GROUP – B

1. What do you mean by object oriented programming ? How it differs from conventional procedural Programming ?[WBUT 2008, 2010]

Ans:

Object oriented programming(OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class(an object) is run, the code will be able to accidentally access other program data. The characteristics of data hiding provides greater system security.

- The definition of a class is reuseable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).

Procedural programming

It is divided into smaller parts known as functions.
Here, importance is not given to **data** but to functions as well as **sequence** of actions to be done.
It follows top down approach.
In procedural programming, data can move freely from one function to another.
Most functions uses global data for sharing that can be accessed freely from one function to another.
It does not have the data hiding property, so data is less secure.
Overloading is not possible in procedural programming.
Examples are C, Visual Basic, FORTAN and Pascal.

Object oriented programming

It is divided into parts called objects.
Here, importance is given to the data rather than procedure or function because it works as a real world.
It follows bottom up approach.
In OOP, objects can move and communicate with each other through member functions.
In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
OOP provides Data Hiding so provides **more security**.
In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Example of OOP are : C++, JAVA, VB.NET, C#.NET.

2. Explain OOPs properties of Java. "Java is a platform independent language" — Explain with the help of JVM.[WBUT 2009]

Ans.

The properties of OOP are given as follows:

- Encapsulation:

It is a process of binding or wrapping the data and the codes that operates on the data into a single entity. This keeps the data safe from outside interface and misuse. One way to think about encapsulation is as a protective wrapper that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper.

- Abstraction:

Abstraction is way of converting real world objects in terms of class. For example creating a class Vehicle and injecting properties into it. E.g

```
public class Vehicle {
public String colour;
public String model;
}
```

- Inheritance:

Inheritance is the property which allows a Child class to inherit some properties from its parent class. In Java this is achieved by using extends keyword. Only properties with access modifier public and protected can be accessed in child class.

```
public class Parent{
public String parentName;
public int parentage;
public String familyName;
}
public class Child extends Parent{
public String childName;
public int childAge;
public void printMyName(){
System.out.println(" My name is "+ chidName+" " +familyName)
}
}
```

In above example the child has inherit its family name from the parent class just by inheriting the class.

- Polymorphism:

Polymorphism gives us the ultimate flexibility in extensibility. The abiltiy to define more than one function with the same name is called Polymorphism. In java,c++ there are two type of polymorphism: compile time polymorphism (overloading) and runtime polymorphism (overriding).

Programs written in Java are interpreted by Java itself, which is installed a platform. The language is platform-independent because programs written in it will work on any computer that has Java installed on it, regardless of operating system. Java programs run in a "virtual machine" (sometimes called the Java Runtime Environment [JRE]. The "virtual machine" acts like an operating system unto itself, so your program always acts the same, regardless of the computer platform it is using.

3. What are the types of inheritances in JAVA? Give example. What is the use of super() keyword in JAVA? [WBUT 2009]

Ans.

There are various types of inheritances in JAVA:

- Single inheritance:

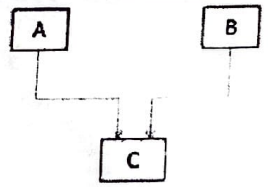
When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.

```
Class A
{
  public void methodA()
  {
   System.out.println("Base class method");
  }
}
```

```
Class B extends A
{
  public void methodB()
  {
   System.out.println("Child class method");
  }
  public static void main(String args[])
  {
   B obj = new B();
   obj.methodA(); //calling super class method
   obj.methodB(); //calling local method
  }
}
```

- Multiple inheritance:

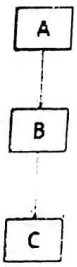
"Multiple Inheritance" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



(b) Multiple Inheritance

- Multilevel inheritance:

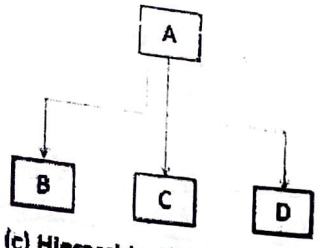
Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



(d) Multilevel Inheritance

- Hierarchial inheritance:

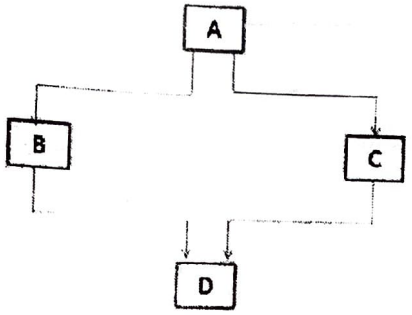
In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.



(c) Hierarchical Inheritance

- Hybrid inheritance:

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple inheritance**. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be.



(e) Hybrid Inheritance

If your method overrides one of its superclass's methods, you can invoke the overridden method through the use of the keyword super. You can also use super to refer to a hidden field (although hiding fields is discouraged). Consider this class, Superclass:

```
public class Superclass {

    public void printMethod() {
        System.out.println("Printed in Superclass.");
    }
}
```

Here is a subclass, called Subclass, that overrides printMethod():

```
public class Subclass extends Superclass {

    // overrides printMethod in Superclass
    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass");
    }
    public static void main(String[] args) {
        Subclass s = new Subclass();
        s.printMethod();
    }
```

```
}
```

4. What is abstract class? Explain with a short program to illustrate this. [WBUT 2009]

Ans.

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways. The GraphicObject class can look something like this:

```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

Each nonabstract subclass of GraphicObject, such as Circle and Rectangle, must provide implementations for the draw and resize methods:

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

5. What is garbage collection? What is static variable? Give example of both. [WBUT 2010]

Ans.

JAVA has a built in garbage collection facility. The heap is the area of memory used for dynamic allocation. In most configurations the operating system allocates the heap in advance to be managed by the JVM while the program is running. This has a couple of important ramifications:

- Object creation is faster because global synchronization with the operating system is not needed for every single object. An allocation simply claims some portion of a memory array and moves the offset pointer forward. The next allocation starts at this offset and claims the next portion of the array.
- When an object is no longer used, the garbage collector reclaims the underlying memory and reuses it for future object allocation. This means there is no explicit deletion and no memory is given back to the operating system.

Static variables are referenced by their classes. This fact makes them de facto GC roots. Classes themselves can be garbage-collected, which would remove all referenced static variables. This is of special importance when we use application servers, OSGi containers or class loaders in general.

6."Objects are passed by reference in JAVA"- Explain with example. [WBUT 2011]

Ans.

Everything in Java is pass-by-value. There is no such thing as "pass-by-reference" in Java. The key to understanding this is that something like

Dog myDog;

is *not* a Dog; it's actually a *pointer* to a Dog.

7.What is the difference between 'default and public' modifier and 'protected and private' access modifier?[WBUT 2011]

Ans.

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public

Private Access Modifier - private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class if public getter methods are present in the class.

Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.

Public Access Modifier - public:

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However if the public class we are trying to access is in a different package, then the public class still need to be imported.

Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Protected Access Modifier - protected:

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

8.a) Write a program to convert given number of days into month and days.[WBUT 2013]

Ans.

```
class DayMonthDemo
{
public static void main(String args[])
{
int num = Integer.parseInt(args[0]);
int days = num%30;
int month = num/30;
System.out.println(num+" days = "+month+" Month and "+days+" days"); }}
```

b) How are this and super used?[WBUT 2013]

Ans.

Within an instance method or a constructor, this is a reference to the *current object* — the object whose method or constructor is being called. You can refer to any member of the current object from within an instance method or a constructor by using this.

Using this with a Field

The most common reason for using the this keyword is because a field is shadowed by a method or constructor parameter.

For example, the Point class was written like this

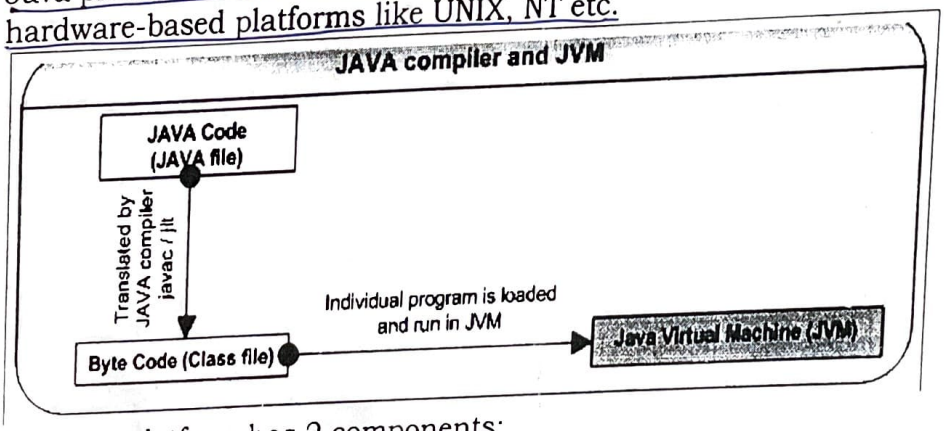
```
public class Point {
    public int x = 0;
    public int y = 0;

