OBJECT ORIENTATED PROGRAMMING THROUGH JAVA

UNIT – I

Introduction to Java: Features of Java, The Java virtual Machine, Parts of Java **Naming Conventions and Data Types:** Naming Conventions in Java, Data Types in Java, Literals **Operators in Java:** Operators, Priority of Operators **Control Statements in Java:** if... else Statement, do... while Statement, while Loop, forLoop, switch Statement, break Statement, continue Statement, return Statement

Input and Output: Accepting Input from the Keyboard, Reading Input with Java.util.Scanner Class, Displaying Output with System.out.printf(), Displaying Formatted Output with String.format()

UNIT – II

Arrays: Types of Arrays, Three Dimensional Arrays (3D array), arrayname.length, Command Line Arguments, **Strings:** Creating Strings, String Class Methods, String Comparison, Immutability of Strings

Introduction to OOPs: Problems in Procedure Oriented Approach, Features of Object-Oriented Programming System (OOPS)

Classes and Objects: Object Creation, Initializing the Instance Variables, Access Specifiers, Constructors, **Methods in Java:** Method Header or Method Prototype, Method Body, Understanding Methods, Static Methods, Static Block, The keyword 'this', Instance Methods, Passing Primitive Data Types to Methods, Passing Objects to Methods, Passing Arrays to Methods, Recursion, Factory Methods

UNIT – III

Inheritance: Inheritance, The keyword 'super', The Protected Specifier, Types of Inheritance, **Polymorphism:** Polymorphism with Variables, Polymorphism using Methods, Polymorphism with Static Methods, Polymorphism with Private Methods, Polymorphism with Final Methods, final Class

Type Casting: Types of Data Types, Casting Primitive Data Types, Casting Referenced DataTypes, The Object Class

Abstract Classes: Abstract Method and Abstract Class, **Interfaces:** Interface, Multiple Inheritance using Interfaces **Packages:** Package, Different Types of Packages, The JAR Files, Interfaces in a Package, Creating Sub Package in a Package, Access Specifiers in Java, Creating API Document

UNIT – IV

Exception Handling: Errors in Java Program, Exceptions, throws Clause, throw Clause, Types of Exceptions, Re – throwing an Exception **Streams:** Stream, Creating a File using FileOutputStream, Reading Data from a File using FileInputStream, Creating a File using FileWriter, Reading a File using FileReader, Zipping and Unzipping Files, Serialization of Objects, Counting Number of Characters in a File, File Copy, File Class

UNIT – V

Threads: Single Tasking, Multi-Tasking, Uses of Threads, Creating a Thread and Running it, Terminating the Thread, Single Tasking Using a Thread, Multi-Tasking Using Threads, Multiple Threads Acting on Single Object, Thread Class Methods, **Deadlock of Threads**, Thread Communication, Thread Priorities, thread Group, Daemon Threads, **Applications of Threads**, Thread Life Cycle.

Applets: Creating an Applet, Uses of Applets, <APPLET> tag, A Simple Applet, An Applet with Swing Components, Animation in Applets, A Simple Game with an Applet, Applet Parameters

UNIT – I

1Q) HISTORY OF JAVA

Ans:

- **In 1990** Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by **James Gosling** was formed to undertake this task.
- Java is a general-purpose object oriented programming language developed by **sun Microsystems of USA in 1991**, **it was Originally called OAK** by James Gosling and his team. The team, known as **Green Project team** by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch sensitive screen.
- Java was designed for the development of software for consumer electronic devices like TVs, VCRs, toasters and such other electronic machines. This goal had a strong impact on the development team to make the language simple, portable and highly reliable.
- The java team which included **Patrick Naughton** discovered that the existing languages like C and C++ had limitations in terms of both reliability and portability. However, they modeled their new language Java on C and C++ but removed a number of features of C and C++ that were considered as sources of problems and thus made java a really simple, reliable, portable and powerful language.
- **In 1993 the WWW** appeared on the internet and transformed the text-based internet into a graphical-rich environment. The Green Project team came up with the idea of developing web applets using the new language that could run on all types of computer connected to internet.
- **In 1994** the team developed a web browser called **"Hot Java"** to located and run applet programs on internet.
- **In 1995 Oak was renamed "Java",** due to some legal problems. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announced their support to java.

2Q) Features / Buzzwords of JAVA

Ans:

Java is OOP language which is having the following features. They are –

1. Object Oriented
2. Simple
3. Platform Independent and Portable
4. Robust and secured
5. Architectural Neutral
6. Multithreaded
7. Distributed
8. Dynamic and Extensible

1. Object-Oriented

- It is a true-object oriented language.
- Almost everything in java is an object.
- OOPS is so integral to java that you must understand its basic principles, such as Data Encapsulation, Polymorphism and inheritance.

2. **Simple**
   - Java was designed to be easy for the professional programming to learn and use effectively.
   - If you already understand the basic concepts of OOP, learning java will be even easier.
   - Java inherits the C/C++ syntax and many of OO features of C++.
3. **Platform Independent and Portable**
   - Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.
   - A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based.
   - The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:
     1. Runtime Environment
     2. API(Application Programming Interface)
   - Java code can be run on multiple platforms, **for example,** Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., **Write Once and Run Anywhere(WORA).**

4. **Robust and Secured**
   Robust simply means strong. Java is robust because:
   - It uses strong memory management.
   - There is a lack of pointers that avoids security problems.
   - There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
   - There are exception handling and the type checking mechanism in Java.
5. **Architecture-neutral**
   - Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
   - In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.
6. **High-performance**

- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
- It is still a little bit slower than a compiled language (e.g., C++).
- Java is an interpreted language that is why it is slower than compiled languages,
  e.g., C, C++, etc.

7. **Multi-threaded**
   - Multi-thread means handling multiple tasks simultaneously. Java supports multi-threaded programming. This means that we need not wait for the application to finish one task before beginning another.
   - The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
   - Threads are important for multi-media, Web applications, etc.
8. **Distributed**
   - Java is distributed because it facilitates users to create distributed applications in Java. It has the ability to share both data and programs.
   - RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
9. **Dynamic and Extensible**
   - Java is a dynamic language.
   - Java is capable of dynamically linking in new class libraries, methods and objects.
   - Java programs support functions written in other languages, such as C and C++. These functions are known as native methods.
   - Native methods are linked dynamically at runtime. Java supports dynamic compilation and automatic memory management

3Q) Write about JAVA TOKENS?

Ans: A Punctuation marks, cammas, semi-colons, characters etc., is called as java tokens. Java contains different types of tokens. They are

1. Identifiers
2. Keywords
3. Operators
4. Data types
5. Strings
6. Special Symbols

1. **Identifiers**
   An identifier is nothing but a function name or variable name. A variable is nothing but a space in a memory where values can be constantly changed. To define identifiers the following rules to be followed. They are
   - The first character of identifier name should always begin with alphabet.
   - In between identifier name there should not be any special symbol except underscore
   - Keywords should not be defined as identifier.
   - Duplicate identifiers cannot be defined.
2. **Keywords:**

The words which are already existed in java language, they are known as keywords. These are also called as **reserve words**. Java contains 48 keywords.

Eg:

Abstract	Assert	Boolean	Break	Byte	Case
Catch	Char	Class	Const	Continue	Default
Do	Double	Else	Enum	Extends	Final
Finally	Float	For	Goto	If	Implements
Import	Instanceof	Int	Interface	Long	Native
Package	Private	Protected	Public	Return	Short
Static	Strictfp	Super	Switch	Synchronized	This
Throw	Throws	Transient	Try	Void	Volatile
While	True	False	Null		

3. **Operators:**

An operator is nothing but symbols which are used to operate the operands. Java support different types of operators. They are

1) Arithmetic operators
2) Relational / Comparison operators
3) Logical Operators
4) Assignment Operators
5) Bitwise Logical Operators
6) Unary operators
7) Ternary Operators

Arithmetic Operators

The arithmetic operators are used to perform arithmetic calculations such as addition, subtraction, multiplication and division. The arithmetic operators are

Operator	Meaning	Examples
+	Addition	A=10,B=20,C=A+B=>30
-	Subtraction	A=10,B=20,C=A-B=>-10
*	Multiplication	A=10,B=20,C=A*B=>200
/	Division	A=10,B=2,C=A/B=>5
%	Modulus	A=10,B=3,C=A%B=>1

Relational / Comparison Operators

The relational or comparison operators are used to compare different operands. The relational operators are

Operator	Meaning	Examples
>	Greater than	A=10,b=5,a>b =>0(true)
<	Less than	A=10,b=5,A<b =>-1(false)
>=	Greater than equals to	A=10,b=5,a>=b =>0(true)
<=	Less than equals to	A=10,b=5,A<=b =>-1(false)
==	equals to	A=10,b=5,A= =b =>-1(false)
!=	not equals to	A=10,b=5,A!=b =>0(true)

Logical Operators

The logical operators are used to combine two or more expressions into one. The logical operators are

Operator	Meaning
&&	And
||	Or
!	Not

Eg for AND:

Exp1	&&	Exp2	Result
T	&&	T	T
T	&&	F	F
F	&&	T	F
F	&&	F	F

Eg for OR:

Exp1	||	Exp2	Result
T	||	T	T
T	||	F	F
F	||	T	F
F	||	F	F

Eg for NOT:

!Exp	Result
T	F
F	T

Bitwise Logical Operators

The Bitwise logical operators are used to perform bit calculations. The Bitwise logical operators are

Operator	Meaning
>>	Right Shift
<<	Left Shift
~	Complement
^	XOR

Unary Operator

The Unary operators are used to perform unary calculations. The unary operators are incrementation and decrementation. The incrementation can be post incrementation or pre-incrementation and the decrementation can be post decrementation or pre-decrementation.

Operator	Meaning	Examples
++	Incrementation	A=5 A++ =>6
--	Decrementation	A=5 A-- =>4

Assignment Operators

The assignment operators are used to assign R.H.S value to L.H.S either before calculation or after calculation.

Operator	Meaning	Examples
=	Assigns RHS value to LHS	**a=10**
+=	Assigns RHS value to LHS after addition	**a=10 a+=3=>13 or a=a+3=>13**
-=	Assigns RHS value to LHS after subtraction	**a=10 a - =3=>7 or a=a - 3=>7**
*=	Assigns RHS value to LHS after multiplication	**a=10 a*=3=>30 or a=a*3=>30**
/=	Assigns RHS value to LHS after Division	**a=10 a/=2=>5 or a=a/2=>5**
%=	Assigns RHS value to LHS after modulus	**a=10 a%=3=>1 or a=a%3=>1**

Ternary Operators

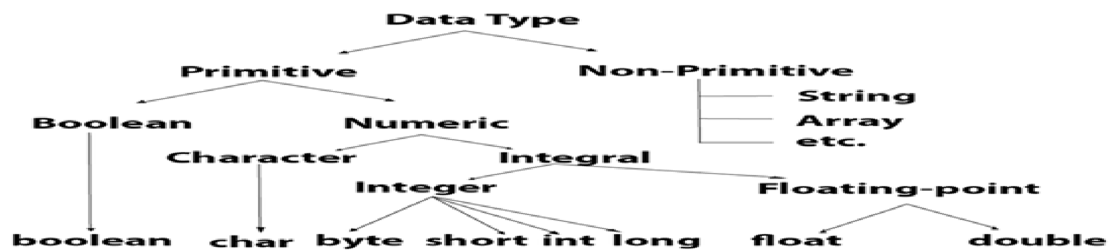
It is also called as conditional operator. The conditional operators are used to execute true statement only when the condition is true otherwise it executes false statement.

Operator	Meaning
?	Question mark
:	colon

Datatypes:

Data types specify the different sizes and values that can be stored in the variable. Java is a statically typed programming language. It means all variables must be declared before its use. That is why we need to declare variable's type and name. There are two types of data types in Java:

1. Primitive Data type
2. Non-primitive Data type



1) **Primitive Data type:**

Java supports eight primitive data types: byte, short, int, long, float, double, char and Boolean. These eight data types are further classified into four groups.

1) Integer
2) Rational Numbers (Floating point)
3) Characters
4) Boolean

Integer Type:

Integer is the whole number without any fractional point. It can hold whole numbers such as 196, -52, 4036, etc. Java supports four different types of integers, they are:

Name	Types	Range	Memory
Integer	Byte	-128 to 127	1 byte
	Short	-32,768 to 32767	2 byte
	Int	-2147,483,648 to +2147,483,647	4 byte
	Long	-9223,372,036,854,755,808 To 9223,372,036,854,755,807	8 bytes

Rational Numbers:

It is used to hold whole numbers containing fractional part such as 36.74, or -23.95. There are two types of floating point storage in java. These are:

Name	Types	Range	Memory
Rational	Float	$3.4*10^{-38}$ to $3.4*10^{+37}$	4 bytes
	Double	$1.7 * 10^{-308}$ to $1.7 * 10^{+307}$	8 bytes

Characters

It is used to store character constants in memory. Java provides a character data type called char whose type consumes a size of two bytes but can hold only a single character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive)

Name	Types	Range	Memory
Character	Char	0 to 216-1 or \u0000 to \uFFFF (0 to 65535)	2 bytes
	String	The String data type is used to store a sequence of characters. String values must be surrounded by double quotes	

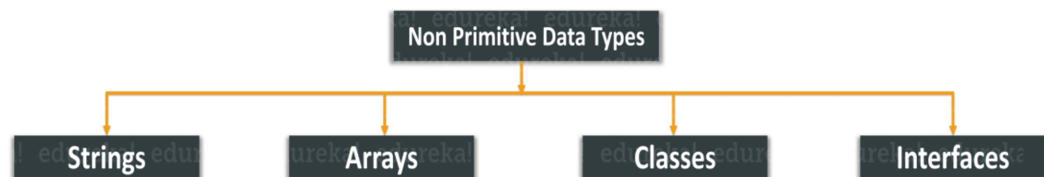
Conditional/Boolean

Boolean type is used to test a particular condition during program execution. Boolean variables can take either *true* or *false* and is denoted by the keyword boolean and usually consumes one byte of storage.

Name	Types	Range	Memory
Boolean	Boolean	True or false	1 bit

2) Non-primitive data type

Non-primitive data types are created by programmer. It is also called as **'Reference Variables'** or **'Object reference'** because it refers a memory location where data is stored.



1. **Strings:**
   String is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The *java.lang.String* class is used to create a string object. If you wish to know more about Java Strings, you can refer to this article on Strings in Java.
2. **Arrays:**
   Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same. A specific element in an array is accessed by its index. If you wish to learn Arrays in detail, then kindly check out this article on Java Arrays.
3. **Classes:**
   A class in Java is a blueprint which includes all your data. A class contains fields(variables) and methods to describe the behavior of an object.
4. **Interface:**
   Like a class, an *interface* can have methods and variables, but the methods declared in *interface* are by default abstract

5. Special Symbols:

The symbols which have special functionality are called as special symbols.

Symbol	Meaning
[ ]	Square Brackets
( )	Parantheses
{ }	Braces
" "	Double Quotes
' '	Single Quotes
;	Semi-colon

4Q) Write about Constants?

Ans: Constant is also called as Literals. A constant is a fixed value that will not be changed ever during the execution of the program. Once the programmer defines a value to a constant, it remains same through the entire program. C supports several types of constants.

1) Integer constants
2) Real constants
3) Character constants
4) String constants
5) Backslash character constants

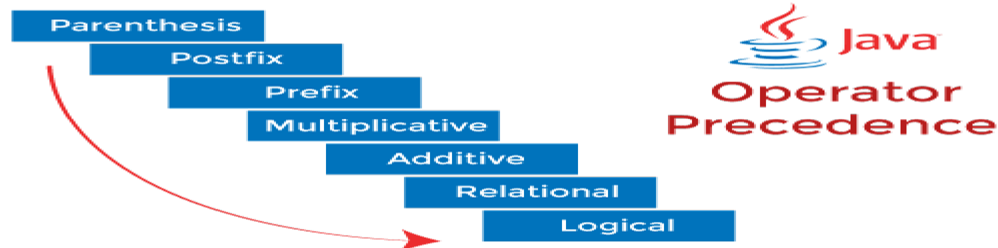
1) **Integer constants:** An integer constant is a sequence of numerical digits. There are three types of integer constants.
   1. **Decimal integer constant** consists of a set of digits 0 to 9. Decimal integers may be either positive or negative.
      Eg: 1234, 3443, -1233, 0, 92929
   2. An **octal integer constant** consists of any combination of digits from the set 0 to 7 with a leading 0. Octal values have no sign.
      Eg: 034 , 027, 0736
   3. **Hexadecimal integer constant** consist a set of digits 0 to 9 and alphabets A to F to represent the values 10 to 15. Each hexadecimal value begins with 0x.
      Eg: 0x20, 0xF5, 0xabf6
2) **Real Constants:**
   The numbers containing fractional parts are called as **real constants**. To represent fixed floating point values, we use real constants. These numbers are represented with a decimal value containing a decimal point. A real constant may be either positive or negative
   Eg: 3.1428, 0.000455, -1.346
   A real number may also be expressed in exponential notation.
   Eg: 2.1565e2
   E2 means multiply the number 10^2
   Then the above value becomes 215.65
3) **Character constants:**
   A character constant contains single character enclosed within a pair of single quotation marks. The character may be an alphabet, digit or any symbol.
   Eg: 'C', '8','$'
4) **String constants:**
   A string constant is a sequence of characters enclosed between a pair of double quotation marks. A string constant may contain alphabets, digits, any symbols and white spaces.
   Eg: "hello world" "1234" "@#$&^"
5) **Backslash Character constants:**
   Java supports backslash character constants that are used to format the output presented to the user. These characters are also called as **escape sequence characters**.

Code	Meaning
\n	New line
\a	Beep sound
\t	Horizontal Tab space
\'	Single quotes
\"	Double Quotes
\\	Backslash
\0	Null value
\r	Carriage return

5Q) What is operator precedence?

Ans:

The **operator precedence** represents how two expressions are bind together. In an expression, it determines the grouping of operators with operands and decides how an expression will evaluate. While solving an expression two things must be kept in mind the first is a **precedence** and the second is **associativity**.



Precedence

Precedence is the priority for grouping different types of operators with their operands. It is meaningful only if an expression has more than one operator with higher or lower precedence. The operators having higher precedence are evaluated first. If we want to evaluate lower precedence operators first, we must group operands by using parentheses and then evaluate.

Associativity

We must follow associativity if an expression has more than two operators of the same precedence. In such a case, an expression can be solved either **left-to-right** or **right-to-left,** accordingly.

Java Operator Precedence Table

The following table describes the precedence and associativity of operators used in Java.

Precedence	Operator	Type	Associativity
15	() [ ] .	Parentheses Array subscript Member Selection	Left to Right
14	++ ==	Unary Post-increment Unary Post-decrement	Right to Left
13	++ == + - ! ~ Type	Unary Pre-increment Unary Pre-decrement Unary Plus Unary Minus Unary logical Negation Unary Bitwise complement Unary Type Cast	Right to Left
12	* / %	Multiplication Division Modulus	Left to Right
11	+ -	Addition Subtraction	Left to Right
10	<< >> >>>	Bitwise Left Shift Bitwise Right Shift with sign extension Bitwise Right Shift with zero extension	Left to Right

9	< <= > >= instanceof	Relational less than Relational less than or equal Relational Greater than Relational Greater than or equal Type comparison (objects only)	Left to Right
8	== !=	Relational is equal to Relational is not equal to	Left to Right
7	&	Bitwise AND	Left to Right
6	^	Bitwise exclusive OR	Left to Right
5	|	Bitwise inclusive OR	Left to Right
4	&&	Logical AND	Left to Right
3	||	Logical OR	Left to Right
2	? :	Ternary Operator	Right to Left
1	= += -= *= /= %=	Assignment Operator Addition Assignment Subtraction Assignment Multiplication Assignment Division Assignment Modulus Assignment	

6Q) Java Naming Convention

Ans:

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc. But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
- All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

Advantage

1. By using standard Java naming conventions, you make your code easier to read for yourself and other programmers.

Naming Conventions of the Different Identifiers

The following table shows the popular conventions used for the different identifiers.

