Name: Bilal Khondannavar **Class: BCA V**

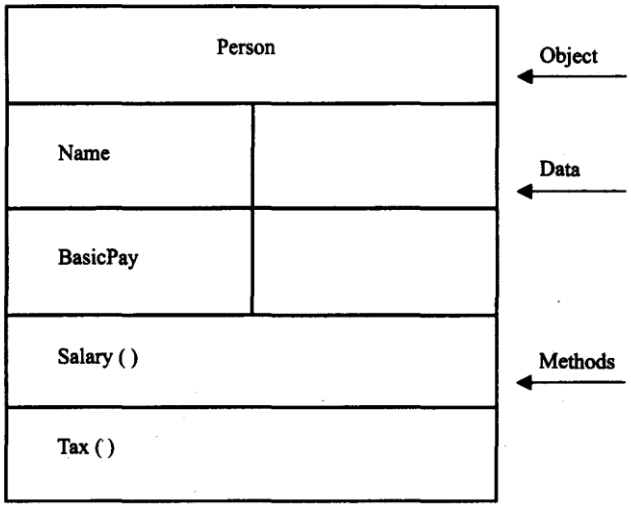
Internet Programming - 2017

1.(a) Explain the concepts of OOP's.

Object-Oriented Programming (OOP) is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts.

4 *Programming with Java: A Primer*

Fig. 1.2



Representation of an object

Classes are user-defined data types and behave like the built-in types of a programming language. For example, the syntax used to create an object is no different than the syntax used to create an integer object in C. If fruit has been defined as a class, then the statement

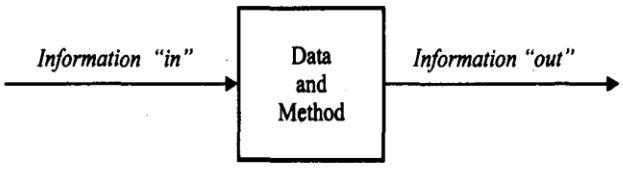
```
Fruit mango;
```

will create an object mango belonging to the class fruit.

Data Abstraction and Encapsulation

The wrapping up of data and methods into a single unit (called class) is known as *encapsulation.* Data encapsulation is the most striking feature of a class. The data is not accessible to' the outside world and only those methods, which are wrapped in the class, can access it. These methods provide the interface between the object's data and the program. This insulation of the data from direct access by 'the program is called *data hiding.* Encapsulation makes it possible for objects to be treated like 'black boxes', each performing a specific task without any concern for internal implementation

Fig. 1.3



Encapsulation—Objects as "black boxes"

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and methods that operate on these attributes. They encapsulate all the essential properties of the objects that are to be created.

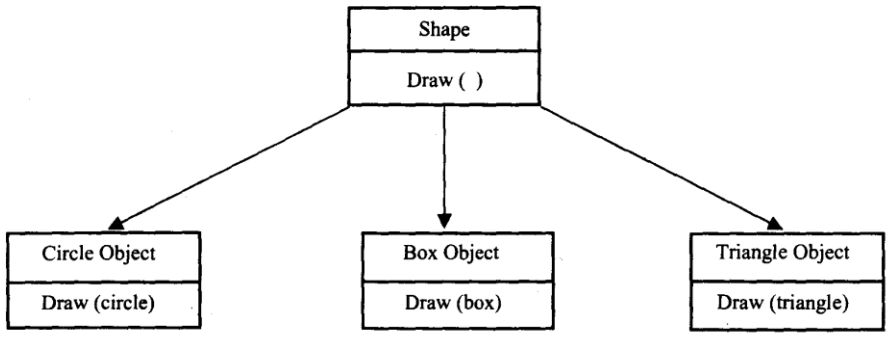
Inheritance. Inheritance is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. For example, the bird robin is a part of the class flying bird, which is again a part of the class bird. As illustrated in Fig. 1.4, the principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived. In OOP, the concept of inheritance provides the idea of reusability: This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one ..The new class will have the combined features of both the classes. Thus the real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side effects into the rest of the classes.

In Java, the derived class is known as 'subclass'. Note that each subclass defines only those features that are unique to it. Without the use of inheritance, each class would have to explicitly include all of its features.

Polymorphism

Polymorphism is another important OOP concept. Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behaviour in different instances. The behaviour depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. Figure 1.5 illustrates that a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings depending on the context.

Fig. 1.5



Polymorphism

Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the

call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime. It is associated with polymorphism and inheritance. A procedure call associated with a polymorphic reference depends on the dynamic type of that reference.

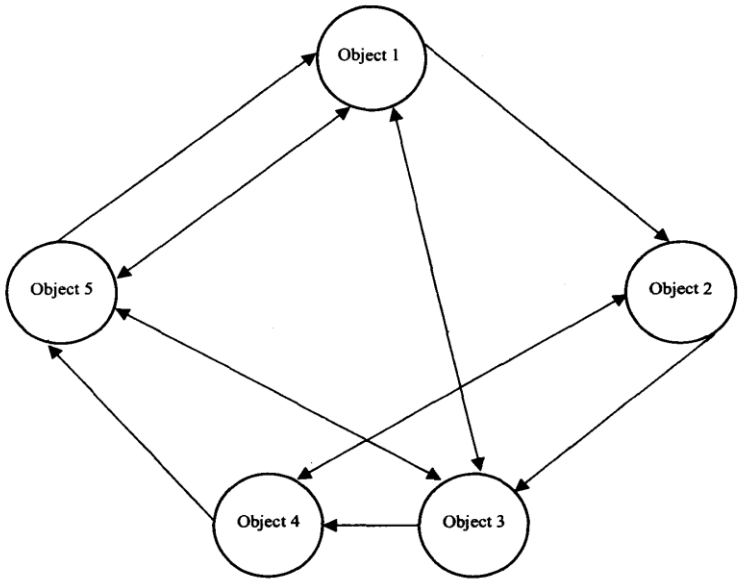
Message Communication

An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:

1. Creating classes that define objects and their behaviour.
2. Creating objects from class definitions.
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another as shown in Fig. 1.6. The concept of message passing makes it easier to talk about building systems that directly model or simulate their real world
counterparts.

