SRI GCSR COLLEGE

Study Material

Programming in JAVA

Prepared by

Department of Computer Science
Sri GCSR College
GMR Nagar: Rajam

UNIT-1

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, for Loop, 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() Arrays: Types of Arrays, Three Dimensional Arrays (3D array), arrayname.length, Command Line Arguments

---

1. Explain about Java Environment.

Java environment includes a large number of development tools that are used for hundreds of classes and methods. They are two categories. They are:

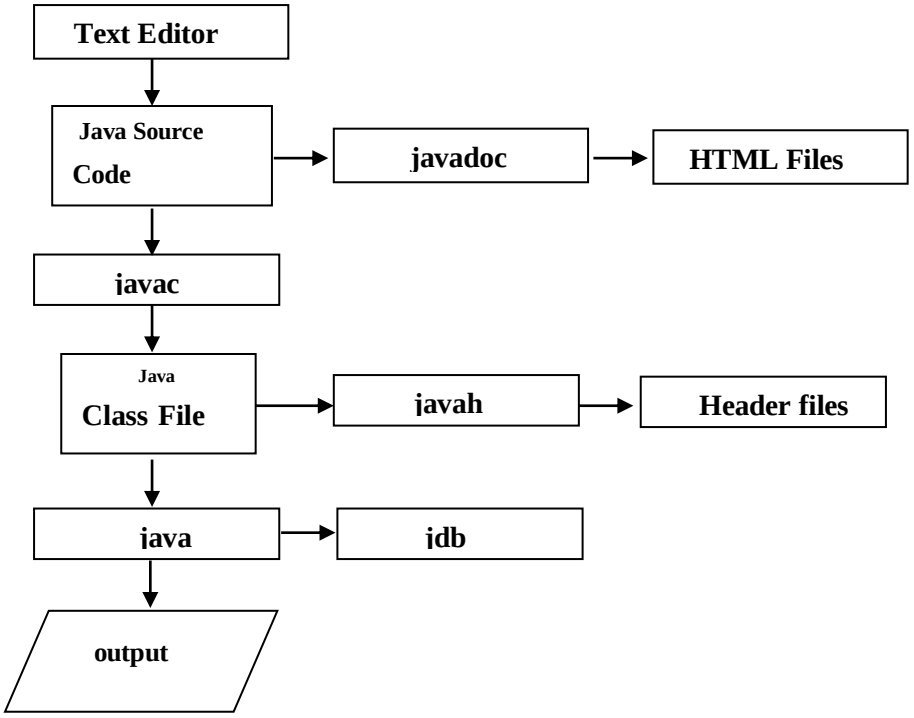
Java Development Kit (JDK):

It contains the development tools such as applet viewer, java, javac, etc.

Table of Java Development Tools

Tool	Use
1. Applet viewer	It enables us to run java applets.
2. Javac (java compiler)	It translates java source code to bytecode.
3. Java (java interpreter)	It interprets java bytecode.
4. Javadoc	It creates HTML format documentation from java source code files.
5. Javah	It produces header files for use with native methods.
6. Javap (java diassembler)	It enables us to convert bytecode files into a program.
7. Jdb (java debugger)	It helps us to find errors in our programs.

Flowchart for the usage of java tools in application development



2. Explain about Java Standard Library (JSL) or Application Programming Interface (API)

It contains the classes and methods such as language support package, utilities package, etc. It is also known as Application Programming Interface (API). Most commonly used packages are:

a) Language Support Package: It is a collection of classes and methods required for implementing basic features of java.

java.lang.*;

b) Utilities Package: It is a collection of classes that contains utility functions such as date and time functions.

java.util.*;

c) Input / Output Package: It is a collection of classes required for input/output manipulations.

java.io.*;

d) Networking Package: It is a collection of classes for communicating with other computers through internet.

java.net.*;

e) AWT Package: The Abstract Window Tool Kit package contains classes that implements graphical user interface.

java.awt.*;

f) Applet Package: It is a collection of classes that allows us to create java applets.