Identifiers Type	Naming Rules	Examples

Class	1. It should start with the uppercase letter.
	2. It should be a noun such as Color, Button, System, Thread, etc.
	3. Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	**1.** It should start with the uppercase letter.
	2. It should be an adjective such as Runnable, Remote, ActionListener.
	3. Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	1. It should start with lowercase letter. It should be a verb such as main(), print(), println().
	2. If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter
		class Employee
		{
		void **draw()**
		{
		//code snippet
		}
		}

Variable	1. It should start with a lowercase letter
	2. It should not start with the special characters
	3. If the name contains multiple words, start it with the lowercase letter followed by an uppercase
	4. Avoid using one-character variables such as x, y, z.
		class Employee
		{
		int **id**;
		//code snippet
		}

Package	1. It should be a lowercase letter such as java, lang.
	2. If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	1. It should be in uppercase letters
	2. If the name contains multiple words, it should be separated by an underscore.
	3. It may contain digits but not as the first letter.
		class Employee
		{
		static final int **MIN_AGE** = 18;
		//code snippet
		}

7Q) How many parts in Java?

Ans: Sun Microsystems Inc. has divided Java into 3 parts. They are

1. Java SE
2. Java EE
3. Java ME.

Java SE

- It is the Java Standard Edition that contains basic core Java classes.
- This edition is used to develop standard applets and applications

Java EE

- It is the Java Enterprise Edition and it contains classes they are beyond Java SE.
- In fact, we need Java SE in order to use many of the classes in Java EE.
- Java EE mainly concentrates on providing business solutions on a network.

Java ME

- It stands for Java Micro Edition. Java ME is for developers who develop code for portable device, such as a PDA or a cellular phone. Code on these devices needs to be small in size and take less memory

8Q) Displaying Formatted Output with String.format()

Ans:

- In java, String format() method returns a formatted string using the given locale, specified format string, and arguments. We can concatenate the strings using this method and at the same time, we can format the output concatenated string.
- If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.
- The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.
- There is two types of string format() method

  public static String format(Locale loc, String form, Object... args)
  public static String format(String form, Object... args)

  Parameter:
  - The locale value to be applied on the format() method
  - The format of the output string.
  - args specifying the number of arguments for the format string. It may be zero or more.

Exception Thrown:

- NullPointerException: If the format is null.
- IllegalFormatException: If the format specified is illegal or there are insufficient arguments.

//Example

```
public class FormatExample
{
	public static void main(String args[])
	{
		String name="Muthu";
		String sf1=String.format("name is %s",name);
		String sf2=String.format("value is %f",32.33434);
		String sf3=String.format("value is %32.12f",32.33434);
		System.out.println(sf1);
		System.out.println(sf2);
		System.out.println(sf3);
		}
	}
```

String Format Specifiers

Format Specifier	Data Type	Output
%a	floating point (except *BigDecimal*)	Returns Hex output of floating point number.
%c	Character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer

%e	floating point	decimal number in scientific notation
%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.

Eg:

```
public class FormatExample2
{
   public static void main(String[] args)
   {   String str1 = String.format("%d", 101);          // Integer value
       String str2 = String.format("%s", "Amar Singh"); // String value
       String str3 = String.format("%f", 101.00);       // Float value
       String str4 = String.format("%x", 101);          // Hexadecimal value
       String str5 = String.format("%c", 'c');          // Char value
       System.out.println(str1);
       System.out.println(str2);
       System.out.println(str3);
       System.out.println(str4);
       System.out.println(str5);
   } }
```

Output:

101
Amar Singh
101.000000
65
C

9Q. Write the structure of Java program?

Ans: The Structure of java program is divided into 6 sections. They are –

1. **Documentation Section**
   The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. The comment about a program can be written in 3 different levels.
    1. // Single line comment
    2. /* Paragraph Comment */
    3. /** Documentation Comment */

2. **Package Statement**
   The first statement allowed in a java file is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package. **The package statement is optional**
   Syn: package pack_name;
3. **Import statements**
   The import statement is very similar to the #include statement in C. This statement instructs the interpreter to load the class contained in the package.
   Syn: import pack_name;
4. **Interface Statements**
   An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance features in the program. Interface is a new concept in java.
5. **Class Definitions**
   A java program may contain multiple class definitions. Classes are the primary and essential elements of a java program. These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of the problem.
6. **Main Method Class**
   Since every java stand-alone program requires a main method as its starting point, this class is the essential part of a java program. A simple java program may contain only this part. The main method creates objects of various classes and establishes communications between them.

10Q) What is Variable? Write about types of variables? Write about the scope and life of variables?

Ans: A variable is nothing but a space in a memory where values can be constantly changed during the program execution.

The declaration of a variable tells the compiler what the variable name is, the data type of the variable and scope of the variable. Each variable must be declared before to use it.

Syntax: datatype var1,var2..;
Eg: int rno;
float height;
char gender;

Initializing a variable:
Initializing a variable is nothing but assigning a value to the variable. This can be done in two ways.

1) By using an assignment statement
2) By using a read statement.

Syn -1: datytype var=value;
Eg: int a=10;
Float height=160.75f;
char gender='M';

Eg 2: int a;

```
float b;
DataInputStream dis=new DataInputStream(System.in);
a=Integer.parseInt(dis.readLine());
b=Float.parseFloat(dis.readLine());
```

Scope of a Variable:

The area of the program where the variable is accessible is called "Scope" of the variable. The scope of a variable defines the life span of that variable. A variable may have any of the scopes like block scope, method scope (local scope), class scope (global scope) etc.,

Types of Variables

Java variables are classified into the following types:

1. Instance Variables
2. Class Variables
3. Local Variables.

Instance Variables:

Instance variables are the variables that are declared inside the class. These variables are created when the objects are instantiated and associated with the objects. They take different values for each object.

Class variables:

Class variables are the variables that are declared inside the class as static variables. These variables are global to entire class thus they are common to entire set of objects created for that class.

Local Variables:

Local variables are the variables that we declared and used inside the methods of a class. These variables are not accessible outside the class in which they are declared. Local variables can also be declared inside the blocks which are defined between a pair of opening and closing braces i.e., {}. The variables which are declared inside a block will be accessible only that block.

Eg:

```
public class sample
{
        static String nm;   // class / Global variable
        intrno;    // Instance variables
        public void display()
        {
        int fees=7000;   // local variables
        System.out.println("Rno =" + rno);
        System.out.println("Name =" + nm);
        System.out.println("Fees =" + fees);
        }
    public static void main(String[] args) throws Exception
    {   Sample ob=new sample();
        sample.nm="raju";
        ob.rno=1001;
        ob.display(); }}
```

11Q) How to display data in java using System class

Ans: Java brings various Streams with its I/O package that helps the user to perform all the **input-output operations.** These streams support all the types of **objects, data-types, characters, files etc** to fully execute the I/O operations.

Before exploring various input and output streams. Let look at **3 standard or default streams** that Java has to provide which are also most common in use:

1. **System.in:**
   This is the **standard input stream** that is used to read characters from the keyboard or any other standard input device.
2. **System.out:**
   - This is the **standard output stream** that is used to produce the result of a program on an output device like the computer screen.
   - Here is a list of the various print functions that we use to output statements:
   1. **print():**
      This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

      Syn: System.out.print(Parameter) ;

```
class Demo_print
{
    public static void main(String[] args)
    {
            System.out.print("S. Muthahar ! ");
            System.out.print(" Cell No: 9885545889 ");
            System.out.print(" Kadapa! ");
    }
}
```

Output: S. Muthahar ! Cell No: 9885545889 Kadapa!

   2. **println():**

This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.

Syn: System.out.println(Parameter) ;

```
// Java code to illustrate println()
class Demo_print
{
        public static void main(String[] args)
        {
                System.out.println("S. Muthahar ! ");
                System.out.println(" Cell No: 9885545889 ");
                System.out.println(" Kadapa! ");
        }
}
```

Output: S. Muthahar !
Cell No: 9885545889
Kadapa!

3. printf():

This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.

Syn:
System.out.printf("<Message><format_Controls><escape sequences>"+arg1+arg2..) ;

Eg: // A Java program to demonstrate working of printf() in Java

```
class JavaFormatter1
{
        public static void main(String args[])
        {   int x = 100;
            System.out.printf("Printing simple integer: x = %d\n", x);
            System.out.printf("Formatted with precison: PI = %.2f\n", Math.PI);
            float n = 5.2f;
            System.out.printf("Formatted to specific width: n = %.4f\n",n);
            n =2324435.3f;
            System.out.printf("Formatted to right margin: n = %20.4f\n",n);
        }
}
```

Output:

Printing simple integer: x = 100
Formatted with precison: PI = 3.14
Formatted to specific width: n = 5.2000
Formatted to right margin: n = 2324435.2500

3. System.err

- This is the **standard error stream** that is used to output all the error data that a program might throw, on a computer screen or any standard output device.
- This stream also uses all the 3 above-mentioned functions to output the error data:
  1. print()
  2. println()
  3. printf()

12Q) How to read input in Java?

Ans: There are several ways to get input from the user in Java. You can to get input by using **Scanner object** and **DataInputStream object**.

1. **Scanner class:**
   - Scanner class in Java is found in the java.util package.
   - Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
   - The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default.
   - It provides many methods to read and parse various primitive values.
   - The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression.
   - It is the simplest way to get input in Java.
   - By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc
   - The Java Scanner class extends Object class and implements Iterator and Closeable interfaces.
   - The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

   Syntax: Scanner ob=new Scanner(System.in);

S.No.	Method	Description
1	Next()	It is used to get the next complete token from the scanner which is in use
2	nextBoolean()	It reads a Boolean value from the user
3	nextByte()	It reads a byte value from the user
4.	nextDouble()	It reads a double value from the user
5	nextFloat()	It reads a float value from the user
6	nextInt()	It reads a integer value from the user
7	nextLine()	It reads a string value from the user
8	nextLong()	It reads a long value from the user
9	nextShort()	It reads a short value from the user
10	hasNext()	It returns true if this canner has another token in its input.

11	hasNextBoolean()	It is used to check if the next token in this scanner’s input can be interpreted as a Boolean using the nextBoolean() method or not
12	hasNextByte()	It is used to check if the next token in this scanner's input can be interpreted as a Byte using the nextBigDecimal() method or not
13	hasNextDouble()	It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextByte() method or not.
14	hasNextFloat()	It is used to check if the next token in this scanner's input can be interpreted as a Float using the nextFloat() method or not.
15	hasNextInt()	It is used to check if the next token in this scanner's input can be interpreted as an int using the nextInt() method or not.
16	hasNextLine()	It is used to check if there is another line in the input of this scanner or not.
17	hasNextLong()	It is used to check if the next token in this scanner's input can be interpreted as a Long using the nextLong() method or not.

12Q) Write about Control Statements?

Ans: The control statements that java supports are

1. Conditional Statements
2. Loops
3. Unconditional Statements

Conditional Statements

The conditional statements are used to control the flow of execution of statements of a program. Java support different types of conditional statements. They are

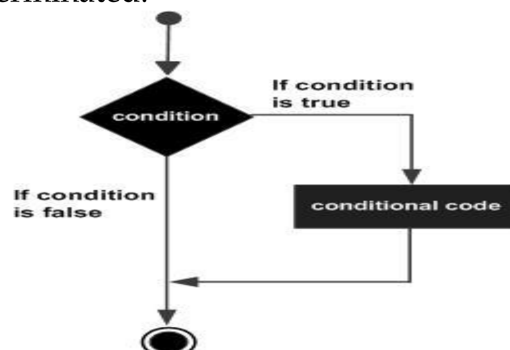
1. Simple if
2. If..else
3. If..else if
4. Neste if
5. Switch
6. Ternary operators

1. **Simple if**:

   This conditional statement executes true statements only when the condition is true, otherwise the, if statement will be terminated.

   Syn:

```
if (expression)
{
        Statement1;
        Statement2;
}
```

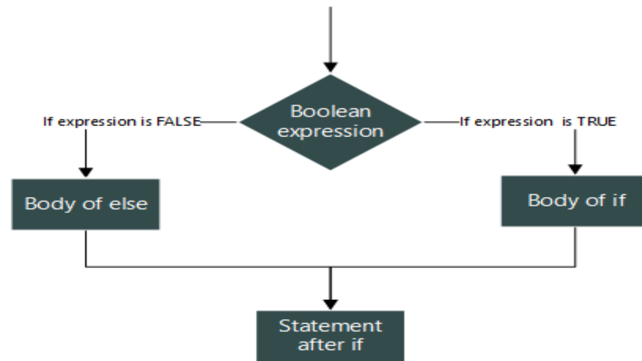


2. if .. else:

This conditional statement can execute true statements only when the condition is true otherwise it executes false statements.

Syn:

```
if( expression )
{       statement1;
        statement2;
  }
else
{
        statement1;
        statement2;
}
```

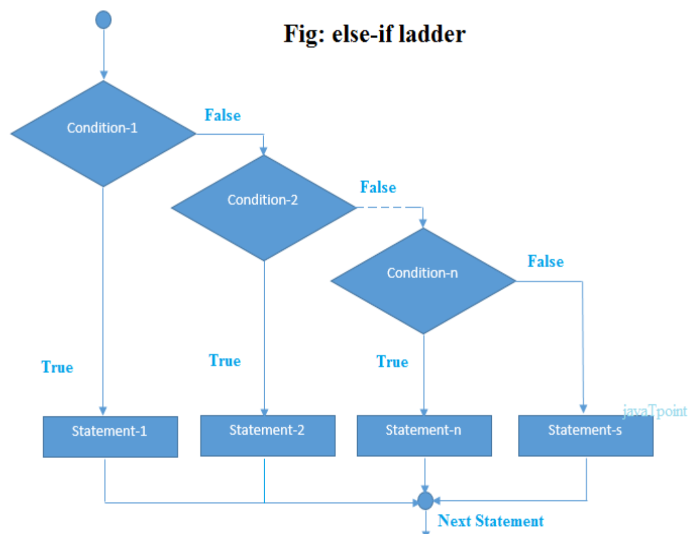


3. if .. else if:

It is also called as a branching statement or Ladder statement. This conditional statement executes statements based on its respective condition.

Syn: if(expression1)

```
{
        statement1;
        statement2;
 }
else if(expression2)
{
        statement1;
        statement2;
}
else
{
        statement1;
        statement2;

 }
```

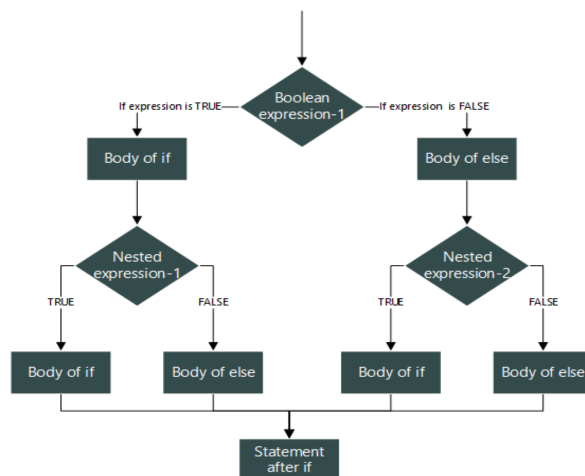


4. Nested if:

A if statement which can execute within a if statement it is called as nested if conditional statement. It is also called as a multi-level branching statement.

Syn: if(exp1)

```
{
  if(exp1.1)
  {     statement1;
        statement2;
  }
  else
  if(exp1.2)
  {     statement1;
        statement2;
      }
 }
else
```



```
if(exp2)
{
    if(exp2.1)
    {   statement1;
        statement2;
        }
    else
    if(exp1.2)
    {   statement1;
        statement2;
        }
  }
```

5. **Switch:**
   It is also called as branching statement. This conditional statement is very similar to if..else if conditional statement. The execution of switch statement is very faster than if..else if conditional statement.
   Syntax: switch(exp)

```
{
    case constant1 :    statement1;
                        statement2;
                        break;
    case constant2:     statement1;
                        statement2;
                        break;
    case constant3:     statement1;
                        statement2;
                        break;
    default:            statement;
}
```

6. **Ternary Operators**
   It is also as conditional operators. These conditional operators are used to execute a true statement only when the condition is true otherwise it executes a false statement
   Syn: exp ? True statement: false statement;

Loops

Executing a statement or group of statements for a repeated number of times, it is called as a loop. Java language contains different types of loops. They are –

1. for .. loop
2. while .. loop
3. do .. while loop

for..loop:

- ⇨ A for..loop is used to execute a statement or group of statements for repeated number of times.
- ⇨ A for.. loop can be used only when we know the number of times the loop has to execute.
- ⇨ A for .. loop can be constructed only by using three steps

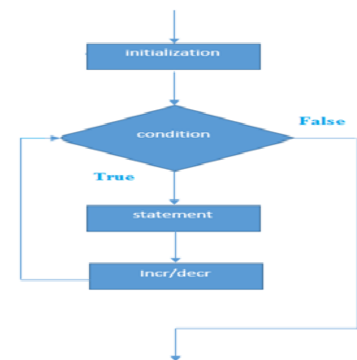
1. **Initialization**:
   It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**:
   It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Increment/Decrement**:
   It increments or decrements the variable value, it is an optional condition.

Syn-1:

```
for(initialvalue; conditionalvalue;changingvalue)
{
        block of statements;
}
```

Syn-2:

```
initial value;
for( ; conditional value; )
{
        block of statements;
        Changing value;
}
```



Nested for..loop:

A for..loop which can execute with in a for .. loop itself, such loop can be called as nested for .. loop. The execution of this loop always depends on the conditional value of the loop.

Syntax:

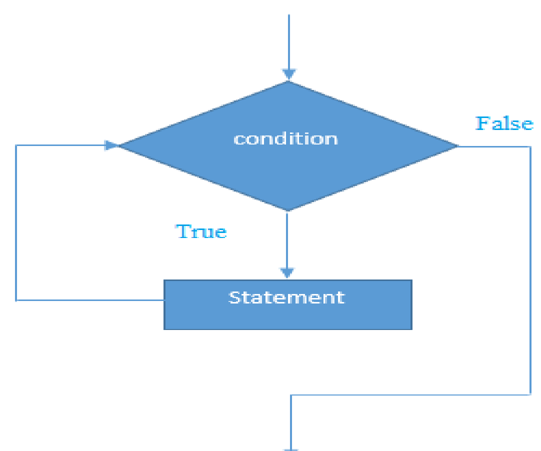
```
for(initialvalue; conditionalvalue; changing value)
{
        for(initialvalue ; conditional value; changing value)
        {       statement1;
                statement2;
        }
}
```

While ..loop:

It is also called as a conditional loop. In this loop a statement or group of statements can be executed for a repeated number of times only when the condition is true otherwise the loop will be terminated.

Syn:

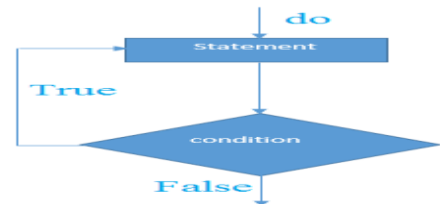
```
initial value;
while(expression)
{
        Block of statements;

        Changing value;
}
```



do ..while loop:

The *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop. The *do-while loop* is executed at least once because condition is checked after loop body.