Fig. 1.6



Network of objects communicating between them

1.(b) How Java is different from C and C++?

Java and C

Java is a lot like C but the major difference between Java and C is that Java is an object-oriented language and has mechanism to define classes and objects. In an-effort to build a simple and safe language, the Java team did not include some of the C features in Java.

- Java does riot include the C unique statement keywords goto, sizoof, and typedef.
- Java does not contain the data types struct, union and enum .
- ' Java does not define the type modifiers keywords auto, extern, register, signed, and unsigned.
- Java does not support an explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use # define, # include, and # ifdef statements.

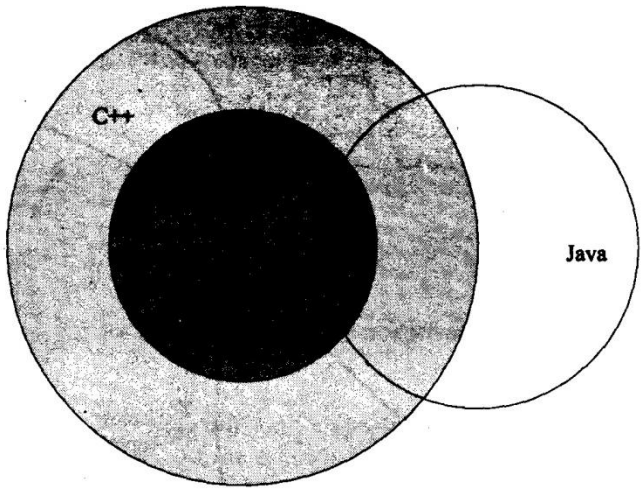
• Java does not support any mechanism for defining variable arguments to functions.
• Java: requires that the functions with no arguments must be declared with empty parenthesis and not with the void keyword as done in C.
• Java adds new operators such as instanceof and> > >.
• Java adds labelled break and continue statements.
• Java adds many features required for object-oriented programming.

Java and C++

Java is a true object-oriented language while c++ is basically C with object-oriented extension. That is what exactly the increment operator ++ indicates. C++ has maintained backward compatibility with C. It is therefore possible to write an old style C program and run it successfully under C++. Java appears to be similar to C++ when we consider only the "extension" part of C++. However, some object-oriented features of C++ make the C++ code extremely difficult to follow and maintain. Listed below are some major C+.. + features that were intentionally omitted from Java or significantly modified .
.• Java does not support operator overloading.
• Java does not have template classes as in C++ .
•. Java does not support multiple inheritance of classes. This is accomplished using a new feature called "interface".
• Java does not support global variables. Every variable and method is declared within a class and forms part' of that class.
• Java does not use pointers.
• Java has replaced the destructor function with a finalize() function.
• There are no header files in Java.

Java also adds some new features. While C+ + is a superset of C, Java is neither a superset nor a subset of C or C+ +. Java may be considered as a first cousin of C+ + and a second cousin of C as illustrated in Fig. 2.1. Amore detailed discussion on the differences between C+ + and Java is available in Appendix: C.

Fig. 2.1



Overlapping of C, C++, and Java

2.(a) Describe the structure of typical Java program.

Java program may contain many classes of which only one class defines a main method. Classes contain data members that operate on the data members of the class. Methods may contain data type declarations and executable statements. To write Java program, we first define classes and put them together. A Java program may contain one or more sections as shown below:

Documentations section
Package Statement
Import Statements
Interface Statements
Class Definitions

Main method Class
{
Main Method Definition
}

Documentation Section

The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer at a larger stage. Comments must explain *why* and *what* of classes and *how* of algorithms. This would greatly help in maintaining the program. In addition to the two style of comments discussed earlier, Java also uses a third style of comment /**…..*/ known as *documentation comment.* This form of comment is used for generating documentation automatically.

Package Statement

The first statement allowed in a Java file is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package. Example:

package student;

The package statement is optional. That is, our classes do not have to be part of a package.

Import Statements

The next thing after a package statement (but before any class definitions) may be a number of import statements. This is similar to the #include statement in C. Example:

```
import student. test;
```

This statement instructs the interpreter to load the test - class contained in the package student. Using import statements, we can have access to classes that are part of other named packages.

Interface Statements

An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance feature in the
program.

Class Definitions

A Java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program. These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of the problem.

Main Method Class

Since every Java stand-alone program requires a main method as its starting point, this class is the essential part of a Java program. A simple Java program may contain only this part. The main method creates objects of various classes and establishes communications between them. On reaching the end of main, the program terminates and the control passes back to the operating system.

2.(b) Briefly explain the features of Java.

The inventors of Java wanted to design a language which could offer solutions to some of the problems encountered in modern programming. They wanted the language to be not only reliable, portable and distributed but also simple, compact and interactive. Sun Microsystems officially describes Java with the following attributes:

- Compiled and Interpreted
- Platform-Independent and Portable
- Object-Oriented
- Robust and Secure
- Distributed
- Familiar, Simple. and Small
- Multithreaded and Interactive
- High Performance
- Dynamic and Extensible

Although the above appears to be a list of buzzwords, they aptly describe the full potential of the language. These features have made Java the first application language of the World Wide Web. Java will also become the premier language for general purpose stand-alone applications.