    //constructor
    public Point(int a, int b) {
        x = a;
        y = b;
    }
}
```

c) What is the main difference between JAVA platforms and other platforms?[WBUT 2013]

Ans.

Java platform is a software-only platform, which runs on top of other hardware-based platforms like UNIX, NT etc.



The Java platform has 2 components:

Java Virtual Machine (JVM) – 'JVM' is a software that can be ported onto various hardware platforms. Byte codes are the machine language of the JVM.

9.Explain inheritance, encapsulation concept. [WBUT 2013]

Ans.

Refer to ques 2 of Group B.

GROUP –C

1. Write a program in JAVA to input a number and check whether it is prime or not.[WBUT 2008]

Ans.

```
import java.util.*;
class Prime
{
        public static void main(String args[])
        {
                int n, i, res;
                boolean flag=true;
                Scanner scan= new Scanner(System.in);
                System.out.println("Please Enter a No.");
                n=scan.nextInt();
                for(i=2;i<=n/2;i++)
                {
                        res=n%i;
                        if(res==0)
                        {
                                flag=false;
                                break;
                        }
                }
                if(flag)
                        System.out.println(n + " is Prime Number");
                else
                        System.out.println(n + " is not Prime Number");
        }
}
```

2.a) Describe the structure of typical Java program.[WBUT 2008]

Ans.

A Typical Java class would have the following.

Package statement
import statements
Class comments (Optional)
Class Declaration {
Instance variable declarations

Constructor declaration
Method declarations }

b) Write a java program to find out the sum of harmonic series :
1 + ½ + 1/3 + ... up to nth term , for any value of n.[WBUT 2008]
Ans.

```
class HarmonicSeries
{
public static void main(String args[])
{
int num = Integer.parseInt(args[0]);
double result = 0.0;
while(num > 0){
result = result + (double) 1 / num;
num--; }
System.out.println("Output of Harmonic Series is "+result); } }
```

3.a)Briefly discuss the features of JAVA language.[WBUT 2009]
Ans.

Simple :

- Java is Easy to write and more readable and eye catching.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drew from C++ thus making Java learning simpler.

Secure :

- Java program cannot harm other system thus making it secure.
- Java provides a secure means of creating Internet applications.
- Java provides secure way to access web applications.

Portable :

- Java programs can execute in any environment for which there is a Java run-time system.(JVM)
- Java programs can be run on any platform (Linux,Window,Mac)
- Java programs can be transferred over world wide web (e.g applets)

Object-oriented :

- Java programming is object-oriented programming language.
- Like C++ java provides most of the object oriented features.
- Java is pure OOP. Language. (while C++ is semi object oriented)

Robust :

- Java encourages error-free programming by being strictly typed and performing run-time checks.

Multithreaded :

- Java provides integrated support for multithreaded programming.

b) Explain the each term of the following statement :

Public static void main (String args [ ])[WBUT 2008, 2009]

Ans.

The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member can be accessed by code outside the class in which it is declared.In this case, main( ) must be declared as public, since it must be called by code outside of its class when the program is started.

The keyword **static** allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java interpreter before any objects are made.

The keyword **void** simply tells the compiler that main( ) does not return a value. As you will see, methods may also return values.
As stated, **main**( ) is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, Main is different from main. It is important to understand that the Java compiler will compile classes that do not contain a main( ) method. But the Java interpreter has no way to run these classes. So, if you had typed Main instead of main, the compiler would still compile your program. However, the Java interpreter would report an error because it would be unable to find the main( ) method.
Any information that you need to pass to a method is received by variables specified within the set of parentheses that follow the name of the method. These variables are called parameters. If there are no parameters required for a given method, you still need to include the empty parentheses. In main( ), there is only one parameter, albeit a complicated one.

String args[ ] declares a parameter named args, which is an array of instances of the class String. Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

c) Define different types of inheritance with example.[WBUT 2009]

Ans.

Refer to ques 3 of Group B.

4.Explain the usage of 'this' and 'super' keywords in Java program.[WBUT 2009]

Ans.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

Refer to ques b of group b.

5. a) What are the features of java language ? Describe.[WBUT 2010]

Ans.

Simple :

- Java is Easy to write and more readable and eye catching.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drew from C++ thus making Java learning simpler.

Secure :

- Java program cannot harm other system thus making it secure.
- Java provides a secure means of creating Internet applications.
- Java provides secure way to access web applications.

Portable :

- Java programs can execute in any environment for which there is a Java run-time system.(JVM)
- Java programs can be run on any platform (Linux,Window,Mac)
- Java programs can be transferred over world wide web (e.g applets)

Object-oriented :

- Java programming is object-oriented programming language.
- Like C++ java provides most of the object oriented features.
- Java is pure OOP. Language. (while C++ is semi object oriented)

Robust :

- Java encourages error-free programming by being strictly typed and performing run-time checks.

Multithreaded :

- Java provides integrated support for multithreaded programming.

b) What do you mean by command line argument ?[WBUT 2010]

Ans.

Public static void main(String args[])

String args[ ] declares a parameter named args, which is an array of instances of the class String. Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

c) Explain operators with reference to java. [WBUT 2010]

Ans.

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:

Assume integer variable A holds 10 and variable B holds 20, then:

Show Examples

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:

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	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.
<	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.

>=	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.
<=	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 following table lists the bitwise operators:

Assume integer variable A holds 60 and variable B holds 13 then:

Show Examples

Ads by OffersWizardAd Options

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which

		is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes	(A || B) is true.

true.

!	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:

There are following assignment operators supported by Java language:

Show Examples

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

Misc 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

6.a) What is the significance of final keyword used as a modifier before a variable? How it is used for a method or a class?[WBUT 2011]

Ans.

Final is a keyword or reserved word in java and can be applied to member variables, methods, class and local variables in Java. Once you make a reference final you are not allowed to change that reference and compiler will verify this and raise compilation error if you try to re-initialized final variables in java.

Any variable either member variable or local variable (declared inside method or block) modified by final keyword is called final variable. Final variables are often declare with static keyword in java and treated as constant. Here is an example of final variable in Java

```
public static final String LOAN = "loan";

LOAN = new String("loan") //invalid compilation error
```

Final variables are by default read-only.

b) Explain about this keyword. Explain with a suitable example.[WBUT 2011]

Ans.

```
class ThisDemo1
{
        int a = 0;
        int b = 0;
        ThisDemo1(int x, int y)
        {
        this.a = x;
        this.b = y;
        }
        public static void main(String [] args)
        {
        ThisDemo1 td = new ThisDemo1(10,12);
        ThisDemo1 td1 = new ThisDemo1(100,23);
        System.out.println(td.a); // prints 10
        System.out.println(td.B)/>; // prints 12
        System.out.println(td1.a);// prints 100
        System.out.println(td1.B)/>;// prints 23
        }
}
```

c) What is the difference between private and static modifiers in a class?[WBUT 2011]

Ans.

Static modifier is used to create variables and methods that will exist independently of any instance created for the class. Static members exists before any instance of the class is created.
Also there will be only one copy of the static member.

To call a static method displayRuns() of the class named Cricket we write

Cricket.displayRuns();

Class name is used to invoke the static method as static member does not depend on any instance of the class.

private static method means you can not invoke the method from outside the class as the method is private to the class.

7.a) What is a wrapper class? Illustrate through example. Is string a wrapper class? [WBUT 2011]

Ans.

Wrapper classes are used to convert any data type into an object. A wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used. Wrapper classes include methods to unwrap the object and give back the data type. It can be compared with a chocolate. The manufacturer wraps the chocolate with some foil or paper to prevent from pollution. The user takes the chocolate, removes and throws the wrapper and eats it.

```
int k = 100;
Integer it1 = new Integer(k);
```

The **int** data type **k** is converted into an object, **it1** using **Integer** class. The **it1** object can be used in Java programming wherever **k** is required an object.

The following code can be used to unwrap (getting back **int** from **Integer** object) the object **it1**.