Syn:
```
initial value;
do
{
        Block of statements;
        Changing value;
} while(expression);
```



Jumping Statements

When the programmer wants to skip some portion of the loop body, they can use jumping statements. Jumping statements are used to transfer the control from one part of the code to another part of the code. Java supports the following statements as jumping statements which can be used to control the flow of the control.

1. break
2. continue

break:

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syn:
```
initial value;
while(expression)
{
        Block of statements;
        Changing value;
        if(exp)
          break;
}
```

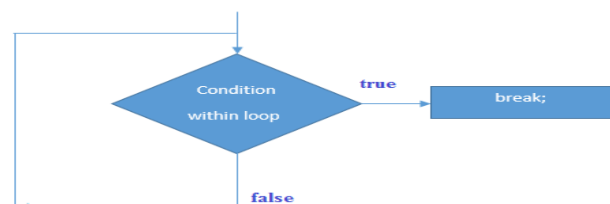


Figure: Flowchart of break statement

Eg:
```
public class BreakEg
{
 public static void main(String[] args)
 {
  for(int i=1;i<=10;i++)
  {
     if(i==5)
          break;
    System.out.print(i);
  }
 }
}
```
Output: 1 2 3 4

Continue:

- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
- The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
- We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax: while(exp)

```
{
        Statement1;
        Statement2
        if(exp)
                continue;
}
```

UNIT – II

1Q) Write about Command Line Arguments?

Ans:

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** numbers of arguments from the command prompt.
- The arguments whatever we passed they can be stored in String array and they can act as string
- The user can enter command-line arguments when invoking the application. When running the java program from java command, the arguments are provided after the name of the class separated by space.

```
// Program to demonstrate command line arguments.
class sample
{
        public static void main(String[] args)
        {
                int a,b;
                a=Integer.parseInt(args[0]);
                b=Integer.parseInt(args[1]);
                System.out.println("Sum =" + (a+b));
        }
 }
C:\>javac sample.java
C:\> java sample 10 20
        Sum = 30
```

2Q) What is Array? Explain different types of Arrays?

Ans:

- An array is a collection of similar type of elements which have a contiguous memory location.
- **Java array** is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- Unlike C/C++, we can get the length of the array using the **length member**. In C/C++, we need to use the sizeof operator.
- In Java, array is an object of a dynamically generated class.
- We can store primitive values or objects in an array in Java.
- Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.

Types of Array

1) **Single Dimension Array**
2) **Double Dimension Array**
3) **Multi-Dimension Array**
4) **Jagged Array**

SINGLE DIMENSION ARRAY:

- ⇨ An array which contains only one row and it can have set of columns, it is called as single dimension array.
- ⇨ It is also called as single subscripted value array or one dimensional array.
- ⇨ Array position always starts from zero
- ⇨ Each section of an array is called as an element.

Syntax: datatype [ ]var=new datatype[size];

(or)

datatypevar[ ]=new [size]datatype;

Eg: int a[ ]= new int[5];

int b[]=new [5]int;

Whenever memory is allocated for array variable, JVM assigns an attribute called **length** to it.

```
int x[ ] = new int[10] ;
System.out.println(x.length ); => 10
```

Initialization of an 1-D array:

You can initialize an array in java either one by one or using a single statement as follows –

Initialization of values in an array –

Eg: - 1. int a[4]= { 10,20,30,40 };

a[0] = 10
a[1] = 20
a[2] = 30
a[3] = 40

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ]. If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

2. int x[] ={10,20,30,40,50};

x[0] = 10
x[1] = 20
x[2] = 30
x[3] = 40
x[4] = 50

DOUBLE DIMENSION ARRAY

It is defined as a collection of one dimensional array is called as double dimension array. It is also called as double sub-scripted value array.

Syntax: datatype var[][]=new datatype[rs][cs];

eg:- int a[ ][ ]=new int[3][3];

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

While allocating memory to a two dimensional array, number of row specification is must but not column specification.

```
eg:-  int a=new int[4][4];
      int b[ ]=new int[3][4];
        System.out.println(b.length);
        System.out.println(b[1].length);
int c[ ][ ] = new int [3][ ]      ;
        c[0]=new int [2];
        c[1]=new int[3];
        c[2]=new int[4];
```

Initialization of an array:

Eg-1:int a[][]={{10,20,30},{40,50,60},{70,80,90}};

a[0][0]	a[0][1]	a[0][2]
10	**20**	**30**
a[0][0]	a[0][1]	a[0][2]
40	**50**	**60**
a[0][0]	a[0][1]	a[0][2]
70	**80**	**90**

Eg:2 int x[][]={{1},{2,3},{4,5,6},{7,8,9,10}}

x[0][0]			
1			
x[1][0]	x[1][1]		
2	**3**		
x[2][0]	x[2][1]	x[2][2]	
4	**5**	**6**	
x[2][0]	x[2][1]	x[2][2]	x[2][3]
7	**8**	**9**	**10**

Multi-Dimensional array:

Three dimensional arrays are also called as space array. In this space rows and columns are taken. Three dimensional array will require three subscripts i.e., three pairs of square brackets

Declaration of three dimensional array

Similar to one and two dimensional arrays most be declared being used. The declaration statements tells the compiler the name of the array, the data type of each element in the array and size of each dimension. A three dimensional array is declared as

Syn: datatype array_name[][][]=new type[size][size][size];

Initialization of 3D array:

You can initialize a three dimensional array in similar way like a two dimensional array

Eg:

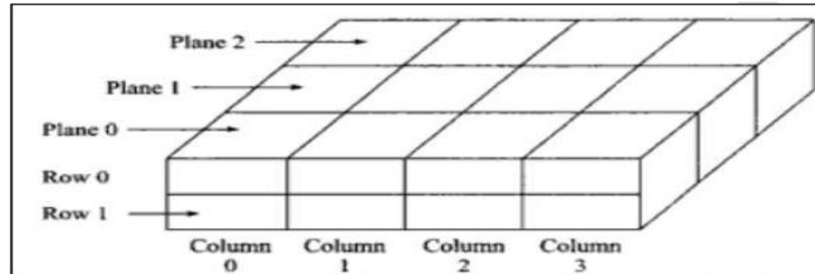
```
Int a[2][3][4]={     { { 1,2,3,4}, {5,6,7,8}, {9,10,11,12}},
                     { {13,14,15,16},{17,18,19,20},{21,22,23,24}}

              };
```

Accessing elements of the 3D array:

Three dimensional arrays contains three subscripts, we will use three for loops to access the elements

For example, arr contains 3*2*4=24 elements.

The arr[3][2][4] can be represented as shown below



Eg:

```
class ThreeD
{
	public static void main(String[] args)
	{
		int a[][][]=new int[2][2][2];
		int i,j,k,m=1;
		for(i=0;i<2;i++)
		{
		  for(j=0;j<2;j++)
		  {
			for(k=0;k<2;k++)
			{
				a[i][j][k]=m;
				m++;
			}
		  }
		 }
	System.out.println("Three Dimensional Array \n");
		for(i=0;i<2;i++)
		{
		  for(j=0;j<2;j++)
		  {
			for(k=0;k<2;k++)
			{
				System.out.print(a[i][j][k]+"  ");
			}
			System.out.println();
		  }
			System.out.println();
		 }
		}
}
```

Jagged Array

A double dimension array can be represented jagged array that means an array which can have any number of rows and each row of array can have different columns.

Syn: datatype array_name[ ][ ]=new datatype[rs][ ];

array_name[index]=new datatype[cs];

Eg: int x[][]={{1},{2,3},{4,5,6},{7,8,9,10}}

x[0][0]
1

x[1][0] x[1][1]
2 3

x[2][0] x[2][1] x[2][2]
4 5 6

x[2][0] x[2][1] x[2][2] x[2][3]
7 8 9 10

3Q) String class / Immutable String class

Ans:

- String is a sequence of characters but it's not a primitive type.
- When we create a string in java, it actually creates an object of type String.
- String is **immutable object** which means that it cannot be changed once it is created.
- String is the only class where operator overloading is supported in java. We can concat two strings using + operator.
- Java provides two useful classes for String manipulation – StringBuffer and StringBuilder.
- A Java string is not a character array and is not NULL terminated.
- It is a final class defined in **java.lang package.**

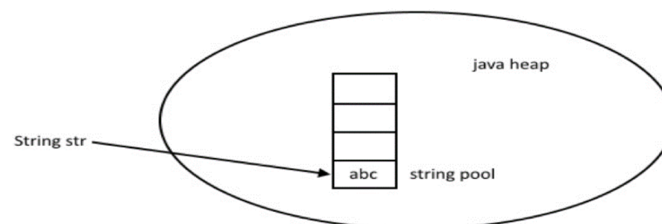
Different Ways to Create String

There are many ways to create a string object in java, some of the popular ones are given below.

1. **Using string literal**

   This is the most common way of creating string. In this case a string literal is enclosed with double quotes.

   Eg: String str = "abc";



   When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool.

2. ***Using new keyword***

We can create String object using new operator, just like any normal java class. There are several constructors available in String class to get String from char array, byte array, StringBuffer and StringBuilder.

Syn: String ob=new String();
Or
String ob=new String("String");

Eg:

```
String str  =  new String("abc");
char[] a = {'a', 'b', 'c'};
String str2  =  new String(a);
```

Eg:

```
public class StrExample
{
public static void main(String args[])
{
        String s1="java";//creating string by java string literal
        char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);//converting char array to string
        String s3=new String("example");//creating java string by new keyword
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        }
}
```

o/p:
java
strings
example

Methods:
String s="java in muthusoft";

1. **charAt():** This function returns nth character from a given string.
   Eg: s.charAt(2) => v
2. **length()**: This function returns the length of a given string.
   Eg: s.length() => 17
3. **concat():** This function is used to combine two strings into one.
   Eg:
   s.concat("kadapa")
   java in muthusoftkadapa
4. **trim():** This function removes trailing and leading spaces of string
   eg: String s1="   computer   ";
   s1.trim() => computer
5. **indexOf():** This function returns the position of the given character.
   Eg: String s="java in muthusoft"
   s.indexOf("java") => 0

6. **lastIndexOf():** This function returns the last index position of given string.
   Eg: String s="java in muthusoft"
   s.lastIndexOf("in") => 5
7. **substring():** This function gives substring starting from nth character
   Eg: String s="java in muthusoft"
   s.substring(5) => in muthusoft
8. **startsWith():**
   This function returns Boolean expression true/false, whether a given string starts with specified charaters or not.
   Eg: String s="java in muthusoft"
   s.startswith("java") => true
9. **endsWith():**
   This function returns Boolean expression true/false, whether a given string ends with specified characters or not.
   Eg: String s="java in muthusoft"
   s.endsWith("in") => false
10. **equals():**
    This function is used to compare two strings and find whether they are exact or not. If they are exact it returns true otherwise it returns false.
    Eg:
    String s1="hello";
    String s2="Hello";
    s1.equals(s2) => false
    s1.equals("hello") => true
11. **equalsIgnoreCase():**
    This function is used to compare two strings and find whether they are similar or not. If they are similar it returns true otherwise it returns false.
    Eg: String s1="hello";
    String s2="Hello";
    s1.equalsIgnoreCase(s2) => true
12. **toLowerCase():** This function is used to convert a string into lower case
    Eg: String s="COMPUTER"
    s.toLowerCase() => computer
13. **toUpperCase():** This function is used to convert a string into upper case.
    Eg: String s="computer"
    s.toUpperCase() => COMPUTER

// Program to implement string functions

```
import java.io.*;
classstrtest
{
public static void main(String[] args) throws Exception
 {
        DataInputStream dis=new DataInputStream(System.in);
        String s1=new String();
        String s2=new String();
```

```
s1=dis.readLine();
s2=dis.readLine();
if(s1.equals(s2))
System.out.println("Exact Strings");
System.out.println("String in uppercase ="+s1.toUpperCase());
System.out.println("String in lowercase ="+s2.toLowerCase());
System.out.println("String combination = " + s1.concat(s2));
System.out.println("String length =" + s1.length());
System.out.println("String length =" + s1.substring(1,5));
 }}
```

4Q) Write about StringBuffer class

Ans:

StringBuffer is similar to a String but they have lot of difference. While String creates, strings of fixed length. StringBuffer creates strings of flexible length that can be modified in terms of both length and content. This is similar to string class defined as final in java.lang package. **StringBuffer is a mutable class**. We can append data to StringBuffer or insert data into StringBuffer without reference. To do this StringBuffer uses 16 bytes of memory to keep track of reference.

Syntax: StringBuffersb=new StringBuffer( );

```
sb.append("java");
sb.length( ) => 4
sb.capacity( ) => 16
sb.toString( ) => java
sb.reverse()=avaj
```

To apply string methods on StringBuffer( ), convert StringBuffer to string using toString() method.

StringBuffer Methods:

1. **toString()**: This method is used to convert object into string.

Eg: StringBuffer sb1=new StringBffer();

```
sb1.append("hello");
sb1.toString( ).toUpperCase( ) =>HELLO
```

2.**append( ):** This method is used to append a string into StringBuffer object.

Eg: StringBuffer sb1=new StringBuffer();

```
sb1.append("hello");
```

3. reverse(): This function is used to reverse a string object.

Eg: StringBuffer sb1=new StringBuffer();

```
sb1.append("hello");
sb1.reverse( ) =>olleh
```

4. length(): This function is used to find the length of object.

Eg: StringBuffer sb1=new StringBuffer();

```
sb1.append("hello");
sb1.length() => 5
```

6. capacity(): This function returns the capacity of stringbuffer object.

Eg: StringBuffer sb1=new StringBffer();

```
sb1.capacity() => 16
```

5Q) Write about String Comparison?

Ans:

We can compare String in Java on the basis of content and reference. It is used in **authentication**, **sorting**, **reference matching** (by == operator) etc. There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

1) By Using equals() Method

- The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:
  1. **public boolean equals(Object another)** compares this string to the specified object.
  2. **public boolean equalsIgnoreCase(String another)** compares this string to another string, ignoring case.

Teststringcomparison1.java

```
class Teststringcomparison1
{
    public static void main(String args[])
    {       String s1="Sachin";
            String s2="Sachin";
            String s3=new String("Sachin");
            String s4="Saurav";
            System.out.println(s1.equals(s2));//true
            System.out.println(s1.equals(s3));//true
            System.out.println(s1.equals(s4));//false
    }}
```

2) By Using == operator

```
class Teststringcomparison3
{
public static void main(String args[])
{
        String s1="Sachin";
        String s2="Sachin";
        String s3=new String("Sachin");
        System.out.println(s1==s2);
        System.out.println(s1==s3); } }
```

3) By Using compareTo() method

The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two String objects. If:

- **s1 == s2** : The method returns 0.
- **s1 > s2** : The method returns a positive value.
- **s1 < s2** : The method returns a negative value.

```
class Teststringcomparison4
{
    public static void main(String args[])
    {
        String s1="Sachin";
        String s2="Sachin";
        String s3="Ratan";
        System.out.println(s1.compareTo(s2));
        System.out.println(s1.compareTo(s3));
        System.out.println(s3.compareTo(s1));
    }
}
```

6Q) Problems in Procedure Oriented Approach

Ans:

- The program code is harder to write when Procedural Programming is employed
- The Procedural code is often not reusable, which may pose the need to recreate the code if is needed to use in another application
- Difficult to relate with real-world objects
- The importance is given to the operation rather than the data, which might pose issues in some data-sensitive cases
- The data is exposed to the whole program, making it not so much security friendly

7Q) Write the Features of OOP?

Ans:

1. Emphasis is on data rather than procedure.
2. Programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure
5. Data is hidden and cannot be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added whenever necessary
8. Follows bottom-up approach in program design

8Q) Write about class, object and methods?

Ans:

Class:

- A class is a group of objects which have common properties.
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.
- A class is the way to bind data and its associated functions together. It allows the data to be hidden if necessary from the external use. When defining a class we are creating a new abstract data type that can be treated like any other built in data type.
- Classes are user-defined data types and behave like the built-in types of a programming language.

- A class in Java can contain **Fields, Methods, Constructors, Blocks & Nested class.** These members of class can be defined by access specifiers. They are –
  1. **Private:** The members defined under this access specifier can be accessed with in a class, functions of a class and friend functions.
  2. **Protected:** The members defined under this access specifier can be accessed with in a class, functions of a class, friend functions and in sub-classes.
  3. **Public:** The members which defined under this access specifier can be accessed anywhere in a program. Mostly methods should be defined as public

  Syn:
```
Class <class_name>
{
        member1;
        member2;
        <returntype> method1([arglist])
        {
                Block of statements;
        }
}
```

Object

- An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.
- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.
- An object has three characteristics:
  1) **State:** represents the data (value) of an object.
  2) **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
  3) **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Syn: class_name ob=new class_name()

// Program to demonstrate class and object

Eg-1:
```
class Student
{   int id;
    String name;
    public static void main(String args[])
    {
        Student s1=new Student();//creating an object of Student
        System.out.println(s1.id);//accessing member through reference variable
        System.out.println(s1.name);
    }
}
```
Output:

0
Null

Eg-2:

```
class Student
{       int id;
        String name;
}
class TestStudent1
{
        public static void main(String args[])
        {       Student s1=new Student();
                System.out.println(s1.id);
                System.out.println(s1.name);
        }
}
```

Output:
0
Null

Anonymous object:

- Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.
- If you have to use an object only once, an anonymous object is a good approach.

new Calculation(); //anonymous object

Eg:

```
class Calculation
{
        void fact(int  n)
        {
                int fact=1;

                for(int i=1;i<=n;i++)
                        fact=fact*i;
                 System.out.println("factorial is "+fact);
         }
public static void main(String args[])
{
   new Calculation().fact(5);//calling method with anonymous object
        }
}
```

Output:
Factorial is 120

9Q) What is Method? Write about Method header/ prototype?
Ans:

A block of statements which performs a particular task, it is called as a function.

Syntax: [modifier] returntype method_name ([parameter list])

```
{   Statement1;
    Statement2;
    return value;
}
```

The following are four basic parts of method declaration,

1. returntype
2. Method name
3. list of parameters
4. Body of the method.
5. Return statement

In the above syntax,

- **'return type'** specifies the type of value the method returns,'
- **Method_name**' is valid identifier.
- '**parameter-list**' is enclosed in parenthesis; given to the method as input. If suppose we don't need any input data then the declaration retains empty parenthesis.
- '**body of method**' describes the operations to be performed on the data. We can overload and override methods. We can declare methods as final. By doing so, the method is redefined in a subclass we can also indicate that a method should always be redefined in a subclass using **abstract**.

Eg:

```
class briefcase
  {
    double w, h, d;
    void vol()
    {
        System.out.println("Volume=" + (w*h*d));            }
}
Class demomethod
{
    public static void main(String[] args)
    {   briefcase b1=new briefcase();
        briefcase b2=new briefcase();
        b1.w=15;
        b1.h=25;
        b1.d=20
        b2.w=4;
        b2.h=7
        b2.d=10
        b1.vol();
        b2.vol();
        } }
```

10Q) Write about the type of Methods

Ans:

There are two ways of passing arguments to method. They are,

1. Call-by-value
2. Call-by-reference

Call-by-value:

In call-by value, the values of actual arguments are passed to the called method. In this method, the copy of actual arguments is made and passed to the formal parameters of the called method. Doing this prevents the values of actual arguments to be changed. Therefore, in call-by-value method, changes made to the formal parameters will have no effect on the actual arguments.

```
Syn:    Class <class_name>
        {
                member1;
                member2;
                [<modifier>][<static>] <void> method1(arg1,arg2)
                {
                        Block of Statements;
                }
                [<modifier>][<static>] <void> method2(arg1,arg2)
                {
                        Block of Statements;
                }
                public static void main(String[] args)
                {
                        Class_name ob=new class_name();
                        Ob.method1(arg1,arg2);
                        Ob.method2(arg1,arg2);
                }
        }
```

//Eg: program to demonstrate the passing of arguments to a calling function.

```
class sample
{
        public void vol(int a, intb,int c)
        {       int vol=a*b*c;
                System.out.println("Volume:"+vol);
        }
}
class impl
{
public static void main(String[] args)
{       int a=10,b=20,c=3;
        sample ob=new sample();
        ob.vol(a,b,c);
          }
  }
```

11Q) Passing Array to Function In Java

Ans:

Generally, the purpose of passing an array to a function is to transfer a large amount of data between methods. To pass an array to a function, just pass the array as function's parameter (as normal variables), and when we pass an array to a function as an argument, in actual the address of the array in the memory is passed, which is the reference. Thus, any changes in the array within the method will affect the actual array values.