Compiled and Interpreted

Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as *bytecode* instructions. Bytecodes are' not machine instructions and Therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. We can thus say that Java is both a compiled and an interpreted language.

Platform-Independent and Portable

The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming on Internet which interconnects different kinds of systems worldwide. We can download a Java applet from a remote computer onto our local system via Internet and execute it locally. This makes the ,Internet an extension of the user's basic system providing practically unlimited number of accessible applets and applications.

Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine- independent .

Object-Oriented

Java is a true object-oriented language. Almost everything in Java is an *object.* All program code and data reside within objects and classes. Java comes with an extensive set of *classes,* arranged in *packages,* that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

Robust and Secure

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage-collected language relieving the programmers virtually all memory management problems. Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system. Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources is everywhere. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

Distributed

Java is designed as a distributed language for creating application on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

3.(a) With example explain command line arguments in Java.

There may be occasions when we may like our program to act in a particular way depending on the input provided at the time of execution. This is achieved in Java programs by using what are known as command line arguments. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution. It may be recalled that was invoked for execution at the command line as follows:

java Test

Here, we have not supplied any command line arguments. Even if we supply arguments, the program does not know what to do with them.
We can write Java programs that can receive and use the arguments provided in the command line. Recall the signature of the main( ) method used in our earlier example programs:
public static void main (String args[ ])
As pointed out earlier, args is declared as an array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements. We can simply access the array elements and use them in the program as we wish. For example, consider the command line
java Test BASIC FORTRAN C++ Java

```
/*
* This program uses command line
* arguments as input.
*/
Class ComLineTest
{
public static void main(String args[ ])
{
int count, i=O;
String string;
count = args.length;
System.out.println("Number of arguments = " + count);
while (i < count)
{
string = args[i];
i = i + 1;
System.out.println(i+ " : " + "Java is " +string+ "!");
}
}
}
```

java **ComLineTest** Simple Object Oriented Distributed Robust Secure
Portable. Multithreaded Dynamic Upon execution, the command line arguments Simple, Object_Oriented, etc. are passed to the program through the array args as discussed earlier. That i& the element args[0] contains Simple, args[1] contains Object_Oriented, and SO on. These elements are accessed using the loop variable i as an index like
name = args[i]
The index i is incremented using a while loop until all the arguments are accessed. The number of arguments is obtained by statement:
count = args.length;
The output of the program would be as follows:
Number of arguments = 8

1 : Java is Simple!
2 : Java is Object_Oriented!
3 : Java is Distributed!
4 : Java is Robust!
5 : Java is Secure!
6 : Java is Portable!
7 : Java is Multithreaded!
8 : Java is Dynamic!
Note how the output statement concatenates the strings while printing.

3.(b) With example explain looping statements in Java.

The Java language provides for three constructs for performing loop operations. They are:
1 The while statement
2 The do statement
3 The for statement

1. The while statement
Syntax:

```
Initialization
While(test condition)
{
    Body of the loop
}
```

The while is an *entry-controlled* loop statement. The *test* condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process of repeated _execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of' the loop. The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements. However, it is a good practice to use braces even if the body has only one statement.
Consider the following code segment:

```
..... ..... .
sum = 0;
n = 1;
while (n <= 10)
{
sum = sum + n * n;
n = n+1;
}
System.out.println ("Sum = "+ sum);
..... .
```

The body of the loop is executed 10 times for n = 1,2, ... , 10 each time adding the square of the value of n, which is incremented inside the loop. The test condition may also be written as i< 11,-the-result would be the same. Program 7.1 illustrates the use of the while for reading a string of characters from the keyboard. The loop terminates when c = '\ n', the newline character.

```
class WhileTest
{
public static void main(String args[ ])
{
StringBuffer string = new StringBuffer();
char c;
System.out.println("Enter a string H);
try
{
while ( (c = ,(char)System.in.read(» != '\n')
{
string. append (c); / / Append character
}
}
catch (Exception e)
{
System.out.println("Error in input");
}
System.out.println(" You have entered");
System.out.println(string);
}
}
```

Given below is the output of Program 7.1:

```
Enter a string
Java is a true Object-Oriented Language
You have entered ...
Java is a true Object-Oriented Language
```

THE DO STATEMENT

The **while** loop construct that we have discussed in the previous section makes a test condition *before* the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the do statement. This takes the form.

```
Initialization;
Do
{
        Body of the loop
}
While(test condition);
```

On reaching the do statement, the program proceeds to evaluate the body of the loop first. At the end of the loop, the *test condition* in the while statement is evaluated. If the condition is

true, the program continues to evaluate the body of the *loop* once again. This process continues as long as the *condition* is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement. Since the *test condition* is evaluated at the bottom of the loop, the do ••••while construct
provides an *control1ed* loop and therefore the body of the loop is always executed at least once.
Consider an example:

```
i = 1;
sum = 0;
do
{
sum = sum + i;
i = i+2;
}
while(sum < 40 || i < 10);
```

THE FOR STATEMENT

The for loop is another *entry- controlled* loop that provides a more concise loop control structure. The general form of the for loop is

for(initialization;test condition;increment)

{

Body of the loop

}

The execution of the for statement is as follows:
1. *Initialization* of the *control 'Variables* is done first, using assignment statements such as i·= 1 and count = O. The variables i and count are known as loop-control variables.
2. The value of the control variable is tested using the *test* condition .The test condition is a relational expression, such as i <10 that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is *incremented* using an assignment statement such as i = i+ 1 and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

```
Program to find power of 2 using for loop
class ForTest
{
public static void main (String args[ ])
```

```
{
long p;
int n;
double q;
System.out.println(~2 to power -n n 2 to power n");
p = 1;
for (n = 0; n < 10; ++n)
{
if (n == 0)
p = 1;
else
p = p * 2;
q = 1.0 / (double)p;

System.out.println(" " + q'+ " " + n +"" + p);
}
}
}
```

3.(c) How java is Architectural neutral ? Explain .

