Solved Assignment/Object Oriented Technologies and Java

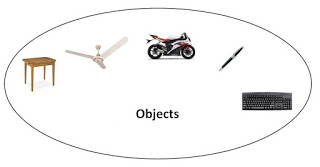
Q.1.(a) What is Object Oriented Programming? Explain basic components of Object Oriented Programming .

**A.1.(a)  Object Oriented Programming:-**

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.

 It simplifies the software development and maintenance by providing some concepts:             **Basic Component of OOP**

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

[](https://3.bp.blogspot.com/-v-_zHaHikb8/WK2WFgFcjmI/AAAAAAAAAHY/7M8hcYnLNrY18t3nJyZnucxofg27Bbe2gCLcB/s1600/p1.jpg)

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

**Collection of objects** is called class. It is a logical entity.

Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

[](https://2.bp.blogspot.com/-TkAL6491bSQ/WK2WHOB7-xI/AAAAAAAAAHc/sY8xqgpnmr87Om5W74A8qTa_-N-g2-8xQCEw/s1600/p2.gif)

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

[](https://4.bp.blogspot.com/-PaMFjHByKMQ/WK2WHYcF97I/AAAAAAAAAHg/OIUTNvzKXAsUtvzdD5CZgoM1j4Wc2VGvACEw/s1600/p3.jpg)

Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Q.1.(b) Explain abstraction and encapsulation with the help of examples?

**A.1.(b) Encapsulation** binds together the data and functions that manipulate the data, keeping it safe from interference and misuse.

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

**Real World Example:**Every time you log into your email account( GMail, Yahoo, Hotmail or official mail), you have a whole lot of processes taking place in the backend, that you have no control over. So your password, would probably be retrieved in an encyrpted form, verified and only then you are given access. You do not have any control, over how the password is verified, and this keeps it safe from misuse.

**Abstraction**is a process of hiding the implementation from the user, only the functionality is exposed here. So you are aware only of what the application does, not how it does it.

**Real World Example:**When you log into your email, compose and send a mail. Again there is a whole lot of background processing involved, verifying the recipient, sending request to the email server, sending your email. Here you are only interested in composing and clicking on the send button. What really happens when you click on the send button, is hidden from you.

Q.1.(c) Explain concept of java virtual machine?

**A.1.(c)**JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM

It is:

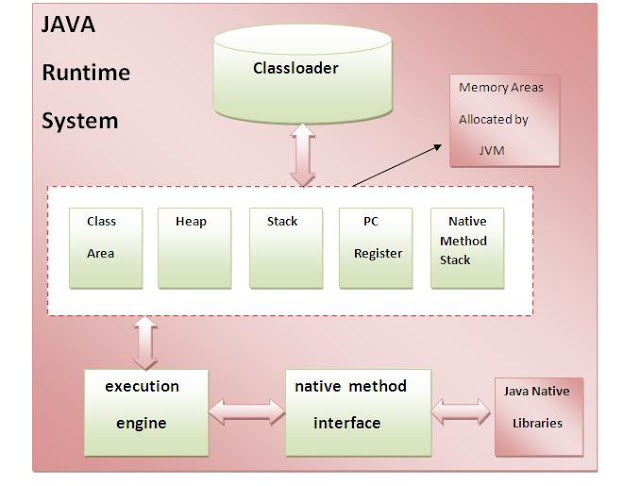
1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

What it does

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.
* **[](https://2.bp.blogspot.com/-2x89FRFgBYg/WK2Xqjurx2I/AAAAAAAAAHs/buxdrilWY4ooI2p5Eof75VaQkRVSPHISwCLcB/s1600/t1.jpg)**

**Q.2.a)** **Write a java program to demonstrate use of different operators available in java.**

**A.2.a)** Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

* Arithmetic Operators
* Relational Operators
* Bitwise Operators
* Logical Operators
* Assignment Operators
* Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator. | A + B will give 30 |
| - (Subtraction) | Subtracts right-hand operand from left-hand operand. | A - B will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator. | A \* B will give 200 |
| / (Division) | Divides left-hand operand by right-hand operand. | B / A will give 2 |
| % (Modulus) | Divides left-hand operand by right-hand operand and returns remainder. | B % A will give 0 |
| ++ (Increment) | Increases the value of operand by 1. | B++ gives 21 |
| -- (Decrement) | Decreases the value of operand by 1. | B-- gives 19 |

