Biyani's Think Tank

Concept based notes

Core Java Programming

(BCA Part-III)

Neha Jain
Deptt. of Information Technology
Biyani Girls College, Jaipur

Published by :
Think Tanks
Biyani Group of Colleges

Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

First Edition: 2016
Price:

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the “Teach Yourself” style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Syllabus

B.C.A. Part-III

Core Java Programming

Unit – I

Overview of Object Oriented Concepts in Java.

Introduction: getting and installing the Java Development Kit, Java features like security, portability, byte code, java virtual machine, object oriented, robust, multithreading, architectural neutral, distributed and dynamic, Java programming language structure and syntax, control statements (The If statement, Logical Operators, The Conditional Operator, the Switch Statement, Variable Scope, Loops).

Unit – II

Java arrays, Java Strings, Operations on Strings and String Buffer Objects, Class, Objects, Methods and Problem solving using classes, objects and relationships.

Inheritance, types of Inheritance, packages and interface, exception handling.

Unit - III

Java utilities like java.lang, java.util, java.io, GUI in Java using AWT and Swing, Event Handling Mechanisms, AWT based effective GUI in Java : Detailed overview of AWT classes, Graphics primitives and UI Components, Layout features, Standalone GUI applications, Layout Managers, Implementation of event driven mechanism, Delegation of even model, Listeners and Adapters, Inner classes.

Unit - IV

Applets : Introduction to Applet coding, Applet life cycle, Graphis facility, Color and Font, Passing parameters to applets, Apletcontext, Inter Applet Communication.

Threading in Java : Fundamentals of Multi-threading Java coding with Thread classes, thread Management in Java, Implicit wait, Using Runnable interface, Thread Management in Java, Implicit wait, Using Runnable interface, Thread Synchronization, Inter thread communication.

Unit – V

Overview of Networking in Java : URL class and its usage through connection, Sockets based connectivity, TCP/IP Sockets and server sockets, Datagram Sockets. Collections in Java-Array List, stack, queue, Hash table, Collection class hierarchy, JDBC and Jar files.

Content

□ □ □

Chapter-1

Introduction of Java

Q.1 Write a note on History of Java?

Ans.: **History of Java:** James Gosling initiated Java language project in June 1991 for use in one of his many set top box projects. The language, initially called 'Oak' after an oak tree that stood outside Gosling's office, also went by the name 'Green' and ended up later being renamed as Java, from a list of random words. Sun released the first public implementation as Java 1.0 in 1995. It promised Write Once, Run Anywhere (WORA), providing no-cost run-times on popular platforms. On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL). On 8 May, 2007, Sun finished the process, making all of Java's core code free and open source, aside from a small portion of code to which Sun did not hold the copyright.

Q.2. What is java? Explain in detail.

Ans.: In the Java whitepaper (available for Sun's website http://java.sun.com), Sun describes Java as follows: Java is a simple, object-oriented, distributed, interpreted, robust, secure, architectural neutral, portable, high performance, multithreaded, and dynamic language. While this seems like a string of buzzwords, but the fact is that these buzzwords actually describe the language and its features. To get a feeling of why Java is important and interesting, let's look at the features behind some of these buzzwords. Java is Simple One of the design goals of Java was to make it much easier to write bug-free code. In order to help programmers with this, the language has to be simple. The simplicity of Java makes it fun to program with, and its programs are easy to write and read. If you have programmed in C or C++, you would know that half of the bugs in your programs are related to memory allocation. With Java you will not have this problem as the Java runtime environment provides automatic memory allocation and garbage collection. Java is Object-Oriented; Object-Oriented (OO) programming was the catch phrase of the 1990's. As a marketing strategy, many companies claim that their software is object oriented, when in fact they are not. An earlier

computer scientist (S. King?) claimed that if someone wanted to sell his cat, he should not say it is clean, nice, beautiful….etc but rather it is object-oriented. The next time you go buy a toaster, make sure it is object-oriented! You will do OO programming this term and more of it next term…. Java is Distributed As I mentioned earlier, the aim of the Green project was to build a distributed system that would allow all consumer electronic devices to talk to one another. Since this was a design goal, Java provides a lot of high-level support for networking. You will see more of this next year! Java is Interpreted Java is an interpreted language. This means that Java programs are not compiled into machine platform-dependant language. But rather they are compiled into byte-codes for the Java Virtual Machine (JVM). To run Java programs, you use the Java interpreter to run the Java byte-codes. Java byte codes are platform-independent, which means they can run on any platform with a Java interpreter. One catch of interpreted code is that it is a bit slower than machine code when it runs. However, with all the Java optimization techniques and Just In-Time Compilers technology (JIT), Java byte-codes will run as fast as C or C++ compiled code. Java is Robust Buggy software can be written in any language, and Java is no exception. However, Java eliminates certain types of programming errors and that makes it easier to write reliable software. Java is a strongly typed language and that allows for extensive compile-time checking. Also, the fact that Java does not have pointers eliminates another class of memory-related bugs.

Q.3. What are the basic features of Java Language?

Ans.: Java Features: Here we list the basic features that make Java a powerful and popular programming language :

- **Platform Independence :**
  - The *Write-Once-Run-Anywhere* ideal has not been achieved (tuning for different platforms usually required), but closer than with other languages.
- **Object Oriented :**
  - Object oriented throughout - no coding outside of class definitions, including main().
  - An extensive class library available in the core language packages.
- **Compiler/Interpreter Combo :**
  - Code is compiled to bytecodes that are interpreted by a Java virtual machines (JVM).

- This provides portability to any machine for which a virtual machine has been written.
- The two steps of compilation and interpretation allow for extensive code checking and improved security.

- **Robust :**
  - Exception handling built-in, strong type checking (that is, all data must be declared an explicit type), local variables must be initialized.
- **Several dangerous features of C & C++ eliminated:**
  - No memory pointers
  - No preprocessor
  - Array index limit checking
- **Automatic Memory Management**
  - Automatic garbage collection - memory management handled by JVM.
- **Security**
  - No memory pointers
  - Programs runs inside the virtual machine sandbox.
  - Array index limit checking
  - Code pathologies reduced by
    - *bytecode verifier* - checks classes after loading
    - *class loader* - confines objects to unique namespaces. Prevents loading a hacked "java.lang. Security Manager" class, for example.
    - *security manager* - determines what resources a class can access such as reading and writing to the local disk.
- **Dynamic Binding**
  - The linking of data and methods to where they are located, is done at run-time.
  - New classes can be loaded while a program is running. Linking is done on the fly.
  - Even if libraries are recompiled, there is no need to recompile code that uses classes in those libraries.

This differs from C++, which uses static binding. This can result in *fragile* classes for cases where linked code is changed and memory pointers then point to the wrong addresses.

- **Good Performance :**
  - Interpretation of bytecodes slowed performance in early versions, but advanced virtual machines with adaptive and just-in-time compilation and other techniques now typically provide performance up to 50% to 100% the speed of C++ programs.
- **Threading :**
  - Lightweight processes, called threads, can easily be spun off to perform multiprocessing.
  - Can take advantage of multiprocessors where available.
  - Great for multimedia displays.
- **Built-in Networking :**
  - Java was designed with networking in mind and comes with many classes to develop sophisticated Internet communications.

Features such as eliminating memory pointers and by checking array limits greatly help to remove program bugs. The garbage collector relieves programmers of the big job of memory management. These and the other features can lead to a big speedup in program development compared to C/C++ programming.

Q.4. What is JVM? Explain its architecture with diagram.

Ans.: Java Virtual Machine: A **Java Virtual Machine** (**JVM**) is a set of computer software programs and data structures which 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. For example, Ada source code can be compiled to Java bytecode, which may then be executed by a JVM. JVMs can also be released by other companies besides Sun (the

developer of Java) -- JVMs using the "Java" trademark may be developed by other companies as long as they adhere to the JVM specification published by Sun (and related contractual obligations).

The JVM is a crucial component of the Java Platform. Because JVMs are available for many hardware and software platforms, Java can be both middleware and a platform in its own right — hence the expression "write once, run anywhere." 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 unique features as Automated Exception Handling which 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 which implement the Java API (Application Programming Interface). The virtual machine and API have to be consistent with each other and are therefore bundled together as the *Java Runtime Environment*.

Architecture of JVM: JVM has various sub components internally. You can see all of them from the above diagram.

1. Class loader sub system: JVM's class loader sub system performs 3 tasks
a. It loads .class file into memory.
b. It verifies byte code instructions.
c. It allots memory required for the program.

2. Run time data area: This is the memory resource used by JVM and it is divided into 5 parts

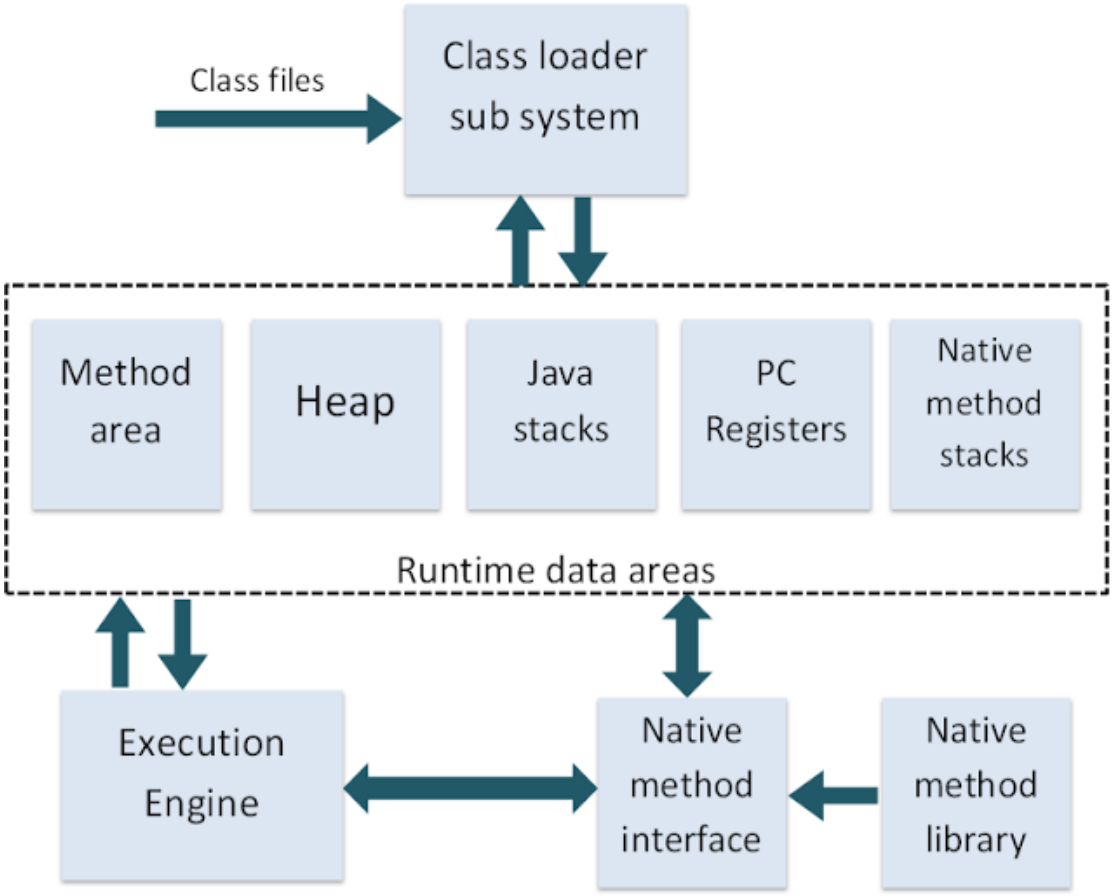
a. Method area: Method area stores class code and method code.

b. Heap: Objects are created on heap.

c. Java stacks: Java stacks are the places where the Java methods are executed. A Java stack contains frames. On each frame, a separate method is executed.

d. Program counter registers: The program counter registers store memory address of the instruction to be executed by the micro processor.

e. Native method stacks: The native method stacks are places where native methods (for example, C language programs) are executed. Native method is a function, which is written in another language other than Java.



JVM Architecture

3. Native method interface: Native method interface is a program that connects native methods libraries (C header files) with JVM for executing native methods.

4. Native method library: holds the native libraries information.

5. Execution engine: Execution engine contains interpreter and JIT compiler, which covert byte code into machine code. JVM uses optimization technique to decide which part to be interpreted and which part to be used with JIT compiler. The HotSpot represents the block of code executed by JIT compiler.

Q.5. What are the Fundamental JDK Tools?

Ans.: These tools are the foundation of the JDK. They are the tools you use to create and build applications.

Tool Name	Brief Description
javac	The compiler for the Java programming language.
java	The launcher for Java applications. In this release, a single launcher is used both for development and deployment. The old deployment launcher, **jre**, is no longer provided.
javadoc	API documentation generator. See Javadoc Tool page for doclet and taglet APIs.
apt	Annotation processing tool. See Annotation Processing Tool for program annotation processing.
appletviewer	Run and debug applets without a web browser.
jar	Create and manage Java Archive (JAR) files. See Java Archive Files page for the JAR specification.
jdb	The Java Debugger. See JPDA for the debugger architecture specifications.
javah	C header and stub generator. Used to write native methods.
javap	Class file disassembler.
extcheck	Utility to detect Jar conflicts.

Chapter-2

Language Features

Q.1 Explain data types in Java?

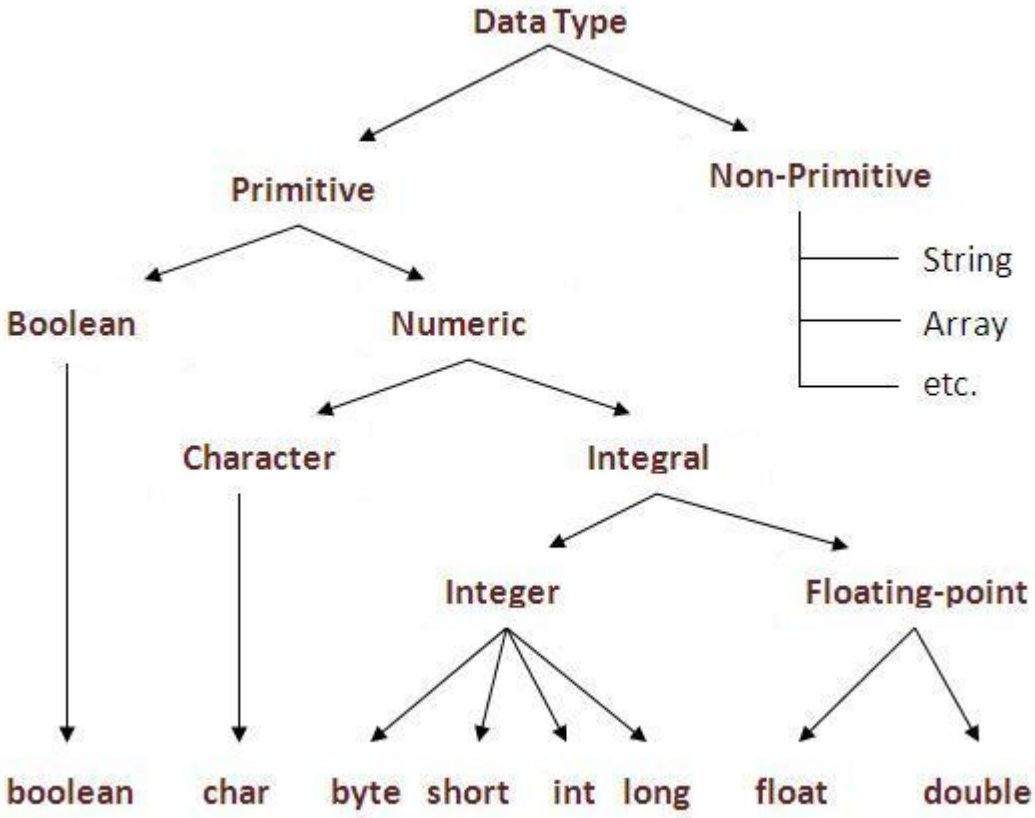
OR

How many primitive data types are there in Java?