The Java compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called the *Java Virtual Machine* and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. Figure 3.6 illustrates the process of compiling a Java program into bytecode which is also referred to as *virtual machine code.*

Fig. 3.6

Process of compilation

The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in Fig. 3.7. Remember that the interpreter is different for different machines.

Fig. 3.7

Process of converting bytecode into machine code

4.(a) What are the differences between a class and inheritance? Give the syntax of class and inheritance.

class	Inheritance
The member of a class can be constant or variable.	The member of an interface are always declared as constants, i.e. their value are final.
The class definition can contains the code for each of its method i.e. the methods can be abstract or non -abstract.	The method in an interface are abstract in nature there is no code associated with them , it is later defined by class that implements the interface.
It can be instantiated by declaring objects.	Cannot be used to declare objects it can only been inherited by a class .
It can be use various access specifier like public, private, or protected.	It can only use a public access specifier.

Syntax:
Class classname
{
Type instance_variable
{
(parameter list1)
//body of the method
}
Parameter_list2
{
//body of the method
}
}
	Syntax:
	Interface name of the interface
	{
	Final fields;
	Abstract method
	Declaration;
	}

4.(b) How multiple inheritance is demonstrated in Java. Explain with example.

IMPLEMENTING INTERFACES

Interfaces are used as "superclasses" whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. This is done as follows:

```
class classname implements interfacename
{
    body of classname
}
```

Here the class *classname* "implements" the interface *interfacename*. A more general form of implementation may look like this:

```
class classname  extends superclass
      implements interface1,interface2, ......
{
      body of classname
}
```

Implementation of interfaces as class types is illustrated by Program 10.1. In this program, first we create an interface Area and implement the same in two different classes, Rectangle and
Circle. We create an instance of each class using the new operator. Then we declare an object of
type Area, the inter face class. Now, we assign the reference to the Rectangle object rect to area.
When we call the compute method of area, the compute method of Rectangle class is invoked.
We repeat the same thing with the Circle object.

```
// InterfaceTestJava
interfaceArea // Interface defined
{
final static float pi =  3.14F;

float compute (float X,  float y);
}
class Rectangle implements Area // Interface imptemented
{
public float compute (float X,  float y)
{
return (x*y) ;
}
}
class Circle implements  Area // Another implementation
{
public float compute (float X,  float y)
{
return (pi *x*x) ;
```

```
}
}
class InterfaceTest
{
public static void main (String args[ ])
{
Rectangle rect =  new Rectangle( );
Circle cir =  new Circle( );
Area area; // Interface object
area =  rect;
System.out.println("Area of Rectangle =  "+ area.compute(10,20»;
area =  cir;
System.out.println("Area of Circle =+ area.compute(l0,0);
}
}
```

The Output is as follows:
Area of Rectangle = 200
Area of Circle = 314

4.(c) Write a note on visibility control in Java.

VISIBILITY CONTROL

We stated earlier that it is possible to inherit all the members of a class by a subclass using the keyword extends. We have also seen that the variables and methods of a class are visible everywhere in the program. However, it may be necessary in some situations to restrict the access to certain variables and methods from outside the class. For example, we may not like the objects of a class directly alter the value of a variable or access a method. We can achieve this in Java by applying *visibility modifiers* to the instance variables and methods. The visibility modifiers are also known as *access modifiers.* Java provides three types of visibility modifiers: public, private and protected. They provide different levels of protection as described below.

public Access

Any variable or method is visible to the entire class in which it is defined. What if we want to make it visible to all the classes outside this class? This is possible by simply declaring the variable or method as public. Example: public `int number;`
public `void sum( )` { ..... }
A variable or method declared as public has the widest possible visibility and accessible everywhere. In fact, this is what we would like to prevent in many programs. This takes us to the next levels of protection.

friendly Access

In many of our previous examples, we have not used public modifier, yet they were still accessible in other classes in the program. When no access modifier is specified, the member defaults to a limited version of public accessibility known as "friendly" level of access. The difference between the "public" access and the "friendly" access is that the public modifier makes fields visible in all classes, regardless of their packages while the friendly access

makes fields visible only in the same package, but not in other packages. . A package in Java is similar to a source file in C.

protected Access

The visibility level of a "protected" field lies in between the public access and friendly access. That is, the protected modifier makes the fields visible not only to all classes and subclasses in the same package but also to subclasses in other packages. Note that non-subclasses in other packages cannot access the "protected" members.

private Access

private fields enjoy the highest degree of protection. They are accessible only within their own class. They cannot be inherited by subclasses and therefore not accessible in sub classes. A method declared as private behaves like a method declared as final. It prevents the method from being sub classed. Also note that we cannot override a non-private method in a subclass and then make it private.

private protected Access

A field can be declared with two keywords private and protected together like:

```
private protected int codeNumber ;
```

This gives a visibility level in between the "protected" access and "private" access. This modifier makes the fields visible in all subclasses regardless of what package they are in. Remember, these fields are not accessible by other classes in the same package. Table 8.1 summarises the visibility provided by various access modifiers.