```
int m = it1.intValue();
System.out.println(m*m); // prints 10000
```

intValue() is a method of **Integer** class that returns an **int** data type.

Wrapper classes are used to "wrap" the primitives data types into objects so that they can be included in the activities which are reseved for the objects. String class wraps the string literals to an object.

a *(primitive) wrapper class* in Java is one of those eight classes that wrap a (=one) primitive value. String *wraps* a char[] so according to this it is *not* a (primitive) wrapper class.

b) What is abstract base class? What is concrete derive classes? Can there be an abstract class with no abstract methods in it?[WBUT 2011]

Ans.

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

```
public abstract class GraphicObject {
  // declare fields
  // declare nonabstract methods
  abstract void draw();
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

A concrete derived class has no virtual functions. It provides functions for its inherited pure virtual functions. This is to say, it provides all missing functionalities of the abstract class.

The derived class that implements the missing functionality of an abstract class is the concrete derived class.

Declaring a class abstract only means that you don't allow it to be instantiated on its own.

Declaring a method abstract means that subclasses have to provide an implementation for that method.

The two are separate concepts, but obviously you can't have an abstract method in a non-abstract class. You can even have abstract classes with final methods but never the other way around.

c) Does JAVA support static or dynamic binding? Explain.[WBUT 2011]

Ans.

1) Static binding in Java occurs during Compile time while Dynamic binding occurs during Runtime.

2) private, final and static methods and variables uses static binding and bonded by compiler while virtual methods are bonded during runtime based upon runtime object.

3) Static binding uses Type(Class in Java) information for binding while Dynamic binding uses Object to resolve binding.

4) Overloaded methods are bonded using static binding while overridden methods are bonded using dynamic binding at runtime. Here is an example which will help you to understand both static and dynamic binding in Java.

This is directly related to execution of code. If you have more than one method of same name (method overriding) or two variable of same name in same class hierarchy it gets tricky to find out which one is used during runtime as a result of there reference in code. This problem is resolved using *static and dynamic binding in Java.* For those who are not familiar with binding operation, its process used to link which method or variable to be called as result of there reference in code. Most of the references is resolved during compile time but some references which depends upon Object and polymorphism in Java is resolved during runtime when actual object is available.

8.a) Write a program in JAVA that demonstrates a three level inheritance. Use the various access controls for the same. [WBUT 2011]

Ans.

```
class A
 {
   void disp()
    {
      System.out.println("Hello m Dikaoo kya");
    }
 }
class B extends A
 {
    void moll()
    {
      System.out.println("nye nye item chiye kya");
    }
 }
class MultiInheri extends B
 {
   void china()
   {
     System.out.println("sara ka sara mall kharab hai");
   }
  public static void main(String args[])
   {
    MultiInheri ab=new MultiInheri();
    ab.disp();
    ab.moll();
    ab.china();
   }
 }
```

b) What is the difference between object oriented programming and procedural programming paradigm? [WBUT 2011]

Ans.

Refer to group b.

c) What is the role of JVM? How it is different from JRE? [WBUT 2011]

Ans.

A Java Virtual Machine (JVM) is a set of computer software programs and data structures that use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a form of computer intermediate language commonly referred to as Java bytecode. This language conceptually represents the instruction set of a stack-oriented, capability architecture. Java Virtual Machines operate on Java bytecode, which is normally (but not necessarily) generated from Java source code; a JVM can also be used to implement programming languages other than Java. The JVM is a crucial component of the Java Platform. JVMs are available for many hardware and software platforms. The use of the same bytecode for all platforms allows Java to be described as "compile once, run anywhere", as opposed to "write once, compile anywhere", which describes cross-platform compiled languages. The JVM also enables such features as Automated Exception Handling that provides 'root-cause' debugging information for every software error (exception) independent of the source code. The JVM is distributed along with a set of standard class libraries that implement the Java API (Application Programming Interface). An application programming interface is what a computer system, library or application provides in order to allow data exchange between them. They are bundled together as the Java Runtime Environment.

8.a) What do you mean by automatic garbage collection in JAVA?[WBUT 2011]

Ans.

Refer to group b.

b) What is the use of finally clause in exception handling? When is the finally clause of a try catch finally statement executed? Illustrate with example. [WBUT 2011]

Ans.

Syntax

```
try {
   try_statements
}
[catch (exception_var_1 if condition_1) {
   catch_statements_1
```

}]
...
[catch (*exception_var_2*) {
  catch_statements_2
}]
[finally {
  finally_statements
}]

Parameters

try_statements
: The statements to be executed.

catch_statements_1, catch_statements_2
: Statements that are executed if an exception is thrown in the try block.

exception_var_1, exception_var_2
: An identifier to hold an exception object for the associated catch clause.

condition_1
: A conditional expression.

finally_statements
: Statements that are executed after the try statement completes. These statements execute regardless of whether or not an exception was thrown or caught.

Description

The try statement consists of a try block, which contains one or more statements, and at least one catch clause or a finally clause, or both. That is, there are three forms of the try statement:

1. try...catch
2. try...finally
3. try...catch...finally

A catch clause contain statements that specify what to do if an exception is thrown in the try block. That is, you want the try block to succeed, and if it does not succeed, you want control to pass to the catch block. If any statement within the try block (or in a function called from within the try block) throws an exception, control immediately shifts to the catch clause. If no exception is thrown in the try block, the catch clause is skipped.

The finally clause executes after the try block and catch clause(s) execute but before the statements following the try statement. It always executes, regardless of whether or not an exception was thrown or caught.

You can nest one or more try statements. If an inner try statement does not have a catch clause, the enclosing try statement's catch clause is entered.

b) Explain inheritance, encapsulation concept. [WBUT 2012]

Ans.

Refer to group b.

c) What is the superkey in JAVA? What is wrapper class? [WBUT 2012]

Ans.

Refer to group b

Refer to group c

9. a) What are adapter classes? [WBUT 2013]

Ans.

An adapter class provides the default implementation of all methods in an event listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface. You can define a new class by extending one of the adapter classes and implement only those events relevant to you.

b) What is multithreading? What are the two different ways to create multithreaded program? [WBUT 2013]

Ans.

Java is a*multithreaded programming language* which means we can develop multithreaded program using Java. A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multithreading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multithreading enables you to write in a way where multiple activities can proceed concurrently in the same program.

c) Discuss each part of the statement: System.out.println(); [WBUT 2013]

Ans.

- System is a built-in class present in java.lang package. This class has a final modifier, which means that, it cannot be inherited by other classes. It contains pre-defined methods and fields, which provides facilities like standard input, output, etc.
- out is a static final field (ie, variable)in System class which is of the type PrintStream (a built-in class, contains methods to print the different data values). static fields and methods must be accessed by using the class name, so (System.out).
- out here denotes the reference variable of the type PrintStream class.
- println() is a public method in PrintStream class to print the data values. Hence to access a method in PrintStream class, we use out.println() (as non static methods and fields can only be accessed by using the refrence varialble)

10.a) How does JAVA implement platform independence?[WBUT 2013]

Ans.

The compiled code is the exact set of instructions the CPU requires to "execute" the program. In Java, the compiled code is an exact set of instructions for a "virtual CPU" which is required to work the same on every physical machine.

So, in a sense, the designers of the Java language decided that the language and the compiled code was going to be platform independent, but since the code eventually has to run on a physical platform, they opted to put all the platform dependent code in the JVM.

This requirement for a JVM is in contrast to your Turbo C example. With Turbo C, the compiler will produce platform dependent code, and there is no need for a JVM work-alike because the compiled Turbo C program can be executed by the CPU directly.

With Java, the CPU executes the JVM, which is platform dependent. This running JVM then executes the Java bytecode which is platform independent, provided that you have a JVM availble for it to execute upon. You might say that writing Java code, you don't program for the code to be executed on the physical machine, you write the code to be executed on the Java Virtual Machine.

The only way that all this Java bytecode works on all Java virtual machines is that a rather strict standard has been written for how Java virtual machines work. This means that no matter what physical platform you are using, the part where the Java bytecode interfaces with the JVM is guaranteed to work only one way. Since all the JVMs work exactly the same, the same code works exactly the same everywhere without recompiling. If you can't pass the tests to make sure it's the same, you're not allowed to call your virtual machine a "Java virtual machine".

b) What is the difference between an instance member and a class member?[WBUT 2013]

Ans.