The Relational Operators

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == (equal to) | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != (not equal to) | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > (greater than) | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < (less than) | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= (greater than or equal to) | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= (less than or equal to) | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true |

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

-----------------

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a  = 1100 0011

The Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && (logical and) | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false |
| || (logical or) | Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. | (A || B) is true |
| ! (logical not) | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true |

The Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator. Assigns values from right side operands to left side operand. | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand. | C -= A is equivalent to C = C – A |
| \*= | Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand. | C \*= A is equivalent to C = C \* A |
| /= | Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| |= | bitwise inclusive OR and assignment operator. | C |= 2 is same as C = C | 2 |

Miscellaneous Operators

There are few other operators supported by Java Language.

Conditional Operator ( ? : )

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

variable x = (expression) ? value if true : value if false

**Q.2.(b)** **Explain followings in context of java, with the help of examples.**

**(i) Access specifiers and Inheritance**

**(ii) Application program and Applet program**

**A.2.(b) (i) Access specifiers and Inheritance**

**Access Specifier/Modifier** in any language defines the **boundary**and scope for accessing the method, variable and class etc.  
  
Java has defined four types of access specifiers. These are :

1.    Public

2.    Private

3.    Protected

4.    Default

If you do not define any access specifier/modifier than java will take “**default**” access specifier as the default for variables/methods/class.

**SYNTAX For declaring a class with access specifier :**

<access-specifier><class-keyword><class-name>

For e.g.

public class Demo

**public access specifier :**

If you declare any method/function as the public access specifier than that variable/method can be used anywhere.

**public Access Specifier Example :**

**package abc;**

**public class AccessDemo**

**{**

**public void test()**

**{**

**System.out.println("Example of public access specifier");**

**}**

**}**

**private access specifier :**

If you declare any method/variable as private than it will only accessed in the same class which declare that method/variable as private. The private members cannot access in the outside world. If any class declared as private than that class will not be inherited. :

**private Access Specifier Example :**

**class AccessDemo**

**{**

**private int x = 56;**

**public void showDemo()**

**{**

**System.out.println("The Variable value is " + x);**

**}**

**private void testDemo()**

**{**

**System.out.println("It cannot be accessed in another class");**

**}**

**}**

**public class AccessExample**

**{**

**public static void main(String[] args)**

**{**

**AccessDemo ad = new AccessDemo();**

**ad.testDemo(); // Private method cannot be used**

**ad.x = 5; // Private variable cannot be used**

**ad.showDemo(); // run properly**

**}**

**}**

**default access specifier :**

If you define any method/variable as default or not give any access specifier than the method or variable will be accessed in only that package in which they are defined.

They cannot be accessed outside the package.

**default Access Specifier Example :**

In one package we define the variable default as below code :

**package abc;**

**class AccessDemo**

**{**

**default int a = 4;**

**}**

**protected access specifier :**

It has same properties as that of private access specifier but it can access the methods/variables in the child class of parent class in which they are declared as protected.

When we want to use the private members of parent class in the child class then we declare those variables as protected.

**protected Access Specifier Example :**

class AccessDemo

 {

      protected int x = 34;

       public void showDemo()

      {

            System.out.println("The variable value is " + x);

       }

 }

 class ChildAccess extends AccessDemo

 {

       // child class which inherits

       // the properties of AccessDemo class

 }

 public class AccessExample

 {

       public static void main(String[] args)

      {

            ChildAccess ca = new ChildAccess();

            ca.showDemo(); // run properly

            ca.x = 45; // run properly

 }

 }

**(ii) Application program and Applet program**

**A.2.(b)(ii)**

**1.Applications**- Java programs that run directly on your machine.

o    Applications must have a *main()*.

o    Java applications are compiled using the **javac** command and run using the **java** command.

**2.Applets**- Java programs that can run over the Internet. The standard client/server model is used when the Applet is executed. The server stores the Java Applet, which is sent to the client machine running the browser, where the Applet is then run.

o    Applets do not require a *main()*, but in general will have a *paint()*.

o    An Applet also requires an HTML file before it can be executed.

o    Java Applets are also compiled using the **javac** command, but are are run either with a browser or with the **appletviewer**command.

Application is a Java class that has a main() method. An applet is a Java class which extends java.applet.Applet. Generally, application is a stand-alone program, normally launched from the command line, and which has unrestricted access to the host system. An applet is a program which is run in the context of an applet viewer or web browser, and which has strictly limited access to the host system. For instance, an applet can normally not read or write files on the host system whereas an application normally can. The actions of both applets and applications can be controlled by SecurityManager objects.