Ans.: The Java programming language is strongly-typed, which means that all variables must first be declared before they can be used. This involves stating the variable's type and name:

int gear = 1;

Doing so tells your program that a field named "gear" exists, holds numerical data, and has an initial value of "1". A variable's data type determines the values it may contain, plus the operations that may be performed on it. In addition to int, the Java programming language supports seven other *primitive data types*. A primitive type is predefined by the language and is named by a reserved keyword. Primitive values do not share state with other primitive values.



The eight primitive data types supported by the Java programming language are :

- **Byte** : The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
- **Short** : The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with
- **Int** : The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive). For integral values, this data type is generally the default choice unless there is a reason (like the above) to choose something else
- **Long** : The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum

value of 9,223,372,036,854,775,807 (inclusive). Use this data type when you need a range of values wider than those provided by int.

- **Float** : The float data type is a single-precision 32-bit IEEE 754 floating point As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency.
- **Double** : The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.
- **Boolean** : The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- **Char** : The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the java.lang.String class.

Q.2. Mention the different types of operators in Java?

Ans.: Operators are special symbols that perform specific operations on one, two, or three *operands*, and then return a result.

The operators in the following table are listed according to precedence order Operators with higher precedence are evaluated before operators with relatively lower precedence. Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first. All binary operators except for the assignment operators are evaluated from left to right; assignment operators are evaluated right to left.

Operator Precedence	
Operators	**Precedence**
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
Operators	**Precedence**
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
Operators	**Precedence**
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Q.3 Explain type Casting?

OR

What is the process of Automatic Type Conversion?

Ans.: It is sometimes necessary to convert a data item of one type to another type. For example when it is necessary to perform some arithmetic using data items of different types (so called mixed mode arithmetic). Under certain circumstances Type conversion can be carried out automatically, in other cases it must be "forced" manually (explicitly).

Automatic Conversion : In Java type conversions are performed automatically when the type of the expression on the right hand side of an assignment operation can be safely promoted to the type of the variable on the left hand side of the assignment. Thus we can safely assign: byte -> short -> int -> long -> float -> double

For example :

```
//64 bit long integer
long myLongInteger;
//32 bit long integer
int myInteger;
myLongInteger=myInteger;
```

The extra storage associated with the long integer, in the above example, will simply be padded with extra zeros.

Explicit Conversion (Casting) : The above will not work the other way round. For example we cannot automatically convert a long to an int because the first requires more storage than the second and consequently information may be lost. To force such a conversion we must carry out an explicit conversion (assuming of course that the long integer will fit into a standard integer). This is done using a process known as a type cast: myInteger = (int) myLongInteger.

This tells the compiler that the type of myLongInteger must be temporarily changed to a int when the given assignment statement is processed. Thus, the cast only lasts for the duration of the assignment. Java type casts have the following form: (T) N where T is the name of a numeric type and N is a data item of another numeric type. The result is of type T.

Q.4. What are the basic control structures in Java?

OR

What are the different control constructs?

OR

Explain the looping constructs?

OR

Explain the conditional constructs?

OR

What is the functioning of 'break and continue' statements?

Ans.: A Java program is a set of statements, which are normally executed sequentially in the order in which they appear. However, in practice, we have a number of situations, where we may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met.

Java language possesses decision making capabilities and supports the following statements known as control or decision making statements :

(1) if statement : It allows the computer to evaluate the expression first and then, depending on whether the value of the expression is 'true' or 'false'.

The general form is :

```
if (test expression)
```

The if statement may be implemented in different forms :

a) Simple if statement

The general form is :

```
if (test expression)
{
        statement-block;
}
statement-x;
```

b) The if---else statement

The general form is :

```
if(test expression)
{
```

```
            true block statements;
}
else
{
            false block statements;
}
statement-x;
```

c) Nested if---else statement

The general form is :

```
if (test condition1)
{
            if(test condition2)
            {
                        statement-1;
            }
else
{
            statement-2;
}
statement-x;
```

d) Else if ladder

The general form is :

```
if (condition1)
            statement-1;
else if (condition2)
            statement-2;
else if (condition)
            statement-n;
else
            default-statement;
statement-x;
```

(2) **Switch Statement :** When one of the many alternatives is to be selected, we can design a program using if statements to control the selection. However, when the number of alternatives increases, the program becomes difficult to read and follow. Then we can use switch statement in such situations.

The general form is :

```
Switch (expression)
{
        case value1:
                block-1;
                break;
        case value-2:
                block-2;
                break;
        ……..
        ……..

        default:
                default-block;
                break;
}
statement-x;
```

(3) **? : operator :**

The general form is :

Conditional expression ? expression1:expression2

The process of repeatedly executing a block of statements is known as looping. The statements in the block may be executed any number of times, from zero to infinite number.

(a) **The while statement :** The simplest of all looping structures in Java is the while statement.

The general format is :

```
Initialization;
while (test condition)
{
        body of the loop
}
```

(b) **The do statement :** In this construct the body of the loop will execute first and the test condition is evaluated.

```
Initialization;
do
{
        body of the loop
}
while(test condition);
```

(c) **The for statement :** This is another entry-controlled loop like while loop. The general format is:

```
For (initialization; test condition; increment/decrement)
{
        body of the loop
}
```

Jumps in Loops : Loops perform a set of operations repeatedly until the control variable fails to satisfy the test condition. Sometimes, it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs.

Jumping out of a loop---We can use the break statement which will immediately exited and the program continues with the statement immediately following the loop.

e.g. while(……)

```
{ …….
        if(condition)
        break;
```

…….

……

}

……

Skipping a part of loop---During the loop operation it may be necessary to skip a part of the body of the loop under certain conditions. We can use continue statement for this.

e.g. while(……..)

{……

if(……..)

continue;

……

}

The statements below continue statement are skipped and control jumps to header part of loop.

□ □ □

Chapter-3

Classes and Objects

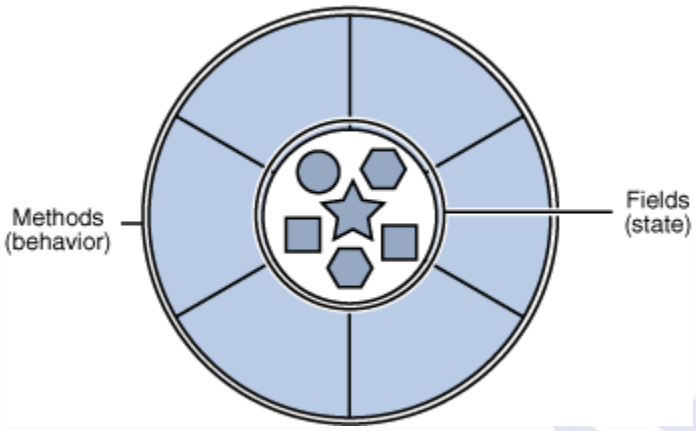
Q.1. What is an Object?

Ans.: Objects are key to understanding *object-oriented* technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.

So, anything that exists in real world is an object. In other words an object is a real life entity.

Real-world objects share two characteristics: They all have ***state*** and ***behavior***. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

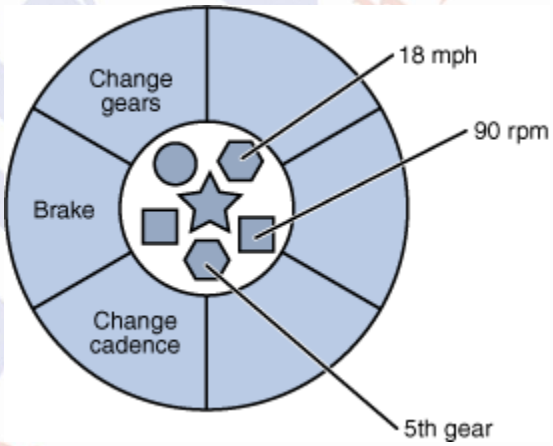
Take a minute right now to observe the real-world objects that are in your immediate area. You'll notice that real-world objects vary in complexity; your desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune). You may also notice that some objects, in turn, will also contain other objects. These real-world observations all translate into the world of object-oriented programming.



A software object.

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in ***fields*** (variables in some programming languages) and exposes its behavior through ***methods*** (functions in some programming languages). Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming.

Consider a bicycle, for example:



A bicycle modeled as a software object.

By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it. For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.

Q.2. What is a Class?

Ans.: In the real world, you'll often find many individual objects all of the same kind.

All the objects that have similar properties and similar behaviour are grouped together to form a class.

In other words we can say that a class is a user defined data type and objects are the instance variables of class.

There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles. A *class* is the blueprint from which individual objects are created.

The following **Bicycle** class is one possible implementation of a bicycle :

```
class Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;
    void changeCadence(int newValue) {
        cadence = newValue;
    }
    void changeGear(int newValue) {
        gear = newValue;
    }
    void printStates() {
  System.out.println("cadence:"+cadence+" speed:"+speed+" gear:"+gear);
    }
}
```

The fields cadence, speed, and gear represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world.

You may have noticed that the Bicycle class does not contain a main method. That's because it's not a complete application; it's just the blueprint for bicycles that might be *used* in an application..

Here's a BicycleDemo class that creates two separate Bicycle objects and invokes their methods :

```
class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
        // Invoke methods on those objects
        bike1.changeCadence(50);
        bike1.changeGear(2);
        bike1.printStates();
        bike2.changeCadence(50);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

The output of this test prints the ending pedal cadence, speed, and gear for the two bicycles:

```
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

Q.3. What do you mean by Garbage Collection?

OR

What do you mean by Memory Management in Java?

Or

How Memory Heaps are avoided by Garbage Collection Process?

Ans.: The name "garbage collection" implies that objects no longer needed by the program are "garbage" and can be thrown away. A more accurate and up-to-date metaphor might be "memory recycling." When an object is no longer referenced by the program, the heap space it occupies can be recycled so that

the space is made available for subsequent new objects. The garbage collector must somehow determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects. In the process of freeing unreferenced objects, the garbage collector must run any *finalizers* of objects being freed.

In addition to freeing unreferenced objects, a garbage collector may also combat heap fragmentation. Heap fragmentation occurs through the course of normal program execution. New objects are allocated, and unreferenced objects are freed such that free portions of heap memory are left in between portions occupied by live objects. Requests to allocate new objects may have to be filled by extending the size of the heap even though there is enough total unused space in the existing heap. This will happen if there is not enough contiguous free heap space available into which the new object will fit. On a virtual memory system, the extra paging (or swapping) required to service an ever growing heap can degrade the performance of the executing program. On an embedded system with low memory, fragmentation could cause the virtual machine to "run out of memory" unnecessarily.

Garbage collection relieves you from the burden of freeing allocated memory. Knowing when to explicitly free allocated memory can be very tricky. Giving this job to the Java virtual machine has several advantages. First, it can make you more productive. When programming in non-garbage-collected languages you can spend many late hours (or days or weeks) chasing down an elusive memory problem. When programming in Java you can use that time more advantageously by getting ahead of schedule or simply going home to have a life.

A second advantage of garbage collection is that it helps ensure program integrity. Garbage collection is an important part of Java's security strategy. Java programmers are unable to accidentally (or purposely) crash the Java virtual machine by incorrectly freeing memory.

A potential disadvantage of a garbage-collected heap is that it adds an overhead that can affect program performance. The Java virtual machine has to keep track of which objects are being referenced by the executing program, and finalize and free unreferenced objects on the fly. This activity will likely require more CPU time than would have been required if the program explicitly freed unnecessary memory. In addition, programmers in a garbage-collected environment have less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.

Q.4. What do you mean by Static Members of a Class?

Ans.: **Static Members of Classes :** In addition to (instance) members, a Java class can include static members that are attached to the class rather than instances of the class. We have already seen how static final fields provide a simple way to define constants.

The static members of a class are not included in the template used to create class instances. There is only one copy of a static field for an entire class--regardless of how many instances of the class are created (possibly none). Similarly, the code in a static method cannot refer to this or to the fields of this because there is no class instance to serve as the receiver for such an access. Of course, a static method can invoke an instance method (or extract an instance field) of class if it explicitly specifies a receiver for the invocation.

Static methods are useful because we occasionally need to write methods where the primary argument is either a primitive value or an object from a class that we cannot modify. For example, the library method Integer.toString (int i) converts an int to the corresponding String. Since an int is not an object, there is no int class to hold such a method. Consequently, the Java library provides a static method toString (int i) in the class Integer.

Finally, all operations on arrays must be expressed in static (procedural) form because array types do not have conventional class definitions; they are built-in to the Java virtual machine.

Q.5. What do you mean by Wrapper Classes?

Ans.: **Wrapper classes** are used to represent primitive values when an Object is required. The wrapper classes are used extensively with Collection classes in the *java.util* package and with the classes in the *java.lang.reflect* reflection package.

Wrapper classes has the following features :

- One for each primitive type: Boolean, Byte, Character, Double, Float, Integer, Long, and Short.
- Byte, Double, Float, Integer and Short extend the abstract Number class.
- All are **public final** i.e. cannot be extended.
- Get around limitations of primitive types.
- Allow objects to be created from primitive types.
- All the classes have two constructor forms :

- a constructor that takes the primitive type and creates an object eg Character(char), Integer(int).
- a constructor that converts a String into an object eg Integer("1"). Throws a NumberFormatException if the String cannot be converted to a number.

NOTE : The character class does not have a constructor that takes a String argument

- All, except Character, have a valueOf(String s) method which is equivalent to new Type(String s)
- All have a typeValue() method which returns the value of the object as it's primitive type. These are all abstract methods defined in Number and overridden in each class
  - public byte byteValue()
  - public short shortValue()
  - public int intValue()
  - public long longValue()
  - public float floatValue()
  - public double doubleValue()
- All the classes override equals(), hashCode() and toString() in Object
  - equals() returns true if the values of the compared objects are the same.
  - hashCode() returns the same hashcode for objects of the same type having the same value.
  - toString() returns the string representation of the objects value.
- All have a public static final TYPE field which is the Class object for that primitive type.
- All have two static fields MIN_VALUE and MAX_VALUE for the minimum and maximum values that can be held by the type.

