullerror;
}

public static void main(String args[])
{
    try
    {
        demo();
    }

    catch(NullPointerException e)
    {
      // display First layer exception
      System.out.println("Caught: " + e);

      // Display the cause of exception
      System.out.println("Original cause: " + e.getCause());
    }}
```

```
Caught: java.lang.NullPointerException: First layer
Original cause: java.lang.ArithmeticException: Arthimetic
```

Figure 1.54 : - Chained exception in action

Above is the code snippet which shows chained exception in action. We have defined a method by name demo. In demo we have thrown a null pointer error. But after that we set the main cause of error using initcause as arthimetic exception. You can also see the display. You can get the main cause of error using getCause method.

(B) What is serialization?

Serialization is a process by which an object instance is converted in to stream of bytes. There are many useful stuff you can do when the object instance is converted in to stream of bytes for instance you can save the object in hard disk or send it across the network.

(I) How do we implement serialization actually?

In order to implement serialization we need to use two classes from java.io package ObjectOutputStream and ObjectInputStream. ObjectOutputStream has a method called writeObject, while ObjectInputStream has a method called readObject. Using writeobject we can write and readObject can be used to read the object from the stream. Below are two code snippet which used the FileInputStream and FileOutputstream to read and write from harddisk.

```
private void SaveObject(myObject obj) throws IOException
{
    //open a FileInputStream associated with the data
    String filePath = "c:\my.obj";
    FileOutputStream fos = new FileOutputStream(filePath);
    // Associate the fileobject with the outputstream
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    // write the object to the outputstream
    oos.writeObject(obj);
    // close and clear all objects
    oos.close();
    fos.close();
    fos =null;
    oos = null;
}
private myobject getObject(string fullfilepath) throws IOException,
ClassNotFoundException
{
    myobject obj = null;
    // open the file and read the same in file inputstream
    FileInputStream fis = new FileInputStream(fullfilepath);
    //Read in the data from the object
    ObjectInputStream ois = new ObjectInputStream(fis);
    obj = (myobject)ois.readObject();
    ois.close();
    fis.close();
    return obj;
}
```

(A) What’s the use of Externalizable Interface?

When performance becomes a important criteria, then you can serialize using the java.io.Externalizable interface instead of the Serializable interface. Externalizable requires that you write the details of reading and writing the object's state to the byte stream. The ObjectOutputStream class no longer simplifies this process. You must use the methods readExternal and writeExternal method to write the object in to stream and read from stream.

(B) What is JAVAdoc utility?

Javadoc parses comments in JAVA source files and produced HTML pages for the same. Below is the syntax for the same

```
javadoc [ options ] [ packagenames ] [ sourcefiles ] [ @files ]
```

Arguments can be in any order.

Options Command-line options that is doctitle, windowtitle, header, bottom etc

Packagenames: - A series of names of packages, separated by spaces, such as java.lang java.lang.reflect java.awt. You must separately specify each package you want to document. Javadoc uses -sourcepath to look for these package names. Javadoc does not recursively traverse subpackages.

sourcefiles :- A series of source file names, separated by spaces, each of which can begin with a path and contain a wildcard such as asterisk (*). The path that precedes the source file name determines where javadoc will look for it. (Javadoc does not use -sourcepath to look for these source file names.)

@files: - One or more files that contain packagenames and sourcefiles in any order, one name per line.

(I) what are JAVAdoc doclets?

Doclets helps us to specify the content and format for Javadoc tool. By default Javadoc uses the standard doclet provided by sun. But you can supply your own customized output for Javadoc.

(I)What is Auto boxing and unboxing?

This is a newly added feature in J2SE 5.0. Auto boxing is the process in which primitive type is automatically boxed in to equivalent type wrapper. There is no need to explicitly do casting for the same. Auto boxing is vice versa of the same. Value of the boxed object is converted in to primitive data type. You can see in the below code snippet we have two samples of Auto boxing and Unboxing.

```
// this will auto box a int to a Integer object automatically

Integer objint = 100;

// this will auto unbox. That means move a integer object to a primitive data type

int i = objint;
```

Note: - Auto boxing and unboxing does prevent lot of errors. So programmer may be tempted to use this feature extensively. For instance the below code would be more efficient if primitive data type was used.