Applets may communicate with other applets running on the same virtual machine. If the applets are of the same class, they can communicate via shared static variables. If the applets are of different classes, then each will need a reference to the same class with static variables.

**Q.3.(a) Create array of objects in a java program and pass it as an argument in a method.**

**A.3.(a)** As we have already said, arrays are capable of storing objects also. For example, we can create an array of Strings which is a reference type variable. However, using a String as a reference type to illustrate the concept of array of objects isn't too appropriate due to the immutability of String objects. Therefore, for this purpose, we will use a class Student containing a single instance variable marks. Following is the definition of this class.

class Student {  
   int marks;  
}

An array of objects is created just like an array of primitive type data items in the following way.

Student[] studentArray = new Student[7];

For instance, the following statement throws a NullPointerException during runtime which indicates that studentArray[0] isn't yet pointing to a Student object.

studentArray[0].marks = 100;

The Student objects have to be instantiated using the constructor of the Student class and their references should be assigned to the array elements in the following way.

studentArray[0] = new Student();

In this way, we create the other Student objects also. If each of the Student objects have to be created using a different constructor, we use a statement similar to the above several times. However, in this particular case, we may use a for loop since all Student objects are created with the same default constructor.

for ( int i=0; i<studentArray.length; i++) {  
studentArray[i]=new Student();  
}

The above for loop creates seven Student objects and assigns their reference to the array elements. Now, a statement like the following would be valid.

studentArray[0].marks=100;

Enhanced for loops find a better application here as we not only get the Student object but also we are capable of modifying it.

for ( Student x : studentArray ) {  
    x.marks = s.nextInt(); // s is a Scanner object  
}

 Following illustrates this concept.

public static void main(String[] args) {  
    Student[] studentArray = new Student[7];  
    studentArray[0] = new Student();  
    studentArray[0].marks = 99;  
    System.out.println(studentArray[0].marks); // prints 99  
    modify(studentArray[0]);  
    System.out.println(studentArray[0].marks); // prints 100 and not 99  
    // code  
}

public static void modify(Student s) {  
    s.marks = 100;  
}

**3.(b) Write a java program to demonstrate different use of final variable.**

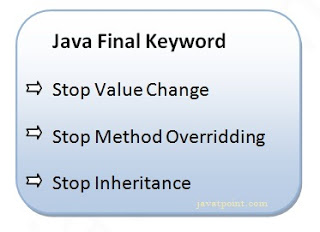
**A.3.(b)**

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

[](https://3.bp.blogspot.com/-XlfDnLZl9jM/WK2bhtbFRFI/AAAAAAAAAIA/cFBu3fZvhkwwAQ-434VLQa8Xe2rTdmQSwCLcB/s1600/j1.jpg)

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

**class** Bike9{

**final** **int** speedlimit=90;//final variable

**void** run(){

  speedlimit=400;

 }

**public** **static** **void** main(String args[]){

 Bike9 obj=**new**  Bike9();

 obj.run();

 }

**Output:Compile Time Error**

3.(c) Write a java program to create Ticket class with proper constructor, to create a railway ticket. Define a method to display ticket details.

**A.3.(c)**

public class Station {

    private int stationID;

    private String stationName;

    // Getters and Setters

}

public class Train {

    private int trainID;

    private String trainName;

    private Map<Station, Double> trainStationsWithFares;

    public Train(int ID, String trainName, Station[] stations) {

    // Initialize ID and name and create a hashmap with all stations and

    // zero fare initially for all stations.

    ....

    trainStationsWithFares = new HashMap<Station, Double>();

    for(Station s : stations) trainStationsWithFares.put(s, new Double(0.0));

    }

    // Getters and Setters (including methods to add new stations with fares and

    // update fares of existing stations

}

public class Passenger {

    private String Name;

    private int id;

    private int age;

    private static final enum { Male, Female } gender;

}

public class TicketDetails {

    private Train t;

    private Station from;

    private Station to;

    private Passenger passenger;

    // Getters and Setters

}

public class TrainTicket {

    private int ID;

    private TicketDetails ticketDetails;

    private Double fare;

    public TrainTicket(TicketDetails ticketDetails)

        throws InvalidTrainException, InvalidFromStationException,

            InvalidToStationException {

        ...

        calculateFare();

    }

    // Calculates fare based on Train and from and to Stations and taxes, etc.

    private void calculateFare() {

        this.fare = ...