Eg-1:

```
import java.util.Scanner;
class array
{
        public int max(int [] array)
        {
                int max = 0;
                for(int i=0; i<array.length; i++ )
                {
                        if(array[i]>max)
                                max = array[i];
                }
        return max;
}
public int min(int [] array)
{
        int min = array[0];
        for(int i = 0; i<array.length; i++ )
        {
                if(array[i]<min)
                        min = array[i];
        }
    return min;
}
public static void main(String args[])
{
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the array range");
        int size = sc.nextInt();
        int[] arr = new int[size];
        System.out.println("Enter the elements of the array ::");
        for(int i=0; i<size; i++)
                arr[i] = sc.nextInt();
        array m = new array();
        System.out.println("Maximum value in the array is::"+m.max(arr));
        System.out.println("Minimum value in the array is::"+m.min(arr));
    }
}
```

Eg-2:

```
class Bsort
  {
```

```
public static void main(String[] args)
{       int[] n={12,24,2,89,34,45};
        System.out.println("Before sorting");
        display(n);
        sort(n);
        System.out.println("\n After Sorting  :");
        display(n);
}
static void display(int n[])
{
        for(int i=0; i<n.length;i++)
        System.out.print(n[i] + " ");
}
static void sort(int n[])
{
        int i, j, temp;
        for(i=0; i<n.length-i;i++)
        {
                for(j=0; j<n.length-i-1;j++)
                {
                        if(n[j]>n[j+1])
                        {
                                temp = n[j];
                                n[j] = n[j+1];
                                n[j+1] = temp;   }
                }
        }
}
    }
```

Call-by-reference:

In call-by-reference method, an address of a variable i.e., the reference or an object is passed as a parameter to the called method. It means the calling method passes the address of a variable to the formal parameters of the called method. In call-by-reference method, the changes made to the formal parameters will affect the actual parameters.

```
Syn:    Class <class_name>
        {       member1;
                member2;
                [<modifier>][<static>] <void> method1(class_name ob)
                {
                        Statement1
                        Statement2
                }
        }
        Class class_name2
        {
```

```
                public static void main(String[] args)
                {
                        class_name ob=new class_name();
                        Ob.method1(ob);
                }
        }
```

Eg:
```
class Operation2
   {
      int data=50;
      void change(Operation2 op)
      {
              op.data=op.data+100;//changes will be in the instance variable
       }
   public static void main(String args[])
   {
            Operation2 op=new Operation2();
            System.out.println("before change "+op.data);
            op.change(op);//passing object
            System.out.println("after change "+op.data);
      }
}
```

Output

```
   before change 50
   after change 150
```

12Q) What is Recursion? Write the example program

Ans:

A function which can call by itself, for repeated number of times, it is called as recursive functions. Recursive functions are used to build data structures like stacks, queues, lists etc.,

// Create a recursive function to display factorial of a given number

```
import java.io.*;
class rectest
{
  public static void main(String[] args) throws Exception
  {
   DataInputStream dis=new DataInputStream(System.in);
   int n,x;
   System.out.println("Enter n value");
      n=Integer.parseInt(dis.readLine());
      x=factorial(n);
      System.out.println("Factorial =" + x);
   }

   public static void factorial(int n)
   {    int f=1;
```

```
        if(n==0)
        f=1;
        else
        f=n * factorial(n-1);
        return f;
      }
}
```

13Q) Write about Method Overloading?

Ans: A method with identical name can be defined for any number of times, whereas its arguments or return type should be differ, it is called as **method overloading**. The invocation of methods depends on the arguments passed or based on the return type.

Syn:

```
Class class_name
{
        public <returntype>method_name(arg1,arg2)
        {       Statement1;
                Statement2;
        }
        public <returntype>method_name(arg1,arg2)
        {
                Statement1;
                Statement2;
        }
}
```

// Eg-1: Program to demonstrate Method Overloading by changing No. of arguments

```
class Adder
{
   static int add(int a,int b)
   {     return a+b;      }
   static int add(int a,int b,int c)
   {     return a+b+c;   }
    public static void main(String[] args)
    {
        System.out.println(add(11,11));
        System.out.println(add(11,11,11));
        }
}
```

Output:

22
33

14Q) Write about Constructors?

Ans:

- A constructor is a special type of member function which is used to initialize the members of class.

- A constructor need not be called explicitly by the user, it can be called automatically when we create an object of the class.
- To create a constructor the following rules to be followed. They are
  1. The constructor name should be same as that of class name.
  2. A constructor may or may not have arguments.
  3. A constructor should not have any return types even void also.
  4. A constructor should not be defined as static

Types of Constructors

Constructors are of different types. They are

1) **Default constructor**
2) **Parameterized constructor.**
3) **Constructor Overloading**
4) **Copy Constructor**

Default Constructor:

- A constructor which doesn't have any arguments
- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type
- It can be called automatically when we create an object.

Syn:
```
class class_name
{
        class_name()
        {
                Default values to members of class;        }
}
```

Eg:-1
```
class Student3
{       int id;
        String name;
        void display()
        {       System.out.println(id+" "+name); }
        public static void main(String args[])
        {       Student3 s1=new Student3();
                Student3 s2=new Student3();
                s1.display();
                s2.display();
                }
 }
```

Output:

0 null
0 null

Parameterized constructor:

- A construct which can have one or more parameters can be called as parameterized constructor.

- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.
- The parameterized constructor can be invoked automatically when we create an object and while creating an object we should pass arguments to a constructor.

Syntax:
```
class class_name
{
      class_name(arg)
      {     initialize single argument;       }
}
```

//Eg: Program to demonstrate parameterized constructor
```
import java.util.*;
class sums
{
	mtable(int n)
	{
		for(int i=1;i<=10;i++)
			System.out.println(n+"x"+i+"="+n*i);
	 }
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter N value");
		int n=sc.nextInt();
		mtable ob=new mtable(n);
		}
	}
```

Constructor Overloading:

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Syntax:
```
Class class_name
	{
		 Class_name( )
		{	Initialize the members;	}
		Class_name(arg1,arg2)
		{	Initialize the members;	}
		public static void main(String[ ] args)
		{	statement1;
			statement2;
			}
		.}
```

// Example program to demonstrate constructor overloading
```
import java.util.*;
class ParaTest
```

```
{
        ParaTest(int a,int b)
        {
                System.out.println("A value =" + a);
                System.out.println("B value =" + b);
         }
        ParaTest(double x,double y)
        {
                System.out.println("X value =" + x);
                System.out.println("Y value =" + y);
         }
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                int a,b;
                System.out.println("Enter two Integers");
                a=sc.nextInt();
                b=sc.nextInt();
                ParaTest ob=new ParaTest(a,b);
                System.out.println("Enter two doubles");
                x=sc.nextDouble();
                y=sc.nextDouble();
                ParaTest ob=new ParaTest(x,y);
        }
   }
```

Copy Constructor

- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
- There are many ways to copy the values of one object into another in Java. They are:
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

Eg: // Java program to initialize the values from one object to another object.

```
class Student
{        int id;
        String name;
    Student(int i,String n)
    {
        id = i;
        name = n;
   }
  Student(Student s)
  {
         id = s.id;
```

```
            name =s.name;
    }
  void display()
 {       System.out.println(id+" "+name);    }
   public static void main(String args[])
   {     Student s1 = new Student(111,"Karan");
         Student s2 = new Student(s1);
         s1.display();
         s2.display();
         }
     }
```

Output:

111 Karan

111 Karan

15Q) Difference between constructor and method

Ans:

Constructor	Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

16Q) Write about static members?

Ans: The members that are declared as static are called as static members. Both variables and methods of a class can be declared as static. Since these members are associated with the class rather than individual objects, the static variables and methods are referred to as class variables.

1) **Static Variables:**
   The variables that are declared inside a class with the keyword static are called as static variables. These variables are declared with the class scope. Static variables are used when we want a variable that is common to all instances of a class. Java creates only one copy for as static variable for the entire class to which it belongs. Static variables can also be used without creating an object.

   Syntax: static <data type><variable name>;

   Eg:
```
   class staticmembers
   {
           static int count;
           static members()
```

```
            {
                    count++;
             }
    }
```

A static variable can also be access without creating any object by using the class name directly.

2) Static Methods:

The methods that are declared inside a class with the keyword static are called as static methods. A static method can be access only static variables and static methods; it cannot access non-static members of the class. Static methods can also be called even without using the object. They are available to other classes by their class name.

Eg: class sample

```
    {
        public static void m1()
        {
                System.out.println("hello world");
        }
        public static void main(String[] args) throws Exception
        {  m1(); }
    }
```

17Q) Differences between Method overloading and Method Overriding?

Ans:

Method Overloading	Method Overriding
Writing two or more methods with the same name but with different signatures is called method overriding	Writing two or more methods with the same name and same signature is called method overriding
Method overloading is done in the same class	Method overriding is done in super and sub-classes
In method overloading, method return type can be same or different	In method overriding, method return types should also be same
JVM decides which method is called depending on the difference in the method signatures	JVM decides which method is called depending on the data type of the object used to call the method
Method overloading is done when the programmer wants to extend the already available features	Method overriding is done when the programmer wants to provide a different implementations for the same feature
Method overloading is code refinement. Same method is refined to perform a different task.	Method overriding is code replacement. The subclass method overrides the super class method
Method overloading is one of the ways that java implements polymorphism.	Method overriding concept is used inheritance.

18Q) Write about this keyword?

Ans:

Here is given the 6 usage of java this keyword.

- this can be used to refer current class instance variable.

- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

Eg:

```
class Student
{
	int rollno;
	String name;
	float fee;
	Student(int rollno,String name,float fee)
	{
		this.rollno=rollno;
		this.name=name;
		this.fee=fee;
	}
void display()
{
	System.out.println(rollno+" "+name+" "+fee);
	}
}
class TestThis
{
	public static void main(String args[])
	{
		Student s1=new Student(111,"Muthahar",5000f);
		Student s2=new Student(112,"sumit",6000f);
		s1.display();
		s2.display();
	}
}
```

Output:

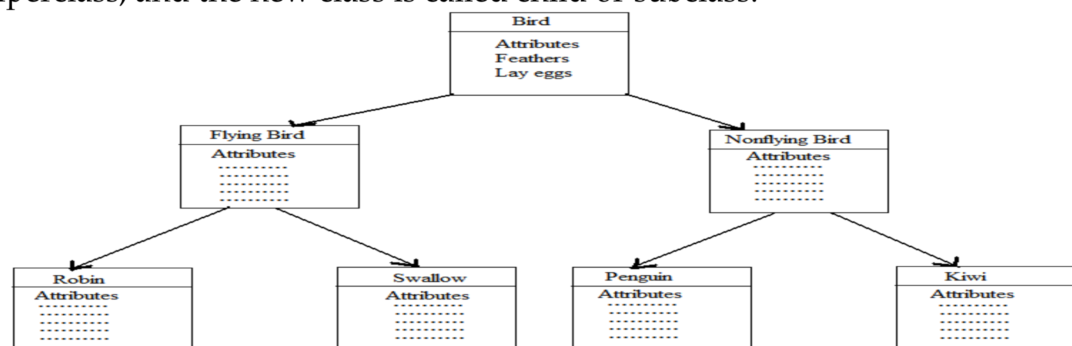
111 Muthahar 5000.0
112 sumit 6000.0

UNIT – III

1Q) Write about Inheritance?

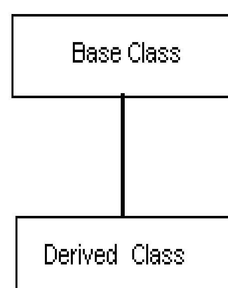
Ans:

- This is the process of providing properties of a parent class to a child class is called as an **inheritance**. A parent class is also called as **base class / super class** and child class is also called as **derived class**.
- In java inheritance is achieved with the help of **"extends"** keyword.
- Java supports different inheritances except multiple inheritances. To perform multiple inheritances in java, JDK provides interface class.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
- In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.



Types of Inheritances:

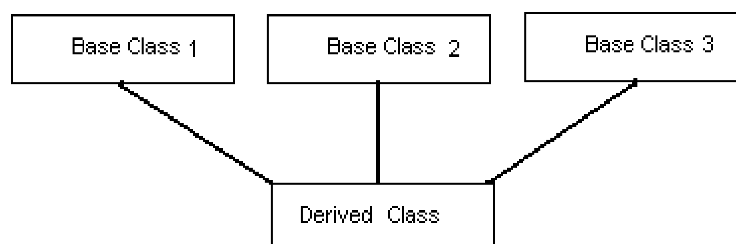
- **Single inheritance**
  - A class which can able to create another class, it is called as single inheritance. It means the features or properties of a base class can be inherited to a child class.
  - **A parent class is also called as a base class** and a **child class is also called as a derived class**.
  - The properties of parent class can be inherited to child class by using the keyword **extends**



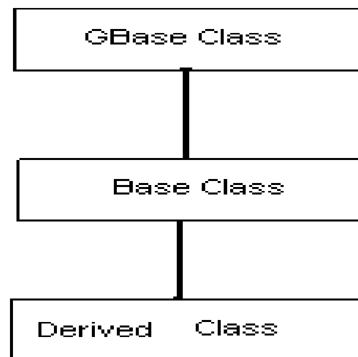
Syn:

```
class superclass-name
{
        field declarations;
        field declarations;
        [specifier][modifier]<returntype> method1(arglist)
        {
                body of method;
        }
        [specifier][modifier]<returntype> method1(arglist)
        {
                body of method;
        }
        .
        .
}
class Subclass-name extends Superclass-name
{
        field declarations;
        field declarations;
        [specifier][modifier]<returntype> method1(arglist)
        {
                body of method;
        }
        [specifier][modifier]<returntype> method1(arglist)
        {
                body of method;
        }
        .
        .
        public static void main(String[] args)
        {
                Code;
        }
}
```

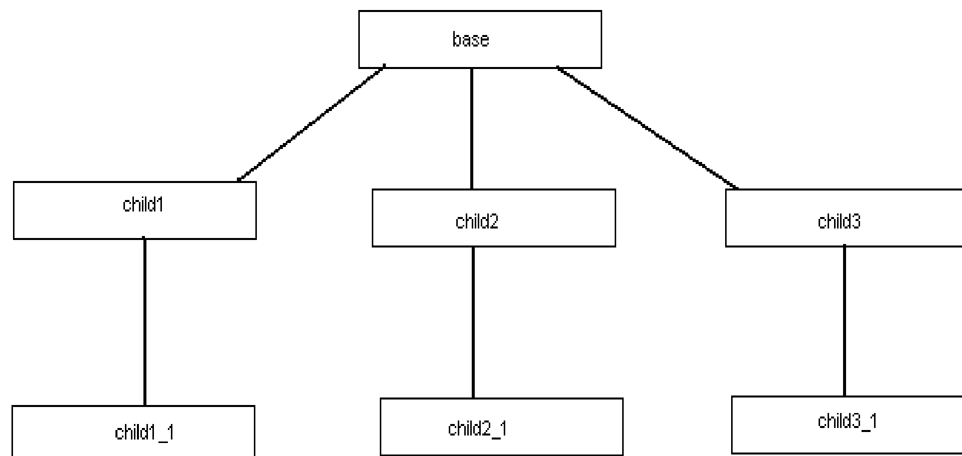
- **Multiple inheritances**
  A class which inherits the properties from different parent classes, it is called as multiple inheritance. It means the properties of different base classes can be inherited to a child class



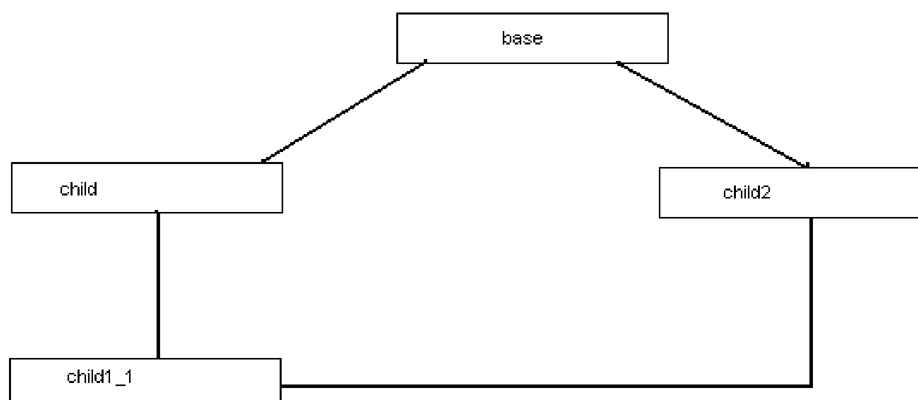
- **Multi-level inheritance**
  A class which can be created in sequence that means a class can contains the properties of grand base and parent classes it is called as multi-level inheritance. A class is derived from another derived class it is called multi-level inheritance.



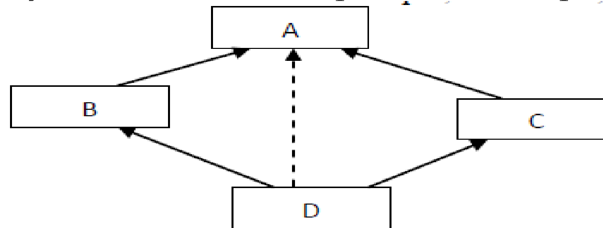
- **Hierarchical inheritance**
  The base class includes all the properties that are common to the subclasses. Subclasses can be constructed by inheriting the properties of the base class. A subclass can serve as a base class for the lower level classes and so on. This process is called hierarchical inheritance.



- **Hybrid Inheritance**
  A class which inherits the properties from the combination of multiple parent classes, it is called as hybrid Inheritance. It means hybrid inheritance is the combination of multi-level and multiple inheritances.



- **Multipath inheritance**
  - Multipath inheritance in C++ is derivation of a class from other derived classes, which are derived from the same base class. In this type of inheritance, there involves other inheritance like multiple, multilevel, hierarchical etc.
  - It is famously known as diamond problem in computer programming.



  - Here, class D is derived from derived classes B & C directly and from class A indirectly. (hierarchical and multiple)
  - Both derived classes inherits the features of base class. Hence when we derive a new class by inheriting features form these two classes derived from the same base class, then same features from the first base is inherited to the finally derived class from two paths. This cause ambiguity in accessing first base class members.

Examples

//Eg-1: Program to demonstrate single inheritance

```
class base
{
        public static void stud_info()
        {       System.out.println("Rno = 1001");
                System.out.println("Name = Sami");
        }
        public static void stud_addr( )
        {       System.out.println("Dno = 18/199")
                System.out.println("Street = G.C Street");
                System.out.println("City = Kadapa");
        }
 }
 public class derived extends first
 {
        public static void stud_co()
        {
                System.out.println("Course = Bsc");
                System.out.println("Fees = 9000");
        }
        public static void main(String[ ] args)
        {       m1();
                m2();
                m3();
        }
  }
```

//Eg -2:

```
class Employee
{   float salary=40000;  }
class Programmer extends Employee
{
        int bonus=10000;
        public static void main(String args[])
        {
                Programmer p=new Programmer();
                System.out.println("Programmer salary is:"+p.salary);
                System.out.println("Bonus of Programmer is:"+p.bonus);
        }
}
```

Output:

Programmer salary is: 40000.0
Bonus of programmer is:10000

2Q) Write about interface with example?