java.applet.*;

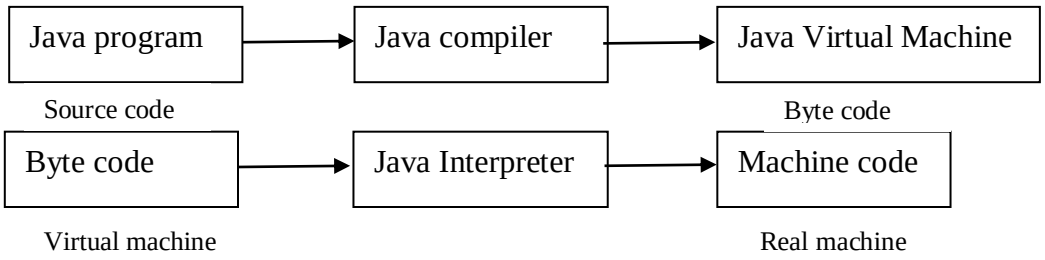
Java Runtime Environment (JRE):

It facilitates the execution of programs development in java. It contains the following:

3. Explain about Java Virtual Machine (JVM):

JVM is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of JRE(Java Run Environment). It stands for Java Virtual Machine

- In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
- First, Java code is complied into bytecode. This bytecode gets interpreted on different machines
- Between host system and Java source, Bytecode is an intermediary language.
- JVM is responsible for allocating memory space.



Software Code Compilation & Execution process

In order to write and execute a software program, you need the following

a) Editor – To type your program into, a notepad could be used for this

b) Compiler – To convert your high language program into native machine code

c) Linker – To combine different program files reference in your main program together.

d) Loader – To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.

e) Execution – Actual execution of the code which is handled by your OS & processor.

With this background, refer the following video & learn the working and architecture of the Java Virtual Machine.

4. Explain the features of JAVA.

Java is an object-oriented, cross platform, multi-purpose programming language developed by **James Gosling, Patrick Naughton** and their team members at Sun Microsystems (Sun) in 1991. It was originally called as OAK by James Gosling later renamed to 'Java'. It was specially designed to develop software for consumer electronic devices like mobiles, TVs, VCRs, micro wave ovens and other electronic machines.. The first publicly available version of Java (Java 1.0) was released in 1995. It has earned a prominent place in the world of computer programming. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the steermanship for Java.

The target of Java is to write a program once and then run this program on multiple operating systems.

Java features:

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java *buzzwords*.

A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

i. Simple:

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on existing programming languages.
- Java has removed many complicated and rarely-used features,

For example:- C language pointers, C++ operator overloading, etc.

- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

ii. Object-oriented:

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are: Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation

iii. Portable:

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

iv. Platform independent:

A platform is the hardware or software environment in which a program runs. 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.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

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).

v. Secured:

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox

vi. Robust

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. Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking. All these points make Java robust.

vii. 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.

viii. 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.

ix. Distributed

Java is distributed because it facilitates users to create distributed applications in Java. 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.

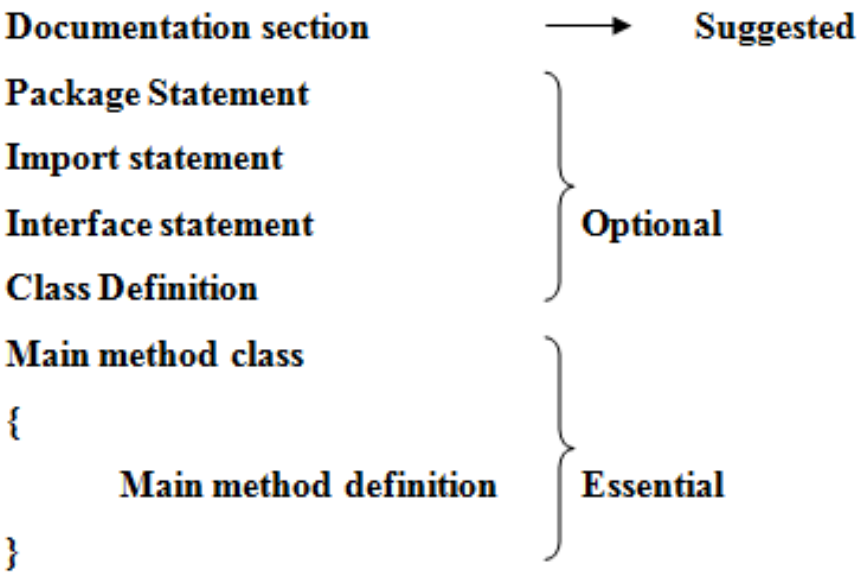
x. Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. 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.