    }

}

// Assuming card payment only for online reservation system for simplicity.

// Design can be modified and enhanced suitably.

public class PaymentDetails {

    private String cardNumber;

    private String cardExpirationMonth;

    private String cardExpirationYear;

    private String cardCVV;

    // Getters and Setters

}

Q.4.(a) What is polymorphism? It provides flexibility in application development? Yes or No? Justify your answer with the help of an example.

**A.4.(a) Polymorphism:-**

* Poly = many: polygon = many-sided, polystyrene = many styrenes (a), polyglot = many languages, and so on.
* Morph = change or form: morphology = study of biological form, Morpheus = the Greek god of dreams able to take any form.

So polymorphism is the ability (in programming) to present the same interface for differing underlying forms (data types).

For example, integers and floats are implicitly polymorphic since you can add, subtract, multiply and so on, irrespective of the fact that the types are different. They're rarely considered as objects in the usual term.

But, in that same way, a class like BigDecimal or Rational or Imaginary can also provide those operations, even though they operate on different data types.

The classic example is the Shape class and all the classes that can inherit from it (square, circle, dodecahedron, irregular polygon, splat and so on).

With polymorphism, each of these classes will have different underlying data. A point shape needs only two co-ordinates (assuming it's in a two-dimensional space of course). A circle needs a center and radius. A square or rectangle needs two co-ordinates for the top left and bottom right corners and (possibly) a rotation. An irregular polygon needs a series of lines.

Yes, It provides flexibility in application development. We can understand with the helpof an Example:-

Java has excellent support of polymorphism in terms of Inheritance, [method overloading](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) and [method overriding](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Method overriding allows Java to [invoke method](http://javarevisited.blogspot.sg/2012/04/how-to-invoke-method-by-name-in-java.html)based on a particular object at run-time instead of declared type while coding. To get hold of concept let's see an **example of polymorphism in Java:**

**public class** TradingSystem{

**public** String getDescription(){

**return** "electronic trading system";

   }

}

**public class** DirectMarketAccessSystem **extends** TradingSystem{

**public** String getDescription(){

**return** "direct market access system";

   }

}

**public class** CommodityTradingSystem **extends** TradingSystem{

**public** String getDescription(){

**return** "Futures trading system";

   }

}

With polymorphism, each of these classes will have different underlying data. A point shape needs only two co-ordinates (assuming it's in a two-dimensional space of course). A circle needs a center and radius. A square or rectangle needs two co-ordinates for the top left and bottom right corners and (possibly) a rotation. An irregular polygon needs a series of lines.

By making the class responsible for its code as well as its data, you can achieve polymorphism. In this example, every class would have its own Draw() function and the client code could simply do:

shape.Draw()

Q.4.(b) Explain the need of package in Java. Explain accessibility rules for package. Also explain how members of a package are imported. Write java program to create your own package for finding area of different shapes.

**A.4.(b)  Package:-**

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

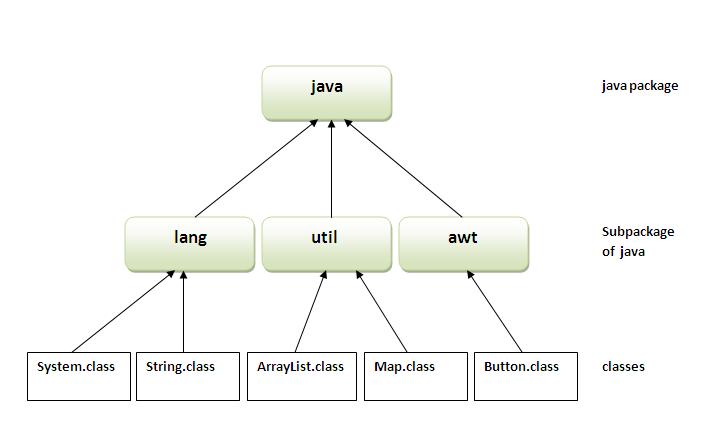
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



**Simple example of java package**

The package keyword is used to create a package in java.