Ans:

- An interface is a collection of **abstract methods** and **constants / final fields.**
- It cannot have any concrete methods in it.
- It tells the classes which implement the interface to do something without how to do it.
- **In java**, interfaces are used to achieve the concept of **multiple inheritance**.
- A java class cannot extend more than one class at a time, but it can implement more than one interface at a time.
- An interface is a collection of method declarations with no data and no bodies. That is, the methods of an interface are always empty that is, they are simply **method signatures**. When a class implements an interface, it must implement all of the methods declared in the interface. In this way, interfaces enforce requirements that an implementing class has methods with certain specified signatures.

Rules:

1. All the methods of an interface are implicitly public and abstract.
2. All the variables of an interface are implicitly public static and final
3. Interface methods should not be static or final.
4. An interface can extend one or more other interfaces.
5. An interface cannot implement another interface or class.

Syn:

```
interface <class_name>
{
        variable declarations;
        method declarations;
}
```

Eg-1:-Program to implement interface class

```
interface geometry
{
        double pi=3.14;
        public void area( );
 }
public class geotest implements geometry
{
        public void area( )
        {
                System.out.println("Area="+geometry.pi*(5*5));
        }
       public static void main(String[] args)
      {
                System.out.println(geometry.pi);
                geotest ob= new geotest();
                Ob.area();
        }
}
```

Extending an interface:

Like classes, interfaces can also be extended. An interface can be sub-interfaced from other interfaces. The interface that extends another interface will inherit all the members of the super interface. The classes that implement the newly developed interface must implement both inherited methods and its own methods of the interface.

Syn:
```
interface <interface_name1>
        {
                variable declarations;
                method declarations;
        }
      interface<interface_name2> extends <interface_name1>
      {
                variable declarations;
                method declarations;
        }
```

// Eg program to demonstrate interface

```
interface base1
{
        public void stud_det();
        public void stud_addr();
}
```

```
interface base2 extends base1
{
        public void stud_co();
}
class child implements base2
{
        public void stud_det()
        {
                S.o.pln("Rno=1001");
                S.o.pln("Name=kiran");
        }
        public void stud_addr()
        {
                S.o.pln("Dno=12/122");
                S.o.pln("Street=knagar");
        }
        public void stud_co()
        {
                S.o.pln("course=bsc");
                S.o.pln("Fees=9000");
        }
        public static void main(String[] args)
        {
                child ob=new child();
                ob.stud_det();
                ob.stud_addr();
                ob.stud_co();
                }
}
```

Output:

Rno = 1001
Name = kiran
Dno = 12/122
Stree = Knagar
Course = bsc
Fees = 9000

Implementing interfaces:

As interface is a collection of abstract methods, we cannot give them a life. To give birth to an interface, it must be implemented by a class. **A class can implement one or more interfaces.** Through classes, we can give birth to the interfaces. The class that implements an interface will acquire all the properties of that interface.

Syn:

```
interface <interface_name1>
{
                Member1;
                Member2;
                public <returntype> method1(arglist);
}
interface <interface_name2>
{
                Members;
                public <returntype> method2(arglist);
}
class <class_name> implements <interface_name1>,<interface_name2>
{
        public void method1(arglist)
        {
                Block of statements;
                }
        public void method2(arglist)
        {
                Block of statements;
                }
  public static void main(String[] args)
  {             Statement1;
                Statement2;
   }
}
```

Eg:

```
interface base1
{
        public void stud_det();
        public void stud_addr();
}
interface base2
{
        public void stud_co();
}
class child implements base1,base2
{
        public void stud_det()
        {        S.o.pln("Rno=1001");
                 S.o.pln("Name=kiran");
        }
        public void stud_addr()
        {       S.o.pln("Dno=12/122");
                S.o.pln("Street=knagar");   }
```

```
        public void stud_co()
        {
                S.o.pln("course=bsc");
                S.o.pln("Fees=9000");
        }
        public  static void main(String[] args)
        {       child ob=new child();
                ob.stud_det();
                ob.stud_addr();
                ob.stud_co();
                }
}
```

3Q) Write about the keyword super?

Ans:

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Eg:

```
class Animal
{
        String color="white";
}
class Dog extends Animal
{
        String color="black";
        void printColor()
        {
                System.out.println(color);//prints color of Dog class
                System.out.println(super.color);//prints color of Animal class
        }
}
class TestSuper1
{
        public static void main(String args[])
        {
                Dog d=new Dog();
                d.printColor();
        }
}
```

4Q) Differences between this and super keywords

Ans:

This	super
this keyword points to a reference of the current class.	super keywords points to a reference of the parent class.
this can be used to access variables and methods of the current class	super can be used to access variables and methods of the parent class from the sub class
this keyword is related to an instance of an object, so it cannot be used inside a static block or static method.	super keyword is related to an instance of an object so it can be used inside a static block or static method.
this keyword is commonly used when an instance variable is shadowed by a parameter of a method	super keyword allows to access to non-private methods and variables of the parent class. but it is not possible to access private members of the parent class inside subclass

Eg:

```
class A
{
        public int x = 0;
        public int y = 0;
}
class B extends A
{
        public int x, y;
        B(int x, int y)
         {
                this.x = x;
                this.y = y;
        }
        void print()
        {
                System.out.println("Base class  : {" + x + ", " + y + "}");
                System.out.println("Super class : {" + super.x + ", " + super.y + "}");
                }
}
class Point
{
        public static void main(String[] args)
        {
                B obj = new B(1, 2);
                obj.print();
        }
}
```

5Q) Write about protected specifier?

Ans:

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- It provides more accessibility than the default modifier.

Eg:

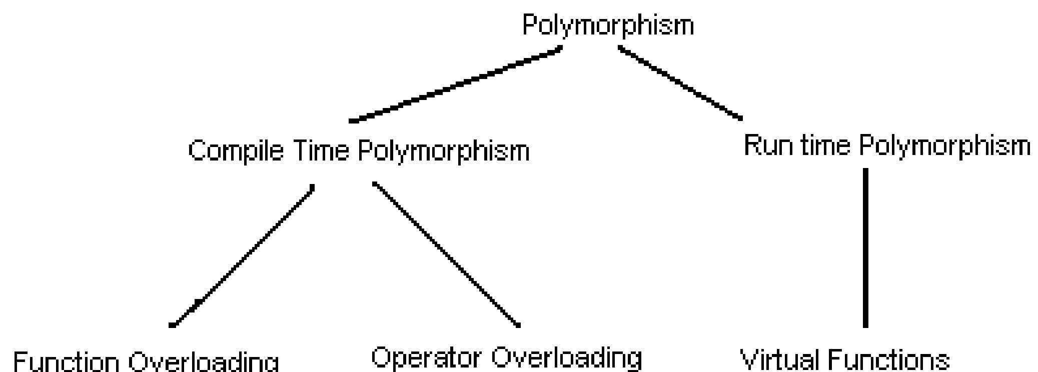
```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
class B extends A{
public static void main(String args[]){
B obj = new B();
obj.msg();
    }
  }
```

6Q) What is Polymorphism? Explain Polymorphism with variables?

Ans:

- Polymorphism is a combination of two Greek words called **poly** and **morphism**.
- **Poly** means **many** and **morphism** means **forms** and thus **polymorphism means many forms**.
- In object oriented programming, polymorphism refers to identically named methods that have different behavior depending on the type of object they refer.
- Polymorphism is the process of defining, a number of objects of different classes into a group and call the methods to carry out the operation of the objects using different function calls.



Polymorphism with variables

- A variable is called polymorphic if it refers to different values under different conditions.
- Object variables represent the behavior of polymorphic variables in Java. It is because object variables of a class can refer to objects of its class as well as objects of its subclasses.

Eg:

```
class ProgrammingLanguage
{
  public void display()
  {
    System.out.println("I am Programming Language.");
  }
}
class Java extends ProgrammingLanguage
{
  public void display()
  {
    System.out.println("I am Object-Oriented Programming Language.");
  }
}
class Main
{
  public static void main(String[] args)
  {
                ProgrammingLanguage pl;
                pl = new ProgrammingLanguage();
                pl.display();

                pl = new Java();
                pl.display();
  }
}
```

7Q) Explain about static polymorphism?

Ans:

Choosing a function in normal way, during compilation time is called as **early binding** or **static binding** or **static linkage.** During compilation time, the compiler determines which function is used based on the parameters passed to the function or the function's return type. The compiler then substitutes the correct function for each invocation. Such compiler based substitutions are called **static linkage.**

With early binding, one can achieve greater efficiency. Function calls are faster in this case because all the information necessary to call the function are hard coded.

This polymorphism is resolved during the compiler time and is achieved through the method overloading.

8Q) Write about Polymorphism with Static Methods

Ans:

- A static method is a method whose single copy in memory is shared by all the objects of the class.
- Static methods belong to the class rather than to the objects. So they are also called class methods.
- When static methods are overloaded or overridden, since they do not depend on the objects, the java compiler need not wait till the objects are created to under which method is called.

Eg:

```
class SimpleCalculator
{
   int add(int a, int b)
   {
       return a+b;
   }
   int  add(int a, int b, int c)
   {
       return a+b+c;
   }
}
public class Demo
{
  public static void main(String args[])
  {
 SimpleCalculator obj = new SimpleCalculator();
      System.out.println(obj.add(25, 25));
      System.out.println(obj.add(25, 25, 30));
  }
}
```

Output of the program

50
80

9Q) Write about Polymorphism with Private Methods

Ans:

Private methods are the methods which are declared by using the access specifier private. This access specifier makes the method not be available outside the class. So other programmers cannot access the private methods. Even private methods are not available in the sub classes. This means there is no possibility to override the private methods of the super class in its sub-classes. Sonly method overloading is possible in case of private methods.

The only way to call the private methods of a class is by calling them within the class. For this purpose, we should create a public method and call the private method from/within it. When the public method is called it calls the private method.

10Q) Write about Polymorphism with Final Methods and final Class?

Ans:

The modifier final is used to finalize classes, methods and variables. Finalizing thing effectively freezes the implementation or value of that thing. More specifically, here is how final works with classes, variables and methods respectively:

- When the modifier final is applied to a class, it means that the class cannot be inherited
- When final is applied to a variable, it means that the variable is constant.
- When final is applied to a method in a class, it means that the method cannot be overridden in the sub-classes.

1) Finalizing classes

- In order to finalize a class, the modifier final is added to the class definition. Typically, it is added after protection modifiers such as private or public.
- A class is declared final for the following reasons –
  1. To prevent inheritance
  2. For better efficiency. Final classes allow programmers to rely on instances of only that class and optimize those instances

Syn:
```
public final class class_name
{
        Member declarations;
        <returntype> method1(arglist)
        {
                Block of statements;
        }
}
```

2) Finalizing Variables

- The value of a finalized variable cannot be changed so it is called as a constant.
- Local variables cannot be declared as static
- To declare constant in java, final variables with initial values are used. This declaration can be done in a program as follows.

Syn:
```
public final class class_name
{
        public static final float pi=3.14;
        public static final String nm=”Muthahar”;
}
```

3) Finalizing Methods:

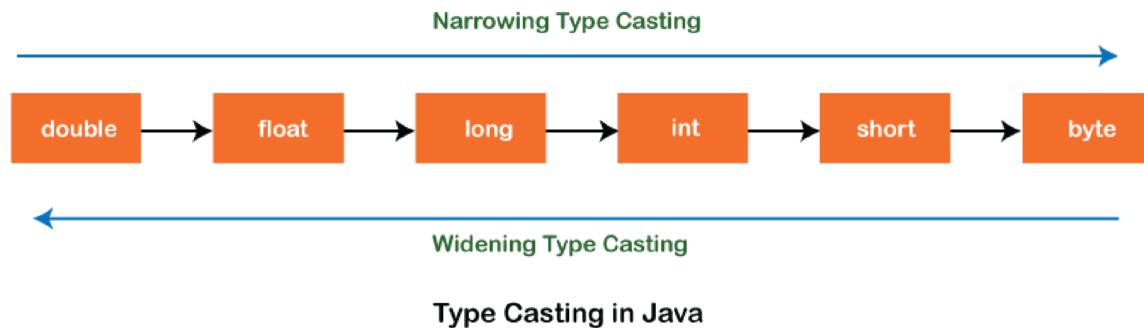
- Methods that cannot be overridden are known as finalized methods.
- The implementations of final methods cannot be redefined in sub-classes.

Syn:
```
public class <class_name>
{
        public final <return_type> method1()
        {
                Block of statements
                }
}
```

11Q) What is type casting? Explain about casting primitive data types

Ans:

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.



Type Casting in Java

Types of Type Casting

There are two types of type casting:

- Widening Type Casting
- Narrowing Type Casting

Widening Type Casting

- Converting a lower data type into a higher one is called **widening** type casting.
- It is also known as **implicit conversion** or **casting down**.
- It is done automatically.
- It is safe because there is no chance to lose data.
- It takes place when:
  - Both data types must be compatible with each other.
  - The target type must be larger than the source type.

Narrowing Type Casting

- Converting a higher data type into a lower one is called **narrowing** type casting.
- It is also known as **explicit conversion** or **casting up**.
- It is done manually by the programmer.
- If we do not perform casting then the compiler reports a compile-time error.

12Q) Write about Casting Referenced DataTypes?

Ans:

A class is a referenced data type. Converting a class type into another class type is also possible through casting. But the classes should have some relationship between them by the way of inheritance.

1. Widening in Referenced Data types

- Objects of a class can be cast into objects of another class if both the classes are related to each other through the property of inheritance, i.e., one class is the parent class, and the other class is the child class.
- This type of casting superclass object (parent class) will hold the sub-class object's properties.

Eg:

class one

```
{
    void show1()
    {
    System.out.println("Super class Method");
    }
}
class Two extends one
{
    void show2()
    {
            System.out.println("Sub class Method");
    }
}
class Cast
{
    public static void main(String[] args)
    {
            one ob;
            ob=(one) new Two(); //ob is referring to sub class object
            ob.show1();
    }
}
```

2. **Narrowing Type casting with objects (downcasting)**
   - Similar to widening typecasting, the object of one class can be narrowed cast into the object of another class when two classes hold the relationship of parent class and child class through inheritance. The class that inherits the properties of another class is called a child class or sub-class, while the inherited class is called a Parent class or superclass.
   - But unlike widening typecasting, the programmer needs to use a cast operator to perform narrowcast explicitly. If we do not perform narrowcasting, the java compiler will throw a “compile-time error”.

   Eg:

```
class one
{
        void show1()
        {       System.out.println("Super class Method");       }
}
class Two extends one
{
        void show2()
        {
                System.out.println("Sub class Method");
        }
}
class Cast
{
```

```
        public static void main(String[] args)
        {
                Two ob;
                ob=(Two) new one(); //ob is referring to super class object
                ob.show1();
        }
}
```

13Q) Write about Object Class?

Ans:

- The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.
- The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Methods of Object class

Method	Description
getClass	This method gives an object that contains the name of a class to which an object belongs
hashCode()	It returns the hashcode number for this object
equals()	This method compares the references of two objects and if they are equal, it returns true otherwise false. The way it compares the objects is dependent on the objects
clone()	It creates and returns the exact copy of this object
toString()	It returns the string representation of this object
notify()	It wakes up single thread, waiting on this object's monitor
notifyAll()	It wakes up all the threads, waiting on this object's monitor
wait()	It causes the current thread to wait for the specified milliseconds until another thread notifies
finalize()	This method is called by the garbage collector when an object is removed from memory

14Q) Write about JAR Files?

Ans:

- A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.
- In simple words, a JAR file is a file that contains a compressed version of .class files, audio files, image files, or directories. We can imagine a .jar file as a zipped file(.zip) that is created by using WinZip software. Even, WinZip software can be used to extract the contents of a .jar. So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.

1. Create a JAR file

- In order to create a .jar file, we can use jar cf command in the following ways as discussed below:
  Syn: jar cf jarfilename inputfiles
  Here, cf represents to create the file. For example , assuming our package pack is available in C:\directory , to convert it into a jar file into the pack.jar , we can give the command as:
  C:\> jar cf pack.jar pack
  Now, pack.jar file is created.

2. **View a JAR file**
   - In order to view the contents of .jar, we can use the command as:
     Syn: jar tf jarfilename
     Here, tf represents the table view of file contents. For example, to view the contents of our pack.jar file, we can give the command:
     C:/> jar tf pack.jar
3. **Extracting a JAR file**
   - In order to extract the files from a .jar file, we can use the commands below listed:
     jar xf jarfilename
     Here, xf represents extract files from the jar files. For example, to extract the contents of our pack.jar file, we can write:
     C:\> jar xf pack.jar
4. **Updating a JAR File**
   The Jar tool provides a 'u' option that you can use to update the contents of an existing JAR file by modifying its manifest or by adding files. The basic command for adding files has this format as shown below:
   Syn: jar uf jar-file input-file(s)
   Here 'uf' represents the updated jar file. For example, to update the contents of our pack.jar file, we can write:
   C:\>jar uf pack.jar
5. **Running a JAR file**
   In order to run an application packaged as a JAR file, the following command can be used as listed:
   Syn: C:\>java -jar pack.jar

15Q) Write about abstract class?

Ans:

- A class that is declared as abstract is called Abstract class.
- Abstract class may have **abstract methods** or **simple / concrete methods**.
- The abstract class cannot be instantiated; that is we cannot create any objects based on an abstract class. But we can create references to an abstract class.
- Abstract classes must be extended by other concrete classes to give them life to use. We go for abstract classes when we know the implementation of some methods but do not know the implementation of some methods.
- Any class that extends an abstract class must implement all the abstract methods of the abstract super class.

- Abstract is a keyword which is also used to specify a method without body. Whenever a specification of method is known but not its implementation, we have to define a method as abstract.
- Abstract stands for “not qualified”.
- Abstract methods can reside only in abstract classes.

Syn:

```
abstract class classname
{
        public abstract void method1( );
        public abstract void method2( );
}
```

// Example to demonstrate abstract class with abstract methods.

```
abstract class arithmetic
{
        public abstract void add( );
        public abstract void diff( );
}
public class arithtest extends arithmetic
{
        public void add( )
        {       System.out.println("Add method");       }
        public void diff( )
        {       System.out.println("diff method");      }
        public static void main(String[ ] args)
        {
        arithtest f=new arithtest( );
                f.add( );
                f.diff( );
        }
}
```

//Example to demonstrate Abstract class without any abstract methods

```
abstract class test
{
        public static void m1( )
        {       System.out.println("method1");  }
        public static void m2( )
        {       System.out.println("method2");  }
}
public class temp extends test
{
        public static void main(String[ ] args)
        {       test.m1( );
                temp t1=new temp( );
                t1.m1( );
        }
}
```

Note

- It is illegal to use static and abstract modifiers together because static say allocate memory implicitly whereas abstract prevents allocation.
- Final and abstract modifiers cannot be used together because final says prevent overriding whereas abstract needs overriding.
- An abstract class can extend another abstract class.

Eg:-
```
public static abstract void m1( );  //error
public static abstract void m2( ); //error
```

Eg – for abstract class inheritance

```
abstract class first
{
        public abstract void m1( );
}
abstract class second extends first
{
        public void m2( )
        {
                System.out.println("in method2");
        }
}
public class temp extends second
{
        public void m1( )
        {
                System.out.println("in method1");
        }
public static void main(String[ ] args)
{       temp t=new temp( );
                t.m1( );
                t.m2( );
        } }
```

16Q) Write about Abstract Methods?