Table 8.1 Visibility of Fields in a Class

Access modifier → / Access location ↓	public	protected	friendly (default)	private protected	private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

5.(a) What is packages? Explain some Java package.

Packages are Java's way of grouping a variety of classes and/or interfaces together. The grouping is usually done according to functionality. In fact, packages act as "containers" for classes.

Package name	Contents
java.lang	Language support classes. These are classes that Java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.
java.util	utility classes such as vectors, hash tables. random numbers, date, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.applet	Classes for creating and implementing applets.

5.(b) Write a program to implement all string operations.

```
/* Program to Implement All String Operations*/
class StringOpe
{
public static void main(String args[])
{
String s1,s2,s3,s4;
s1="Internet";
s2="Programming";
int k,m,n;
System.out.println("\nThe length of First String is : "+s1.length());
System.out.println("\nThe length of Second String is : "+s2.length());
System.out.println("\nThe First character of s1 is : "+s1.charAt(0));
System.out.println("\nThe First character of s2 is : "+s2.charAt(0));
System.out.println("\nThe Concatenation of 2 String is : "+s1+"
"+s2);
if(s1.equals(s2))
System.out.println("\nBoth Strings are Equal");
else
System.out.println("\nBoth Strings are Not Equal");
k=s1.compareTo(s2);
if(k==0)
System.out.println("\nBoth are Equal");
else if(k>0)
System.out.println("\nFirst String is bigger ");
else
System.out.println("\nSecond String is bigger ");
s3=s1.toLowerCase();
System.out.println("\nThe "+s1+" converted to Lower Case Letters :
"+s3);
s4=s2.toUpperCase();
```

```
System.out.println("\nThe "+s2+" converted to Upper Case Letters :
"+s4);
}
}
```

Output

```
F:\Programs>javac StringOpe.java
F:\Programs>java StringOpe
The length of First String is : 8
The length of Second String is : 11
The First character of s1 is : I
The First character of s2 is : P
The Concatenation of 2 String is : Internet Programming
Both Strings are Not Equal
Second String is bigger
The Internet converted to Lower Case Letters : internet
The Programming converted to Upper Case Letters : PROGRAMMING
```

5.(c) How vector class differs from an arrays?

Vector class contained in the java.util package. This class can be used to create a generic dynamic array known as vector that can hold objects of any type and any number. The objects do not have to be homogeneous . Arrays can be easily implemented as vectors. Vectors are created like arrays as follows:

```
vector  invect = new Vector();   //declaring without size.
Vector list      = new Vector();   //declaring size
```

Note that a vector can be declared without specifying any size explicitly. A vector can accommodate an unknown number of items. Even, when a size is specified, this can be overlooked and a different number of items may be put into the vector. Remember, in contrast, an array must always have its size specified.

Vectors possess a number of advantages over arrays.

1. It is convenient to use vectors to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

A major constraint in using vectors is that we cannot directly store simple data types in a vector; we can only store objects.

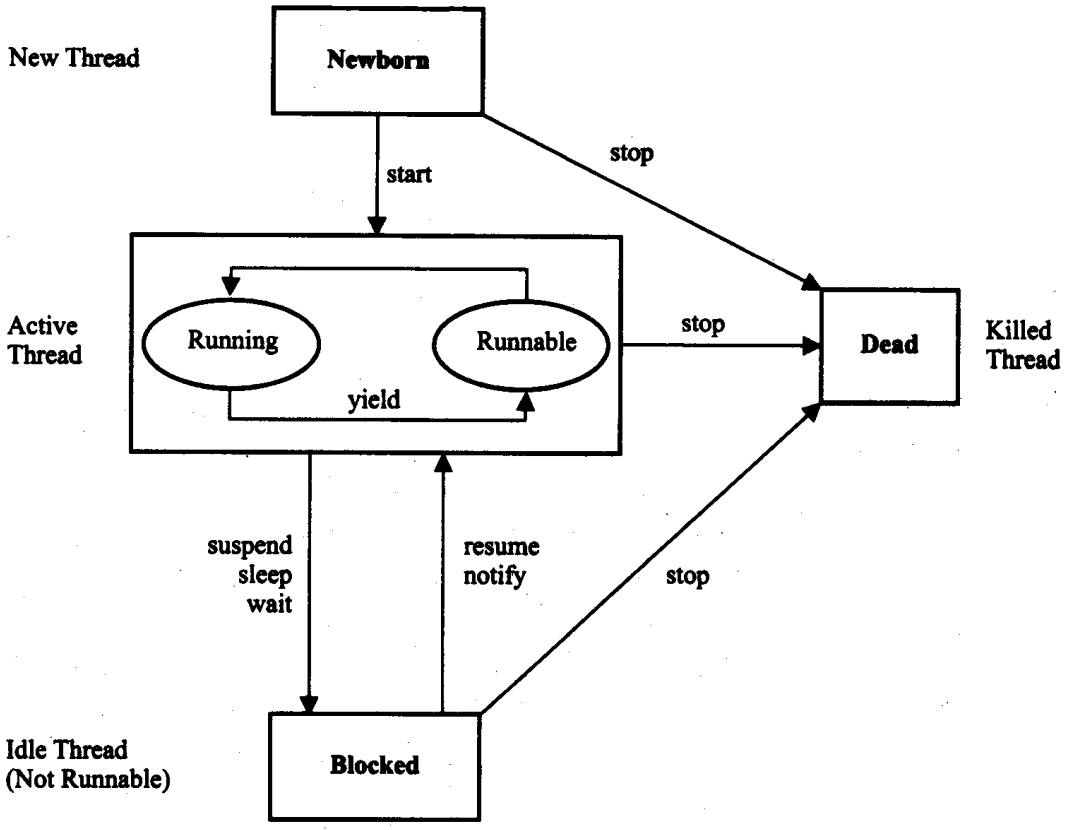
6.(a) With a neat diagram explain the life cycle of thread.