//save as Simple.java

package mypack;

public class Simple{

 public static void main(String args[]){

    System.out.println("Welcome to package");

   }

}

**Program for finding area of different shapes.**

class FindLargestShape

{

    public static void main(String arg[])

    {

        Rectangle r = new Rectangle(10, 4);

        Square s = new Square(7);

        Circle c = new Circle(3.5);

        System.out.println("Rectangle Area : " + r.getArea());

        System.out.println("Square Area : " + s.getArea());

        System.out.println("Circle Area : " + c.getArea());

        System.out.println();

        if ((r.getArea() > c.getArea()) && (r.getArea() > s.getArea()))

        {

            System.out.println("Rectangle has the largest area.");

        }

        else if( s.getArea() > c.getArea() )

        {

            System.out.println("Square has the largest area.");

        }

        else

        {

            System.out.println("Circle has the largest area.");

        }

    }

}

class Rectangle

{

    double length;

    double breadth;

    Rectangle(double length, double breadth)

    {

        this.length = length;

        this.breadth = breadth;

    }

    double getArea()

    {

        return length \* breadth;

    }

}

class Square

{

    double side;

    Square(double side)

    {

        this.side = side;

    }

    double getArea()

    {

        return side \* side;

    }

}

class Circle

{

    double radius;

    Circle(double radius)

    {

        this.radius = radius;

    }

    double getArea()

    {

        return (22.0/7.0) \* radius \* radius;

    }

}

OUTPUT

Rectangle Area : 40.0

Square Area : 49.0

Circle Area : 38.5

Square has the largest area.

Q.5.(a) What is abstract class? Explain need of abstract class with the help of an example.

**A.5.(a)** A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Before learning java abstract class, let's understand the abstraction in java first.

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

**Abstract class in Java**

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

**Example abstract class**

abstract class A{}

**Abstract method**

A method that is declared as abstract and does not have implementation is known as abstract method.

**Example abstract method**

abstract void printStatus();   //no body and abstract

**Example of abstract class that has abstract method**

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

abstract class Bike{

  abstract void run();

}

class Honda4 extends Bike{

void run(){System.out.println("running safely..");}

public static void main(String args[]){

 Bike obj = new Honda4();

 obj.run();

}

}

**OUTPUT OF THE PROGRAM**

**running safely..**

**Q.5.(b) What is an exception? Explain haw an exceptions are handled in Java.**

**Write a java program to handle different arithmetic exceptions while managing a medical store billing process.**

**A.5.(b)**

**A**n exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

* A user has entered an invalid data.
* A file that needs to be opened cannot be found.
* A network connection has been lost in the middle of communications or the JVM has run out of memory.
* Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

* **Checked exceptions** − A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a *FileNotFoundException* occurs, and the compiler prompts the programmer to handle the exception.

The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an exception handler.  
  
  
import java.util.Scanner;  
  
class Billing{  
    String name;  
   int Price,total,Quantity,d;  
    public void input(){  
        Scanner rd=new Scanner(System.in);  
        System.out.println("enter the name of medicine");  
        name=rd.next();  
        System.out.println("enter the price of medicine");  
        Price=rd.nextInt();  
        System.out.println("enter the Quantity of medicine");  
        Quantity=rd.nextInt();  
        System.out.println("enter the Discount in percent");  
        d=rd.nextInt();  
    }  
   
 void process()  
    {  
     
      try{  
        
        total=Price\*Quantity;  
        total=total/d;  
        output();  
        }  
        catch(ArithmeticException e){  
          System.out.println("Arithmetic Exception");  
      }  
        
    }  
    void output(){  
        System.out.println("Your medicine "+name);  
        System.out.println("Price "+Price);  
        System.out.println("Quantity "+Quantity);  
        System.out.println("Discount "+d+"%");  
        System.out.println("total ammount "+total);  
    }  
    
}  
  
public class ProgramRun {  
  
 public static void main(String[] args) {  
     Billing obj=new Billing();  
          
     obj.input();  
     
     obj.process();  
    }  
}

Q.6.(a) What is I/O stream in java? Write a program in java to create a file and count the number of words in it.

**A.6.(a)**

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

* **InPutStream** − The InputStream is used to read data from a source.
* **OutPutStream** − The OutputStream is used for writing data to a destination.