Ans:

- A method declared using the **abstract** keyword within an abstract class and does not have a definition (implementation) is called an abstract method.
- When we need just the method declaration in a super class, it can be achieved by declaring the methods as abstracts.
- Abstract method is also called subclass responsibility as it doesn't have the implementation in the super class. Therefore a subclass must override it to provide the method definition.

  Syn: abstract return_type method_name([argument-list]);

// Example to demonstrate abstract class with abstract methods.

```
abstract class arithmetic
{
        public abstract void add( );
        public abstract void diff( );
}
public class arithtest extends arithmetic
{
        public void add( )
        {
        System.out.println("Add method");
        }
        public void diff( )
        {
        System.out.println("diff method");
         }
        public static void main(String[ ] args)
        {
        arithtest f=new arithtest( );
                f.add( );
                f.diff( );
        }
}
```

17Q) What is package? Write the benefits of package?

Ans:

Packages are java's way of grouping a variety of classes or interfaces together. The grouping is usually done according to functionality. In fact, packages act as containers for classes. By organizing our classes into packages we achieve the following **benefits**.

1. The classes contained in the packages of other programs can be easily reused.
2. In packages, classes can be unique compared with classes in other packages, i.e., two classes in two different packages can have the same name. They may be referred by their fully qualified name, comprising the package name and the classname.
3. Packages provide a way to "hide" classes thus preventing other programs or packages from accessing classes that are meant for internal use only.
4. Packages also provide a way for separating "design" from "coding". First we can design classes and decide their relationships, and then we can implement the java code needed for the methods.

Types of packages:

1) **Built in Packages**
2) **User-defined Packages**

Built in Packages

These packages consist of a large number of classes which are a part of Java **API**. Some of the commonly used built-in packages are:

1) **java.lang:**
   It Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2) **java.io:**
   It contains classed for supporting input / output operations.
3) **java.util:**
   It contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4) **java.applet: It** Contains classes for creating Applets.
5) **java.awt:**
   It contain classes for implementing the components for graphical user interfaces (like button, ;menus, etc).
6) **java.net: It c**ontain classes for supporting networking operations.

User-defined Packages

These are the packages that are defined by the user. The package can be created by using the package statement.

Steps to create package

1. Declare the package at the beginning of a file.
   package pack_name;
2. Define the class that is to be put in the package and declare it public.
3. Create a sub-directory under the directory where the main source files are stored.
4. Store the listing as the classname.java in the sub-directory

Eg: package package1;

```
        public class classA
        {
                public void displayA( )
                {       System.out.println("Class A");          }
                public void displayB( )
                {       System.out.println("Class B");          }
        }
```

c:\>javac –d . classA.java

```
import package1.classA;
class test
{
        public static void main(String[ ] args)
        {       classA ob=new classA( );
                ob.displayA( );
                ob.displayB( );
        }
}
```

18Q) What about accessing of package?

Ans:
A java system package can be accessed either using a fully qualified class name or using import statement. We generally use import statement when the package name is too long or when there are several references to a particular package. We can use the same approach for accessing user defined packages also. The below is the general form of import statement for searching a class.

```
import pack1[.pack2[.pack3].classname;
```

Here, pack1 is the name of the top level package; pack2 is the name of the package which is inside pack1 and so on. In this way we can have several packages in a package hierarchy. We should specify explicit class name finally. The statement should end with a semicolon. Multiple import statements are valid.

```
import firstpack.secondpack.classname;
```

After defining this statement, all the members of the class one can be accessed directly using the classname or its objects directly without using the name of package. There is also one more way as shown below.

```
import nameofpackage.*;
```

Here, **'nameofpackage'** may represent a single package or hierarchy of packages. The '*' represent that the compiler should search this entire package hierarchy when it encounters a class name. The big advantage of this is we need not use long package names in the program repeatedly and on the other hand if we follow this approach it will be very difficult to find from which package a particular member came.

Java provides many levels of protection to the variables and method to be visible within classes, subclasses and packages. Classes and packages both encapsulate the name space and scope of variables and methods. Package acts as a container for the related classes and sub packages. A class acts as a container for data and code. Because of interaction between classes and packages Java provides following four categories of visibilities for classes, members,

1. Subclasses in the same package
2. Non-subclasses in the same package
3. Subclasses in different packages
4. Classes that are neither in the same package nor a subclass.

19Q) Explain about interfaces in package

Ans:
It is also possible to write interfaces in a package. But whenever, we create an interface the implementation of classes should also be created. We cannot create an object to the interface but we can create objects for implementation classes and use them. We write an interface to display system date and time in the package

Eg:
```
package mypack;
```

```
public interface MyDate
{
        void showDate();
}
```

Compile the preceding code and observe that the java compiler creates a sub directory with the name mypack and store MyDate.lass file there.

```
package mypack;
import mypack.MyDate;
import java.util.*;
public class DateImpl implements MyDate
{
        public void showDate()
        {
            Date d=new Date();
            System.out.println(d);
        }
}
```

When the preceding code is completed, DateImpl.class is created in the same package mypack. DateImpl class contained showDate() method which can be called and used in any other program.

```
import mypack.DateImpl;
class DateDisplay
{
  public static void main(String[] args)
 {
        DateImpl obj=new DateImpl();
        obj.showDate();
  }
}
```

20Q) Explain about access specifiers or visibility access in java?

Ans:

There are two types of modifiers in Java:

1) access modifiers and
2) non-access modifiers.

Access modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private:**
   The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

2. **Default:**
   The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:**
   The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:**
   The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

21Q) CREATING API DOCUMENT

Ans: We can create document api in java by the help of javadoc tool. In the java file, we must use the documentation comment /**... */ to post information for the class, method, constructor, fields etc.

```
package com.abc;
public class M
{
        public static void  cube(int n)
        {
            System.out.println(n*n*n);}
            }
```

To create the document API, you need to use the javadoc tool followed by java file name. There is no need to compile the javafile.

On the command prompt, you need to write:

```
javadoc M.java
```

to generate the document api. Now, there will be created a lot of html files. Open the index.html file to get the information about the classes.

Unit – IV

1Q) What is Exception? Write about types of Exceptions?

Ans:

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

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

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner. Based on these, we have **three categories of Exceptions.**

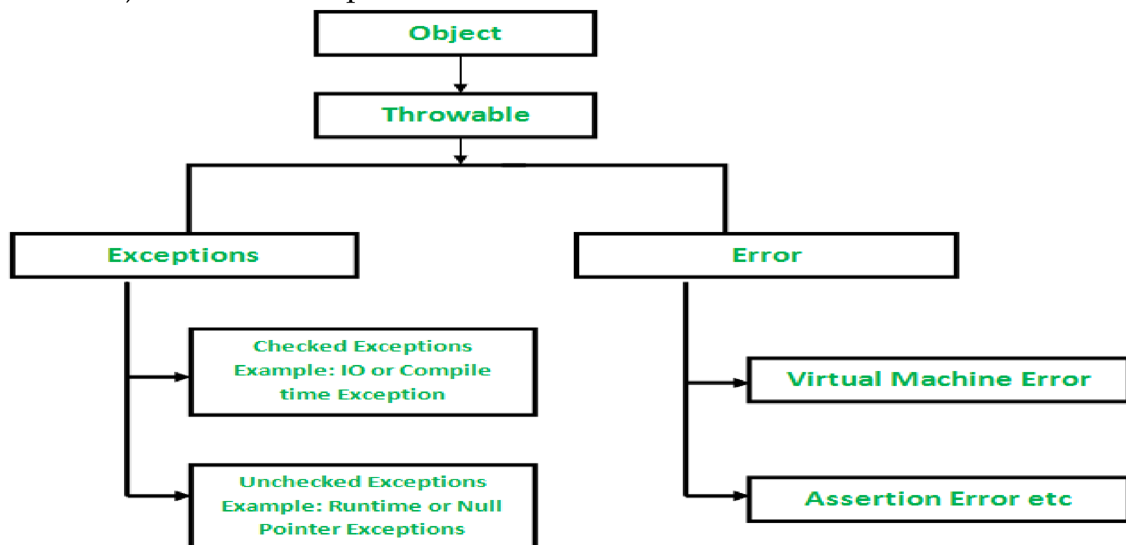
1. **Checked exceptions**
   A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.
1. **Unchecked exceptions**
   An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
2. **Errors**
   These are not exceptions at all, but problems that arise beyond the control of 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.

2Q) Write about Exception Hierarchy?

Ans:

- 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 abnormal conditions that happen in case of system failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment.
  Eg: JVM is out of memory. Normally, programs cannot recover from errors.
- **The Exception class has two main subclasses:**
  1) IOException class

2) RuntimeException Class.



3Q) Write the Built-in Exceptions in Java

Ans: Java defines several exception classes inside the standard package **java.lang**. The most general of these exceptions are subclasses of the standard type RuntimeException. Since java.lang is implicitly imported into all Java programs, most exceptions derived from RuntimeException are automatically available.

Java defines several other types of exceptions that relate to its various class libraries. Following is the list of Java Unchecked RuntimeException.

S.No.	Exception	Description
1	**ArithmeticException**	Arithmetic error, such as divide-by-zero.
2	**ArrayIndexOutOfBoundsException**	Array index is out-of-bounds.
3	**ArrayStoreException**	Assignment to an array element of an incompatible type.
4	**ClassCastException**	Invalid cast.
5	**IllegalArgumentException**	Illegal argument used to invoke a method.
6	**IllegalStateException**	Environment or application is in incorrect state.
7	**IllegalThreadStateException**	Requested operation not compatible with the current thread state.
8	**IndexOutOfBoundsException**	Some type of index is out-of-bounds.
9	**NegativeArraySizeException**	Array created with a negative size.
10	**NullPointerException**	Invalid use of a null reference.

11	**NumberFormatException**	Invalid conversion of a string to a numeric format.
12	**StringIndexOutOfBounds**	Attempt to index outside the bounds of a string.

Following is the list of Java **Checked Exceptions** Defined in java.lang.

S.No.	Exception	Description
1	**ClassNotFoundException**	Class not found.
2	**CloneNotSupportedException**	Attempt to clone an object that does not implement the Cloneable interface.
3	**IllegalAccessException**	Access to a class is denied.
4	**InstantiationException**	Attempt to create an object of an abstract class or interface.
5	**InterruptedException**	One thread has been interrupted by another thread.
6	**NoSuchFieldException**	A requested field does not exist.
7	**NoSuchMethodException**	A requested method does not exist.

4Q) Write about Exception Handlers

Ans: There are 5 exception handlers in which try, catch and finally are blocks and throw and throws are the keywords.

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
Catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
Finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
Throw	The "throw" keyword is used to throw an exception.
Throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

1) **Try and catch blocks:**
Whenever there is a possibility of an exception being generated in a program, it is better to handle it explicitly. **To do this, the try and catch blocks are used**. The advantages of using the try and catch are that it fixes the error and presents the program from terminating abruptly. The statements of a program should be included within a try block which we may think that it may generate/cause an error. During run time of a program if error occurs within the try block then it holds that error and handed over to catch block.

The catch block should immediately follow the try block. The catch block can have statements explaining the cause of the exception generated. It is mandatory to specify both try and catch blocks. A try block can have any number of catch blocks.

Syntax – 1

```
try
{
            statement1
            statement2;
            statement3;
            statement4;
            statement5;
    }
    catch(Exception e)
    {
            statement;
            }
```

Syntax –2

```
try
{
            statement1
            statement2;
            statement3;
            statement4;
}
catch(Exception e1)
{
    statement;
 }
catch(Exception e2)
{
   statement;
 }
catch(Exception e3)
{
    statement;
}
```

Finally

- Java supports finally block that can be used to handle an exception that is not caught by any of the catch statements.
- Finally block can be used to handle any exception generated within a try block. It may be added 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 an exception is thrown. As a result, we can use it to perform certain house-keeping operations such as **closing files and releasing system resources.**

Syn-1:

```
try
{
        statement1
        statement2;
        statement3;
}
catch(Exception e)
{
        Statement;
}
finally
{
        statement1;
        statement2;
}
```

Syn-2:

```
try
{
        Statement1;
        Statement2;
        Statement3;
}
finally
{
        Statement1;
        Statement2;
        Statement3;
}
```

throw:

This keyword is used to raise an exception explicitly by the user, followed to throw keyword only exception object is valid. Throw keyword is used to raise built-in or user-defined exceptions explicitly by the users. Generally exceptions are raised by JVM.

Syntax – throw new throwable_subclass;

eg:- throw new ArithmeticException( );

throws:

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing checkup before the code being used.

User-defined Exceptions

- If you are creating your own Exceptions that is known as custom exception or user-defined exception
- Java custom exceptions are used to customize the exception according to user need. By the help of custom exception, you can have your own exception and message.

Eg:

```
class InvalidAgeException extends Exception
{
InvalidAgeException()
{
        System.out.println("Age is not Valid");
 }
 InvalidAgeException(String s)
 {
    super(s);
        }
}
```

Eg:

```
class TestCustomException1
{
  static void validate(int age)throws InvalidAgeException
  {
    if(age<18)
     throw new InvalidAgeException("not valid");
    else
     System.out.println("welcome to vote");
    }
  public static void main(String args[])
  {
     try
    {
            validate(13);
     }
    catch(Exception m)
    {
        System.out.println("Exception occured: "+m);
        }
      System.out.println("rest of the code...");
          }
}
```

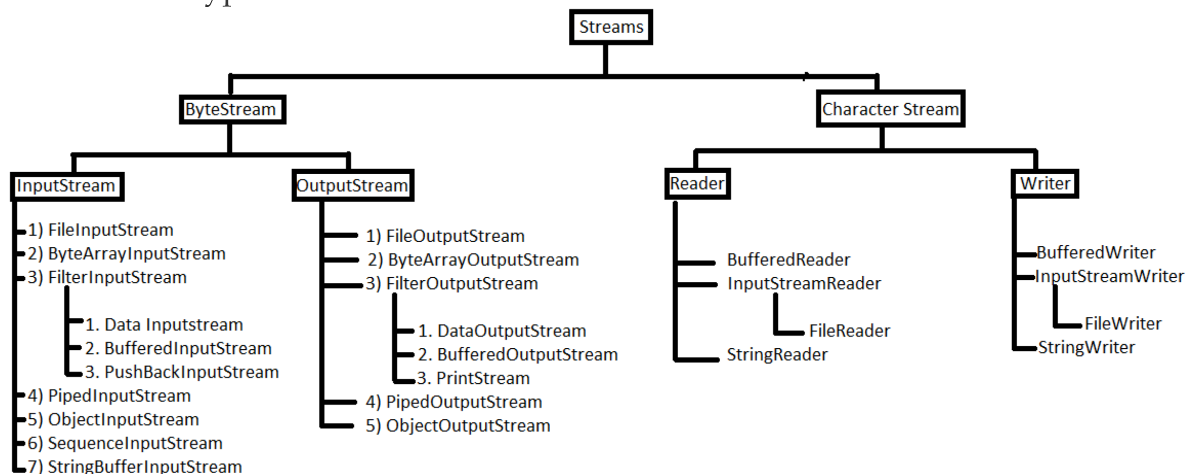
5Q) Write about Streams

Ans:

- A Stream is a path of communication between the source of information and the destination.
- Streams can be dealt under three headings – **Input Streams**, **Output streams** and **Readers / Writers**. All of them are abstract classes. All methods of these classes throw an IOException on error conditions.
- Streams are the sequence of bits (data). There are two types of streams:
  1. **Input Streams**
     Input streams are used to read the data from various input devices like keyboard, file, network, etc.
  2. **Output Streams**
     Output streams are used to write the data to various output devices like monitor, file, network, etc.

Types of Streams:

There are two types of streams based on data:



1. Byte Stream:

- Java byte streams are used to perform input and output of 8-bit bytes.
- Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.
- The ByteStream is classified into two types. They are
  1. Input Stream
  2. Output Stream

Input Stream:

- These are used to read byte data from various input devices.
- InputStream is an **abstract class** and it is the super class of all the input byte streams.
- List of Byte Input Streams are
  1. FileInputStream
  2. ObjectInputStream
  3. ByteArrayInputStream
  4. BufferedInputStream
  5. DataInputStream

1) FileInputStream

- Java FileInputStream class is a part of java.io package.
- FileInputStream obtains input bytes from a file in a file system.
- FileInputStream is used for reading streams of raw bytes such as image data.
- FileInputStream is a subclass of InputStream class.
- The FileInputStream class provides the connection to a disk file.

Constructors

Constructors	Description
FileInputStream(File file)	It creates a FileInputStream by opening a connection to a file.
FileInputStream(FileDescriptor fdobj)	Creates a FileInputStream by using the FileDescriptor object fdobj.
FileInputStream(String name)	Creates a FileInputStream by opening a connection to a file.

Methods

Method	Description
available()	Returns an approximation of the number of bytes that can be read from this file input stream.
close()	Close the input stream file and releases the system resources.
finalize ()	Make sure that close () method of this input stream is called properly.
read(byte[] b)	Read the data from the given input stream in the form of an array of bytes.
skip(long n)	Skip or discard the n bytes of data from the input stream.

Eg: Program to read the data from a file.

```
import java.io.*;
public class FisDemo
{
public static void main(String args[])throws FileNotFoundException, IOException
{
    FileInputStream fis = new FileInputStream("abc.txt");
     int i;
     System.out.println("ASCII value of the character:");
     while((i=fis.read()) != -1)
     {
         System.out.print(i+" : ");
       System.out.println((char)i);
       }
   }
}
```

Eg-2:

```
import java.io.*;
class FiTest
{
	public static void main(String[] args) throws Exception
	{	int v;
	FileInputStream fis=new FileInputStream("c:/AdJava/DisTest.java");
		while((v=fis.read())!=-1)
		{
			System.out.print((char)v);
		}
	}
}
```

2) DataInputStream

- The Java DataInputStream class enables you to read Java primitives (int, float, long etc.) from an InputStream instead of only raw bytes.
- You wrap an InputStream in a DataInputStream and then you can read Java primitives from the DataInputStream. That is why it is called *DataInputStream* - because it reads data (numbers) instead of just bytes.
- The DataInputStream is handy if the data you need to read consists of Java primitives larger than one byte each, like int, long, float, double etc. The DataInputStream expects the multi byte primitives to be written in network byte order

Syn -1: DataInputStream dis=new DataInputStream(System.in);

Syn -2: FileInputStream fis=new FileInputStream("Filename");
DataInputStream dis=new DataInputStream(fis);

or

DataInputStream dis=new DataInputStream(new FileInputStream("Filename"));

Methods:

Methods.	Description
readBoolean()	This method reads one input byte and returns true if that byte is nonzero, false if that byte is zero.
readByte()	This method reads and returns one input byte.
readChar()	This method reads two input bytes and returns a char value.
readDouble()	This method reads eight input bytes and returns a double value.
readFloat()	This method reads four input bytes and returns a float value.
readInt()	This method reads four input bytes and returns an int value.
readLong()	This method reads eight input bytes and returns a long value.
readShort()	This method reads two input bytes and returns a short value.