xi. Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

5. **Explain the structure of Simple java program.**



- **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.

  Example: **// Addition of two numbers**

- **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.

  Example: **package arithmetic;**

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

  import arithmetic. Addition;

- **Interface statement:** 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 interface_name;

- **Class definition:** A Java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program. The number of classes used depends on the complexity of the problem.

  class class_name

```
{
        Data members;
        methods
}
```

- **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. The main method creates objects of various classes and establishes communications between them.

```
class Sample
{
        public static void main (String args[])
        {
                -------
        }       }
```

6. Explain the Difference between Java and C Language.

C	**JAVA**
C is a Procedural Programming Language.	Java is Object-Oriented language.
C is a middle-level language because binding of the gaps takes place between machine level language and high-level languages	Java is a high-level language because translation of code takes place into machine language using compiler or interpreter.
C is a compiled language that is it converts the code into machine language so that it could be understood by the machine or system.	Java is an Interpreted language that is in Java, the code is first transformed into bytecode and that bytecode is then executed by the JVM (Java Virtual Machine).
C generally breaks down to functions.	Java breaks down to Objects.
Memory allocation can be done by malloc in C	Memory allocation can be done by a new keyword in Java.
free is used for freeing the memory in C.	A compiler will free up the memory internally by calling the garbage collector.
C supports pointers.	Java does not supports pointers.
C supports storage classes	Whereas JAVA does not supports storage classes
It has 32 keywords.	It has 50 keywords.

7. Explain the Difference between Java and C++ Language.

C++	JAVA
C++ was Influenced by Influenced by Ada, ALGOL 68, C, ML, Simula, Smalltalk, etc. languages.	Java was Influenced by Ada 83, Pascal, C++, C#, etc. languages.
Platform dependent, should be compiled for different platforms.	Platform independent, Java bytecode works on any operating system.
C++ is a Compiled Language.	Java is both Compiled and Interpreted Language.
Memory management in C++ is manual	Memory management is system controlled
It supports both single and multiple inheritance	It supports only single inheritance. multiple inheritances are achieved using interfaces
It supports both method and operator overloading.	It supports only method overloading and doesn't allow operator overloading.
It strongly supports pointers	It has limited supports of pointers
It supports direct system library calls, making it suitable for system-level programming.	It doesn't support direct native library calls but only Java Native Interfaces.
C++ is both a procedural and an object-oriented programming language.	Java is only an object-oriented programming language.

8. Explain Naming Conventions in Java

In java, it is good practice to name class, variables, and methods name as what they are actually supposed to do instead of naming them randomly.

- Class names should be **nouns**, in mixed cases with the **first** letter of each internal word capitalized. Interfaces names should also be capitalized just like class names.
- Use whole words and must avoid acronyms and abbreviations.

```
class Student { }
class S=Integer {}
class Scanner {}
```

- Methods should be **verbs**, in mixed case with the **first letter lowercase** and with the first letter of each internal word capitalized.

```
public static void main(String [] args)  {}
```

Variable names should be short yet meaningful.

Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.