During the life time of a thread, there are many states it can enter. They include:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state ..
5. Dead state

A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in Fig. 12.3.

Fig. 12.3



State transition diagram of a thread

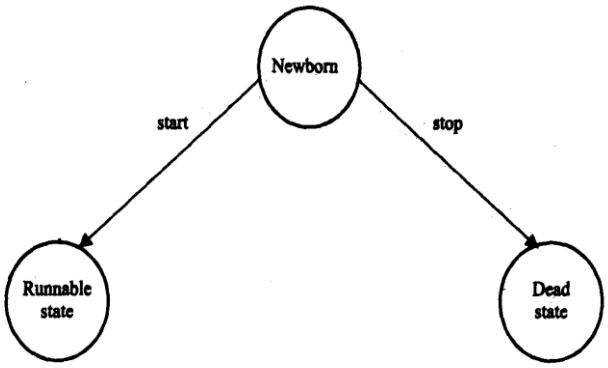
Newborn State

When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state, we can do only one of the following things with it:

• Schedule it for running using start( ) method.
• Kill it using stop( ) method.

If scheduled, it moves to the runnable state (F'ig. 12.4).1fwe attempt to use any other method at this stage, an exception will be thrown.

Fig. 12.4



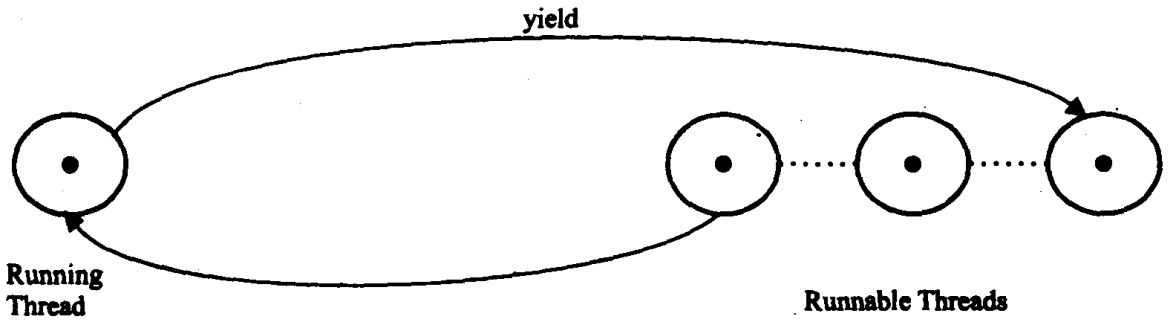
Scheduling a newborn thread

Runnable State

The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. That is, the thread has joined the queue of threads that are

waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion, i.e., first-come, first-serve manner. The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as *time-slicing.*
However, if we want a thread to relinquish control to another thread of equal priority before its turn comes, we can do so by using the yield( ) method (Fig. 12.5).

Fig. 12.5



Relinquishing control using yield( ) method

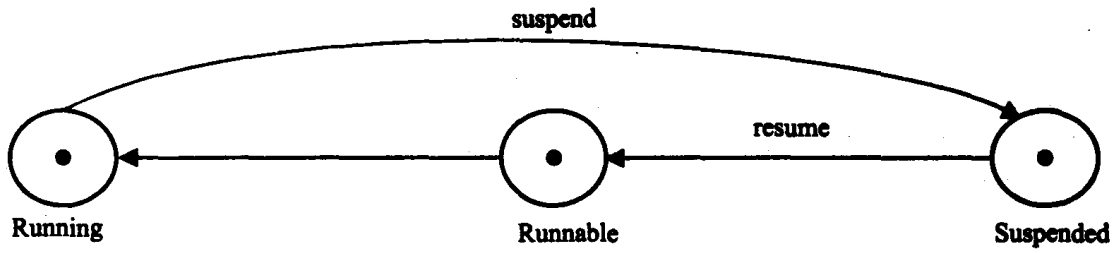
Running State
Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it· is preempted by a higher priority thread. A running thread may relinquish its control in one of the following situations.

Blocked State
A thread is said to be *blocked* when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

Dead State
Every thread has a life cycle. A running thread ends its life when it has completed executing its run()method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. A thread can be killed as soon it is born, or while it is running, or even when it is in "not runnable" (blocked) condition.

1. It has been suspended using **suspend( )** method. A suspended thread can be revived by using the **resume( )** method. This approach is useful when we want to suspend a thread for some time due to certain reason, but do not want to kill it.

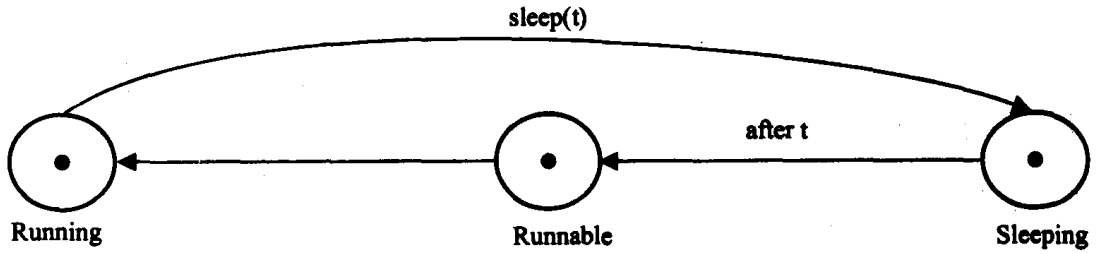
Fig. 12.6



Relinquishing control using suspend( ) method

2. It has been made to sleep. We can put a thread to sleep for a specified time period using the method **sleep(*time*)** where *time* is in milliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.