Eg:

```
import java.io.*;
class  DisTest
{
        public static void main(String[] args) throws IOException
        {       int rno,fees;
                String nm,co;
                DataInputStream dis=new DataInputStream(System.in);
                System.out.println("Enter Rno,Name,Course and Fees");
                rno=Integer.parseInt(dis.readLine());
                nm=dis.readLine();
                co=dis.readLine();
                fees=Integer.parseInt(dis.readLine());
                System.out.println("Rno =" + rno);
                System.out.println("Name =" + nm);
                System.out.println("Course="+ co);
                System.out.println("Fees ="+ fees);
        }
}
```

Eg:-2

```
import java.io.*;
class readStud
{
  public static void main(String[] args) throws Exception
  {
    DataInputStream dis=new DataInputStream(new FileInputStream("student.dat"));
    int rno,fees;
    String nm,co;
    do
    {
        try
        {
                rno=dis.readInt();
                nm=dis.readLine();
                co=dis.readLine();
                fees=dis.readInt();
                System.out.println(rno +"   " +nm +"   "+ co +"   "+ fees);
        }
        catch(Exception e)
        {
                break;
        }
        }while(true);
    }
}
```

OutputStream

- These are used to write byte data to various output devices.
- Output Stream is an abstract class and it is the superclass for all the output byte streams.
- List of Byte Output Streams:
    1. FileOutputStream
    2. ObjectOutputStream
    3. ByteArrayOutputStream
    4. BufferedOutputStream
    5. DataOutputStream

1) FileOutputStream

- The *Java FileOutputStream* class, java.io.FileOutputStream, makes it possible to write a file as a stream of bytes.
- The Java FileOutputStream class is a subclass of Java OutputStream meaning you can use a FileOutputStream as an OutputStream.

Constructors

Constructor	Description
FileOutputStream(Filename)	This creates a file output stream to write to the file represented by the specified *File* object.
FileOutputStream(filename, boolean)	This creates a file output stream to write to the file represented by the specified File object.

Methods:

Methods	Description
Close()	This method closes this file output stream and releases any system resources associated with this stream.
Finalize()	This method cleans up the connection to the file, and ensures that the close method of this file output stream is called when there are no more references to this stream.
write(byte[] b)	This method writes *b.length* bytes from the specified byte array to this file output stream.
write(int b)	This method writes the specified byte to this file output stream.

Eg:

```
import java.io.*;
class FileCopy
{
	public static void main(String[] args) throws Exception
	{
		int v;
		FileInputStream fis=new FileInputStream("c:/AdJava/FiTest.java");
```

```
        FileOutputStream fos=new FileOutputStream("c:/AdJava/shdc.java");
        while((v=fis.read())!=-1)
        {       fos.write(v);           }
        System.out.println("File Copied...");
    }}
```

DataOutputStream

- The Java DataOutputStream class enables you to write Java primitives to OutputStream's instead of only bytes.
- You wrap an OutputStream in a DataOutputStream and then you can write primitives to it. That is why it is called a *DataOutputStream* - because you can write int, long, float and double values to the OutputStream, and not just raw bytes.
- Often you will use the Java DataOutputStream together with a **Java DataInputStream**. You use the DataOutputStream to write the data to e.g. a file, and then use the DataInputStream to read the data again.
- Using the DataOutputStream and DataInputStream together is a handy way to be able to write larger primitives than bytes to an OutputStream and be able to read them in again, ensuring the right byte order is used etc.

Syn: FileOutputStream fos=new FileOutputStream("Filename");
DataOutputStream ds=new DataOutputStream(fos);

or

DataOutputStream ds=new DataOutputStream
(new FileOutputStream("filename"));

Methods

Metods	Description
size()	This method returns the current value of the counter written, the number of bytes written to this data output stream so far.
write(int b)	This method writes the specified byte (the low eight bits of the argument b) to the underlying output stream.
writeBoolean(boolean v)	This method writes a boolean to the underlying output stream as a 1-byte value.
writeByte(int v)	This method writes out a byte to the underlying output stream as a 1-byte value.
writeBytes(String s)	This method writes out the string to the underlying output stream as a sequence of bytes.
writeChar(int v)	This method writes a char to the underlying output stream as a 2-byte value, high byte first.
writeDouble(double v)	This method converts the double argument to a long using the double To Long Bits method in class Double, and then writes that long value to the underlying output stream as an 8-byte quantity, high byte first.

writeFloat(float v)	This method converts the float argument to an int using the float To Int Bits method in class Float, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.
writeInt(int v)	This method writes an int to the underlying output stream as four bytes, high byte first.
writeLong(long v)	This method writes a long to the underlying output stream as eight bytes, high byte first.
writeShort(int v)	This method writes a short to the underlying output stream as two bytes, high byte first.

Eg:

```
import java.io.*;
class StudTest
{
	public static void main(String[] args) throws Exception
	{
		FileOutputStream fos=new FileOutputStream("student.dat");
		DataOutputStream ds=new DataOutputStream(fos);
		DataInputStream dis=new DataInputStream(System.in);
		int rno,fees;
		String nm,co;
		char ch='y';
		do
		{
			System.out.println("Enter Rno,Name,Course and Fees");
			rno=Integer.parseInt(dis.readLine());
			nm=dis.readLine();
			co=dis.readLine();
			fees=Integer.parseInt(dis.readLine());
			ds.writeInt(rno);
			ds.writeBytes(nm+"\n");
			ds.writeBytes(co+"\n");
			ds.writeInt(fees);
		System.out.println("Do you want to add another record [y/n]:");
		ch=(char)System.in.read();
		System.in.skip(2);
		}while (ch=='y');
		dis.close();
		ds.close();
		fos.close();
		}
	}
```

2. Character Stream:

- Character Stream Classes are used to read characters from the source and write **characters** to destination.
- Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.
- There are two kinds of Character Stream classes –

1. Reader classes

- Reader class is a base class of all the classes that are used to read characters from a file, memory or console.
- Reader is an abstract class and hence we can't instantiate it but we can use its subclasses for reading characters from the input stream.
- The Reader class contains number of sub-classes. They are
  1. BufferedReader
  2. StringReader
  3. PipedReader
  4. CharArrayReader
  5. FileReader

Methods:

Methods	Description
read()	This method reads a characters from the input stream.
close()	This method closes this output stream and also frees any system resources connected with it.

1) FileReader

- The *Java FileReader* class, java.io.FileReader makes it possible to read the contents of a file as a stream of characters.
- It works much like the **FileInputStream** except the FileInputStream reads bytes, whereas the FileReader reads characters.
- The FileReader is intended to read text, in other words. One character may correspond to one or more bytes depending on the character encoding scheme.
- The Java FileReader is a subclass of the Java Reader class,

Constructor

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Methods

Methods	**Description**
read()	Reads a single character. Returns an int, which represents the character read.
Close()	It is used to close the FileReader class

Eg:

```
import java.io.FileReader;
public class FileReaderExample
{
        public static void main(String args[])throws Exception
        {
                FileReader fr=new FileReader("D:\\testout.txt");
                int i;
                while((i=fr.read())!=-1)
                System.out.print((char)i);
                fr.close();
        }
}
```

2. Writer Classes

- Writer class and its subclasses are used to write characters to a file, memory or console.
- Writer is an abstract class and hence we can't create its object but we can use its subclasses for writing characters to the output stream.
- The Writer class contains number of sub-classes. They are
    1. StringWriter
    2. BufferedWriter
    3. FileWriter
    4. PipedWriter
    5. CharArrayWriter

Methods:

Methods	Description
flush()	This method flushes the output steam by forcing out buffered bytes to be written out.
write(int c)	This method writes a character to the output stream.
close()	This method closes this output stream and also frees any resources connected with this output stream.

1) FileWriter

- The *Java FileWriter* class, makes it possible to write characters to a file. In that respect the Java FileWriter works much like the **FileOutputStream** except that a FileOutputStream is byte based, whereas a FileWriter is character based.

- The FileWriter is intended to write text, in other words. One character may correspond to one or more bytes, depending on the character encoding scheme in use.
- The Java FileWriter class is a subclass of the Java Writer class.
  Syn: FileWriter fW = new FileWriter("Filename", boolean);

Constructor:

Constructor	Description
FilterWriter(Writer out)	It creates InputStream class Object

Methods

Method	Description
close()	It closes the stream, flushing it first.
flush()	It flushes the stream.
write(int c)	It writes a single character.
write(String str, int off, int len)	It writes a portion of a string.

Eg:

```
import java.io.*;
class A
{
public static void main(String... ar)
{
char[] arr= {'H', 'e', 'l' , 'l' , 'o', '-'};
String str="How are you today?";
try
{
        File file= new File("D:\\TextBook.txt");
        FileWriter fw= new FileWriter(file);
        for(char ch : arr) //For-each loop to write each character to a file
        fw.write(ch);
        fw.write(str); //Writing a String to a file
        fw.flush();
        fw.close();
}
catch(IOException e)
{
        System.out.println(e);
    }
  }
}
```

6Q). Explain about Zipping a File?

Ans:

1. Attach the input file **"file1"** to FileInputStream for reading data.
2. Take the output file **"file2"** and attach it to FileOutputStream. This will help to write data into file2
3. Attach FileUputStream to DeflaterOutputStream for compressing the data.
4. Now rad dadta from FileInputStream and write it into DeflaterOutputStream. It will compress the data and send it to FileOutputStream which stores the compressed data into the output file.

```
import java.io.*;
import java.util.zip.*;
class Zip
{
	public static void main(String[] args) throws Exception
	{
		FileInputStream fis=new FileInputStream("file1");
		FileOutputStream fos=new FileOutputStream("file2");
		DeflaterOutputStream dos=new DeflaterOutputStream(fos);
		int data;
		while((data=fis.read())!-1)
			dos.write(data);
		fis.close();
		dos.close();
	}
}
```

output:

```
c:\2mscs>javac Zip.java
c:\2mscs>java Zip
```

7Q) Explain about Unzipping a file?

Ans:

The file with the name **"file2"** contains compressed data and suppose we want to obtain original uncompressed data from this file.

Steps:

1. Attach the file compressed file "file2" to FileInputStream. This helps to read data from "file2"
2. Attach the output file "file3" to FileOutputStream. This will help to write uncompressed data into file3.
3. Attach FileInputStream to InflaterInputStream so that the data read from FileInputStream goes into InflaterInputStream. Now InflaterInputStream uncompresses the data
4. Now, read uncompressed data from InflaterInputStream and write it into FileOutputstream. This will write the uncompressed data to "file3".

Eg:
```
import java.io.*;
import java.util.zip.*;
class UnZip
{
	public static void main(String[] args) throws Exception
	{
		FileInputStream fis=new FileInputStream("file2");
		FileOutputStream fos=new FileOutputStream("file3");
		InflaterInputStream iis=new InflatgerInputStream(fis);
		int data;
		while((data!=iis.read())!=-1)
		fos.write(data);
		fos.close();
	}
}
```

8Q) Write about Serialization of objects?

Ans:

- **Serialization in Java** is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.
- The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.
- For serializing the object, we call the **writeObject()** method of ObjectOutputStream class, and for deserialization we call the **readObject()** method of ObjectInputStream class.
- We must have to implement the Serializable interface for serializing the object.
- **Serializable** is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The **Cloneable** and **Remote** are also marker interfaces.
- The **Serializable** interface must be implemented by the class whose object needs to be persisted.
- The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Advantages

- It is mainly used to travel object's state on the network (that is known as marshalling).

Student.java
```
import java.io.Serializable;
```

```
public class Student implements Serializable
{
	int id;
	String name;
	public Student(int id, String name)
	{
		this.id = id;
		this.name = name;
	}
}
```

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

public ObjectOutputStream(OutputStream out) throws IOException {}	It creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	It creates an ObjectInputStream that reads from the specified InputStream.

Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	It reads an object from the input stream.
2) public void close() throws IOException {}	It closes ObjectInputStream.

Example of Java Serialization

Persist.java

```
import java.io.*;
class Persist
{
	public static void main(String args[])
	{
	try
	{
	Student s1 =new Student(211,"ravi");
	FileOutputStream fout=new FileOutputStream("f.txt");
	ObjectOutputStream out=new ObjectOutputStream(fout);
	out.writeObject(s1);
	out.flush();
	out.close();
	System.out.println("success");
	}
	catch(Exception e){System.out.println(e);}
	}
}
```

Output:
Success

Q) Java File Class

Ans: The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

Constructor

Constructor	Description
File(File parent, String child)	It creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	It creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child)	It creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri)	It creates a new File instance by converting the given file: URI into an abstract pathname.

Methods

Modifier and Type	Method	Description

static File	createTempFile(String prefix, String suffix)	It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
Boolean	createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
Boolean	canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
Boolean	canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
Boolean	canRead()	It tests whether the application can read the file denoted by this abstract pathname.
Boolean	isAbsolute()	It tests whether this abstract pathname is absolute.
Boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
Boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path	toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
URI	toURI()	It constructs a file: URI that represents this abstract pathname.
File[]	listFiles()	It returns an array

		of abstract pathnames denoting the files in the directory denoted by this abstract pathname
Long	getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name.
String[]	list(FilenameFilter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
Boolean	mkdir()	It creates the directory named by this abstract pathname.

Java File Example 1

```
import java.io.*;
public class FileDemo
{
public static void main(String[] args)
{
try {
    File file = new File("javaFile123.txt");
    if (file.createNewFile())
    {
        System.out.println("New File is created!");
    }
    else
    {
         System.out.println("File already exists.");
    }
  } catch (IOException e) {
    e.printStackTrace();
}
}
}
```

Output:

New File is created!

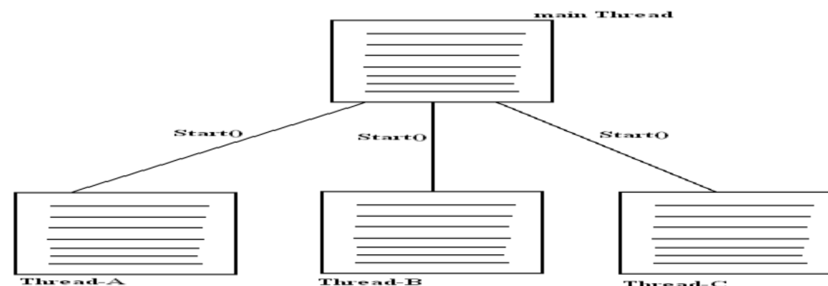
1Q) What is a Thread? Explain different ways of creating Threads?

Ans: A thread of execution is an individual process that has its own flow of control A thread is also called as a light weight process since multiple threads can share some resources among them. Each thread is associated with its own class stack.

Java Support the most powerful and useful feature called Multi-threading to execute multiple individual processes at a time simultaneously. In Multi-threading, a program is divided into two or more sub programs called processes which can be implemented at the same time.

A thread is process that has a single flow control. Each thread has a beginning, a body and an end. **The programs which have only one flow of control is called single-threaded programs** and **that having more than one flow of control are called multi-threaded programs**.

Every java program must use at least one thread to complete its execution. The main() method also runs in one thread called main thread. In main call stack, main() method is at the bottom of the stack.



Creating Threads:

In java, a thread means two different things,

1. An Instance of class java.lang.Thread
2. A thread of execution

A thread in java begins as an instance of java.lang.Thread. Creating threads in java is simple. Threads are implemented in the form of objects that contain a method called run(). The code which we want to execute in a separate thread must be placed in the run() method. The thread of execution always begins by invoking the run() method.

In java, we can define and instantiate a new thread in two different ways,

1. Defining a class that extends Thread class
2. Defining a class that implements runnable interface.

Extending the thread class:

In this methods, we define a class that extends the java.lang.Thread class and override the run() method. This is the simplest way of defining code to run in a separate thread. This gives us access to all the thread methods directly. This method includes the following steps.

a) Declare a class as extending java.lang.Thread class
b) Override the run() method by placing the code of the process
c) Create an object to the newly defined class
d) Call the start() method to invoke the thread for execution.

Syntax: class class_name extends Thread

```
{
  public static void main(String[] args) throws Exception
  {
        Thread t1=new class_name();
        Thread t2=new class_name();
        t1.start();
        t2.start();
  }
  public void run()
  {
        if(Thread1)
        {
              Statement1;
              Statement2;
        }
        if(thread2)
        {     Statement1;
              Statement2;
        }
   }
  }
```

Implements runnable interface

In this method, we define a class that implements the Runnable interface and implement the run() method which belongs to Runnable interface. This method gives the advantage when compared to method-1 that is we can extend our thread lass to any other class. It includes the following steps.

a) Declare a class that implements Runnable interface
b) Implement the run() method
c) Create Thread object and attach the Runnable class to it.
d) Call the start() method to invoke the thread for execution.

Syntax:

```
class class_name implements Runnable
{
    public static void main(String[] args) throws Exception
    {
        Runnable r=new syntest();
        Thread t1=new class_name(r);
        Thread t2=new class_name(r);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        }


        public void run()
```

```
    {
        if(Thread1)
        {
            Statement1;
            Statement2;
            }
        if(thread2)
        {
            Statement1;
            Statement2;
         }
    }
}
```

2Q) Write the uses of Threads?

Ans

1. Threads minimize the context switching time.
2. Use of threads provides concurrency within a process.
3. Efficient communication.
4. It is more economical to create and context switch threads.
5. Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

3Q) Write about the life cycle of threads?

Ans:

Thread can be defined as a sequence of instructions which can run independently. Threads can share data among other threads. Threads concept is the origin of multi-user systems. A program can be made executed quickly by dividing the program instructions into groups called threads. Each thread has a life cycle. In its life cycle the thread has five states. They are,

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

Born State:

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

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

If scheduled, it moves to the runnable state. If we attempt to use any other method at this stage, an exception will be thrown(interruptedException).

Runnable state:

If a thread is ready for execution and if the thread is waiting for the availability of processor then the thread is said to be in runnable state. In this state, the threads which are waiting for the availability of processor form a queue and whenever a thread gets processor it becomes active and moves to the running state. All the threads waiting in the queue have their own priorities. The thread with highest priority will go to the running state first. If the priorities of all threads are equal then the threads are equal then the threads are activated to first come first served fashion.

Running state:

A thread is said to be in running state if it is being executed i.e., when a processor is being allotted to the thread. A running thread may relinquish from control on its own or it relinquishes when it is being preempted by a high priority thread. A running thread may relinquishes from control for three reasons,

i) When a thread is being suspended using **suspend()** method, it moves from running state to blocked stated. The blocked thread can be renewed using **resume()** method.

ii) When a thread is made to sleep using **sleep(n)**. The thread gets blocked for given milliseconds.

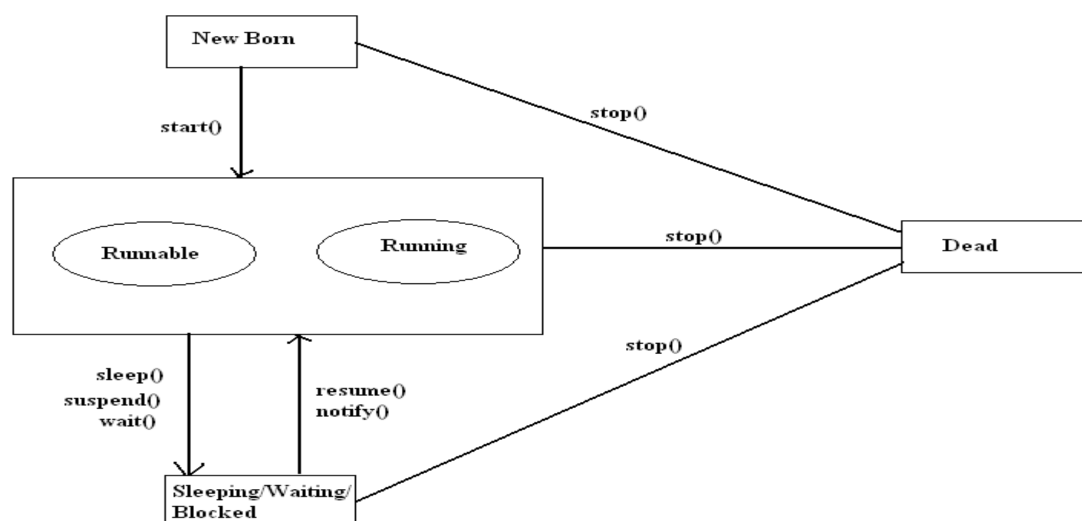
iii) When a thread is asleep to **wait()** method, the thread gets blocked and the thread can be renewed using **notify()** method.