- Should be mnemonic i.e, designed to indicate to the casual observer the intent of its use.
- **One-character variable names should be avoided** except for temporary variables.
- Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

```
int[] marks;
double double answer,
```

- Should be **all uppercase** with words separated by underscores ("_").
- There are various constants used in predefined classes like Float, Long, String etc.

```
num = PI;
```

- The prefix of a unique package name is always written in **all-lowercase ASCII letters** and should be one of the top-level domain names, like com, edu, gov, mil, net, org.
- Subsequent components of the package name vary according to an organization's own internal naming conventions.

```
java.util.Scanner ;
java.io.*;
```

9. Explain Constants and variables in JAVA:

Constants:

Constants in Java refer to fixed values that do not change during the execution of a program.

Integer constant:

An integer constant refers to a sequence of digits.

There are three types of integer namely decimal integer, octal integer and hexadecimal integer.

A. Decimal integers consist of digits 0 through 9, preceded by an optional minus sign. Spaces, commas and non-digit characters are not permitted between digits.

Example: 127, -131, 65534 etc.,

B. An octal integer constant consists of digits from the 0 through 7, with leading O.

Example: O37, O435, o346 etc.

C. A sequence of digits preceded by Ox or OX is considered as hexa-decimal integer. They may also include alphabets A through F or a through f. A letter A through F represents the numbers 10 through 15.

Example: OX2, OX9F, OXbcd etc.,

Real Constants:

Numbers containing fractional parts like 17.548. Such numbers are called real constants.

A real number may also be expressed in exponential notation. The general form is

mantissa E exponent

Mantissa is either a real number or an integer. The exponent is an integer with an optional + or – sign. The mantissa and the exponent can be written in either lower case or uppercase. Example: 215.65 may be written as 2.1565e2.

e2 means multiple by 102.

Single character constant:

A single character constant contains a single character enclosed within a pair of single quote marks.

Examples: '5' 'x' ';' ' '

Note that the character constant '5' is not the same as the number 5.

String character constant:

A string constant is a sequence of characters enclosed between double quotes. The character may be alphabets, digits; special characters and blank spaces.

Example: "2009" "Programming with Java" "3+6=9" "X"

Backslash character constant:

Java supports some special backslash character constants that are used in output methods. For example the symbol '\n' stands for new line character.

List of backslash character constants

Constant	Meaning
'\b'	Back space
'\f'	Form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\"	Single quote
'\"'	Double quote
'\\'	backslash

Variable:

A variable is a data name that may be used to store a data value. A variable may take different values at different times during execution. A variable name can be chosen by the programmer in a meaningful way so as to reflect what it represents in the program.

a. Rules for a variable:

b. They must begin with a letter.

c. The upper and lowercase are distinct.

d. It should not be a keyword.

e. White spaces are not allowed.

f. Variable names can be of any length.

Declaration of variables:

In Java, variables are the name of storage locations. After designing suitable variable names, we must declare them to the compiler.

- It tells the name of the variable.
- It specifies the type of data.
- The place of declaration decides the scope of the variable.

Syntax: type variable_1, variable_2, ... variable_n;

Example: int a,b;

Assigning values to variables:

A variable must be given value after it has been declared but before it is used in an expression.

Syntax: variable name=value; Example: x=0;

It is also possible to assign a value to a variable at the time of its declaration.

Syntax: type variable name=value; Example: int y=2;

The process of giving initial values to variable is known as the initialization. If the variable is not initialized, it is automatically set to zero.

Scope of variables: The area of the program where the variable is accessible is called its scope.

Instance variables: Instance and class variable are declared inside a class. Instance variables are created, when the objects are instantiated and therefore they are associated with the objects. They take different values for each object.

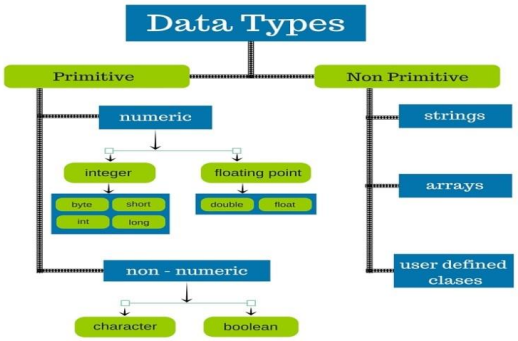
Class variables: Class variable are global to a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variable.