Void :

- There is also a wrapper class for Void which cannot be instantiated.

NOTE : The constructors and methods described above do NOT exist for the Void class although it does have the TYPE field.

Character :

- Contains two methods for returning the numeric value of a character in the various number systems :
  - public static int digit(char ch, int radix)
  - public static int getNumber(char ch)
- And one method to return the character value of a number :
  - public static char forDigit(int digit, int radix)
- Has two case conversion methods :
  - public static char toLowerCase(char ch)
  - public static char toUpperCase(char ch)
- Also contains a variety of other methods to test wether a character is of a specific type eg isLetter(), isDefined(), isSpaceChar(), etc.
- GetType() returns an int that defines a character's Unicode type.

Integer, Short, Byte and Long :

- All have parseType methods eg parseInt(), parseShort(), etc that take a String and parse it into the appropriate type.
- The Integer and Long classes also have the static methods toBinaryString(), toOctalString() and toHexString() which take an integer value and convert it to the appropriate String representation.

Float and Double :

- Both classes have static fields which define POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN.
- And the following methods to test a value :
  - public boolean isNan()
  - public static boolean isNaN(type value)
  - public boolean isInfinite()
  - public static boolean isInfinite(type value)
- Float also has a constructor that takes a double value.
- both classes have methods to convert a value into a bit pattern or vice versa :
  - public static int floatToIntBits(float value)
  - public static float intBitsToFloat(int bits)
  - public static long doubleToLongBits(double value)

- public static double longBitsToDouble(long bits)

Chapter-4

String Handling

Q.1. What is string handling in java? Explain.

Ans.: Introduction: Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.

Creating Strings

The most direct way to create a string is to write −

```
String greeting = "Hello world!";
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

Example

```
public class StringDemo {
  public static void main(String args[]) {
    char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
    String helloString = new String(helloArray);
    System.out.println( helloString );
  }
}
```

This will produce the following result −

Output

```
hello.
```

Note − The String class is immutable; so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.

Creating a String object

String can be created in number of ways; here are a few ways of creating string object.

1) Using a String literal

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

```
String str1 = "Hello";
```

2) Using another String object

```
String str2 = new String(str1);
```

3) Using new Keyword

```
String str3 = new String("Java");
```

4) Using + operator (Concatenation)

```
String str4 = str1 + str2;
```

or,

```
String str5 = "hello"+"Java";
```

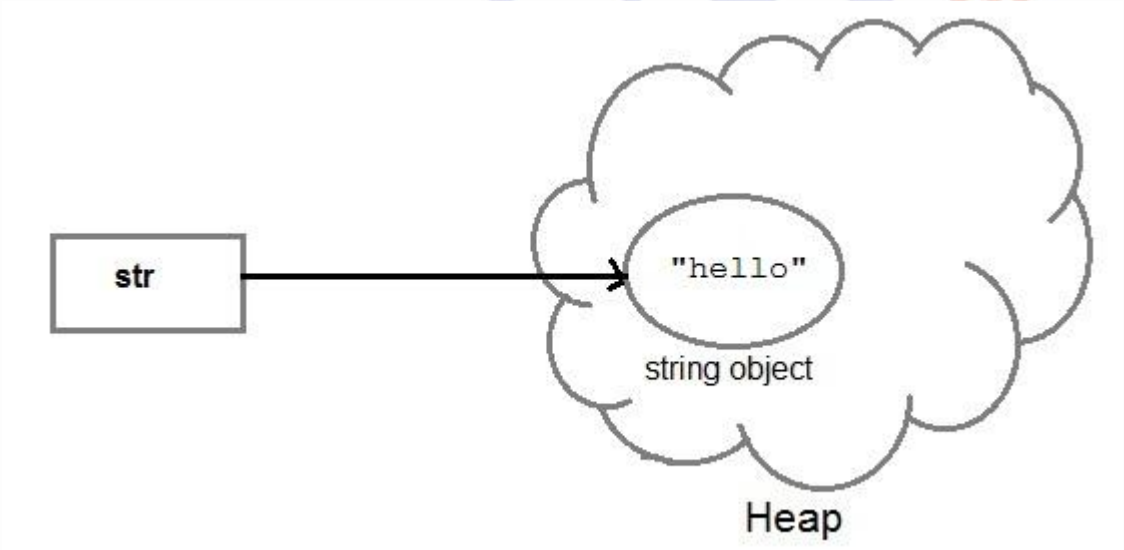
Each time you create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned.

If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as string constant pool inside the heap memory.

String object and how they are stored

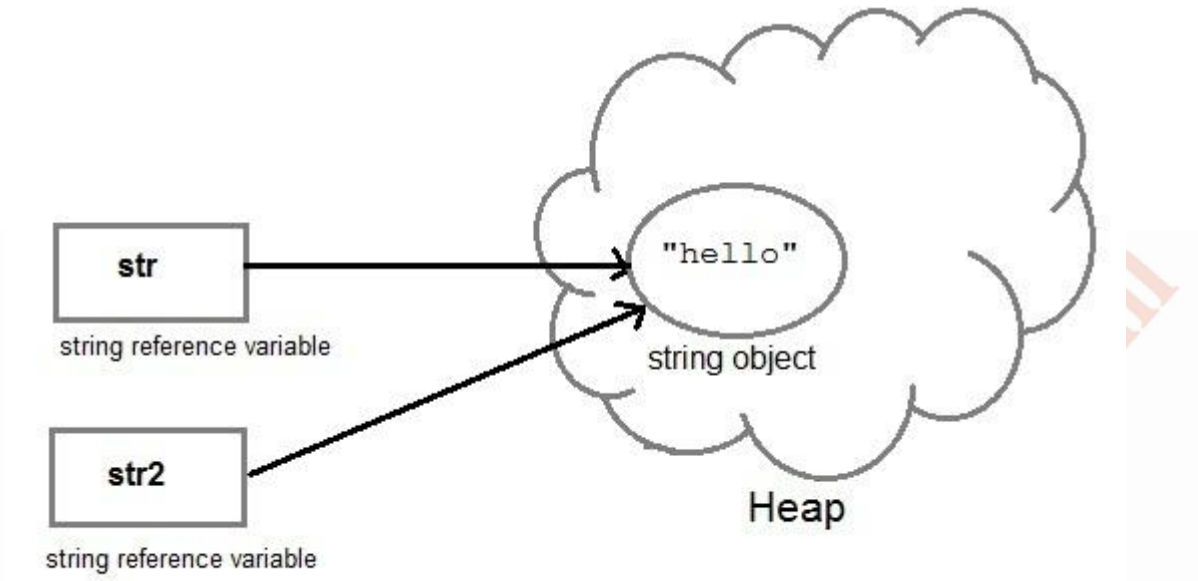
When we create a new string object using string literal, that string literal is added to the string pool, if it is not present there already.

String str= "Hello";



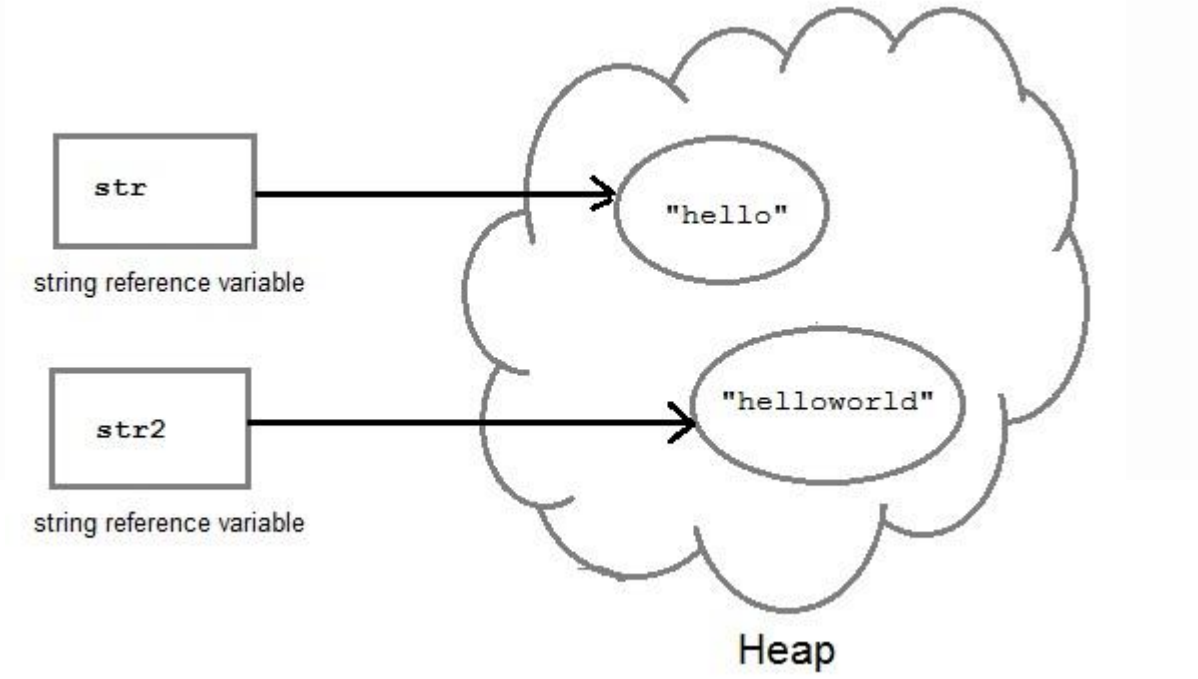
And, when we create another object with same string, then a reference of the string literal already present in string pool is returned.

String str2=str;



But if we change the new string, its reference gets modified.

str2=str2.concat("world");



Concatenating String

There are 2 methods to concatenate two or more string.

- Using concat() method
- Using + operator

1) Using concat() method

```
string s = "Hello";

string str = "Java";

string str2 = s.concat(str);

String str1 = "Hello".concat("Java");    //works with string literals too.
```

2) Using + operator

```
string str = "Rahul";

string str1 = "Dravid";

string str2 = str + str1;

string st = "Rahul"+"Dravid";
```

String Comparison

String comparison can be done in 3 ways.

- Using equals() method
- Using == operator
- By CompareTo() method

Using equals() method

equals() method compares two strings for equality. Its general syntax is,

boolean equals (Object str)

It compares the content of the strings. It will return true if string matches, else returns false.

String s = "Hell";

String s1 = "Hello";

String s2 = "Hello";

s1.equals(s2); //true

s.equals(s1) ; //false

Using == operator

== operator compares two object references to check whether they refer to same instance. This also, will return true on successful match.

String s1 = "Java";

String s2 = "Java";

String s3 = new string ("Java");

test(Sl == s2) //true

test(s1 == s3) //false

By compareTo() method

compareTo() method compares values and returns an int which tells if the string compared is less than, equal to or greater than th other string. Its general syntax is,

int compareTo(String str)

To use this function you must implement the Comparable Interface. compareTo() is the only function in Comparable Interface.

String s1 = "Abhi";

String s2 = "Viraaj";

String s3 = "Abhi";

s1.compareTo(S2); //return -1 because s1 < s2

s1.compareTo(S3); //return 0 because s1 == s3

s2.compareTo(s1); //return 1 because s2 > s1

Q.2. What is an immutable object?

Ans.: An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper class's objects are immutable.

Q.3. What is String Buffer Class? Explain.

Ans.: StringBuffer class is used to create a mutable string object. It represents growable and writable character sequence. As we know that String objects are immutable, so if we do a lot of changes with String objects, we will end up with a lot of memory leak.

So StringBuffer class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors. They are,

1. StringBuffer ( )
2. StringBuffer (int size)
3. StringBuffer (String str)

- StringBuffer (charSequence [ ]ch)StringBuffer() creates an empty string buffer and reserves room for 16 characters.
- stringBuffer(int size) creates an empty string and takes an integer argument to set capacity of the buffer.

Example showing difference between String and StringBuffer

```
class Test {
 public static void main(String args[])
 {
 String str = "study";
 str.concat("tonight");
 System.out.println(str);      // Output: study

 StringBuffer strB = new StringBuffer("study");
 strB.append("tonight");
 System.out.println(strB);    // Output: studytonight
 }
}
```

Q.4. What is String Builder Class?Explain.

Ans.: StringBuilder is identical to StringBuffer except for one important difference it is not synchronized, which means it is not thread safe. Its because StringBuilder methods are not synchronised.

StringBuilder Constructors

StringBuilder ( ), creates an empty StringBuilder and reserves room for 16 characters**.StringBuilder (int size), create an empty string and takes an integer argument to set capacity of the buffer.**

1. StringBuilder (String str), create a StringBuilder object and initialize it with string str.

Example of StringBuilder

```
class Test {
 public static void main(String args[])
 {
  StringBuilder str = new StringBuilder("study");
  str.append( "tonight" );
  System.out.println(str);
  str.replace( 6, 13, "today");
  System.out.println(str);
  str.reverse();
  System.out.println(str);
  str.replace( 6, 13, "today");
 }
}
```

Output :

```
studytonight
studyttoday
yadottyduts
```

Chapter-5

Packages and Interfaces

Q.1. What are Packages?

Ans.: Introduction : Many times when we get a chance to work on a small project, one thing we intend to do is to put all java files into one single directory. It is quick, easy and harmless. However if our small project gets bigger, and the number of files is increasing, putting all these files into the same directory would be a problematic for us. In java we can avoid this sort of problem by using Packages.

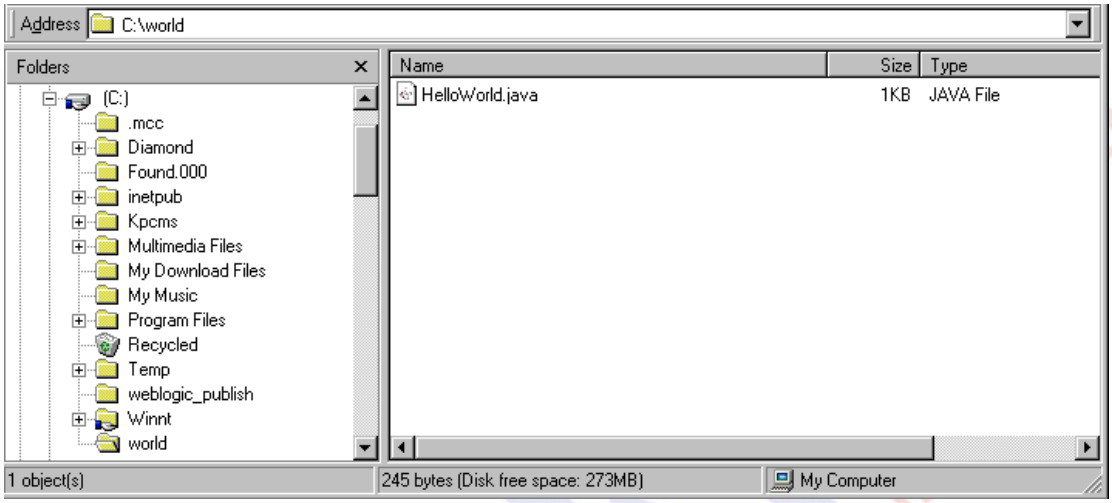
Packages are nothing more than the way we organize files into different directories according to their functionality, usability as well as category they should belong to.

Packaging also help us to avoid class name collision when we use the same class name as that of others. For example, if we have a class name called "Vector", its name would crash with the Vector class from JDK. However, this never happens because JDK use java.util as a package name for the Vector class (java.util.Vector). So our Vector class can be named as "Vector" or we can put it into another package like com.mycompany.Vector without fighting with anyone. The benefits of using package reflect the ease of maintenance, organization, and increase collaboration among developers.

How to create a Package : Suppose we have a file called HelloWorld.java, and we want to put this file in a package **world**. First thing we have to do is to specify the keyword **package** with the name of the package we want to use (**world** in our case) on top of our source file, before the code that defines the real classes in the package, as shown in our HelloWorld class below :

```
package world;
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

One thing you must do after creating a package for the class is to create nested subdirectories to represent package hierachy of the class. In our case, we have the **world** package, which requires only one directory. So, we create a directory **world** and put our HelloWorld.java into it.