When you declare a member variable such as aFloat in MyClass:

```
class MyClass {
   float aFloat;
}
```

you declare an *instance variable*. Every time you create an instance of a class, the runtime system creates one copy of each the class's instance variables for the instance.

Instance variables are in constrast to *class variables* (which you declare using the static modifier). The runtime system allocates class variables once per class regardless of the number of instances created of that class. The system allocates memory for class variables the first time it encounters the class. All instances share the same copy of the class's class variables. You can access class variables through an instance or through the class itself.

Methods are similar: Your classes can have instance methods and class methods. Instance methods operate on the current object's instance variables but also have access to the class variables. Class methods, on the other hand, cannot access the instance variables declared within the class (unless they create a new object and access them through the object). Also, class methods can be invoked on the class, you don't need an instance to call a class method.

By default, unless otherwise specified, a member declared within a class is an instance member. The class defined below has one instance variable--an integer named x--and two instance methods--x and setX--that let other objects set and query the value of x:

```
class AnIntegerNamedX {
   int x;
   public int x() {
      return x;
   }
   public void setX(int newX) {
      x = newX;
   }
}
```

Every time you instantiate a new object from a class, you get a new copy of each of the class's instance variables. These copies are associated with the new object. So, every time you instantiate a new AnIntegerNamedX object

from the class, you get a new copy of x that is associated with the new AnIntegerNamedX object.

c) What is polymorphism? Differentiate between compile time and run time polymorphism.[WBUT 2013]

Ans.

Refer to group b.

d) Write a program to input three number and print the highest among them.[WBUT 2013]

Ans.

```
import java.util.Scanner;

class LargestOfThreeNumbers
{
  public static void main(String args[])
  {
    int x, y, z;
    System.out.println("Enter three integers ");
    Scanner in = new Scanner(System.in);

    x = in.nextInt();
    y = in.nextInt();
    z = in.nextInt();

    if ( x > y && x > z )
      System.out.println("First number is largest.");
    else if ( y > x && y > z )
      System.out.println("Second number is largest.");
    else if ( z > x && z > y )
      System.out.println("Third number is largest.");
    else
      System.out.println("Entered numbers are not distinct.");
  }
}
```

11. Write short notes on:

a) Abstract class.

b) Exception handling.

c) Interface

d) JVM

Ans.

a)Refer to group b

b)Refer to group c

c)Refer to group b

d)Refer to group c

ARRAYS AND STRING HANDLING

GROUP- A

[WBUT 2008]

1. String a = "3";
System.out.println(a);
The output is
a) 3 b) a

c) compiler error d) runtime exception

Ans. a

a) 3 c

2. int arr [] = new int[3];
arr[3] = 7;
System.out.println(arr[3]);
The output is
a) 3 b) 7

c) compiler error d) runtime exception

Ans.d)runtime exception

3. String s = "WBUT"
System.out.println(s.charAt(2));

The output is

a) U b) Exception occurs c) 2 d) B.

Ans.

a)U

4. What is the return type of read ( ) method of InputStream class ?

a) String b) Void c) Float d) None of these.

Ans.

d)none of these

[WBUT 2011]

5.String s="s";

System.out.println(s);

The output is

a)s b)"s"

c)Compile error d)Runtime error

Ans.

a)s

[WBUT 2012]

6.Consider the following statements:
String s="BCA-602B";

System.out.println(s.charAt(2));

The output is

a)C b)A

c)Exception occur d)2.

Ans.

b)A

7.Consider the following code snippet:

String river=new String("Columbia");

System.out.println(river.length());

What is printed?

a)6 b)7

c)8 d)Columbia

e)River

Ans.

c)8

8.You read the following statement in a JAVA program that compiles and executes

Submarine.dive(depth);

What can you say for sure?

a)Depth must be an int

b)Dive must be a method

c)Dive must be the name of an instance field

d)Submarine must be the name of a class

e)Submarine must be a method

Ans. a, b, e.

9.String str="ABCD";

System.out.println(str.charAt(2));

The output is

a)C b)2

c)exception occurs d)none of these

Ans. a

a)C

GROUP – B

1.Display the triangle as follow:

1

23

456

78910...N*/[WBUT 2012]

Ans.

```
class Output1
{
 public static void main(String args[]){
 int c=0;
 int n = Integer.parseInt(args[0]);
for(int i=1;i<=n;i++)
{
for(int j=1;j<=i;j++)
{
 if(c!=n)
{
 c++;
 System.out.print(c+" ");
 }
 else
break loop1;
 }
 System.out.print("\n");
}
}
}
```

GROUP – C

1.Write a program in JAVA to reverse a string.[WBUT 2008]

Ans.

```
import java.util.*;

class ReverseString
{
  public static void main(String args[])
  {
    String original, reverse = "";
    Scanner in = new Scanner(System.in);
```

```
        System.out.println("Enter a string to reverse");
        original = in.nextLine();

        int length = original.length();

        for ( int i = length - 1 ; i >= 0 ; i-- )
          reverse = reverse + original.charAt(i);

        System.out.println("Reverse of entered string is: "+reverse);
    }
}
```

2.Explain two dimensional array with example.[WBUT 2010]
Ans.

Two-dimensional arrays are defined as "an array of arrays". Since an array type is a first-class Java type, we can have an array of ints, an array of Strings, or an array of Objects. For example, an array of ints will have the type int[]. Similarly we can have int[][], which represents an "array of arrays of ints". Such an array is said to be a two-dimensional array.
The command

```
int[][] A = new int[3][4];
```

declares a variable, A, of type int[][], and it initializes that variable to refer to a newly created object. That object is an array of arrays of ints. Here, the notation int[3][4] indicates that there are 3 arrays of ints in the array A, and that there are 4 ints in each of those arrays.
To process a two-dimensional array, we use nested for loops. We already know about for loop. A loop in a loop is called a Nested loop. That means we can run another loop in a loop.

Notice in the following example how the rows are handled as separate objects.

Code: Java

```
int[][] a2 = new int[10][5];
 // print array in rectangular form
 for (int i=0; i<a2.length; i++) {
    for (int j=0; j<a2[i].length; j++) {
       System.out.print(" " + a2[i][j]);
    }
    System.out.println("");
}
```

In this example, "int[][] a2 = new int[10][5];" notation shows a two-dimensional array. It declares a variable a2 of type int[][],and it initializes that variable to refer to a newly created object. The notation int[10][5]

indicates that there are 10 arrays of ints in the array a2, and that there are 5 ints in each of those arrays.

3.a) What is the difference between string and string buffer? [WBUT 2011]

Ans.

String object is immutable whereas StringBuffer/StringBuilder objects are mutable.

By immutable, we mean that the value stored in the String object cannot be changed. Then the next question that comes to our mind is "If String is immutable then how am I able to change the contents of the object whenever I wish to?" . Well, to be precise it's not the same String object that reflects the changes you do. Internally a new String object is created to do the changes.

So suppose you declare a String object:

String myString = "Hello";

Next, you want to append "Guest" to the same String. What do you do?

myString = myString + " Guest";

When you print the contents of myString the output will be "Hello Guest". Although we made use of the same object(myString), internally a new object was created in the process. So, if you were to do some string operation involving an append or trim or some other method call to modify your string object, you would really be creating those many new objects of class String.

b)Write a java program to add two string using string buffer.[WBUT 2011]

Ans.

```
public static String ugly(String foo, String bar) {
   StringBuffer buffer = new StringBuffer();
   buffer.append (foo);
   buffer.append (" ");
   buffer.append (bar);
   return buffer.toString();
}
public static String neat(String foo, String bar) {
  return foo + " " + bar;
}
```

CONSTRUCTORS

GROUP- A

[WBUT 2008]

1.Which of the following statement is true regarding constructor?

a)All cases must be define as a constructor.

b)A constructor can be defined private.

c)A constructor can return a value.

d)A constructor must initialise all fields of a class.

Ans.

b)A constructor can be defined private.

[WBUT 2009]

2.Which member function of a class is used to create object of that class?

a)Constructor b)New

c)Object d)None of these.

Ans.

b)New

3.How many default constructor one can have in a class?

a)1 b)4

c)0 d)2

Ans.

a)1

[WBUT 2010]

4.A constructor can be inherited using the keyword

a)final b)static

c)super d)none of these

Ans.

c)super

5. Which of the following statement is true regarding constructor?

a)All cases must be define as a constructor.

b)A constructor can be defined private.

c)A constructor can return a value.

d)A constructor must initialise all fields of a class.