Local variables: Variables declared and used inside methods are called local variable. They are not available for the outside the method definition. Local variables can also be declared inside program blocks that are defined between an opening brace and a closing brace.

10. Explain different data types supported by JAVA:

Data type is the representation of the kind of data that we use in the program. Every variable in Java has a data type. Data types specify the size and type of values that can be stored. Java language is rich in its data types.

A. **Integer types:** Integer types can hold whole numbers along with the negative numbers. The size of the values that can be stored depends on the integer data type. Java supports four types of integers. They are byte, short, int and long. Java doest not support the concept of unsigned types.



Type	Size	Default value	Minimum Value	Maximum Value
float	4 bytes	0.0f	3.4e-038	3.4e+038
double	8 bytes	0.0d	1.7e-308	1.7e+308

Type	Size	Default value	Minimum Value	Maximum Value
byte	1 byte	0	-128(-2^7)	127(2^7-1)
short	2 bytes	0	-32,768(-2^{15})	32,767(2^{15}-1)
int	4 bytes	0	-2,147,483,648(-2^{31})	2,147,483,647(2^{31}-1)
long	8 bytes	0L	-9,232,372,036,854,775,808(-2^{63})	-9,232,372,036,854,775,807(2^{63}-1)

B. **Floating Point Types:** Floating point type can hold numbers containing fractional part such as 22.365 and -89.365. There are two kinds of floating point storage in java.

- Floating point numbers are treated as double precision quantities. To force them to be in single precision mode, we must append f or F to the number.
- Double-precision types are used when we need greater precision in storage of floating point number.

C. **Character type:** In order to store character constant in memory, java provides a character data type called char. The char type assumes a size of 2 bytes but basically it can hold only a single character.

D. **Boolean type:** Boolean type is used when we want to test a particular condition during the execution of the program. There are only two values that a Boolean type can take true or false. Boolean type is denoted by the keyword Boolean and uses only one bit of storage.

11. Explain about Java Tokens.

The smallest individual units in a program are known as tokens. Java program is a collection of tokens, comments and white spaces. Java language includes five types of tokens.

1. Keywords
2. Identifiers
3. Literals (Constants)
4. Operators
5. Separators

1. Keywords: Keywords are the reserved or pre-defined words. Java language has 60 keywords. Since keywords have specific meaning in Java, we cannot use them as names for variables, classes, methods and so on. All keywords are to be written in lower-case letters. Since Java is case-sensitive.

The following table lists the java keyword

abstract	Continue	for	New	switch
assert***	Default	goto*	Package	synchronized
boolean	Do	if	Private	this
break	Double	implements	protected	throw
byte	Else	import	Public	throws
case	enum****	instanceof	Return	transient
catch	Extends	int	Short	try
char	Final	interface	Static	void
class	Finally	long	strictfp**	volatile
const*	Float	native	Super	while

2. Identifiers: Identifiers refer to the names of classes, methods, variables, arrays, objects, labels, packages and interfaces in a program. These are user- defined names and consist of a sequence of letters and digits with a letter as a first character.

Rules for identifiers:

- They can have alphabets, digits, underscore and dollar sign.
- They must begin with a letter.
- The upper and lowercase are distinct.
- The variable name should not be a keyword.
- White spaces are not allowed.
- They can be of any length.
- Identifiers cannot match any of Java's reserved words.

Examples: These identifiers are valid:

MyClass

$amount

totalGrades;

TotalGrades;…..etc

3. Literals (Constants): Literals in Java are a sequence of characters that represent fixed values that do not change during the execution of a program.**Example of literals:**

Integer literals:	33 0 -9
Floating-point literals:	3 0.3 3.14
Character literals:	'(' 'R' 'r' '{'
Boolean literals:	(predefined values)true false
String literals:	"language" "0.2" "r" ""

4. Operators: An operator is a symbol that takes one or more arguments and performs certain mathematical or logical manipulations.