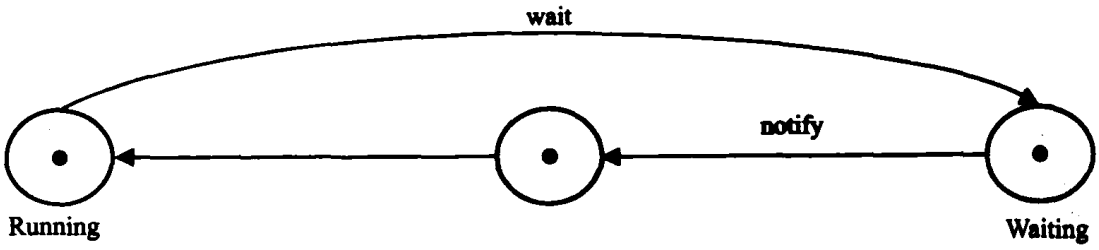
Fig. 12.7



Relinquishing control using sleep( ) method

3. It has been told to wait until some event occurs. This is done using the **wait( )** method. The thread can be scheduled to run again using the **notify( )** method.

Fig. 12.8



Relinquishing control using wait( ) method

6.(b) Explain thread priorities with an example.

THREAD PRIORITY

In Java, each thread is assigned a priority, which affects the order in which it is scheduled for running. The threads that we have discussed so far are of the same priority. The threads of the same priority are given equal treatment by the Java scheduler and, therefore, they share the processor on a first-come, first-serve basis.

Java permits us to set the priority of a thread using the setPriority( ) method as follows:

ThreadName.setPriority(int Number);

The int Number is an integer value to which the thread's priority is set. The Thread class defines several priority constants:
MIN PRIORITY = 1
NORM_PRIORITY = 5
MAXPRIORITY = 10
The int Number may assume one of these constants or any value between 1 and 10. Note that the default. setting is NORM_RIORITY. Most user-level processes should use NORM_PRIORITY, plus or minus 1. Back-ground tasks as network I/O and screen repainting should use a value very near to the lower limit. We should be very cautious when trying to use very high priority values. This may defeat the very purpose of using multithreads. By assigning priorities to threads, we can ensure that they are given the attention (or lack of it) they deserve. For example, we may need to answer an input as quickly as possible. Whenever multiple threads are ready for execution, the Java system chooses the highest priority thread and executes it. For a thread of lower priority to gain control, one of the following things should happen:

1. It stops running at the end of run().
2. It is made to sleep using sleep( ).
3. It is told to wait using wait( ).
However, if another thread of a higher priority comes along, the currently running thread will be *preempted* by the incoming thread thus forcing the current thread to move to the runnable state. Remember that the highest priority thread always preempts any lower priority threads.

```
class A extends Thread
{
public void run( )
{
System.out.println(“threadA started");
for(int i-l; i<=4; i++)
{
System.out.println("\tFrom Thread A : i =" +i);
}
System.out.println("Exit from A");
}
}
class B  extends Thread
{
public void run( )
{
System.out.println("threadB started");
for(int j=l; j<=4; j++)
{ System.out.println("\tFrom Thread B : j = " +j);
}
```

```
System.out.println("Exit from B ");
}
}
class C extends Thread
{
public void rune)
{
System.out.println("threadC started");
for(int k=l; k<=4; k++)
{
System.out.println("\tFrom Thread C : k •.." +k);
}
System.out.println("Exit from C ");
}
}
class ThreadPriority
{
public static void main (String args[ ])
{
A threadA = new A( );
B threadB = new B( );
C threadC = new C( );
threadC.setPriority(Thread.MAX_PRIORITY);
threadB.set Priority (threadA.getPriority( )+1);
threadA.setPriority(Thread.MIN_PRIORITY);
System.out.println("Start thread A");
threadA.start( );
System.out.printlri("Start thread B");
threadB.start( );
System.out.println("Start thread CW
);
threadC.start.( );
System.out.println("End of main thread-);
}
}
```

Output of Program 12.3:
Start thread A
Start thread B
Start thread C
threadB started
From Thread B : j. = 1
From Thread B : j =. 2
threadC started
From Thread C : k = 1
From Thread C : k = 2
From Thread C : k = 3
From Thread C : k = 4

Exit from C
End of main thread
From Thread B : j = 3
From Thread B : j 7 4
Exit from B
threadA started
From Thread A : i = 1
From Thread A : i = 2
From Thread A : i = 3
From Thread A : i = 4
Exit from A

7.(a) Explain applets in Java.

Applets are small Java programs that are primarily used in Internet computing. They can be transported over the Internet from one computer to another and run using the Applet Viewer or
any Web browser that supports Java. An applet, like any application program, can do many things for us. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games.