Java provides strong but flexible support for I/O related to files and networks but this tutorial covers very basic functionality related to streams and I/O. We will see the most commonly used examples one by one −

**Byte Streams**

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

**Character Streams**

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

program in java to create a file and count the number of words in it.

import java.io.FileReader;

import java.io.BufferedReader;

import java.util.StringTokenizer;

class NumberWords {

public static void main(String[] args) throws Exception {

FileReader fr = new FileReader("c:/test.txt");

BufferedReader br = new BufferedReader(fr);

String line = "", str = "";

int a = 0;

int b = 0;

while ((line = br.readLine()) != null) {

str += line + " ";

b++;

}

System.out.println("Totally " + b + " lines");

System.out.println(str);

StringTokenizer st = new StringTokenizer(str);

while (st.hasMoreTokens()) {

String s = st.nextToken();

a++;

}

System.out.println("File has " + a + " words are in the file");

}

}

**OUTPUT:**

Totally 6 lines

hi hello              how are you?    when will you come here?

File has 10 words are in the file.

**Q.6.(b)Create an Applet program to display details of a quiz competition. Insert a background image in this program.**

**A.6(b)**

 import java.applet.\*;

import java.awt.\*;

import java.net.\*;

import java.io.IOException.\*;

public class BackgroundApplet extends Applet {

     Image backGround;

     public void init() {

          // set the size of the applet to the size of the background image.

          // Resizing the applet may cause distortion of the image.

          setSize(300, 300);

          // Set the image name to the background you want. Assumes the image

          // is in the same directory as the class file is

          backGround = getImage(getCodeBase(), "save.GIF");

          BackGroundPanel bgp = new BackGroundPanel();

          bgp.setLayout(new FlowLayout());

          bgp.setBackGroundImage(backGround);

          // Add the components you want in the Applet to the Panel

   bgp.add(new TextField("The competition is open to Macquarie University undergraduate  and co-curricular students only"));

bgp.add(new TextField("Only photos entered in the correct session timeframe for the unit will be accepted"));

bgp.add(new TextField("Only photos entered through the online entry form will be accepted in the competition.");

bgp.add(new TextField("Entries must be taken during your PACE activity in one of the sessions from March 2016 - March 2017.");

          // set the layout of the applet to Border Layout

          setLayout(new BorderLayout());

          // now adding the panel, adds to the center

          // (by default in Border Layout) of the applet

          add(bgp);

     }

}

class BackGroundPanel extends Panel {

     Image backGround;

     BackGroundPanel() {

          super();

     }

     public void paint(Graphics g) {

          // get the size of this panel (which is the size of the applet),

          // and draw the image

          g.drawImage(getBackGroundImage(), 0, 0,

              (int)getBounds().getWidth(), (int)getBounds().getHeight(), this);

     }

     public void setBackGroundImage(Image backGround) {

          this.backGround = backGround;

     }

     private Image getBackGroundImage() {

          return backGround;

     }

}

**Q.6.(c) Write and explain different constructors of String class.**

**A.6.(c)**

Constructor in Java

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.**Constructor in java** is a *special type of method* that is used to initialize the object.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

**String** is nothing but a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java, string is an immutable object which means it is constant and can cannot be changed once it has been created. In this tutorial we will learn about String class and String methods in detail along with many other Java String tutorials.

Creating a String

There are two ways to create a String in Java

1. String literal
2. Using new keyword

**String literal**

In java, Strings can be created like this: Assigning a String literal to a String instance:

String str1 = "Welcome";

String str2 = "Welcome";

**The problem with this approach**: As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword above. The compiler does that task for us it creates a string object having the string literal (that we have provided , in this case it is “Welcome”) and assigns it to the provided string instances.

**Using New Keyword**

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this.

String str1 = new String("Welcome");

String str2 = new String("Welcome");

**Class constructors: -**

1. String()

This initializes a newly created String object so that it represents an empty character sequence.

2

String(byte[] bytes)

This constructs a new String by decoding the specified array of bytes using the platform's default char set.

3

String(byte[] bytes, Charset charset)

This constructs a new String by decoding the specified array of bytes using the specified charset.

4

String(byte[] bytes, int offset, int length)

This constructs a new String by decoding the specified subarray of bytes using the platform's default charset

5

String(byte[] bytes, int offset, int length, Charset charset)

This constructs a new String by decoding the specified subarray of bytes using the specified charset.

6

String(byte[] bytes, int offset, int length, String charsetName)

This constructs a new String by decoding the specified subarray of bytes using the specified charset.

7

String(byte[] bytes, String charsetName)

This constructs a new String by decoding the specified array of bytes using the specified charset.

8

String(char[] value)

This allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

9

String(char[] value, int offset, int count)

This allocates a new String that contains characters from a subarray of the character array argument.