Q.2. What are Interfaces?

OR

How do we implement multiple inheritance in 'Java'?

OR

How do we declare and implement Interfaces?

Ans.: Interfaces and Multiple Inheritance : Interfaces have another very important role in the Java programming language. Interfaces are not part of the class hierarchy, although they work in combination with classes. The Java programming language does not permit multiple inheritance (inheritance is discussed later in this lesson), but interfaces provide an alternative.

In Java, a class can inherit from only one class but it can implement more than one interface. Therefore, objects can have multiple types: the type of their own class and the types of all the interfaces that they implement. This means that if a variable is declared to be the type of an interface, its value can reference any object that is instantiated from any class that implements the interface.

Defining an Interface: An interface declaration consists of modifiers, the keyword interface, the interface name, a comma-separated list of parent interfaces (if any), and the interface body. For example :

public interface GroupedInterface extends Interface1,

```
                    Interface2, Interface3 {
  // constant declarations
  double E = 2.718282;  // base of natural logarithms
  // method signatures
  void doSomething (int i, double x);
  int doSomethingElse(String s);
}
```

The public access specifier indicates that the interface can be used by any class in any package. If you do not specify that the interface is public, your interface will be accessible only to classes defined in the same package as the interface.

An interface can extend other interfaces, just as a class can extend or subclass another class. However, whereas a class can extend only one other class, an interface can extend any number of interfaces. The interface declaration includes a comma-separated list of all the interfaces that it extends.

The Interface Body : The interface body contains method declarations for all the methods included in the interface. A method declaration within an interface is followed by a semicolon, but no braces, because an interface does not provide implementations for the methods declared within it. All methods declared in an interface are implicitly public, so the public modifier can be omitted.

An interface can contain constant declarations in addition to method declarations. All constant values defined in an interface are implicitly public, static, and final. Once again, these modifiers can be omitted.

Implementing an Interface : To declare a class that implements an interface, you include an implements clause in the class declaration. Your class can implement more than one interface, so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class.

```
public interface Relatable
{
  public int isLargerThan(Relatable other);
}
public class RectanglePlus implements Relatable {
   public int width = 0;
   public int height = 0;
```

```
 public Point origin;
 // four constructors
 public RectanglePlus() {
origin = new Point(0, 0);
 }
 public RectanglePlus(Point p) {
origin = p;
 }
 public RectanglePlus(int w, int h) {
origin = new Point(0, 0);
width = w;
height = h;
 }
 public RectanglePlus(Point p, int w, int h) {
origin = p;
width = w;
height = h;
 }
 // a method for moving the rectangle
 public void move(int x, int y) {
origin.x = x;
origin.y = y;
 }
 // a method for computing the area of the rectangle
 public int getArea() {
return width * height;
 }
 // a method to implement Relatable
 public int isLargerThan(Relatable other) {
            RectanglePlus otherRect = (RectanglePlus)other;
```

```
            if (this.getArea() < otherRect.getArea())
                return -1;
            else if (this.getArea() > otherRect.getArea())
                return 1;
            else
                return 0;
    }
}
```

Because RectanglePlus implements Relatable, the size of any two RectanglePlus objects can be compared.

Q.4 Explain Inheritance with example?

Ans.: In the Java language, classes can be *derived* from other classes, thereby *inheriting* fields and methods from those classes.

Definitions : A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

Excepting Object, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object.

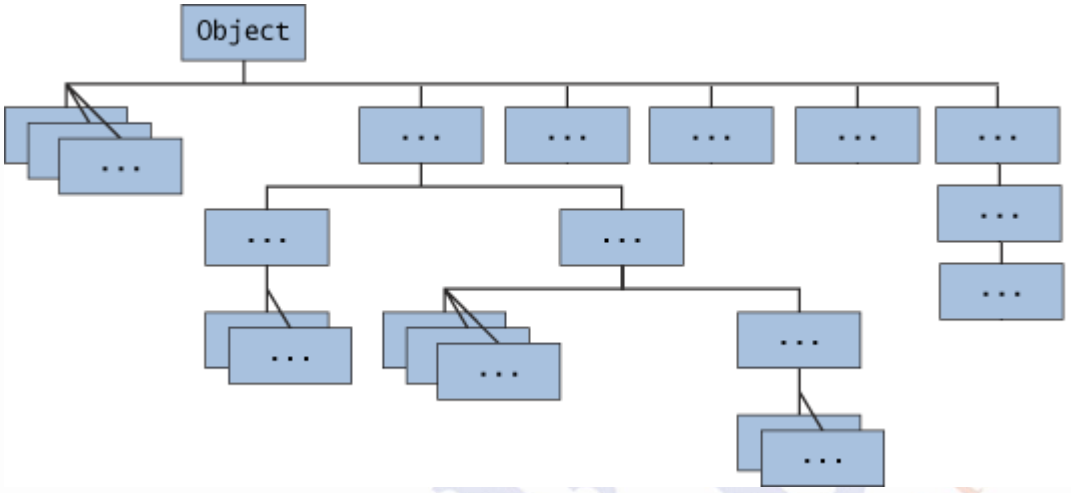
Classes can be derived from classes that are derived from classes, and so on, and ultimately derived from the topmost class, Object. Such a class is said to be *descended* from all the classes in the inheritance chain stretching back to Object.

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

The Java Platform Class Hierarchy : The Object class, defined in the java.lang package, defines and implements behavior common to all classes—

including the ones that you write. In the Java platform, many classes derive directly from Object, other classes derive from some of those classes, and so on, forming a hierarchy of classes.



All Classes in the Java Platform are Descendants of Object

At the top of the hierarchy, Object is the most general of all classes. Classes near the bottom of the hierarchy provide more specialized behavior.

```
/*An Example of Inheritance*/
public class Bicycle {
   // the Bicycle class has three fields
   public int cadence;
   public int gear;
   public int speed;

   // the Bicycle class has one constructor
   public Bicycle(int startCadence, int startSpeed, int startGear) {
      gear = startGear;
      cadence = startCadence;
      speed = startSpeed;
   }
   // the Bicycle class has four methods
   public void setCadence(int newValue) {
```

```
        cadence = newValue;
    }
    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
    public void speedUp(int increment) {
        speed += increment;
    }
}
```

A class declaration for a MountainBike class that is a subclass of Bicycle might look like this:

```
public class MountainBike extends Bicycle {
    // the MountainBike subclass adds one field
    public int seatHeight;
    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight, int startCadence, int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }
    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

MountainBike inherits all the fields and methods of Bicycle and adds the field seatHeight and a method to set it. Except for the constructor, it is as if you had written a new MountainBike class entirely from scratch, with four fields

and five methods. However, you didn't have to do all the work. This would be especially valuable if the methods in the Bicycle class were complex and had taken substantial time to debug.

Q.5. What are Abstract Methods and Classes?

Ans.: Abstract Methods and Classes : 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, the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {
  // declare fields
  // declare non-abstract 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, the subclass must also be declared abstract.

When an Abstract Class Implements an Interface : A class that implements an interface must implement *all* of the interface's methods. It is possible, however, to define a class that does not implement all of the interface methods, provided that the class is declared to be abstract. For example,

```
abstract class X implements Y {
  // implements all but one method of Y
}
class XX extends X {
  // implements the remaining method in Y
}
```

In this case, class X must be abstract because it does not fully implement Y, but class XX does, in fact, implement Y.

Class Members : An abstract class may have static fields and static methods. You can use these static members with a class reference—for example, AbstractClass.staticMethod()—as you would with any other class.

Q.6. What are Final Classes and Methods?

Ans.: Writing Final Classes and Methods :

Final Methods : You can declare some or all of a class's methods *final*. You use the final keyword in a method declaration to indicate that the method cannot be overridden by subclasses. The Object class does this—a number of its methods are final.

You might wish to make a method final if it has an implementation that should not be changed and it is critical to the consistent state of the object. For example, you might want to make the getFirstPlayer method in this ChessAlgorithm class final :

```
class ChessAlgorithm {
   enum ChessPlayer { WHITE, BLACK }
   ...
   final ChessPlayer getFirstPlayer() {
      return ChessPlayer.WHITE;
   }
   ...
}
```

Methods called from constructors should generally be declared final. If a constructor calls a non-final method, a subclass may redefine that method with surprising or undesirable results.

Final Variables : To prevent the subclasses from overrding the member variables of the superclass, we can declare them as final using the final as a modifier.

e.g. final int SIZE =55;

Final Classes : You can also declare an entire class final – this prevents the class from being subclassed. This is particularly useful, for example, when creating an immutable class like the String class. You can use final modifier with class as follows :

e.g. final class A {

}

□ □ □

Chapter-6

Exception Handling in Java

Q.1. What is an Exception?

OR

Explain how Exceptions are handled using try-catch Block?

OR

What is a Finally Block?

Ans.: The term *exception* is shorthand for the phrase "exceptional event."

Definition : An *exception* is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*. After a method throws an exception, the runtime system attempts to find something to handle it.

Some of the predefined exception classes are :

ArithmeticException,

ArrayIndexOutOfBoundException,

IOException etc.

The try Block : The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block. In general, a try block looks like the following.

```
try {
   code
}
```

catch and finally blocks . . .

The segment in the example labeled *code* contains one or more legal lines of code that could throw an exception.

A catch Block : A catch block defined by the keyword catch "catches" the exception "thrown" by the try block and handles it appropriately.The catch block is added immediately after the try block.

The general form is :

```
……..
……
try
{
        statement;
}
catch(Exception type e)
{
        statement;
}
……..
……
```

Multiple catch Statements : It is possible to have more than one catch statements in the catch block.

e.g.

```
…….
……
try
{
        statement;
}
catch(Exception-Type-1 e)
{
```

```
        statement;
}
catch(Exception-Type-2 e)
{
        statement;
}
.
.
.
.
.
catch(Exception –Type-N e)
{
        statement;
}
……..
…….
```

Using Finally Statement : Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements. Finally block can be used to handle any exception generated within a try block. It may be immediately after the try block or after the last catch block.

When a finally block is defined, this is guaranteed to execute, regardless of whether or not in exception is thrown.

Throwing our own Exceptions : There may be times when we would like to throw our own exceptions. We can do this by using the keyword throw as follows :

throw new Throwable_subclass;

e.g. throw new Arithmetic Exception();

□ □ □

Chapter-7

I/O in Java

Q.1. What are Streams?

OR

What is the use of DataInputStream and DataOutputStream?

Ans.: Java uses the concept of streams to represent the ordered sequence of data, a common characteristic shared by all the input/output devices. A stream presents a uniform, easy-to-use, object-oriented interface between the program and the input/output devices.

A stream in Java is a path along which data flows. Both the source and the destination may be physical devices or programs or other streams in the same program.

The concept of sending data from one stream to another has made streams in Java a powerful tool for file processing also.

Stream Classes : The **java.io** package contains a large number of stream classes that provide capabilities for processing all types of data. These classes may categorize into two groups based on the data type on which they operate.

(1) Byte Stream Classes : Provide support for handling I/O operations on bytes.

(2) Character Stream Classes : Provide support for managing I/O operations on characters.

Byte Stream Classes : ByteStream classes have been designed to provide functional features for creating and manipulating streams and files for reading and writing bytes. Since the streams are unidirectional, they can transmit bytes in only one direction and, therefore, Java provides two kinds of byte stream classes: input stream classes and output stream classes.

a) **Input Stream Classes :** Input stream classes are that used to read 8-bit bytes include a super class known as Input Stream and a number of subclasses for supporting various input-related functions.

The InputStream class defines methods for performing input functions such as :

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

Some methods of InputStream are read(), skip(n),reset(), close() etc.

The class DataInputStream extends FilterInput Stream and implements the interface DataInput. Therefore the DataInputStream class implements the methods described in DataInput in addition to using the methods of Input Stream class.

Some methods of DataInputStream are readShort(), readInt(), readLong(), readFloat(), readLine() etc.

b) **Output Stream Classes :** Output stream classes are derived from the base class OutputStream. Like InputStream, the OutputStream is an abstract class and therefore we cannot instantiate it.

The OutputStream includes methods that are designed to perform the following tasks :

- Writing bytes
- Closing streams
- Flushing streams

Some methods of OutputStream are write(), close(), flush() etc.

The DataOutputStream, implements the interface DataOutput and therefore implements methods like writeShort(), writeBytes(), writeInt(), writeLong() etc.

This page shows you how to use the DataInputStream and DataOutputStream classes from java.io using an example, DataIOTest, that reads and writes tabular data like this.

19.99 12 Java T-shirt

9.99 8 Java Mug

DataOutputStream, like other filtered output streams, must be attached to some other OutputStream. In this case, its attached to a FileOutputStream set up to write to a file on the file system named invoice1.txt.

```
DataOutputStream dos = new DataOutputStream(
                        new FileOutputStream("invoice1.txt"));
```

Next, DataIOTest uses DataOutputStream's specialized writeXXX() methods to write the invoice data (contained within arrays in the program).

```
for (int i = 0; i < prices.length; i ++) {
   dos.writeDouble(prices[i]);
   dos.writeChar('\t');
   dos.writeInt(units[i]);
   dos.writeChar('\t');
   dos.writeChars(descs[i]);
   dos.writeChar('\n');
}
dos.close();
```

Note that this code snippet closes the output stream when its finished. The close() method flushes the stream before closing it.

Next, DataIOTest opens a DataInputStream on the file just written :

```
DataInputStream dis = new DataInputStream(
                        new FileInputStream("invoice1.txt"));
```

DataInputStream, like other filtered input streams, must be attached to some other InputStream. In this case, its attached to a FileInputStream set up to read from a file on the file system named invoice1.txt. DataIOTest then just reads the data back in using DataInputStream's specialized readXXX() methods to read the input data into Java variables of the correct type.