Blocked State:

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

Dead State:

This is the final state in the life cycle of a thread. 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() method at any state thus causing a premature death to it. A thread can be killed as soon as it is born or while it is running, or even when it is in "not runnable" condition.



4Q) How terminating a Thread? Explain?

Ans:

- A thread in Java program will terminate (or move to dead state) automatically when it comes out of run() method. But if we want to stop a thread from running or runnable state, we will need to call stop() method of Thread class. The stop() method is generally used when we desire premature death of a thread.
- The general syntax for calling stop method in java programming is as follows:
  static void stop()
- Since the stop() method is static in nature, therefore, it can be called by using Thread class name. When the stop() method is called on a thread, it causes the thread to move to the dead state.

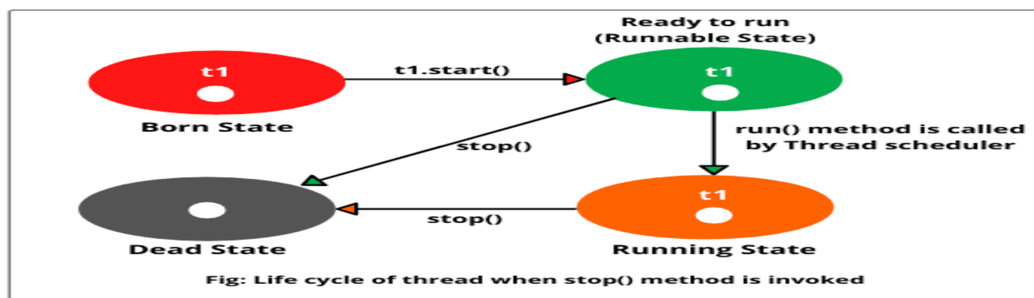


Fig: Life cycle of thread when stop() method is invoked

- As you can observe in the above figure, a thread is terminated when it moves into the dead state either from running or runnable state. If any other thread does not interrupt a thread, it terminates normally.
- If a running thread is interrupted then it throws **InterruptedException** and is terminated. After a thread is gone into the dead state, it cannot be made alive even after the calling of start() method.
- If we will try to call start() method on a dead thread, start() method throws an exception named **IllegalThreadStateException.**

Eg:-1

```
public class Kill extends Thread
{
  static Thread t;
  public void run()
  {
        System.out.println("Thread is running");
        t.stop(); // Calling stop() method on Kill Thread.
        System.out.println("Learn Java step by step");
        }
  public static void main(String[] args)
  {     Kill k = new Kill();
              t = new Thread(k);
        t.start(); // Calling start() method.
     }
}
```

Output: Thread is running

Eg:-2

```
public class Thread1 implements Runnable
{
        public void run()
        {     System.out.println("First child thread");      }
}
public class Thread2 implements Runnable
{
        static Thread t2;
        public void run()
        {
                for(int i = 0; i <= 10; i ++)
                {
                        System.out.println("Second child thread: " +i);
                         if(i==5)
                         {
                                t2.stop(); // Calling stop() method to kill running thread.
                        }
                }
        }
}
public class MyThreadClass
{
        public static void main(String[] args)
        {
                Thread1 th1 = new Thread1();
                 Thread2 th2 = new Thread2();
                Thread t1 = new Thread(th1);
                Thread t2 = new Thread(th2);
         t1.start();
         t1.stop(); // Calling stop() method to kill runnable thread.
         t2.start();
         }
}
```

Output:

```
Second child thread: 0
Second child thread: 1
Second child thread: 2
Second child thread: 3
Second child thread: 4
Second child thread: 5
Exception in thread "Thread-1" java.lang.NullPointerException
        at threadProgram.Thread2.run(Thread2.java:13)
        at java.lang.Thread.run(Unknown Source)
```

5Q) Explain about single tasking using a Thread?

Ans:
In java, threads refer to the path that a code follows during its execution. The Java Virtual Machine (J.V.M.) creates the main thread at the start of the program when the main() method of the program is invoked.

We can perform a single task using the multiple threads by writing a single run() method for all the threads.

Eg:
```
class MyThread extends Thread
{
   public void run()
   {      task1();
          task2();
          task3();
    }
   void task1()
   {      System.out.println("This is Task 1");     }
   void task2()
   {      System.out.println("This is Task 2");     }
   void task3()
   {      System.out.println("This is Task 3");     }
}
class Single
{
   public static void main(String args[])
   {
          MyThread t1=new MyThread();
          t1.start();
   }
}
```
Output:
This is Task-1
This is Task-1
This is Task-1

6Q) Explain about Multiple Threads acting a single object

Ans:
Creating more than one thread to perform multiple tasks is called **multithreading in Java**. In multiple threading programming, multiple threads are executing simultaneously that improves the performance of CPU because CPU is not idle if other threads are waiting to get some resources.

Multiple threads share the same address space in the heap memory. Therefore, it is good to create multiple threads to execute multiple tasks rather than creating multiple processes. Look at the below picture.

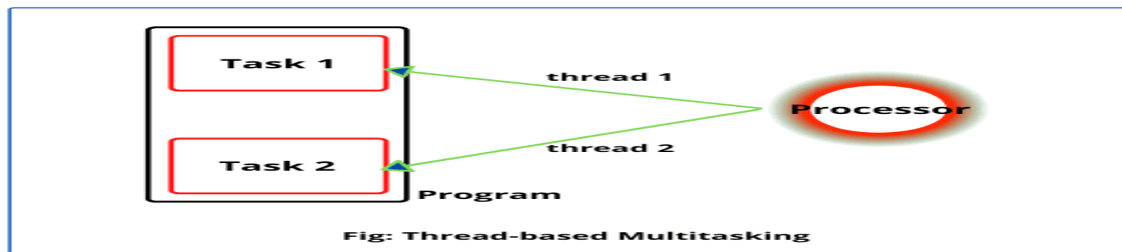


Fig: Thread-based Multitasking

Eg:

```
public class MultipleThread implements Runnable
{
        String task;
        MultipleThread(String task)
        {       this.task = task;   }
         public void run()
         {
                for(int i = 1; i <= 5; i++)
                {       System.out.println(task+ ":" +i);
                        try {   Thread.sleep(1000);      }
                        catch (InterruptedException e) {
                        e.printStackTrace();
                }
          }
         }
public static void main(String[] args)
{       Thread nThread = Thread.currentThread();
        System.out.println("Name of thread: " +nThread);
        MultipleThread mt = new MultipleThread("Hello Java");
        Thread t1 = new Thread(mt);
        Thread t2 = new Thread(mt);
        Thread t3 = new Thread(mt);
        t1.start();
        t2.start();
        t3.start();
        int count = Thread.activeCount();
        System.out.println("No of active threads: " +count);
        }}
```

Output:

```
Name of thread: Thread[main,5,main]
No of active threads: 4
Hello Java:1
Hello Java:1
Hello Java:1
Hello Java:2
Hello Java:2
. . . . . .
. . . . . .
Hello Java:5
```

7Q) Thread communication

Ans:

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:

1) wait()
2) notify()
3) notifyAll()

1) **wait()**

   The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

   The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

2) **notify()**

   The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

   Syntax: public final void notify()

3) **notifyAll()**

   Wakes up all threads that are waiting on this object's monitor.

   Syntax: public final void notifyAll()

Eg:

```
class Customer
{
	int amount=10000;
	synchronized void withdraw(int amount)
	{
		System.out.println("going to withdraw...");
		if(this.amount<amount)
		{
			System.out.println("Less balance; waiting for deposit...");
			try
			{	wait(); }
			 catch(Exception e){}
		}
		this.amount-=amount;
		System.out.println("withdraw completed...");
	}
```

```
        synchronized void deposit(int amount)
        {
                System.out.println("going to deposit...");
                this.amount+=amount;
                System.out.println("deposit completed... ");
                notify();   }
}
class Test
{
        public static void main(String args[])
        {       final Customer c=new Customer();
                new Thread(){
                public void run(){c.withdraw(15000);}
                }.start();
                new Thread(){
                public void run(){c.deposit(10000);}
                }.start();
        }}
```

Output:

going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed

8Q) Write about Thread Information?

Ans: Every thread carries properties along with it. They are **Thread Name, Thread Priority and Thread Group Name**

Two or more threads can be grouped with help of a thread group, to execute functionality collectively. By default all the child threads created in main thread are placed under main thread group. **Default priority** of main thread is **5** which is known as "**Normal Priority**". By default child thread acquires the same priority as that of parent thread.

Priority for a thread can be set between1 to 10 any number. **1 is minimum priority,10 is maximum priority** and **5 is Normal Priority**. To support these values thread has static final variables. They are –

1. Thread.MIN_PRIORITY -1
2. Thread.NORM_PRIORITY – 5
3. Thread.MAX_PRIORITY – 10

Methods:

1. setPriority() is a method from thread class to change the priority of a thread.
2. **getPriority()** is a method to get the priority of a thread
3. **setName()** is a method to set the name for the thread
4. **getName()** is a method to get the name of the thread
5. **getThreadGroup()** is a method to get the name of the thread group

9Q) Thread class methods

Ans:

Thread class provides various static methods that are as follows:

1. **currentThread():**
   The currentThread() returns the reference of currently executing thread. Since this is a static method, so we can call it directly using the class name. The general syntax for currentThread() is as follows:
   Syn: public static Thread currentThread()
2. **sleep():**
   The sleep() method puts currently executing thread to sleep for specified number of milliseconds. This method is used to pause the current thread for specified amount of time in milliseconds.
   Since this method is static, so we can access it through Thread class name. The general syntax of this method is as follows:
   Syn:
   public static void sleep(long milliseconds) throws InterruptedException
   public static void sleep(long mseconds, int nseconds) throw InterruptedException
4. **yield():**
   The yield() method pauses the execution of current thread and allows another thread of equal or higher priority that are waiting to execute. Currently executing thread give up the control of the CPU. The general form of yield() method is as follows:
   Syn: public static void yield()
5. **activeCount():** This method returns the number of active threads.
   Syn: public static int activeCount()

10Q) Write anpit Daemon Thread?

Ans:

- **Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.
- There are many java daemon threads running automatically e.g. gc, finalizer etc.
- You can see all the detail by typing the **jconsole in the command prompt.** The jconsole tool provides information about **the loaded classes, memory usage, running threads etc.**
- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- It is a low priority thread.
- The java.lang.Thread class provides two methods for java daemon thread.
  a. **Public void setDaemon(Boolean):**
     It is used to mark the current thread as daemon thread or user thread
  b. **public boolean isDaemon():**
     It is used to check that current is daemon.

Eg:

```
public class TestDaemonThread1 extends Thread
{
        public void run()
        {
                if(Thread.currentThread().isDaemon())//checking for daemon thread
                        System.out.println("daemon thread work");
                else
                        System.out.println("user thread work");
        }
 public static void main(String[] args)
{
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();
  t1.setDaemon(true);//now t1 is daemon thread
  t1.start();//starting threads
  t2.start();
  t3.start();
   }
}
```

Output

daemon thread work
user thread work
user thread work

Eg-2:

```
class TestDaemonThread2 extends Thread
{
        public void run()
        {
                System.out.println("Name: "+Thread.currentThread().getName());
                System.out.println("Daemon: "+Thread.currentThread().isDaemon());
         }
        public static void main(String[] args)
        {
                TestDaemonThread2 t1=new TestDaemonThread2();
                TestDaemonThread2 t2=new TestDaemonThread2();
                t1.start();
                t1.setDaemon(true);//will throw exception here
                t2.start();
        }
}
```

Output: exception in thread main: java.lang.IllegalThreadStateException

10Q) What is Applet? Write about types of Applet?

Ans:

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

Types of Applet

Applets are of 2 types. They are

1) **Local Applet**
2) **Remote Applet**

Local Applet

- An applet developed locally and stored in a local system is known as **local applet**.
- When a web page is trying to find a local applet, it does not need to use the internet and therefore the local system does not require the internet connection. It simply searches the directories in the local system and locates and loads the specified applet.

Remote Applet

- A **remote applet** is that which is developed by someone else and stored on a remote computer connected to the Internet. If our system is connected to the Internet, we can download the remote applet onto our system via the Internet and run it.
- In order to locate and load a remote applet, we must know the applet's address on the Web. This address is known as Uniform Resource Locator and must be specified in the applet's HTML document as the value of the CODEBASE attribute.

11Q) What is Applet? Write about the life cycle methods of Applet?

Ans:

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

LIFE CYCLE METHODS OF APPLET

Every Java applet inherits a set of default behaviors from the **Applet** class. As a result, when an applet is loaded, it undergoes a series of changes in its state. The applet states include Born or Initialization state, Running state, Idle State, Dead or destroyed state. Applet uses following methods to perform its respective task.

- init( )
- start( )
- paint( )
- stop( )

- destroy( )

init():

init() method is for initialization. Control entry part for an applet is through init() method. This method invokes only once for the first time when the applet is loaded.

start():

This method is executed after the init() method. In case, java capable browser is used to run the applet, any time it is reloaded, the execution begins from the start() method.

paint():

This method is implicit after start() method. This is the only method of applets life cycle which needs a **parameter of graphics type**. paint() method is to draw graphics on the applet. The paint method is from java.awt package.

stop():

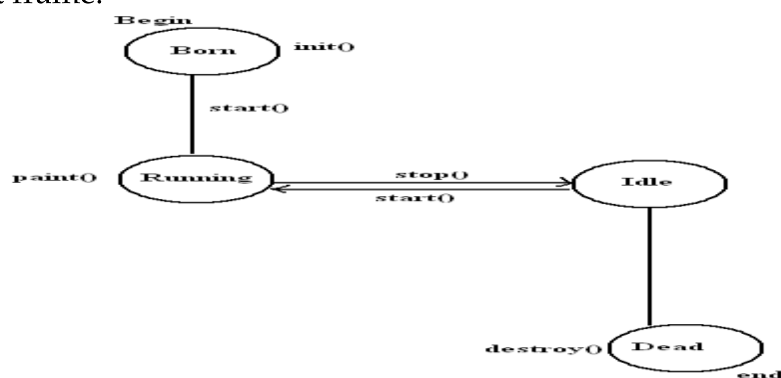
The stop() method is used to halt the running of an applet. A stopped applet can be started with the help of start() method.

destroy( ):

This method is used to free the memory occupied by the variables and objects initialized in the applet. Any clean up activity that needs to be performed can be done in this method. The destroy method is different from the finalize() method. The destroy() method applies only to applets whereas the finalize() method is used generally to clean up a single object.

repaint( ):

This method is used in case an applet is to be repainted. The repaint() calls the update() method., to clear the screen of any existing content. The update() method in turn calls the paint() method that then draws the contents of the current frame.



12Q) Write the differences between Application and Applet?

Ans:

APPLICATIONS	APPLETS
1. Stand-alone Applications are java programs	1. Applets are java programs

2. Stand-alone applications are full featured programs.	2. Applets are not full-featured application program. They are usually written to accomplish a small task.
3. Stand-alone applications require main() method for initiating the execution of the code.	3. Applets do not use the main() method for initiating the execution of the code. Applet, when loaded, automatically call certain methods of Applet class to start and execute the applet code.
4.Stand-alone applications run independently.	4. Applets cannot be run independently. They run from inside a web page using a special feature known as HTML tags.
5. Application programs can read and write to the files at the local computer	5. Applets cannot read from or write to the files in the local computer
6. Stand-alone application cannot communicate with other servers	6. Applets cannot communicate with other servers on the network.
7. Stand-alone applications can dynamically include other libraries or functions of other languages.	7. Applets are restricted from using libraries from other languages such as C.

13Q) Applet Tag

Ans:

- HTML <applet> tag was used to embed the Java applet in an HTML document. This element has been deprecated in HTML 4.0 and instead of it we can use <object> and newly added element <embed>.
- The use of Java applet is also deprecated, and most browsers do not support the use of plugins.

Syn: <applet code="URL" height="200" width="100">.............</applet>

Attributes:

Attribute name	Value	Description
Code	URL	It specifies the URL of Java applet class file.
Width	pixels	It specifies the display width of the applet panel.
Height	pixels	It specifies the display height of applet panel
Align	○ left ○ right ○ top ○ middle ○ bottom	It specifies the position of applet application relative to surrounding content.

Alt	text	It is used to display alternative text in case browser does not support Java.
codebase	URL	It specifies the exact or relative URL of applets .class file specified in the code attribute.
Hspace	pixels	It specifies the horizontal space around the applet.
Vspace	pixels	It specifies the vertical space around the applet.
Name	name	It specifies the name for the applet

14Q) How to pass Parameter in Applet

Ans:

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

Syntax: public String getParameter(String parameterName)

Example of using parameter in Applet:

```
import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet
{
	public void paint(Graphics g)
	{	String str=getParameter("msg");
		g.drawString(str,50, 50);
	}
 }
```

myapplet.html

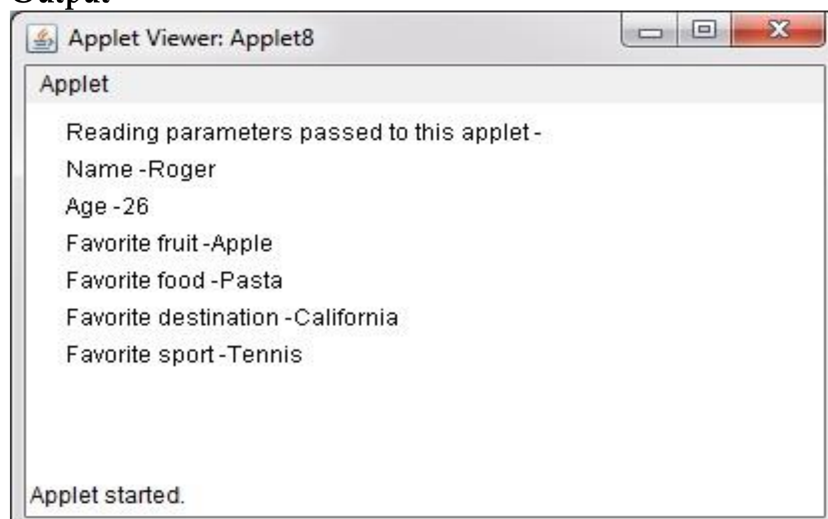
```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

EG-2:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Applet8" width="400" height="200">
<param name="Name" value="Roger">
```

```
<param name="Age" value="26">
<param name="Sport" value="Tennis">
<param name="Food" value="Pasta">
<param name="Fruit" value="Apple">
<param name="Destination" value="California">
</applet>
*/
public class Applet8 extends Applet
{
        String name;
        String age;
        String sport;
        String food;
        String fruit;
        String destination;
        public void init()
        {       name = getParameter("Name");
                age = getParameter("Age");
                food = getParameter("Food");
                fruit = getParameter("Fruit");
                destination = getParameter("Destination");
                sport = getParameter("Sport"); }
        public void paint(Graphics g)
        {       g.drawString("Reading parameters passed to this applet -", 20, 20);
                g.drawString("Name -" + name, 20, 40);
                g.drawString("Age -" + age, 20, 60);
                g.drawString("Favorite fruit -" + fruit, 20, 80);
                g.drawString("Favorite food -" + food, 20, 100);
                g.drawString("Favorite destination -" + name, 20, 120);
                g.drawString("Favorite sport -" + sport, 20, 140);
        }
}
```

Output



15Q) Animation in Applet

Ans:

Applet is mostly used in games and animation. For this purpose image is required to be moved.

//Example of animation in applet

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{
  Image picture;
  public void init()
  {
    picture =getImage(getDocumentBase(),"bike_1.gif");
  }
  public void paint(Graphics g)
  {
    for(int i=0;i<500;i++){
      g.drawImage(picture, i,30, this);
        try{Thread.sleep(100);}catch(Exception e){}
    }
  }
}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>  </html>
```