Local and Remote Applets

We can embed applets into Web pages in two ways. One, we can write our own applets and embed them into Web pages. Second, we can download an applet from a remote computer system and then embed it into a Web page. An applet developed locally and stored in a local system is known as a *local applet* When a Web page is trying to find a local applet, it does not need to use the Internet and therefore the local system does not require the Internet connection. It simply searches the directories in the local system and locates and loads the specified applet.
A *remote applet* is that which is developed by someone else and stored on a remote computer connected to the Internet. If our system is connected to the Internet, we can download the remote applet onto our system via the Internet and run it .
In order to locate and load a remote applet, we must know the applet's address on the Web. This address is known as *Uniform Resource Locator (URL)* and must be specified in the applet's HTML document as the value of the CODEBASE attribute. Example

```
CODEBASE = http://www.netserve.com/applets
```

In the case of local applets, CODEBASE may be absent or may specify a local directory. In this chapter we shall discuss how applets are created, how they are located in the Web documents and how they are loaded and run in the local computer.
The steps involved in developing and testing an applet are:
1. Building an applet code (.java file)
2. Creating an executable applet (.class file)
3. Designing a Web page using HTML tags
4. Preparing <APPLET> tag
5. Incorporating <APPLET>tag into the Web page
6. Creating HTML file

7. Testing the applet code

7.(b) Write a Java program to demonstrate exception.

```
/* Program to find Sort list of elements in Ascending and Descending orderand show
Exception Handling Mechanism*/
import java.io.*;
class ExpDemo
{
public static void main(String args[])
{
try
{
DataInputStream di=new DataInputStream(System.in);
System.out.println("Enter limit");
int n=Integer.parseInt(di.readLine());
int i;
int a[]=new int[50];
System.out.println("Enter array elements");
for(i=0;i<n;i++)
{
try{
a[i]=Integer.parseInt(di.readLine());
}
catch(NumberFormatException e)
{
System.out.println("Numbers only....!!!");
i--;
}
}
System.out.println("The array elements are");
for(i=0;i<n;i++)
System.out.println(a[i]);
for(i=0;i<n;i++)
{
for(int j=i+1;j<n;j++)
{
if(a[i]>a[j])
{
int temp=a[i];
a[i]=a[j]; a[j]=temp;
}
}
}
System.out.println("The Sorted elements in ascending order are");
for(i=0;i<n;i++)
System.out.println(a[i]);
System.out.println("\nThe Sorted elements in descending order are");
```

```
for(i=n-1;i>=0;i--)
System.out.println(a[i]);
}
catch(IOException e)
{
System.out.println(e);
}
catch(Exception e)
{
System.out.println("Error Occurred");
}
}
}
```

7.(c) Explain input/output stream classes.

Input steam classes: It extracts(i.e. read) data from the source(file) and sends it to the program. Input stream classes that are used to read 8- bit bytes include a super class known as InputStream and a member of subclasses for supporting various input -related functions.

The super class inputStream is an abstract class, and therefore, we cannot create instances of this class. Rather, we must use the subclasses that inherit from this class. The InputStream class defines methods for performing input functions such as

- Reading bytes
- Closing bytes
- Marking positions in streams
- Skipping ahead in a stream
- Finding the number of bytes in a stream.

Output steam classes: It takes data from the program and sends (i.e. writes) it to the destination(file). Output stream classes are derived from the base class OutputStream . The output stream is also an abstract class and therefore we cannot instantiate it. The several sub classes of the OutputStream can be used for performing the output operations. The output Stream includes the methods that are designed to perform the following tasks:

- Writing bytes
- Closing streams
- Flushing streams

8. Write short notes on any four:

a) Java tokens

A Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements Most statements contain expressions, which describe the actions carried out on data. Smallest individual units in a

program are known as tokens. The compiler recognizes them for building up expressions and statements.
Java language includes five types of tokens. They are:
• Reserved Keywords
• Identifiers
• Literals
• Operators
• Separators

b) Applications of OOP's

Real-business systems are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in this type of applications because it can simplify a complex problem. The promising areas for application of OOP includes:
• Real-time systems
• Simulation and modelling
• Object-oriented databases
• Hypertext, hypermedia and expert text
• AI and expert systems
• Neural networks and parallel programming
• Decision support and office automation systems
• CIM/CAD/CAD system

c) Java operators

Java supports a rich set of operators. We have already used several of them, such as =, +, - and *. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.
Java operators can be classified into a number of related categories as below:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators
- Conditional operators.
- Bitwise operators
- Special operators

d) Constructors

We know that all objects that are created must be given initial values. We have done this earlier using two approaches. The first approach uses the dot operator to access the instance variables and then assigns values to them individually. It can be a tedious approach to initialize all the variables of all the objects. The second approach takes the help of a method like getData to initialize each object individually using statements like,

```
rectl.getData(15,lO);
```

It would be simpler and more concise to initialize an object when it is first created. Java supports a special type of method, called a *constructor,* that enables an object to initialize itself when it is created. Constructors have the same name as the class itself. Secondly, they do not specify a return type, not even void. This is because they return the instance of the class itself. Let us consider our Rectangle class again. We can now replace the getData method by a constructor method as shown below:

```
class Rectangle
{
int length ;
int width ;
Rectangle(int x, int y) // Constructor method
{
length = x ;
width = Y ;
}
int rectArea( )
{
return(length * width);
}
}
```

e) I-O Package (2017 – 5A)

f) Java History

Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991. Originally called ***Oak*** by James Gosling, one of the inventors of the language, Java was designed for the development of software for consumer electronic devices like TVs, VCRs, toasters and such other electronic machines. This goal had a strong impact on the development team to make the language simple, portable and highly reliable. The Java team which included Patrick Naughton discovered that the existing languages like C and C+ + had limitations in terms of both reliability and portability. However, they modelled their new language Java on C and C++ but removed a number of features of C and C++ that were considered as sources of problems and thus made Java a really simple, reliable, portable, and powerful language.