```
while (!EOF) {
   try {
      price = dis.readDouble();
      dis.readChar();       // throws out the tab
```

```
        unit = dis.readInt();
        dis.readChar();       // throws out the tab
        desc = dis.readLine();
        System.out.println("You've ordered " + unit + " units of " + desc + " at " +
price);
        total = total + unit * price;
    } catch (EOFException e) {
        EOF = true;
    }
}
System.out.println("For a TOTAL of: " + total);
dis.close();
```

When all of the data has been read, DataIOText displays a statement summarizing the order and the total amount owed, and closes the stream.

Note the loop that DataIOTest uses to read the data from the DataInputStream. Normally, when reading you see loops like this:

```
while ((input = dis.readLine()) != null) {
    . . .
}
```

The readLine() method returns some value, null, that indicates that the end of the file has been reached.

CharacterStreamClasses : Character streams can be used to read and write 16-bit Unicode characters. There are two kinds of character stream classes, namely, reader stream classes and writer stream classes.

a) **Reader Stream Classes :** Reader stream classes are designed to read character from the files. Reader class is the base class for all other classes. These classes are functionally very similar to the input stream classes, except input streams use bytes as their fundamental unit of information, while reader streams use characters.

The **Reader** class contains methods that are identical to those available in the **InputStream** class, except Reader is designed to handle characters.

b) **Writer Stream Classes :** Like output stream classes, the writer stream classes are designed to perform all output operations on files. Only

difference is that while output stream classes are designed to write bytes, the writer stream classes are designed to write characters.

The **Writer** class is an abstract class which acts as abase class for all the other writer stream classes. This base class provides support for all output operations by defining methods that are identical to those in **OutputStream** class.

Q.2. What do you mean by Serialization and Object Persistence?

Ans.: Serialization involves saving the current state of an object to a stream, and restoring an equivalent object from that stream. The stream functions as a container for the object. Its contents include a partial representation of the object's internal structure, including variable types, names, and values. The container may be transient (RAM-based) or persistent (disk-based). A transient container may be used to prepare an object for transmission from one computer to another. A persistent container, such as a file on disk, allows storage of the object after the current session is finished. In both cases the information stored in the container can later be used to construct an equivalent object containing the same data as the original.

For an object to be serialized, it must be an instance of a class that implements either the Serializable or Externalizable interface. Both interfaces only permit the saving of data associated with an object's variables. They depend on the class definition being available to the Java Virtual Machine at reconstruction time in order to construct the object. The Serializable interface relies on the Java runtime default mechanism to save an object's state. Writing an object is done via the writeObject() method in the ObjectOutputStream class (or the ObjectOutput interface).

Sometimes you may wish to prevent certain fields from being stored in the serialized object. The Serializable interface allows the implementing class to specify that some of its fields do not get saved or restored. This is accomplished by placing the keyword transient before the data type in the variable declaration. In addition to those fields declared as transient, static fields are not serialized (written out), and so cannot be deserialized (read back in).

Adding object persistence to Java applications using serialization is easy. Serialization allows you to save the current state of an object to a container, typically a file. At some later time, you can retrieve the saved data values and create an equivalent object. Depending on which interface you implement, you can choose to have the object and all its referenced objects saved and restored automatically, or you can specify which fields should be saved and

restored. Java also provides several ways of protecting sensitive data in a serialized object, so objects loaded from a serialized representation should prove no less secure than those classes loaded at application startup.

□ □ □

Chapter-8

AWT based effective GUI in Java

Q.1. Explain Delegation of Event Model?

Ans.: The delegation event model came into existence with JDK1.1. In this model, an event is sent to the component from which it originated. The component registers a listener object with the program. The listener object contains appropriate event-handlers that receive and process the events. e.g., when you click a button, the action to be performed is handled by an object registered to handle the button click event.

NOTE : By registering a listener object with the program, the component enables the d elegation of events to the listener object for processing.

Every event has a corresponding listener interface that specifies the methods that are required to handle the event. Event objects are sent to registered listeners. To enable a component to handle events, you must register an appropriate listener for it.

NOTE : When you use interfaces for creating listeners, the listener class has to override all the methods that are declared in the interface. Some of the interfaces have only one method, whereas others have many. Even if you want to handle only one event, you have to override all the methods. To overcome this, the event package provides seven adapter classes.

Q.2 How do we implement Nesting of Classes?

OR

What are Inner Classes?

Ans.: **Nested Classes :** The Java programming language allows you to define a class within another class. Such a class is called a *nested class* and is illustrated here :

```
class OuterClass {
   ...
```

```
    class NestedClass {
        ...
    }
}
```

Terminology : Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are simply called *static nested classes*. Non-static nested classes are called *inner classes*.

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class. As a member of the OuterClass, a nested class can be declared private, public, protected, or *package private*. (Recall that outer classes can only be declared public or *package private*.)

Q.3. Reasons to use nested classes.

Ans.: There are several compelling reasons for using nested classes, among them :

- It is a way of logically grouping classes that are only used in one place.
- It increases encapsulation.
- Nested classes can lead to more readable and maintainable code.

a) **Logical Grouping of Classes :** If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

b) **Increased Encapsulation :** Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

c) **More Readable, Maintainable Code :** Nesting small classes within top-level classes places the code closer to where it is used.

Static Nested Classes : As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class — it can use them only through an object reference.

Note : A static nested class interacts with the instance members of its outer class (and other classes) just like any other top-level class. In effect, a static nested class is behaviorally a top-level class that has been nested in another top-level class for packaging convenience.

Static nested classes are accessed using the enclosing class name:

OuterClass.StaticNestedClass

For example, to create an object for the static nested class, use this syntax:

OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

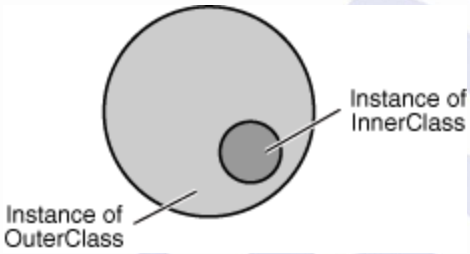
Inner Classes : As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

Objects that are instances of an inner class exist *within* an instance of the outer class. Consider the following classes :

class OuterClass {

```
    ...
    class InnerClass {
        ...
    }
}
```

An instance of InnerClass can exist only within an instance of OuterClass and has direct access to the methods and fields of its enclosing instance. The next figure illustrates this idea.



An InnerClass Exists Within an Instance of OuterClass

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

OuterClass.InnerClass innerObject = outerObject.new InnerClass();

□ □ □

Chapter-9

Applets

Q.1. What are Applets?

OR

What are the advantages and disadvantages of Applet Programming?

Ans.: A **Java applet** is an applet delivered in the form of Java bytecode. Java applets can run in a Web browser using a Java Virtual Machine (JVM), or in Sun's AppletViewer, a stand-alone tool for testing applets. Java applets were introduced in the first version of the Java language in 1995. Java applets are usually written in the Java programming language but they can also be written in other languages that compile to Java bytecode such as Jython.

Applets are used to provide interactive features to web applications that cannot be provided by HTML. Since Java's bytecode is platform independent, Java applets can be executed by browsers for many platforms, including Windows, Unix, Mac OS and Linux.There are open source tools like applet2app which can be used to convert an applet to a stand alone Java application/windows executable/linux executable. This has the advantage of running a Java applet in offline mode without the need for internet browser software.

Technical Information : Java applets are executed in a sandbox by most web browsers, preventing them from accessing local data. The code of the applet is downloaded from a web server and the browser either embeds the applet into a web page or opens a new window showing the applet's user interface. The applet can be displayed on the web page by making use of the deprecated applet HTML element [1], or the recommended object element [2]. This specifies the applet's source and the applet's location statistics.

A Java applet extends the class java.applet.Applet, or in the case of a Swing applet, javax.swing.JApplet. The class must override methods from the applet class to set up a user interface inside itself (Applet is a descendant of Panel which is a descendant of Container).

Advantages : A Java applet can have any or all of the following advantages :

- It is simple to make it work on Linux, Windows and Mac OS i.e. to make it cross platform.
- The same applet can work on "all" installed versions of Java at the same time, rather than just the latest plug-in version only. However, if an applet requires a later version of the JRE the client will be forced to wait during the large download.
- It is supported by most web browsers.
- It will cache in most web browsers, so will be quick to load when returning to a web page but may get stuck in the cache and have issues when new versions come out.
- It can have full access to the machine it is running on if the user agrees.
- It can improve with use: after a first applet is run, the JVM is already running and starts quickly, benefitting regular users of Java but the JVM will need to restart each time the browser starts fresh.
- It can run at a comparable (but generally slower) speed to other compiled languages such as C++, but many times faster than JavaScript.
- It can move the work from the server to the client, making a web solution more scalable with the number of users/clients.

Disadvantages : A Java applet is open to any of the following disadvantages :

- It requires the Java plug-in, which isn't available by default on all web browsers.
- an implementation of the Sun Java plug-in does not exist for 64-bit processors.
- It cannot start until the Java Virtual Machine is running, and this may have significant startup time the first time it is used.
- If untrusted, it has severely limited access to the user's system - in particular having no direct access to the client's disk or clipboard.
- Some organizations only allow software installed by the administrators. As a result, many users cannot view applets by default.
- Applets may require a specific JRE.

Q.2. How communication is possible in between Applications?

OR

What do you mean by Inter Applet Communication?

Ans.: Getting two or more applets within a single Web page to talk to each other has some benefits. Although this applet capability has been around since the earliest version of Java, it's not often used, because there's more emphasis placed on getting applets to communicate with servers.

While this is understandable given the current fashion of client/server programming, it's still a valuable skill for developers to learn. Another reason the technique isn't used much is that complicated Web-borne applets are usually shown in a single window. If there's a lot of information to show, the designers simply make the applet larger.

However, in terms of Web page design, it's better in some cases to place small bits of Java-based functionality in different parts of the page, leaving the rest to be filled with text and images. To do this, you need multiple applet windows that are, in some sense, part of the same program.

Method : The secret of inter-applet communication (which we'll abbreviate to IAC) is the method **AppletContext.getApplets()**. This method provides us with an **Enumeration** of all the applets running on the same page as the calling applet. From this **Enumeration**, you can take actual **Applet** objects, allowing you to freely call methods on it.

What we'll first give names to the applets on the page and then allow them to send text strings to each other using the names as destinations.

Here's an API for this :

public void send(String appletName, String message);

protected String rcv();

the **send()** method sends a string to another applet with a given name;

the **rcv()** method returns the next string that has been sent to you.

Q.3. Explain the life cycle of an Applet?

Ans.: An applet can react to major events in the following ways :

- It can *initialize* itself.
- It can *start* running.
- It can *stop* running.

- It can perform a *final cleanup*, in preparation for being unloaded.

All applets have the following four methods :

public void init();

public void start();

public void stop();

public void destroy();

They have these methods because their superclass, java.applet.Applet, has these methods.

In the superclass, these are simply do-nothing methods.

The **init()** method is called exactly once in an applet's life, when the applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and set up the user interface. Most applets have init() methods.

The **start()** method is called at least once in an applet's life, when the applet is started or restarted. In some cases it may be called more than once. Many applets you write will not have explicit start()methods and will merely inherit one from their superclass. A start() method is often used to start any threads the applet will need while it runs.

The **stop()** method is called at least once in an applet's life, when the browser leaves the page in which the applet is embedded. The applet's start() method will be called if at some later point the browser returns to the page containing the applet. In some cases the stop() method may be called multiple times in an applet's life. Many applets you write will not have explicit stop()methods and will merely inherit one from their superclass. Your applet should use the stop() method to pause any running threads. When your applet is stopped, it *should* not use any CPU cycles.

The **destroy()** method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform any final clean-up. For example, an applet that stores state on the server might send some data back to the server before it's terminated. many applets will not have explicit destroy() methods and just inherit one from their superclass.

Q.4. How parameters are passed to Applets?

Ans.: Passing Parameters to Applets : Parameters are passed to applets in NAME=VALUE pairs in **<PARAM>** tags between the opening and closing APPLET tags. Inside the applet, you read the values passed through the PARAM tags with the getParameter() method of the java.applet.Applet class.

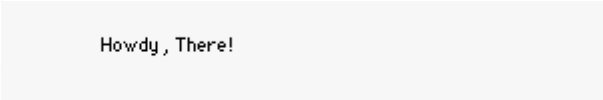
The program below demonstrates this with a generic string drawing applet. The applet parameter "Message" is the string to be drawn.

```
import java.applet.*;
import java.awt.*;
public class DrawStringApplet extends Applet {
  private String defaultMessage = "Hello!";
  public void paint(Graphics g) {
    String inputFromPage = this.getParameter("Message");
    if (inputFromPage == null) inputFromPage = defaultMessage;
    g.drawString(inputFromPage, 50, 25);
  }
}
```

You also need an HTML file that references your applet. The following simple HTML file will do :

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
This is the applet:<P>
<APPLET code="DrawStringApplet" width="300" height="50">
<PARAM name="Message" value="Howdy, there!">
This page will be very boring if your browser doesn't understand Java.
</APPLET>
</BODY>
</HTML>
```

Of course you are free to change "Howdy, there!" to a "message" of your choice. You only need to change the HTML, not the Java source code. PARAMs let you customize applets without changing or recompiling the code.



However rather than hardcoding the message to be printed it's read into the variable inputFromPage from a PARAM element in the HTML.

You pass getParameter() a string that names the parameter you want. This string should match the name of a PARAM element in the HTML page. getParameter() returns the value of the parameter. All values are passed as strings. If you want to get another type like an integer, then you'll need to pass it as a string and convert it to the type you really want.

The PARAM element is also straightforward. It occurs between <APPLET> and </APPLET>. It has two attributes of its own, NAME and VALUE. NAME identifies which PARAM this is. VALUE is the string value of the PARAM. Both should be enclosed in double quote marks if they contain white space.

An applet is not limited to one PARAM. You can pass as many named PARAMs to an applet as you like. An applet does not necessarily need to use all the PARAMs that are in the HTML. Additional PARAMs can be safely ignored.

□ □ □

Chapter-10

Threading in Java

Q.1. What is Multithreading?

OR

What are Threads and how are they implemented in Java?

OR

Explain various states of life cycle of a Thread?

Ans.: Computer users take it for granted that their systems can do more than one thing at a time. They assume that they can continue to work in a word processor, while other applications download files, manage the print queue, and stream audio. Even a single application is often expected to do more than one thing at a time. For example, that streaming audio application must simultaneously read the digital audio off the network, decompress it, manage playback, and update its display.

The Java platform is designed from the ground up to support concurrent programming, with basic concurrency support in the Java programming language and the Java class libraries. Since version 5.0, the Java platform has also included high-level concurrency APIs.

In concurrent programming, there are two basic units of execution: *processes* and *threads*. In the Java programming language, concurrent programming is mostly concerned with threads. However, processes are also important.

Processes : A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.

Processes are often seen as synonymous with programs or applications. However, what the user sees as a single application may in fact be a set of cooperating processes. To facilitate communication between processes, most operating systems support *Inter Process Communication* (IPC) resources, such as pipes and sockets.

Threads : Threads are sometimes called *lightweight processes*. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process.

Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.