10

String(int[] codePoints, int offset, int count)

This allocates a new String that contains characters from a subarray of the Unicode code point array argument.

11

String(String original)

This initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

12

String(StringBuffer buffer)

This allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

13

String(StringBuilder builder)

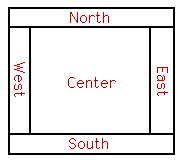
Q.7.(a) What is layout manager? Explain different layouts available in java for GUI programming. What is default layout of an Applet? Explain how to set the layout of an applet.

**A.7.(a)**

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

**BorderLayout**

A BorderLayout places one component in the center of a container. The central component is surrounded by up to four other components that border it to the "North", "South", "East", and "West", as shown in the diagram at the right. Each of the four bordering components is optional. The layout manager first allocates space to the bordering components. Any space that is left over goes to the center component.

**Flow Layout**

A FlowLayout simply lines up its components without trying to be particularly neat about it. After laying out as many items as will fit in a row across the container, it will move on to the next row. The components in a given row can be either left-aligned, right-aligned, or centered, and there can be horizontal and vertical gaps. If the default constructor, "new FlowLayout()" is used, then the components on each row will be centered and the horizontal and vertical gaps will be five pixels. The constructor

FlowLayout(int align, int hgap, int vgap)

**Grid Layout**

A GridLayout lays out components in a grid of equal sized rectangles. The illustration shows how the components would be arranged in a grid layout with 3 rows and 2 columns. If a container uses a GridLayout, the appropriate add method takes a single parameter of type Component (for example: add(myButton)). Components are added to the grid in the order shown; that is, each row is filled from left to right before going on the next row.

The constructor for a GridLayout with R rows and C columns takes the form GridLayout(R,C). If you want to leave horizontal gaps of H pixels between columns and vertical gaps of V pixels between rows, use GridLayout(R,C,H,V) instead.

**GridBagLayout**

A GridBagLayout is similar to a GridLayout in that the container is broken down into rows and columns of rectangles. However, a GridBagLayout is much more sophisticated because the rows do not all have to be of the same height, the columns do not all have to be of the same width, and a component in the container can spread over several rows and several columns. There is a separate class, GridBagConstraints, that is used to specify the position of a component, the number of rows and columns that it occupies, and several additional properties of the component.

Using a GridBagLayout is rather complicated. I will not explain it here; if you are interested, you should consult a Java reference.

**CardLayout**

CardLayouts differ from other layout managers in that in a container that uses a CardLayout, only one of its components is visible at any given time. Think of the components as a set of "cards". Only one card is visible at a time, but you can flip from one card to another. Methods are provided in the CardLayout class for flipping to the first card, to the last card, and to the next card in the deck. A name can be specified for each card as it is added to the container, and there is a method in the CardLayout class for flipping directly to the card with a specified name. (The container object has to be passed as a parameter to each of these methods.)

import java.awt.\*;  
import java.applet.\*;

public class LayoutExample extends Applet   
    {  
     Button okButton1;  
     Button okButton2;  
     Button okButton3;  
     Button okButton4;  
     Button okButton5;

     public void init()  
     {  
  // sets the LayoutManager to BorderLayout  
        setLayout(new BorderLayout());  
        okButton1 = new Button("Centered Button");  
          okButton2 = new Button("Cold North");  
          okButton3 = new Button("Go West");  
          okButton4 = new Button("At East");  
          okButton5 = new Button("Hot South");  
  // always says where the component should be placed when adding  
  // Options are center,East,West,Nort and South  
        add(okButton1,"Center");  
          add(okButton2,"North");  
          add(okButton3,"West");  
          add(okButton4,"East");  
          add(okButton5,"South");  
        }  
    }

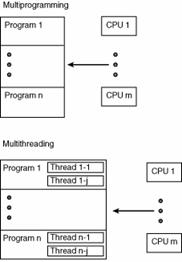
Q.7.(b)What is multithreading? Explain how threads are synchronized in java.

**A.7.(b)  Multithreading and Thread Synchronization**

*Multithreaded* programs are similar to the single-threaded programs that you have been studying. They differ only in the fact that they support more than one *concurrent* thread of execution-that is, they are able to simultaneously execute multiple sequences of instructions. Each instruction sequence has its own unique flow of control that is independent of all others. These independently executed instruction sequences are known as *threads*.