Ans.

b)A constructor can be defined private.

[WBUT 2011]

6.How many default constructors can a class have, when it has constructor?

a)1 b)0

c)2 d)Any number

Ans.

a)1

[WBUT 2012]

7.A constructor

a)must have the same name as the class it is declared within.

b)is used to create objects.

c)may be declared private.

d)none of these.

e)a,b,c

Ans.

e)a,b,c.

8.Which of the following may be a part of class definition?

a)Instance variables.

b)Instance methods.

c)Constructors.

d)All of these.

Ans.

d)All of these.

[WBUT 2013]

9. Which of the following statement is true regarding constructor?

a)All cases must be define as a constructor.

b)A constructor can be defined private.

c)A constructor can return a value.

d)A constructor must initialise all fields of a class.

Ans.

b)A constructor can be defined private.

GROUP – B

1.How will you call parameterized constructor and overrided method from parent class in sub-class?[WBUT 2013]

Ans.

```
class A
{
   public void myMethod()
   { /* ... */ }
}

class B extends A
{
   public void myMethod()
   { /* Another code */ }
}
  public void myMethod()
  {
     // B stuff
     super.myMethod();
     // B stuff
  }
```

2.a)In JAVA, explain how to call a constructor from another.[WBUT 2013]

Ans.

```
public class Foo
{
   private int x;
```

```
    public Foo()
    {
        this(1);
    }

    public Foo(int x)
    {
        this.x = x;/*we call from another constructor*/
    }
}
```

b)Can a super class object reference a sub class object? Explain.[WBUT 2013]

Ans.

We can do this using interface or polymorphism. Such as:

```
abstract class Shape {

    abstract double getArea();

}

class Rectangle extends Shape{
    double h, w;

    public Rectangle(double h, double w){

        this.h = h;
        this.w = w;
    }

    public double getArea(){
        return h*w;
    }
}

class Circle extends Shape{
    double radius;

    public Circle(double radius){
        this.radius = radius;
    }

    public double getArea(){
        return Math.PI * Math.sqrt(radius);
    }
}
```

```
class Triangle extends Shape{
    double b, h;

    public Triangle(double b, double h){
        this.b = b;
        this.h = h;
    }

    public double getArea(){
        return (b*h)/2;
    }


}

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

    //USAGE
    //Without polymorphism
    Triangle t = new Triangle(3, 2);
    Circle c = new Circle(3);
    Rectangle r = new Rectangle(2,3);

    System.out.println(t.getArea());
    System.out.println(c.getArea());
    System.out.println(r.getArea());

    //USAGE with Polymorphism

    Shape s[] = new Shape[3];
    s[0] = new Triangle(3, 2);
    s[1] = new Circle(3);;
    s[2] = new Rectangle(2,3);

    for(Shape shape:s){
        System.out.println(shape.getArea());
    }

    }
}
```

GROUP – C

1.a) Write a program using constructor overloading to calculate area of a rectangle and circle.[WBUT 2008]

Ans.

```
import java.util.*;
class geofig
{
   double area(double r)
   {
      return(3.14*r*r);
   }
    float area(float l,float b)
   {
      return(l*b);
   }
  }

class geo
{
   public static  void main(String arg[])
   {
      Scanner sc =new Scanner(System.in);
      geofig g = new geofig();
      System.out.println("enter double value for radius of circle");
      double r =sc.nextDouble();
      System.out.println("area of circle="+g.area(r));
      System.out.println("enter float value for length and braedth of
rectangle");
      float l =sc.nextFloat();
      float b =sc.nextFloat();
      System.out.println("area of rectangle="+g.area(l,b));
      }
}
```

METHODS

GROUP –A

[WBUT 2008]

1.Dynamic method dispatcher is used for

a)resolving method overriding.

b)resolving multilevel inheritance.

c)resolving multiple inheritance anomaly.

d)none of these.

Ans.

a)resolving method overriding

2. What is the return type of read ( ) method of InputStream class ?

a) String b) Void c) Float d) None of these.

Ans.

d)none of these.

[WBUT 2009]

3.The ability to declare different methods with the same name in a class is known as

a)Method overloading

b)Method overriding

c)Recursion

d)None of these.

Ans.

a)Method overloading

4.Data Input is

a)an abstract class defined in java.io

b)a class we can use to read primitive data types

that defines method to open files

d)an interface that defines methods to read primitive datatypes.

Ans.

d)

[WBUT 2010]

5.Dynamic method dispatcher is used for

a)resolving method overriding.

b)resolving multilevel inheritance.

c)resolving multiple inheritance anomally.

d)none of these.

Ans.

a)resolving method overriding

[WBUT 2013]

6.Dynamic method dispatcher is used for

a)resolving method overriding.

b)resolving multilevel inheritance.

c)resolving multiple inheritance anomally.

d)none of these.

Ans.

7.What is the return type of read() method of InputSream class?

a)String b)Float

c)Void d)None of these

Ans.

d)none of these.

GROUP – B

1. What are the differences between method overloading and method overriding ? Give example for both.[WBUT 2008, 2009, 2010, 2011]

Ans.

The difference between overriding and overloading in Java is a common source of confusion – but it is fairly easy to understand with the examples we present below. Let's start the discussion by talking more about method *overloading* first. Method overloading in Java occurs when two or more methods in the same class have the exact same name but different parameters (remember that method parameters accept values passed into the method).

The conditions for method overloading

1.) The *number* of parameters is different for the methods.
2.) The parameter *types* are different (like changing a parameter that was a float to an int).

Examples of Method Overloading in Java – both valid and invalid:

```
//compiler error - can't overload based on the
//type returned -
//(one method returns int, the other returns a float):

int changeDate(int Year) ;
float changeDate (int Year);

//compiler error - can't overload by changing just
//the name of the parameter (from Year to Month):

int changeDate(int Year);
int changeDate(int Month) ;

//valid case of overloading, since the methods
//have different number of parameters:

int changeDate(int Year, int Month) ;
int changeDate(int Year);

//also a valid case of overloading, since the
//parameters are of different types:

int changeDate(float Year) ;
int changeDate(int Year);
```

Overriding methods is completely different from overloading methods. If a derived class requires a different definition for an inherited method, then that method can be redefined in the derived class. This would be considered

overriding. An overridden method would have the exact same method name, return type, number of parameters, and types of parameters as the method in the parent class, and the only difference would be the definition of the method.

Example of method overriding

Let's go through a simple example to illustrate what method overriding would look like:

```
public class Parent {

public int someMethod() {

 return 3;

 }
}


public class Child extends Parent{

// this is method overriding:
public int someMethod() {

 return 4;

}

}
```

2.What do you mean by interface ?[WBUT 2008, 2011]

Ans.

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways:

- An interface can contain any number of methods.

- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces:

The interface keyword is used to declare an interface. Here is a simple example to declare an interface:

Example:

Let us look at an example that depicts encapsulation:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
   //Any number of final, static fields
   //Any number of abstract method declarations\
}
```

Interfaces have the following properties:

- An interface is implicitly abstract. You do not need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example:

```
/* File name : Animal.java */
interface Animal {

   public void eat();
```

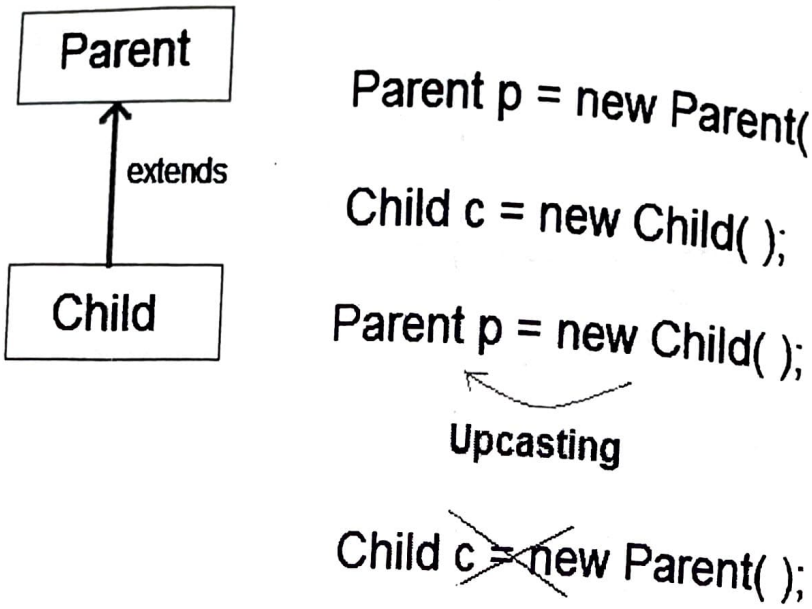
public void travel();

}

3.What is dynamic method dispatch? How it is accomplished? [WBUT 2011]

Ans.

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime.This is how java implements runtime polymorphism.



Dynamic Method Dispatch is related to a principle that states that an super class reference can store the reference of subclass object. However, it can't call any of the newly added methods by the subclass but a call to an overridden methods results in calling a method of that object whose reference is stored in the super class reference. It simply means that which method would be executed, simply depends on the object reference stored in super class object.

GROUP – C

1.What do you mean by method overloading? Explain with example.[WBUT 2008]

Ans.

Refer to ques 1 of group b.

2.Write short note on static variable and method.[WBUT 2009]

Ans.

The Java programming language supports static methods as well as static variables. Static methods, which have the static modifier in their declarations, should be invoked with the class name, without the need for creating an instance of the class, as in

ClassName.methodName(args)

Note: You can also refer to static methods with an object reference like instanceName.methodName(args)
but this is discouraged because it does not make it clear that they are class methods.

A common use for static methods is to access static fields. For example, we could add a static method to the Bicycle class to access the numberOfBicycles static field:

```
public static int getNumberOfBicycles() {
    return numberOfBicycles;
}
```

Not all combinations of instance and class variables and methods are allowed:

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.

3.What is the purpose of runnable interface? What is the collection interface? Explain its usage.[WBUT 2011]

Ans.

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has not yet been stopped.

In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

A Collection represents a group of objects known as its elements. The Collection interface is used to pass around collections of objects where maximum generality is desired. For example, by convention all general-purpose collection implementations have a constructor that takes a Collection argument. This constructor, known as a *conversion constructor*, initializes the new collection to contain all of the elements in the specified collection, whatever the given collection's subinterface or implementation type. In other words, it allows you to *convert* the collection's type.

Suppose, for example, that you have a Collection<String> c, which may be a List, a Set, or another kind of Collection. This idiom creates a new ArrayList (an implementation of the List interface), initially containing all the elements in c.

```
List<String> list = new ArrayList<String>(c);
```
Or — if you are using JDK 7 or later — you can use the diamond operator:
```
List<String> list = new ArrayList<>(c);
```

The Collection interface contains methods that perform basic operations, such as int size(), boolean isEmpty(), boolean contains(Object element), boolean add(E element), boolean remove(Object element), and Iterator<E> iterator().

It also contains methods that operate on entire collections, such as boolean containsAll(Collection<?> c), boolean addAll(Collection<? extends E> c), boolean removeAll(Collection<?> c), boolean retainAll(Collection<?> c), and void clear().

4.What is aggregation? How is aggregation different from generalization and association?[WBUT 2011]

Ans.

Refer to group b

JAVA APPLETS

GROUP –A

[WBUT 2008]

1.In JAVA applet, stop method can be invoked for a thread

a)once

b)twice

c)compile error

d)none of these

Ans.

d)none of these

2. Java bytecode is saved in files with names that end with

a) .class
b) .code
c) .java
d) .exe

Ans.

d).exe

[WBUT 2011]

3. In java applet, 'Stop()' method can be invoked for a thread

a) once
b) twice
c) compiler error
d) none of these.

Ans.

d)none of these

[WBUT 2012]

4.What is the difference between a Java applet and a Java application?

a)An application in general be trusted wheras an applet can't.

b)An applet must be executed in a browser environment.

c)An applet is not able to access the files of the computer it runs on.

d)(a),(b) and (c).

Ans.

b)

[WBUT 2013]

5. In java applet, 'Stop()' method can be invoked for a thread
a) once b) twice
c) compiler error d) not at all.

Ans.

d)

GROUP – B.

1. What is Synchronization ? When is it used ?[WBUT 2010]

Ans.

The Java programming language provides two basic synchronization idioms: *synchronized methods* and *synchronized statements*. The more complex of the two, synchronized statements, are described in the next section. This section is about synchronized methods.

To make a method synchronized, simply add the synchronized keyword to its declaration:

```
public class SynchronizedCounter {
   private int c = 0;

   public synchronized void increment() {
      c++;
   }

   public synchronized void decrement() {
      c--;
   }

   public synchronized int value() {
      return c;
   }
}
```

If count is an instance of SynchronizedCounter, then making these methods synchronized has two effects:

- First, it is not possible for two invocations of synchronized methods on the same object to interleave. When one thread is executing a synchronized method for an object, all other threads that invoke

synchronized methods for the same object block (suspend execution) until the first thread is done with the object.

- Second, when a synchronized method exits, it automatically establishes a happens-before relationship with *any subsequent invocation* of a synchronized method for the same object. This guarantees that changes to the state of the object are visible to all threads.

Note that constructors cannot be synchronized — using the synchronized keyword with a constructor is a syntax error. Synchronizing constructors doesn't make sense, because only the thread that creates an object should have access to it while it is being constructed.

GROUP – C

1. What is applet ? Describe an Applet life cycle. Develop an Applet that receives three numeric values as input from the user and then displays the largest of the three on the screen , also write the HTMl code.[WBUT 2008, 2009, 2010, 2013]

Ans.

An applet is a small Internet-based program written in Java, a programming language for the Web, which can be downloaded by any computer. The applet is also able to run in HTML. The applet is usually embedded in an HTML page on a Web site and can be executed from within a browser. An applet is a Java class that extends the java.applet.Applet class. A main() method is not invoked on an applet instead instance of Graphics class is used.

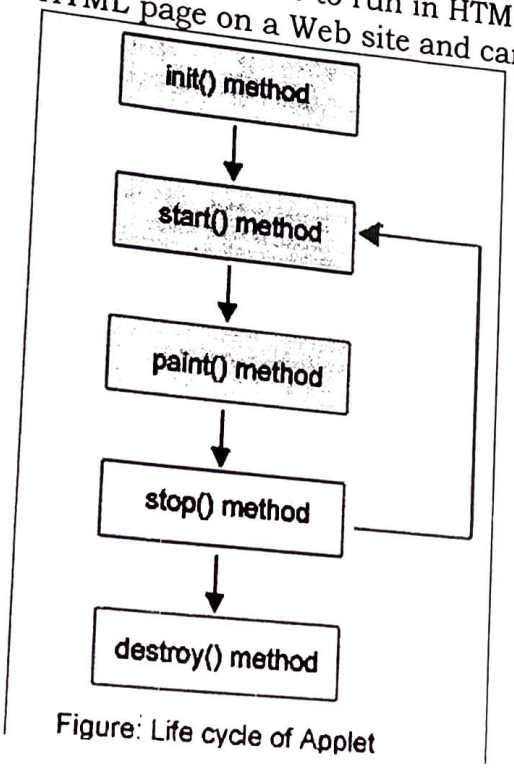


Figure: Life cycle of Applet

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

publicclass MaxOf3No extends Applet
{
    TextField T1,T2,T3;
```

```
publicvoid init(){
   T1 = new TextField(10);
   T2 = new TextField(10);
   T3 = new TextField(10);

   add(T1);
   add(T2);
   add(T3);

   T1.setText("0");
   T2.setText("0");
   T3.setText("0");
}

publicvoid paint(Graphics g){
    int a, b, c,result;
    String str;

    g.drawString("Enter value to Check the Maximum of 3 ",10,50);

    str=T1.getText();
    a=Integer.parseInt(str);
    str=T2.getText();
    b=Integer.parseInt(str);
    str=T3.getText();
    c=Integer.parseInt(str);

    g.setColor(Color.blue);
    if (a>b) {
       if (a>c)
          result=a;
       else
          result=c;
    }
 else{
      if (b>c)
         result=b;
      else
         result=c;
  }
  g.drawString("Maximnum of 3 No is "+result,10,70);
  showStatus("MAXIMUM OF 3 NUMBERS");
}

public boolean action(Event e, Object o){
    repaint();
    returntrue;
}

}
```

2.a) Briefly explain the Applet life cycle. [WBUT 2012]
b) What are the main differences between Java application & Java applet ?

c)Write an applet program where you may input two numbers and get the result of the product of those two numbers?[WBUT 2009]

Ans.