Multithreaded execution is an essential feature of the Java platform. Every application has at least one thread — or several, if you count "system" threads that do things like memory management and signal handling. But from the application programmer's point of view, you start with just one thread, called the *main thread*.

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this:

Provide a Runnable object. The Runnable interface defines a single method, run, meant to contain the code executed in the thread. The Runnable object is passed to the Thread constructor, as in the HelloRunnable example :

```
public class HelloRunnable implements Runnable {
   public void run() {
      System.out.println("Hello from a thread!");
   }
   public static void main(String args[]) {
      (new Thread(new HelloRunnable())).start();
   }
}
```

Subclass Thread : The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run, as in the HelloThread example :

```
public class HelloThread extends Thread {
   public void run() {
      System.out.println("Hello from a thread!");
   }
```

```
    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

Notice that both examples invoke Thread.start in order to start the new thread.

The first idiom, which employs a Runnable object, is more general, because the Runnable object can subclass a class other than Thread. The second idiom is easier to use in simple applications, but is limited by the fact that your task class must be a descendant of Thread.

The Thread class defines a number of methods useful for thread management. These include static methods, which provide information about, or affect the status of, the thread invoking the method. The other methods are invoked from other threads involved in managing the thread and Thread object.

Stopping a Thread : Whenever we want to stop a thread from running further, we may do so by calling its stop() method which results in causing thread to dead state.

Blocking a Thread : A thread can also be temporarily suspended or blocked from entering into the runnable and subsequently running state by using either of the following thread methods :

```
sleep()          //blocked for a specified time

suspend()        //blocked until further orders

wait()           //blocked until certain conditions occurs.
```

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

(1) **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.

(2) **Runnable State :** It 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.

(3) **Running State :** It means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control (using suspend(), sleep(), or notify())on its own or it is preempted by a higher priority thread.

(4) **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.

(5) **Dead State :** 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.

□ □ □

Chapter-11

Overview of Networking

Q.1. Explain how socket based connectivity is useful in Client/Server Applications?

Ans.: In Client/Server applications, the server provides services like processing database queries or modifying the data in the database. The communication that occurs between the client and the server must be reliable. The data must not be lost and must be available to the client in the same sequence in which it was sent by the server.

Transmission control protocol(TCP) provides a reliable, point-to-point communication channel for Client-Server applications to communicate with each other. To communicate over TCP, client and server program establish a connection and bind a socket. Sockets are used to handle the communication links between applications over the network. Further communication between the client and the server is through the socket.

The advantage of the socket model using TCP over other communication models, such as NetBEUI and Apple Talk, is that the server is not affected by the source of client requests. It services all requests, as long as the clients follow the TCP/IP protocol suite. This means that the client can be any kind of computer. No longer is the client restricted to the UNIX, Windows, DOS, or Macintosh platforms. Therefore, all the computers in a network implementing TCP/IP can communicate with each other through sockets.

Java was designed as a networking language. It makes network programming easier by encapsulating connection functionality in the Socket classes, that is, the Socket class to create a client socket and the Server Socket class to create a server socket.

The different socket classes are outlined below :

Socket is the basic class that supports the TCP protocol. TCP is a stream network connection protocol. The Socket class provides methods for stream I/O, which makes reading from and writing to a socket easy. This class is indispensable to the programs written to communicate on the Internet.

ServerSocket is a class used by the Internet server programs for listening to client requests. ServerSocket does not actually perform the service; instead, it

creates a Socket object on behalf of the client. The communication is performed through the object created.

Q.2 Explain TCP/IP sockets and Datagram sockets.

Ans.: Client Server and Sockets :

- From a programmer's viewpoint, the Internet is the largest **client/server** system implemented to date.
- The Internet has well-defined **protocols** used between the clients and the servers.
- In fact the whole of the Internet is underpinned by just two protocols: the **Internet Protocol (IP)** and the **Transmission Control Protocol (TCP).**
- One of the most important ways of implementing client server applications is by using **TCP/IP sockets**.
- Most high level programming languages and common OS's now support the use of sockets - though in this module we are largely concerned with **Java**.

Introduction to Sockets :

- ARPA funded the University of California at Berkeley to provide a UNIX implementation of the TCP/IP protocol suite.
- What was developed was termed the socket interface, although you might hear it called the Berkeley -socket interface or just Berkeley sockets. It was written in C.
- Today, the socket interface is the most widely used method for accessing a TCP/IP network.
- A socket is nothing more than a convenient abstraction. It represents a connection point into a TCP/IP network, devices communicate with each other by sending or receiving data through a socket.

Sockets :

- When two computers want to converse they can each use a socket. Quite often, one computer is termed the server - this opens a socket and listens for connections.

- The other computer is termed the client; it calls the server socket to start the connection. To establish a connection, all that's needed is a destination address and a port number.
- A port is a particular address on the server which is usually represented as a simple integer value - 80 is the standard port for a HTTP (web) server.
- Each computer in a network has a unique IP address. Ports represent individual connections within that address.

Socket Transmission Modes :

- Sockets have two major modes of operation: connection-oriented and connectionless.
- Connection-oriented sockets use TCP/IP and operate like a telephone; they must establish a connection and a hang up. Everything that flows between these two
- events arrives in the same order it was sent.
- Connectionless sockets operate like the postal service and delivery is not guaranteed. Multiple pieces of mail may arrive in a different order than they were sent.
- Which mode to use is determined by an application's needs. Some applications, such as a time s**erver**, don't really need reliability of delivery. Many other applications however do require guaranteed delivery.

UDP and Datagram Sockets :

- Connectionless operation uses the **User Datagram Protocol** (**UDP**). Like TCP, UDP runs on top of IP.
- A **datagram** is a self- contained unit that has all the information needed to attempt its delivery.
- A socket in this mode does not need to connect to a destination socket; it simply sends the datagram to the destination and keeps its fingers crossed.
- The UDP protocol promises only to make a best-effort delivery attempt. Connectionless operation is **fast** and efficient, but not guaranteed.
- UDP is often used in streaming video and audio data to one or more destinations (called **multicast**).

TCP/IP Sockets :

- Connection-oriented operation uses the Transport Control Protocol (TCP).
- A socket in this mode needs to connect to the destination before sending data.
- Once connected, the sockets are accessed using a streams interface: open-read -write-close.
- Everything sent by one socket is received by the other end of the connection in exactly the same order it was sent. If any errors occur, then TCP can request that packets are resent, ensuring 100% data reliability.
- Connection-oriented operation is slower than connectionless, but it is guaranteed.

□ □ □

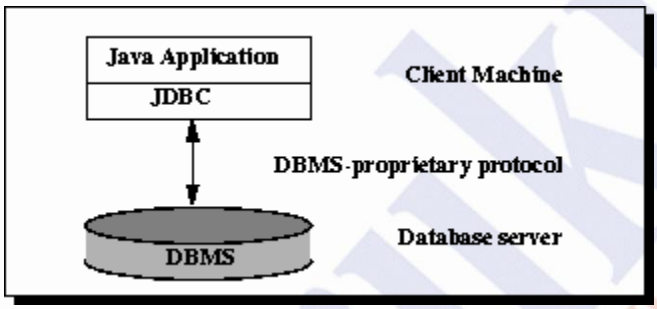
Chapter-12

Java Database Connectivity

Q.1. Explain JDBC Architecture?

Ans.: JDBC Architecture : The JDBC API supports both two-tier and three-tier processing models for database access.

Two-tier Architecture for Data Access :



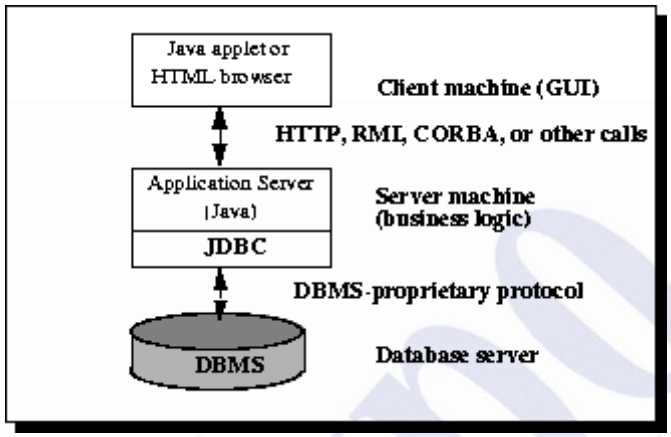
In the two-tier model, a Java application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

Three-tier Architecture for Data Access :

Until recently, the middle tier has often been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java bytecode into efficient machine-specific code and technologies such as Enterprise JavaBeans™, the Java

platform is fast becoming the standard platform for middle-tier development. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features.



With enterprises increasingly using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of a three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed transactions, and disconnected rowsets. The JDBC API is also what allows access to a data source from a Java middle tier.

Q.2. What do you understand by JDBC API?

Ans.: Java Database Connectivity (JDBC) is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

The Java 2 Platform, Standard Edition, version 1.4 (J2SE) includes the JDBC 3.0 API[1] together with a reference implementation JDBC-to-ODBC Bridge, enabling connections to any ODBC-accessible data source in the JVM host environment. This Bridge is native code (not Java), closed source, and only *appropriate for experimental use and for situations in which no other driver is available*.

Overview : JDBC has been part of the Java Standard Edition since the release of JDK 1.1. The JDBC classes are contained in the Java package java.sql. Starting with version 3.0, JDBC has been developed under the Java Community Process. JSR 54 specifies JDBC 3.0 (included in J2SE 1.4), JSR 114

specifies the JDBC Rowset additions, and JSR 221 is the specification of JDBC 4.0.

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The **Driver Manager** is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. JDBC represents statements using one of the following classes :

- **Statement** – the statement is sent to the database server each and every time.
- **PreparedStatement** – the statement is cached and then the execution path is pre determined on the database server allowing it to be executed multiple times in an efficient manner.
- **CallableStatement** – used for executing stored procedures on the database.

Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information.

Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types.

There is an extension to the basic JDBC API in the javax.sql package that allows for scrollable result sets and cursor support among other things.

Example : The method Class.forName(String) is used to load the JDBC driver class.

□ □ □

Chapter-13

JavaScript Fundamentals

Q.1. How can you work with the frames in Java Script?

Ans.: Some browsers (including the latest Netscape and Microsoft browsers) support *frames,* which enable you to divide the browser window into multiple panes. Each frame can contain a separate URL or the output of a script.

Using JavaScript Objects for Frames : When a window contains multiple frames, each frame is represented in JavaScript by a frame object. This object is equivalent to a window object, but it is used for dealing with that frame. The frame object's name is the same as the NAME attribute you give it in the <frame> tag.

Keyword, parent, enables you to refer to the main window.

Each frame object in a window is a child of the parent window object. Suppose you define a set of frames using the *HTML* below:

<frameset ROWS="*,*" COLS="*,*">

<frame NAME="topleft" SRC="topleft.htm">

<frame NAME="topright" SRC="topright.htm">

<frame NAME="bottomleft" SRC="botleft.htm">

<frame NAME="bottomright" SRC="botright.htm">

</frameset>

This simply divides the window into quarters. If you have a JavaScript program in the topleft.htm file, it would refer to the other windows as parent.topright, parent.bottomleft, and so on. The keywords window and self would refer to the topleft frame.

Note : If you use nested framesets, things are a bit more complicated. window still represents the current frame, parent represents the frameset containing the current frame, and top represents the main frameset that contains all the others.

The Frames Array : Rather than referring to frames in a document by name, you can use the frames array. This array stores information about each of the

frames in the document. The frames are indexed starting with zero and beginning with the first <frame> tag in the frameset document.

For example, you could refer to the frames defined in the previous example using array references :

- parent.frames[0] is equivalent to the topleft frame.
- parent.frames[1] is equivalent to the topright frame.
- parent.frames[2] is equivalent to the bottomleft frame.
- parent.frames[3] is equivalent to the bottomright frame.

You can refer to a frame using either method interchangeably, and depending on your application, you should use the most convenient method. For example, a document with 10 frames would probably be easier to use by number, but a simple two-frame document is easier to use if the frames have meaningful names.

Q.2. What do you understand by Document Object Model?

Ans.: One advantage that JavaScript has over basic *HTML* is that scripts can manipulate the Web document and its contents. Your script can load a new page into the browser, work with parts of the browser window and document, open new windows, and even modify text within the page dynamically.

To work with the browser and documents, JavaScript uses a hierarchy of parent and child objects called the Document Object Model, or *DOM*. These objects are organized into a tree-like structure, and represent all of the content and components of a Web document.

Like other objects you've explored, the objects in the *DOM* have *properties,* which describe the Web page or document, and *methods,* which allow you to work with parts of the Web page.

When you refer to an object, you use the parent object name followed by the child object name or names, separated by periods. For example, JavaScript stores objects to represent images in a document as children of the document object. *For instance,* the following refers to the image9 object, a child of the document object, which is a child of the window object:

window.document.image9

The **window** object is the parent object for all the objects.

DOM Levels : The *W3C* (World-Wide Web Consortium) has recently developed the *DOM* level 1 standard. This standard defines not only basic

objects, but an entire set of objects that encompass all parts of an *HTML* document. A level 2 *DOM* standard is also under development.

The basic object hierarchy is informally referred to as *DOM* level 0, and the objects are included in the *DOM* level 1 standard.

The Level 1 and Level 2 *DOM* objects allow you to modify a Web page in real time after it has loaded. This is called dynamic *HTML* (*DHTML*).

Q.3. How can you create Interactive Forms in Java Script?

Ans.: Using the 'form' Object with JavaScript : Each form in your *HTML* page is represented in JavaScript by a form object, which has the same name as the NAME attribute in the <form> tag you used to define it.

Alternately, you can use the forms array to refer to forms. This array includes an item for each form element, indexed starting with 0. For example, if the first form in a document has the name form1, you can refer to it in one of two ways:

document.form1

document.forms[0]

The 'form' Object's Properties : Along with the elements, each form object also has a list of properties, most of which are defined by the corresponding <form> tag. You can also set these from within JavaScript. They include the following :

- Action is the form's ACTION attribute, or the program to which the form data will be submitted.
- Encoding is the MIME type of the form, specified with the ENCTYPE attribute. In most cases, this is not needed.
- Length is the number of elements in the form. You cannot change this property.
- Method is the method used to submit the form, either GET or POST.
- Target specifies the window in which the result of the form (from the *CGI* script) will be displayed. Normally, this is done in the main window, replacing the form itself.

Submitting and Resetting Forms : The form object has two methods, submit and reset. You can use these methods to submit the data or reset the form yourself, without requiring the user to press a button. One reason for this is to

submit the form when the user clicks an image or performs another action that would not usually submit the form.

Note : If you use the submit method to send data to a server or by email, Netscape will prompt the user to verify that she wants to submit the information. There's no way to do this behind the user's back.

Detecting Form Events : The form object has two event handlers, onSubmit and onReset. You can specify a group of JavaScript statements or a function call for these events within the **<form>** tag that defines the form.

If you specify a statement or function for the onSubmit event, the statement is called before the data is submitted to the *CGI* script. You can prevent the submission from happening by returning a value of false from the onSubmit event handler. If the statement returns true, the data will be submitted. In the same fashion, you can prevent a Reset button from working with an **onReset** event handler.

Scripting Form Elements : The most important property of the form object is the elements array, which contains an object for each of the form elements. You can refer to an element by its own name or by its index in the array. For example, the following two expressions both refer to the first element in the order form, the name1 text field:

document.order.elements[0]

document.order.name1

Q.4. Explain how Cookies are implemented in Java Script?

OR

What are Cookies?

Ans.: Cookies : Cookies were originally invented by Netscape to give 'memory' to web servers and browsers. The HTTP protocol, which arranges for the transfer of web pages to your browser and browser requests for pages to servers, is *state-less*, which means that once the server has sent a page to a browser requesting it, it doesn't remember a thing about it. So if you come to the same web page a second, third, hundredth or millionth time, the server once again considers it the very first time you ever came there.

This can be annoying in a number of ways. The server cannot remember if you identified yourself when you want to access protected pages, it cannot remember your user preferences, it cannot remember anything. As soon as personalization was invented, this became a major problem.

Cookies were invented to solve this problem. There are other ways to solve it, but cookies are easy to maintain and very versatile.

How Cookies work : A cookie is nothing but a small text file that's stored in your browser. It contains some data :

(1) A **name-value** pair containing the actual data

(2) An **expiry date** after which it is no longer valid

(3) The **domain and path** of the server it should be sent to

As soon as you request a page from a server to which a cookie should be sent, the cookie is added to the HTTP header. Server side programs can then read out the information and decide that you have the right to view the page you requested or that you want your links to be yellow on a green background.

So every time you visit the site the cookie comes from, information about you is available. This is very nice sometimes, at other times it may somewhat endanger your privacy. Fortunately more and more browsers give you the opportunity to manage your cookies.

Q.5 Write the procedure to create Custom Java Script Objects?

Ans.: How to create your own basic Object : Creating an object requires two steps:

- First, declare the object by using an object function.
- Lastly, instantiate the newly created object by using the "new" keyword.

Lets take this one step at a time. We will now proceed to create an object called "userobject", which, at this stage, does nothing :

Step 1 : Declare the object by using an object function

The first step towards creating an object requires us to define an object function. An object function is virtually identical in syntax as a regular function, although there are some differences which will surface later on. The object function is used to define and declare an object :