If your computer has only a single CPU, you might be wondering how it can execute more than one thread at the same time. In single-processor systems, only a single thread of execution occurs at a given instant. The CPU quickly switches back and forth between several threads to create the illusion that the threads are executing at the same time. Single-processor systems support *logical concurrency*, not *physical concurrency*. Logical concurrency is the characteristic exhibited when multiple threads execute with separate, independent flows of control. On multiprocessor systems, several threads do, in fact, execute at the same time, and physical concurrency is achieved. The important feature of multithreaded programs is that they support logical concurrency, not whether physical concurrency is actually achieved.

Many programming languages support *multiprogramming*. Multiprogramming is the logically concurrent execution of multiple programs. For example, a program can request that the operating system execute programs A, B, and C by having it spawn a separate process for each program. These programs can run in parallel, depending upon the multiprogramming features supported by the underlying operating system. Multithreading differs from multiprogramming in that multithreading provides concurrency within the context of a single process and multiprogramming provides concurrency between processes. Threads are not complete processes in and of themselves. They are a separate flow of control that occurs within a process. Figure illustrates the difference between multithreading and multiprogramming.



Q.7.(c)Explain the need of JDBC? Explain steps involved in connecting a databases using JDBC.

**A.7.(c)**JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

* Making a connection to a database.
* Creating SQL or MySQL statements.
* Executing SQL or MySQL queries in the database.
* Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

* Java Applications
* Java Applets
* Java Servlets
* Java ServerPages (JSPs)
* Enterprise JavaBeans (EJBs).
* All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

5 Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

* Register the driver class
* Creating connection
* Creating statement
* Executing queries
* Closing connection

**1) Register the driver class**

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

**Syntax of** **forName() method**

public static void forName(String className)throws ClassNotFoundException

**Example to register the OracleDriver class**

Class.forName("oracle.jdbc.driver.OracleDriver");

**2)Create the connection object**

The getConnection() method of DriverManager class is used to establish connection with the database.

**Syntax of** **getConnection() method**

1) public static Connection getConnection(String url)throws SQLException

2) public static Connection getConnection(String url,String name,String password)

throws SQLException

**Example to establish connection with the Oracle database**

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

**3) Create the Statement object**

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

**Syntax of createStatement() method**

public Statement createStatement()throws SQLException

**Example to create the statement object**

Statement stmt=con.createStatement();

**4) Execute the query**

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

**Syntax of executeQuery() method**

public ResultSet executeQuery(String sql)throws SQLException

**Example to execute query**

ResultSet rs=stmt.executeQuery("select \* from emp");

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

**Syntax of close() method**

public void close()throws SQLException

**Example to close connection**

con.close();

Q.8.(a) What is socket? Explain stream sockets and datagram sockets.

**A.8.(a) Definition:** A socket is one end-point of a two-way communication link between two programs running on the network.

A server application normally listens to a specific port waiting for connection requests from a client. When a connection request arrives, the client and the server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a *socket* to it. The client talks to the server by writing to the socket and gets information from the server by reading from it. Similarly, the server gets a new local port number (it needs a new port number so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it.

The client and the server must agree on a protocol--that is, they must agree on the language of the information transferred back and forth through the socket.

**Stream Socket:**

* A dedicated point-to-point channel between a client and server.
* Use TCP (Transmission Control Protocol) for data transmission.
* Lossless and reliable.
* Sent and received in the same order.

**Datagram Socket:**

* No dedicated point-to-point channel between a client and server.
* Use UDP (User Datagram Protocol) for data transmission.
* May lose data and not 100% reliable.
* Data may not received in the same order as sent.

Q.8.(b)What is RMI? Explain RMI architecture.

**A.8.(b)**

**RMI (Remote Method Invocation)**

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

Q.8.(c) What is servlet ? Explain servlet life cycle.

**A.8.(c)** Servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

* The servlet is initialized by calling the **init ()** method.
* The servlet calls **service()** method to process a client's request.
* The servlet is terminated by calling the **destroy()** method.
* Finally, servlet is garbage collected by the garbage collector of the JVM.
* **The init() method :**

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

* **The service() method :**

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

* The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

public void destroy() {

// Finalization code...

}

Architecture Digram:

The following figure depicts a typical servlet life-cycle scenario.

* + First the HTTP requests coming to the server are delegated to the servlet container.
  + The servlet container loads the servlet before invoking the service() method.
  + Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.
  + 