Various states, an applet, undergoes between its object creation and object removal (when the job is over) is known as life cycle. Each state is represented by a method. There exists 5 states represented by 5 methods. That is, in its life of execution, the applet exists (lives) in one of these 5 states.

These methods are known as "callback methods" as they are called automatically by the browser whenever required for the smooth execution of the applet. Programmer just write the methods with some code but never calls.

Following are the methods.

Brief Description of Life Cycle Methods

Following is the brief description of the above methods.

1. init(): The applet's voyage starts here. In this method, the applet object is created by the browser. Because this method is called before all the other methods, programmer can utilize this method to instantiate objects, initialize variables, setting background and foreground colors in GUI etc.; the place of a constructor in an application. It is equivalent to born state of a thread.
2. start(): In init() method, even through applet object is created, it is in inactive state. An inactive applet is not eligible for microprocessor time even though the microprocessor is idle. To make the applet active, the init() method calls start() method. In start() method, applet becomes active and thereby eligible for processor time.
3. paint(): This method takes a java.awt.Graphics object as parameter. This class includes many methods of drawing necessary to draw on the applet window. This is the place where the programmer can write his code of what he expects from applet like animation etc. This is equivalent to runnable state of thread.
4. stop(): In this method the applet becomes temporarily inactive. An applet can come any number of times into this method in its life cycle and can go back to the active state (paint() method) whenever would like. It is the best place to have cleanup code. It is equivalent to the blocked state of the thread.

5. destroy(): This method is called just before an applet object is garbage collected. This is the end of the life cycle of applet. It is the best place to have cleanup code. It is equivalent to the dead state of the thread.

3.a) What is a local applet?

b)What are the application of applet tag?[WBUT 2010]

Ans.

A LOCAL applet is the one which is stored on our computer system.When browser try to access the applet, it is not necessary for our computer to be connected to The Internet.

b) We use the applet tag to deploy applets to a multi-browser environment.

For complete details on the applet tag, see the following:

- Deploying With the Applet Tag
- HTML 4.01 Specification

Note: The HTML specification states that the applet tag is deprecated, and that you should use the object tag instead. However, the specification is vague about how browsers should implement the object tag to support Java applets, and browser support is currently inconsistent. Oracle therefore recommends that you continue to use the applet tag as a consistent way to deploy Java applets across browsers on all platforms

PACKAGES AND THREADS

GROUP –A

[WBUT 2013]

1.Which of the following JSP expressions are valid?

a) <%="Sorry"+"for the"+"break"%>

b) <%="Sorry"+"for the"+"break";%>

c) <%="Sorry"%>

d) <%="Sorry";%>

Ans.

b)<%="Sorry"+"for the"+"break";%>

2.A class can be converted to a thread by implementing the interface

a)Thread b)Runnable

Ans.

b)Runnable

3.Which is not allowed in EJB programming?

a)Thread management.

b)Transient field.

c)Listening on a socket.

Ans.

GROUP – B

1. Discuss the various levels of access protection available for packages and their implications.[WBUT 2009]

Ans.

There are two good uses for package level visibility

1) Defining "internal" classes in a public API. Commonly you would define your interfaces and core factories as public and the "internal" implementations as package level. Then the public factories can construct the package level implementation classes and return them as instances of the public interfaces. This nicely allows users to only access the stuff they should.

The downside is that you have to have all this stuff in the same package, which almost never is a good idea for any reasonably-sized API. JSR 294/modules/Project Jigsaw in Java 7 will hopefully provide an alternative by specifying a new visibility modifier (module) that can be used to access classes within a module across packages without making them visible outside the module. You can find an example of how this would work in this article.

2) Unit testing is the other common use case. Frequently you'll see a src tree and a test tree and stuff that would otherwise be private is instead package level so that unit tests in the same (parallel) package are able to access otherwise hidden methods to check or manipulate state.

2.What is a thread? What is multithreading? How to create a thread in a program?[WBUT 2012]

Ans.

a *thread* is a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time. Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time. The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time.

Multithreaded applications deliver their potent power by running many threads concurrently within a single program. From a logical point of view, multithreading means multiple lines of a single program can be executed at the same time, however, it is not the same as starting a program twice and

saying that there are multiple lines of a program being executed at the same time. In this case, the operating system is treating the programs as two separate and distinct processes. Under Unix, forking a process creates a child process with a different address space for both code and data. However, fork() creates a lot of overhead for the operating system, making it a very CPU-intensive operation. By starting a thread instead, an efficient path of execution is created while still sharing the original data area from the parent.

The first method of creating a thread is to simply extend from the Thread class. Do this only if the class you need executed as a thread does not ever need to be extended from another class. The Thread class is defined in the package java.lang, which needs to be imported so that our classes are aware of its definition.

```
import java.lang.*;
public class Counter extends Thread
{
        public void run()
        {
        ....
        }
}
```

The above example creates a new class Counter that extends the Thread class and overrides the Thread.run() method for its own implementation. The run() method is where all the work of the Counter class thread is done. The same class can be created by implementing Runnable:

```
import java.lang.*;
public class Counter implements Runnable
{
        Thread T;
        public void run()
        {
        ....
        }
}
```

3.What is the package? What is the difference between throw and throws keyword? [WBUT 2012]
Ans.

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A Package can be defined as a grouping of related types(classes, interfaces, enumerations and annotations) providing access protection and name space management.

Some of the existing packages in Java are::

- **java.lang** - bundles the fundamental classes
- **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

1) You can declare multiple exception thrown by method in throws keyword by separating them in common e.g. throws IOException, ArrayIndexBoundException etc, while you can only throw one instance of exception using throw keyword e.g. throw new IOException("not able to open connection").

2) **throws** keyword gives a method flexibility of throwing an Exception rather than handling it. with throws keyword in method

signature a method suggesting its caller to prepare for **Exception** declared in throws clause, specially in case of checked Exception and provide sufficient handling of them. On the other hand **throw keyword** transfer control of execution to caller by throwing an instance of Exception. throw keyword can also be used in place of return as shown in below example:

```
private static boolean shutdown() {

    throw new UnsupportedOperationException("Not yet implemented");
}
```

as in below method shutdown should return boolean but having throw in place compiler understand that this method will always **throw exception** .

3) throws keyword cannot be used anywhere exception method signature while **throw keyword** can be used inside method or static initializer block provided sufficient exception handling as shown in example.

```
static{
    try {
        throw new Exception("Not able to initialized");
    } catch (Exception ex) {
        Logger.getLogger(ExceptionTest.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
```

worth remembering is that static initializer block should complete normally.

4) **throw keyword** can also be used to break a switch statement without using break keyword as shown in below example:

```
int number = 5;
switch(number){
        case 1:
            throw new RuntimeException("Exception number 1");
        case 2:
            throw new RuntimeException('Exception number 2");
    }
```

4.Indicate the difference between PATH and CLASSPATH.[WBUT 2013]

Ans.

PATH is nothing but setting up an environment for operating system. Operating System will look in this PATH for executables.

Classpath is nothing but setting up the environment for Java. Java will use to find compiled classes (i.e. .class files).