```
function userobject(parameter){

}
```

The parameter is optional, and with it, allows us to pass in values to an object. For example, in the pre-built object window.alert, the parameter is the text passed in to be alerted. Now, with just the above object function, we have in essence just created a new object called "userobject"! It does nothing at this

stage, and will continue to do until we add in properties and methods. To use this object, all we have to do is instantiate it, by using the keyword "new".

Step 2 : Instantiate the newly created object by using the "new" keyword

Once we've defined an object function, we have to instantiate it to actually use it. Instantiating an object function means using the keyword "new" in front of the object name, and then creating an instance of the object by assigning it to a variable :

```
<script type="text/javascript">
function userobject(parameter){
}
//myobject is now an object of type userobject!
var myobject=new userobject("hi")
</script>
```

"myobject" is now an object...an instance of "userobject", to be exact.

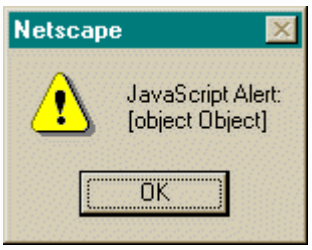
If you're a little confused at this stage, consider a more familiar example :

```
var image1=new Image(20,20)
```

The above should be review to us; we created an instance of the pre-built image object by assigning it to the variable image1. Well, this familiar process is exactly what we'll doing with the custom object above.

If you're the kind that need to actually see and touch an object before you believe its an object, the window.alert method can help :

```
<script type="text/javascript">
function userobject(parameter)
{
}
//myobject is now an object of type userobject!
var myobject=new userobject("hi")
alert(myobject)
</script>
```



How to add properties to your own Object : Thus far, our object "userobject" cannot do anything but take up space in a document. With some properties, that should all change. To add properties to a user defined object, directly embed the properties into the object function, with each property proceeded

by the keyword "this" plus dot (.): In the below example, we'll extend "userobject" to contain two properties, each containing a string of text:

```
function userobject(parameter){
this.firstproperty=parameter
this.secondproperty="This is the second property"
}
```

Now, to use these properties, simply access them like accessing any other property:

```
<script>
var myobject=new userobject("hi there.")
//alerts "hi there."
alert(myobject.firstproperty)
//writes "This is the second property"
document.write(myobject.secondproperty)
</script>
```

How to add methods to your own object : Adding methods to a user defined object is a bit more complicated. We need to first declare and define a function for each method, then associate this function with the object function. For the sake of simplicity, we will simply call functions defined for methods "method functions." Lets get a clean start, and create a new object called "circle" that will contain methods that compute the area and diameter of a circle, respectively.

The first step to adding methods is to implement the method functions. Method functions define what a method does :

```
//first method function
function computearea(){
var area=this.radius*this.radius*3.14
return area
}
//second method function
function computediameter(){
```

```
var diameter=this.radius*2
return diameter
}
```

In the above case, we've created two method functions, "computearea" and "computediamter", which calculates various aspects of a circle. The two functions, as you can see, are just functions, with one major distinction. Take the first one, for example :

```
function computearea(){
var area=this.radius*this.radius*3.14
return area
```

this.radius looks like a property of a custom object to me. Since a method function will eventually be connected to the custom object, it has access to the properties of the object. We haven't defined the properties yet, but we will, and the method functions will use them in its calculation.

We will now associate the two method functions above to the new object "circle", so they become methods of the object :

```
<script type="text/javascript">
/*the below creates a new object, and gives it the two methods defined earlier*/
function circle(r){
//property that stores the radius
this.radius=r
this.area=computearea
this.diameter=computediameter
}
</script>
```

Finally, to use these methods, instantiate the object, and access the methods just like any other method :

```
<script type="text/javascript">
var mycircle=new circle(20)
//alerts 1256
```

```
alert("area="+mycircle.area())
//alerts 400
alert("diameter="+mycircle.diameter())
</script>
```

□ □ □

BACHELOR OF COMPUTER APPLICATIONS
(Part III) EXAMINATION
(Faculty of Science)
(Three – Year Scheme of 10+2+3 Pattern)
PAPER 318
Internet Application Development
OBJECTIVE PART- I

Year - 2011

Time allowed : One Hour *Maximum Marks : 20*

The question paper contains 40 multiple choice questions with four choices and students will have to pick the correct one. (Each carrying ½ marks.).

1. To view web pages you need:
   (a) Browser
   (b) WWW
   (c) TCP/IP
   (d) All of the above ( )

2. Live communication on the internet can be done using:
   (a) E-mail
   (b) Newsgroups
   (c) IRC
   (d) None of the above ( )

3. The protocol that web clients and servers use to communicate with each other is called:
   (a) HTML
   (b) HTTP
   (c) URL
   (d) None of the above ( )

4. Web documents are stores as text files with the extension:
   (a) .htm
   (b) .html

(c) both (a) and (b)
(d) none of the above ( )

5. What of the following is a Search Engine?
(a) Microsoft (b) Yahoo
(c) Alta Vista (d) Both (b) and (c)
(e) Google ( )

6. What will be the result of the following
(a) 38
(b) 25
(c) 9
(d) 12 ( )

7. A package is a collection of :
(a) Classes
(b) Interfaces
(c) Editing tools
(d) Classes and interfaces ( )

8. The methods wait 9 ( ) and notify ( ) are defined in:
(a) java. lang. string
(b) java . lang. runnable
(c) java . lang. object
(d) java. lang. thread ( )

9. Which of the following methods belongs to the string class?
(a) length ( )
(b) comperato ( )
(c) equals ( )
(d) All of the above ( )

10. Which exception is thrown by the read ( ) method of input stream class?
(a) Exception
(b) IO exception
(c) File not found exception
(d) None of the above
( )

11. GUI stands for:
(a) Graphical Unique Interface
(b) Graphical User Interface
(c) Graphical User Information
(d) None of the above ( )

12. Which not a wrapper class?

(a) Random
(b) Byte
(c) Integer
(d) Short ( )

13. Internet e-mail is based on standards, known is:
(a) Protocols
(b) Networks
(c) Both (a) and (b)
(d) None of the above ( )

14. The FTP is a member protocol of the.....suite.
(a) HTTP
(b) TCP/IP
(c) SMTP
(d) None of the above ( )

15. HTML tags are enclosed in:
(a) Angle brackets
(b) Parenthesis
(c) Curly braces
(d) None of the above ( )

16. Links are also known is:
(a) Anchors
(b) Hotspots
(c) Both (a) and (b)
(d) None of the above ( )

17. Which of the case statement?
(a) If.....else statement
(b) switch statement
(c) Loop statement
(d) None of the above ( )

18. Which object provides a list of the URL?
(a) Event object
(b) History object
(c) Location object
(d) Both (b) and (c) ( )

19. Which methods appends a value to the end of an array?
(a) join ( )
(b) push ( )
(c) pop ( )

(d) none of the above ( )

20. Which event occurs when the user presents the mouse button?
(a) onfocus
(b) onchange
(c) onblur
(d) None of the above ( )

21. CSS stands for:
(a) Casading style sheet
(b) Client style sheet
(c) Current style sheet
(d) None of the above ( )

22. The data from teh database on the server is displayed in the table in DHTML, this features is called:
(a) Data inheritance
(b) Data binding
(c) Data collection
(d) None of the above ( )

23. The STYLE attributes is used to apply style sheet to.....
(a) More than two elements
(b) Individual elements
(c) Whole document elements
(d) None of the above ( )

24. Variant data type contain:
(a) Empty
(b) Boolean
(c) Byte
(d) All of the above ( )

25. Which keyword is used to stop the current execution of the loop?
(a) Switch
(b) If
(c) Break
(d) Both (a) and (c) ( )

26. Which button provides an interface to select an option among the multiple choices?
(a) Radio Button
(b) Check Button
(c) List Button
(d) Control Button ( )

27. Which is the example of web browser?
    (a) Java
    (b) C++ & C
    (c) Netscape Navigator
    (d) Both () and (b) ( )

28. JDBC is known as:
    (a) Java Database Client
    (b) Java Database Connection
    (c) Java Database Current
    (d) Java Database connectivity ( )

29. AWT stands for:
    (a) Advance Window Terminator
    (b) Active Window Time
    (c) Advance Windowing toolkit
    (d) Advance Window Toolkit

30. Full form of API is:
    (a) Applet Programming Interface
    (b) Application programming Integer
    (c) Application Programming Interface
    (d) Applet programming Integer ( )

31. Which of the Java debugger?
    (a) JDBC or jdbc (b) JDK or jdk
    (c) JDB or jdb (d) None of the above ( )

32. Full form of JDK is:
    (a) Java Developed Kit
    (b) Java Developers Kit Tool
    (c) Java Developers Kit
    (d) Java Developed Kit Tool ( )

33.
 tag is used to:
    (a) Line break
    (b) Line border
    (c) Paragraph break
    (d) Bold border ( )

34. Object in Java:
    (a) Run time Entity
    (b) Blue print of another object of the class
    (c) Compile Time
    (d) All of the above ( )

35. Which feature is not in Java?
    (a) Procedural
    (b) Object oriented
    (c) Abstraction
    (d) Polymorphism ( )

36. When we implement the Runnable interface, we must define the method:
    (a) start ( )
    (b) init ( )
    (c) run ( )
    (d) runnable ( ) ( )

37. When we invoke repaint ( ) for a component, the AWT invokes the method:
    (a) draw ( )
    (b) show ( )
    (c) update ( )
    (d) paint ( ) ( )

38. Which of the following methods can be used to change the size of a component?
    (a) dimension ( )
    (b) setsize ( )
    (c) resize ( )
    (d) Both (b) and (c) ( )

39. Which of the following keywords are used to control access to a class members?
    (a) abstract
    (b) interface
    (c) public
    (d) all of the above ( )

40. The keywords reserved but not used in the initial version of Java are:
    (a) const
    (b) inner
    (c) goto
    (d) all of the above ( )

DESCRIPTIVE PART-II

Year- 2011

Time allowed : 2 Hours ***Maximum Marks : 30***

Attempt any four descriptive types of questions out of the six. All questions carry 7½ marks each.

Q.1

(a) What is DHTML?

Ans **Dynamic HTML** is used to create animated web sites by using a combination of a static markup language a client side scripting language (such as JavaScript), a presentation definition language (such as CSS).

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, *after* the page has been fully loaded and during the viewing process. Thus the dynamic characteristic of DHTML is the way it functions while a page is viewed, not in its ability to generate a unique page with each page load.

```
<html>
  <head>
     <title>DHTML example</title>
  </head>
  <body>
   <div id="navigation"></div>

   <script>
    var init = function () {
     myObj = document.getElementById("navigation");
      };
       window.onload = init;
   </script>

   <script src="myjavascript.js"></script>
```

```
</body>
</html>
```

(b) **Define hypertext and Hypermedia ?**

Ans **Hypertext** is text displayed on a computer or other electronic device with references (hyperlinks) to other text or sound or animations. Hypermedia simply combines hypertext that the reader can immediately access, usually by a mouse click, keypress sequence or by touching the screen. Apart from running text, hypertext may contain tables, images and other presentational devices. Hypertext is the underlying concept defining the structure of the World wide web.It is an easy-to-use and flexible format to share information over the Internet.

Hypermedia - Hypermedia is a superset of hypertext. Hypermedia documents contain links not only to other pieces of text, but also to other forms of media - sounds, images, and movies. Images themselves can be selected to link to sounds or documents. This means that browsers might not display a text file, but might display images and multimedia.

(c) What is web browser ?

Ans A **web browser** is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. An *information resource* is identified by a Uniform Resource Identifier (URI) and may be a web page, image, video, or other piece of content.Hyperlinks present in resources enable users easily to navigate their browsers to related resources. A web browser can also be defined as an application software or program designed to enable users to access, retrieve and view documents and other resources on the Internet.

Although browsers are primarily intended to access the World Wide Web, they can also be used to access information provided by web servers in private networks or files in file systems. The major web browsers are Firefox, Google Chrome, Internet Explorer, Opera, and Safari.

he first web browser was invented in 1990 by Sir Tim Berners-Lee. It was called WorldWideWeb and was later renamed Nexus.

Every browser features a toolbar that allows you to perform various functions like:

- Go back to the first page you started on the internet which is called Home.
- Book your favorite websites
- Print content you find interesting on web pages

- Check your web history, like the websites you visited in the past
- You can go forward and backwards to see the previous sites you viewed

(d) Define the term dynamic binding ?

Ans. Dynamic binding also refres to the run time polymorphism or late binding. If same message is pass to different object each object response is differently depending upon its own class it is called Dynamic polymorphism.
For acheaving run time polymorphism in java we use three things.

(1) Inhertance must be there

(2) Method overriding must be there

(3) Super class variable refer to the direct or indirect sub class object.

In java dynamic binding is a default binding. But in c++ it is achieve through virtual function.