```
// the below example shows a bad use of autoboxing and unboxing

Double dbla, dblb, dblc;
```

dbla = 11;

dblb = 22;

*dblc = Math.sqrt(dbla*dbla + dblb*dblb);*

Note :- In the coming question we will be answering three question in on shot.

(I) How much subclasses you can maximum in Inheritance?

In one of my old JAVA projects I had an inheritance depth of five. Believe me I never liked that code. It's bad to have a huge inheritance depth. A maintainable inheritance depth should be maximum 5. Anything above that is horrible. There is no limit as such specified anywhere that there is some limit on the inheritance sub classing (In case you come across mail me at shiv_koirala@yahoo.com). But depending on environments you will get stack over flow error.

(I)Can you explain transient and volatile modifiers?

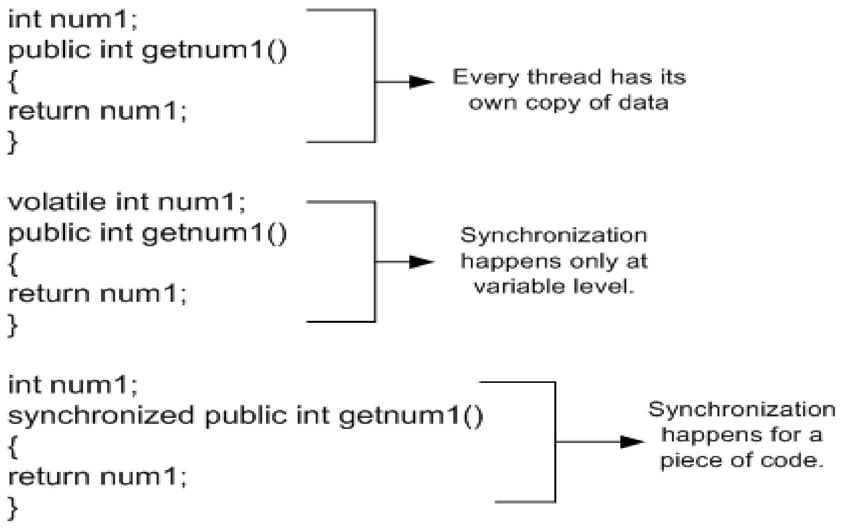

```
int num1;
public int getnum1()
{
return num1;
}

volatile int num1;
public int getnum1()
{
return num1;
}

int num1;
synchronized public int getnum1()
{
return num1;
}
```
Every thread has its own copy of data
Synchronization happens only at variable level.
Synchronization happens for a piece of code.

Figure 1.55 : - Volatile keyword in action

To understand the concept we have three variations defined above.

√ Simple variable.

- √ Variable declared as volatile.
- √ And method which is synchronized. You will understand why we are talking about synchronized methods when the fundamental is about volatile.

Before we start dealing with the above fundamentals lets try to understand the concept of main memory in java. Main memory is a memory which at any moment of time holds the correct value of variable. So when variables are updated its possible that value in main memory can be different from the value of the actual variables. In short it's possible that for a given moment of time which can be very small the main memory and the variable value can be different. Ok now that we know about the main memory fundamentals lets go ahead with the above three cases and then try to understand the concept of volatile variables.

For the first case i.e. simple variable every thread will have its own local copy. At any given moment of time it's possible that the variable data is not as same as the main memory.

For the second case because the variable is defined as volatile it is not allowed to have a local copy of a variable that is different from the value currently held in "main" memory. Effectively, a variable declared volatile must have its data synchronized across all threads, so that whenever you access or update the variable in any thread, all other threads immediately see the same value. Of course, it is likely that volatile variables have a higher access and update overhead than "plain" variables, since the reason threads can have their own copy of data is for better efficiency.

Ok now you must definitely have a question if volatile variables are for this then what's does the synchronized keyword do?. Volatile only synchronizes the value of one variable between thread memory and "main" memory, synchronized synchronizes the value of all variables between thread memory and "main" memory, and locks and releases a monitor to boot. Clearly synchronized is likely to have more overhead than volatile.