For example,assume that Java is installed in C:\jdk1.5.0\ and your code is in C:\Sample\example.java, then your PATH and CLASSPATH should be set to:

PATH=C:\jdk1.5.0\bin;%PATH%
CLASSPATH=C:\Sample

GROUP –C

1 What is thread ? How do we set the priorities for thread ? Describe with an example. Write a small program which will synchronize among two threads.[WBUT 2008]
Ans.
Refer to ques 1 of group b
As an application program may have many Threads ,to run all threads at a time,Thread scheduler is run.Though in theory,*higher priority threads* get more CPU time than *lower priority threads*,but in practice CPU time may vary from one operating system to other,on how OS is implementing multitask.

final void setPriority(int level);
the *level* must be within the range *MIN_PRIORITY* and *MAX_PRIORITY* ...
Currently *MIN_PRIORITY* is 1 and *MAX_PRIORITY* is 10.
default value is *NORM_PRIORITY* which is 5.

```
class CounterThread extends Thread {

 private final int rangeStart;

 private final int rangeEnd;

private final int[] nums;

private int count;

 private CounterThread(final int rangeStart, final int rangeEnd, final int[]
nums) {

super();

this.rangeStart = rangeStart;
```

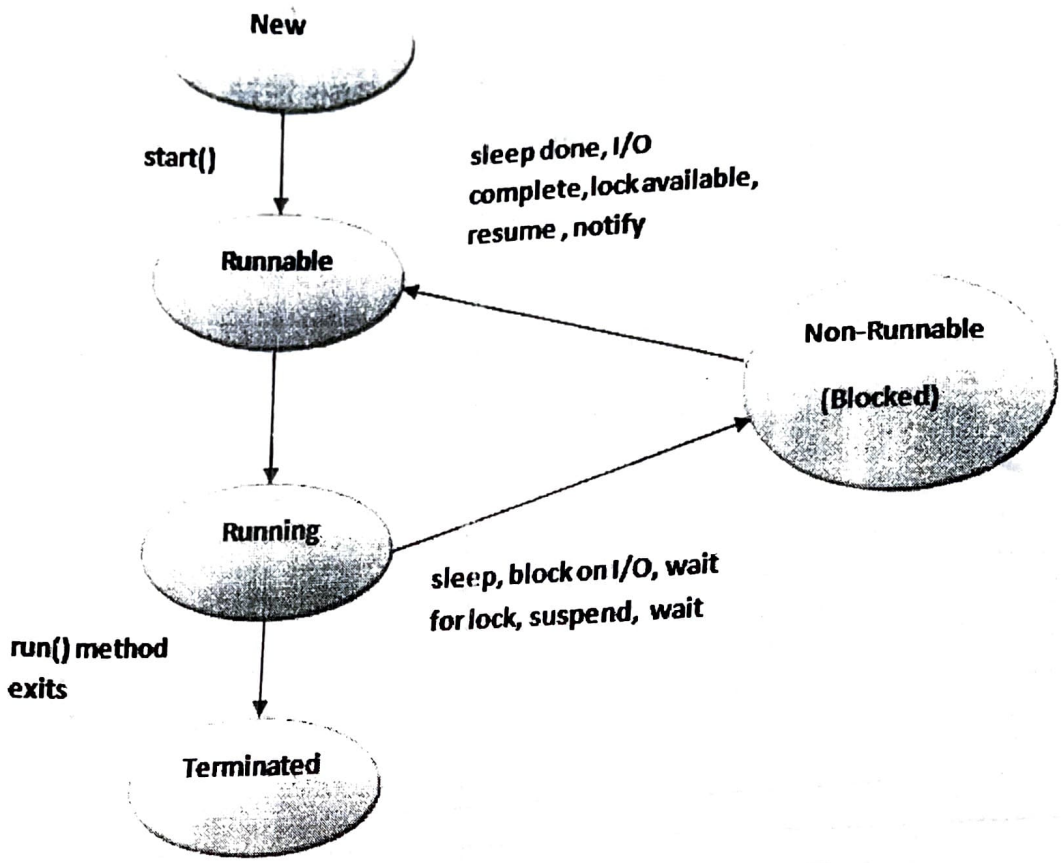
```
this.rangeEnd = rangeEnd;
this.nums = nums;
count = 0;
}
public void run() {
for(int i = rangeStart; i <= rangeEnd; ++i) {
count += nums[i];
}}}
```

2. a) Discuss the lifecycle of thread .
b) Write a small program with output to show thread .
c) What do we mean by thread priority ?[WBUT 2008,2009, 2013]

Ans.

A thread can be in one of the five states in the thread. According to sun, there is only 4 states new, runnable, non-runnable and terminated. There is no running state. But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread is controlled by JVM. The thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



1)New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2)Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3)Running

The thread is in running state if the thread scheduler has selected it.

4)Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5)Terminated

A thread is in terminated or dead state when its run() method exits.

3.a)Explain the methods by which you can block the thread.

b)What is the difference between runnable state and running state of thread.

c)What is meant by alive() and join()?

d)What is thread synchronisation?[WBUT 2009, 2011, 2012]

Ans.

a)Using the Thread *yield* method.

Because of the platform dependent nature of Java threading you cannot be certain if a thread will ever give up its use of CPU resources to other threads. On some operating systems the threading algorithm may automatically give different threads a share of the CPU time, on others one thread might simply hog processor resources. For this reason the Java Thread class has a static method called *yield* which causes the currently running thread to yield its hold on CPU cycles. This thread returns to the "ready to run" state and the thread scheduling system has a chance to give other threads the attention of the CPU. If no other threads are in a "ready to run state" the thread that was executing may restart running again.

It is hard to demonstrate the benefits of using the yield method without being able to run some sample code on two different implementations on Java with different thread scheduling systems. Assuming that option is not available to you it is worth describing two main different scheduling systems used in operating systems.

Time slicing/preemptive

Each thread gets a set amount of CPU time for executing. Once it has used up its time with the CPU, it is removed from accessing the CPU and any other waiting Threads get a chance at CPU time. When each thread has had its chance with the CPU the cycle starts again.The beauty of this approach is that you can be confident that each thread will get at least some time executing.

Non time slicing/Cooperative

A priority system is used to decide which thread will run. A thread with the highest priority gets time with the CPU. A program under this system needs to be created in such a way that it "voluntarily" yield access to the CPU.

b) The different states of threads are as follows:

1) **New** – When a thread is instantiated it is in New state until the start() method is called on the thread instance. In this state the thread is not considered to be alive.

2) **Runnable** – The thread enters into this state after the start method is called in the thread instance. The thread may enter into the Runnable state from Running state. In this state the thread is considered to be alive.
3) **Running** – When the thread scheduler picks up the thread from the Runnable thread's pool, the thread starts running and the thread is said to be in Running state.
4) **Waiting/Blocked/Sleeping** – In these states the thread is said to be alive but not runnable. The thread switches to this state because of reasons like wait method called or sleep method has been called on the running thread or thread might be waiting for some i/o resource so blocked. 5) Dead – When the thread finishes its execution i.e. the run() method execution completes, it is said to be in dead state. A dead state can not be started again. If a start() method is invoked on a dead thread a runtime exception will occur.

c) The main thread must be the last thread to finish. Sometimes this is accomplished by calling **sleep()** within **main( )**, with a long enough delay to ensure that all child threads terminate prior to the main thread. However, this is hardly a satisfactory solution, and it also raises a larger question: How can one thread know when another thread has ended? Fortunately, **Thread** provides a means by which you can answer this question.

Two ways exist to determine whether a thread has finished. First, you can call **isAlive( )** on the thread. This method is defined by **Thread**, and its general form is shown here:

final boolean isAlive( )

The **isAlive( )** method returns **true** if the thread upon which it is called is still running. It returns **false** otherwise. While **isAlive( )** is occasionally useful, the method that you will more commonly use to wait for a thread to finish is called **join( )**, shown here:

final void join( ) throws InterruptedException

This method waits until the thread on which it is called terminates. Its name comes from the concept of the calling thread waiting until the specified thread *joins* it. Additional forms of **join( )** allow you to specify a maximum amount of time that you want to wait for the specified thread to terminate

d) When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issue. For example if multiple threads try to write within a same file then they may corrupt the data because one of the threads can overrite data or while one thread is opening the same file at the same time another thread might be closing the same file.

So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block. Following is the general form of the synchronized statement:

```
synchronized(objectidentifier) {
   // Access shared variables and other shared resources
}
```

4.What is Unicode ? Explain the advantages of using Unicode.[WBUT 2010]

Ans.

Unicode is a computing industry standard designed to consistently and uniquely encode characters used in written languages throughout the world. The Unicode standard uses hexadecimal to express a character. For example, the value 0x0041 represents the Latin character A. The Unicode standard was initially designed using 16 bits to encode characters because the primary machines were 16-bit PCs.

When the specification for the Java language was created, the Unicode standard was accepted and the char primitive was defined as a 16-bit data type, with characters in the hexadecimal range from 0x0000 to 0xFFFF.

Because 16-bit encoding supports 2^{16} (65,536) characters, which is insufficient to define all characters in use throughout the world, the Unicode standard was extended to 0x10FFFF, which supports over one million characters. The definition of a character in the Java programming language could not be changed from 16 bits to 32 bits without causing millions of Java applications to no longer run properly. To correct the definition, a scheme was developed to handle characters that could not be encoded in 16 bits.

The characters with values that are outside of the 16-bit range, and within the range from 0x10000 to 0x10FFFF, are called *supplementary characters* and are defined as a pair of char values.