```
Class Shape
{
Public void area()
{
}}
Class Rectangle extend Shape
{
Public void area()
{
}}
Class Square extend Shape
{
Public void area()
{
}}
Class Demo
{
Public static void main(String args[])
{
Shape obj;
Obj=new Rectangle();
Obj.area();
Obj=new Square();
Obj.area();
}
}
```

Q.2

(a) How do we create tables in html? Explain with suitable examples?

Ans.

HTML tables are use to present data in rows and columns, we can also create HTML tables to organize information on our web page.

The process of creating an HTML table is similar to the process that we used to create our web page and any elements that we may have already included in your page, such as links or frames. Coding HTML tables into your web page is fairly easy since you need only understand a few basic table codes.

II. Creating a basic table

The basic structure of an HTML table consists of the following tags:

- Table tags: **<TABLE> </TABLE>**
- Row tags: **<TR> </TR>**
- Cell tags: **<TD> </TD>**

```
<html>
<head><title> use of table</title>
</head>
<body>
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
</body>
</html>
```

HTML Table Headers

Header information in a table are defined with the <th> tag.

All major browsers display the text in the <th> element as bold and centered.

```
<table border="1">
<tr>
<th>Header 1</th>
<th>Header 2</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How the HTML code above looks in our browser:

Header 1	Header 2
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

(b) What is hypertext link? What is the method to link to a specific place within the same document?

Ans. A **hyperlink** is a reference to data that the reader can directly follow, or that is followed automatically. A hyperlink points to a whole document or to a specific element within a document. Hypertext is text with hyperlinks. A software system for viewing and creating hypertext is a *hypertext system*, and to create a hyperlink is *to hyperlink* (or simply *to link*). A user following hyperlinks is said to *navigate* or *browse* the hypertext.

A hyperlink has an *anchor*, which is the location within a document from which the hyperlink can be followed; the document containing a hyperlink is known as its **source** document. words and terms in the text are hyperlinked to definitions of those terms. Hyperlinks are often used to implement reference mechanisms, such as tables of contents, footnotes, bibliographies, indexes, letters and glossaries.

In some hypertext, hyperlinks can be bidirectional: they can be followed in two directions, so both ends act as anchors and as targets

Q.3 What are applets ? explain applet life cycle?

Ans Applet is java program that can be embedded into HTML pages. Java applets runs on the java enables web browsers such as mozila and internet explorer. Applet is designed to run remotely on the client browser, so there are some restrictions on it. Applet can't access system resources on the local computer. Applets are used to make the web site more dynamic and entertaining.

Advantages of Applet:

- Applets are cross platform and can run on Windows, Mac OS and Linux platform
- Applets can work in any of the java plug-in
- Applets are supported by most web browsers
- Applets are cached in most web browsers, so will be quick to load when returning to a web page
- User can also have full access to the machine if user allows

Disadvantages of Java Applet:

- Java plug-in is required to run applet
- JVM is compulsary to run the applet
- If applet is not already cached in the machine, it will be downloaded from internet and will take time
- Its difficult to desing and build good user interface in applets compared to HTML technology

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

public class HelloWorld extends Applet {
 public void init() { }

 public void stop() { }

 public void paint(Graphics g) {
  g.drawString("Hello, world!", 20,10);

  g.drawArc(40,30,20,20,0,360);
 }
}
```

```
HTML>
<HEAD>
<TITLE>HelloWorld</TITLE>
</HEAD>
<BODY>
<H1>A Java applet example</H1>
 <APPLET code="HelloWorld.class" WIDTH="200" HEIGHT="300">
</APPLET>
</BODY>
</HTML>
```

The Life cycle of An Applet

Introduction

Applet runs in the browser and its lifecycle method are called by JVM when it is loaded and destroyed. Here are the lifecycle methods of an Applet:

init(): This method is called to initialized an applet only one time.

start(): This method is called after the initialization of the applet.

stop(): This method can be called multiple times in the life cycle of an Applet.

destroy(): This method is called only once in the life cycle of the applet when applet is destroyed.

(b) What are frames and framesets? How they are created and what re their applications?

Ans. Introduction to frames and framesets

HTML frames allow authors to present documents in multiple views, which may be independent windows or subwindows. Multiple views offer designers a way to keep certain information visible, while other views are scrolled or replaced. For example, within the same window, one frame might display a static banner, a second a navigation menu, and a third the main

document that can be scrolled through or replaced by navigating in the second frame.

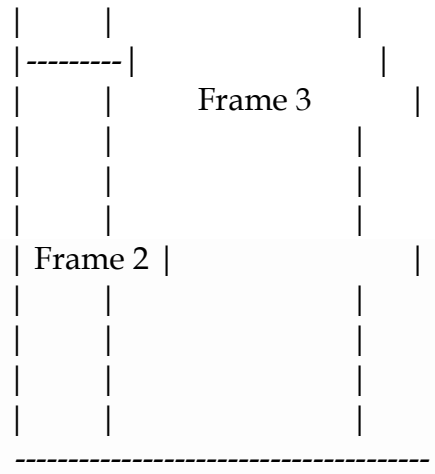
Here is a simple frame document:

```
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
 <FRAMESET rows="100, 200">
    <FRAME src="contents_of_frame1.html">
    <FRAME src="contents_of_frame2.gif">
 </FRAMESET>
 <FRAME src="contents_of_frame3.html">
 <NOFRAMES>
    <P>This frameset document contains:
    <UL>
      <LI><A href="contents_of_frame1.html">Some neat contents</A>
      <LI><IMG src="contents_of_frame2.gif" alt="A neat image">
      <LI><A href="contents_of_frame3.html">Some other neat contents</A>
    </UL>
 </NOFRAMES>
</FRAMESET>
</HTML>
```

The **FRAMESET** element is a *frame container* for dividing a window into rectangular subspaces called *frames*. In a Frameset document, the outermost **FRAMESET** element takes the place of **BODY** and immediately follows the **HEAD**.

that might create a frame layout something like this:

```
 ---------------------------------------
|       |                     |
|       |                     |
| Frame 1 |                       |
|       |                     |
```



When to use frames

From a design viewpoint, there are at least two valid uses of frames:

- Integrated into the page design of a single page, to provide separate areas for material such as navigation
- As the mechanism for associating material from a specific author (such as comments) with other pages that normally stand on their own

Q.4 Write short note on TCP/IP protocol ?

Ans. The **Internet protocol suite** is the set of communication protocol used for the internet and similar networks, and generally the most popular protocol stack for wide area network. It is commonly known as **TCP/IP**, because of its most important protocols: Transmission Control Protocol (TCP) and Internet Protocol (IP), which were the first networking protocols defined in this standard. It is occasionally known as the **DoD model** due to the foundational influence of the ARPANET in the 1970s (operated by DARPA, an agency of the United States Department of Defense).

TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. It has four abstraction layers, each with its own protocols.[1][2] From lowest to highest, the layers are:

1. The link layer (commonly Ethernet) contains communication technologies for a local network.
2. The internet layer (IP) connects local networks, thus establishing internetworking.

3. The transport layer (TCP) handles host-to-host communication.
4. The application layer (for example HTTP) contains all protocols for specific data communications services on a process-to-process level (for example how a web browser communicates with a web server).

(b) Write short note on JDBC and JDK?

Ans. **Java Database Connectivity** in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

JDBC has been developed under the Java Community Process that allows multiple implementations to exist and be used by the same application. JDBC provides methods for querying and updating the data in Relational Database Management system such as SQL, Oracle etc.

The Java application programming interface provides a mechanism for dynamically loading the correct Java packages and drivers and registering them with the JDBC **Driver Manager** that is used as a connection factory for creating JDBC connections which supports creating and executing statements such as SQL INSERT, UPDATE and DELETE. Driver Manager is the backbone of the jdbc architecture.

Generally all Relational Database Management System supports SQL and we all know that Java is platform independent, so JDBC makes it possible to write a single database application that can run on different platforms and interact with different Database Management Systems.

Java Database Connectivity is similar to Open Database Connectivity (ODBC) which is used for accessing and managing database, but the difference is that JDBC is designed specifically for Java programs, whereas ODBC is not depended upon any language.

In short JDBC helps the programmers to write java applications that manage these three programming activities:

1. It helps us to connect to a data source, like a database.
2. It helps us in sending queries and updating statements to the database and
3. Retrieving and processing the results received from the database in terms of answering to your query.

The **Java Development Kit** (**JDK**) is an Oracle Corporation product aimed at Java developers. Since the introduction of Java, it has been by far the most widely used Java Software Development Kit (SDK). Sun announced that it would be released under the GNU General Public License (GPL), thus making it free software. Sun contributed the source code to the OpenJDK.

JDK contents

- java– the loader for Java applications.
- javac – the compiler, which converts source code into Java bytecode
- appletviewer – this tool can be used to run and debug Java applets without a web browser
- apt– the annotation-processing tool
- extcheck– a utility which can detect JAR-file conflicts
- idlj– the IDL-to-Java compiler. This utility generates Java bindings from a given Java IDL file.
- javadoc – the documentation generator, which automatically generates documentation from source code comments
- jar– the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files.

Q.5 What do you understand by exception handling explain in detail?

Ans. Exception handling is a very important yet often neglected aspect of writing robust software. When an error occurs in a Java program it usually results in an exception being thrown. How you throw, catch and handle these exception matters. There are several different ways to do so. Not all are equally efficient and fail safe.

The three categories of exceptions:

- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compliation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a

stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

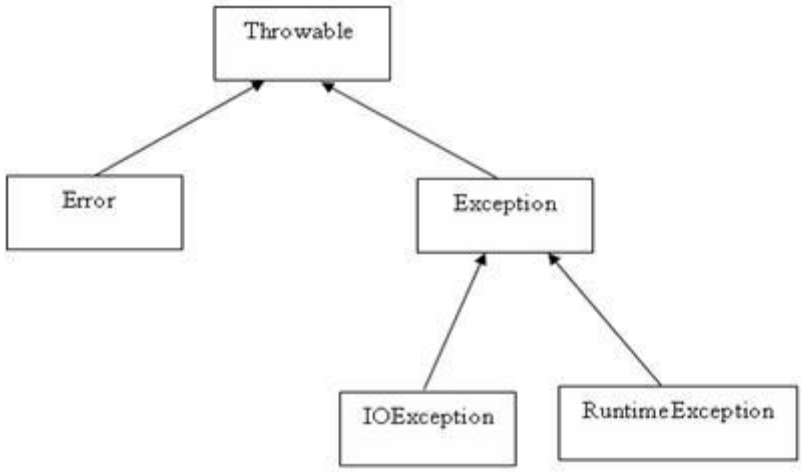
```
import java.io.*;

public class exceptionHandle{
  public static void main(String[] args) throws Exception{
  try{
  int a,b;
  BufferedReader in =
  new BufferedReader(new InputStreamReader(System.in));
  a = Integer.parseInt(in.readLine());
  b = Integer.parseInt(in.readLine());
  }
  catch(NumberFormatException ex){
  System.out.println(ex.getMessage()
  + " is not a numeric value.");
  System.exit(0);
  }
  }
}
```

Exception Hierarchy:

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses : IOException class and RuntimeException Class.



(b) What is class and object in java?

Ans Java is an Object Oriented Language. As a language that has the Object Oriented feature Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

- **Object -** Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.
- **Class -** A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.

Objects in Java:

If we consider the real-world we can find many objects around us, Cars, Dogs, Humans etc. All these objects have a state and behavior.

If we consider a dog then its state is . name, breed, color, and the behavior is . barking, wagging, running

If you compare the software object with a real world object, they have very similar characteristics.

Software objects also have a state and behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java:

A class is a blue print from which individual objects are created.

A sample of a class is given below:

```
public class Dog{
  String breed;
  int age;
  String color;

  void barking(){
  }

  void hungry(){
  }

  void sleeping(){
  }
}
```

A class can contain any of the following variable types.

- **Local variables .** variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables .** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables .** Class variables are variables declared with in a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kind of methods. In the above example, barking(), hungry() and sleeping() are variables.

Below mentioned are some of the important topics that need to be discussed when looking into classes of the Java Language.

Q.6 What is multi-threading ? explain with an example?

Ans. A running instance is known as process. Multiple parts of process that are running simultaneous is known as thread. In term of operating system it is known as multithreading .

In java we can create thread using two methods.

(1) Extending the thread class

(2) Implementing the runnable interface

Extending the thread class:-

(i) Create a sub class of thread

(ii) Override the run method

(iii) Call the start method of thread class with instance of the sub class

Ex…

```
Class one extends Thread
{
    Public void run()
    {
    Int I;
    For(i=1;i<=10;i++)
            {System.out.print("I am one");}}}
Class Two extends Thread
{
    Public void run()
    {
    Int I;
    For(i=1;i<=10;i++)
            {System.out.print("I am Two");}}}
Class demo
{
Public static void main(String args[])
{
   One o =new One();
  o.start();
Two t=new Two();
 t.statrt();
```

```
int I;
        For(i=1;i<=10;i++)
                {
                System.out.print("I am main");}}}
```

Implementing the runnable interface:-

(i) Create a class that implement the runnable interface

(ii) Define the run method of the interface

(iii) Pass the instance of the thread class to the object of the class

```
 Class one implement runnable
{
Public void run()
{
Int i;
For(i=1;i<=10;i++)
{System.out.print("I am one");}}}
Class Two implement runnable
{
Public void run()
{
Int I;
For(i=1;i<=10;i++)
{System.out.print("I am Two");}}
Class demo
{
Public static void main(String args[])
{
Thread t1=new Thread();
One o =new One(t1);
  o.start();
Thread t2=new Thread();
Two t=new Two(t2);
 t.statrt();
int i;
For(i=1;i<=10;i++)
{    System.out.print("I am main");
}}}
```

(b) What are interfaces and packages? Explain their differences?

Ans. An interface is a collection of abstact methods and constents.methods are by default public and abstract and varaibles are public static and final.we need

not to specify their attribute.in java multiple inheritance is not allow but multilpe inheritance of interfaces are allowed.
A package is a grouping of classes and interfaces. The purpose of grouping classes is that we can access them easily only using the import keyword. Package is very similar to grouping items within a folder or directory on a file system. A class is found within a package, but this does not have an impact on the class' behavior.
An interface, however, is a .java file that is used (implemented) by another class to tell the outside world that it conforms to a certain specification. For example, you might have a "Runnable" interface that has a "run()" method in it, by having a class that is "Runnable" (implements Runnable) anyone using that class knows that it must have a "run()" method defined. This is used when you have several different classes that have the same interface.

Interfaces have more in common with abstract classes than they do with packages. An interface, by definition, cannot have any implemented methods; an abstract class, in contrast, can define some methods and leave some methods to be implemented by a subclass. Also, a class can implement many interfaces, but can only extend one (abstract) class.

ex.

```
Package finance;
Public class Distance
{


}

Import finance.*;
Class usedistance
{
Public static void main(String args[])
{
Distance d1=new Distance();
.....}}
```