Types of Operators:

1. Arithmetic operator
2. Relational operator
3. Logical operator
4. Increment and decrement operator
5. Bit wise operator
6. Assignment operator
7. Conditional operator
8. Special operator

5. Separators:

Separators are symbols used to indicate where groups of code are divided and arranged. They basically define the shape and function of our code.

Parentheses	( )	Used to enclose parameters in method definition
braces	{ }	Used to contain the values of automatically initialized arrays
brackets	[ ]	Used to declare array types and for dereferencing array values.
semicolon	**:**	Used to separate statements
comma	**,**	Used to separate consecutive identifies in a variable declaration.
Period	**.**	Used to separate package names from sub-packages and classes also used to separate a variable or method from a reference variable

12.Explain different operators supported by JAVA

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

Operands Operands are the values on which the operators act upon. An operand can be a value or a variable

Types of Operators:

The operators can be classified into three types based on the **Number of operands**

1. Unary if it acts on a single operand (Eg : ++, --)
2. Binary if it requires two operands. (Eg: +, *)
3. Ternary if it requires three operands. The conditional operator is the only ternary operator in Java.

The operators can be classified into the following categories **based on the operations they perform**

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bitwise Operators
5. Assignment Operators
6. Conditional Operators
7. Special Operators

Arithmetic Operators:-

Java's arithmetic operators are used for performing basic arithmetic operations on numerals and characters. The following table lists arithmetic operators

Operator	**Name**	**Example**	**Result**	**Description**
a + *b*	Addition	3 + 5	8	Adds the two operands(Can also be used for String Concatenation)
a – *b*	Subtraction	3 - 5	-2	Subtracts the second operand from the first Operand.
a * *b*	Multiplication	3 * 5	15	Multiplie both the operands.
a / *b*	Division	15 /3	5	Divides the first operand by the second.
a % b	Modulo Division	3 % 2	1	Returns the remainder after dividing the first number by the second

Arithmetic expressions:

An arithmetic expression is a combination of variables, constants and arithmetical operators arranged as per the syntax of the language.

Example: (m+n) * (x+y)

There are three types of arithmetic expressions. They are:

1. **Integer Arithmetic Expression:** It is an expression in which all the operands are integers and the result is also an integer.

   Ex: a=5; b= 6;

   a + b = 11

2. **Real Arithmetic Expression:** It is an expression in which all the operands are real and the result is real.

   Ex: a=2.5; b= 3.5;

   a + b = 6.0

3. **Mixed Mode Arithmetic Expression:** It is an expression in which all the operands are both integers and real and the result is real.

   Ex: a=2.5; b= 6;

   a + b = 8.5

Relational Operators:

Relational Operators are used for comparing numerals and the characters. The result of applying relational operators is either true or false.

The following table lists the relational operators

Operator	Name	Example	Result	Description
a > *b*	Greater than	3 > 5	False	Returns true if the first operand is greater than the second operand
a < *b*	Lesser than	5 < 5	False	Returns true if the first operand is lesser than the second operand
a >= *b*	Greater than or equal	3 >= 5	False	Returns true if the first operand is greater than or equal to the second operand.
a <= *b*	Lesser than or equal to	15 <= 15	True	Returns true if the first operand is lesser than or equal to the second operand.
a == b	Equal To	3 == 2	False	Returns true if the first operand is equal to the second operand.
a != b	Not equal to	3 != 2	True	Returns true if the first operand is not equal to the second operand.

Logical Operators

Relational operators enable you to compare two variables to determine whether they are equal or if one is greater than the other, and so on. But when you want to check to see if a variable is in between a range of values or to check multiple conditions we use logical operators. The following table lists the logical operators

Operator	Name	Example	Result	Description
a && b	Logical AND	(3>5) && (2>5)	True	Returns true if the both operands is true.
a || b	Logical OR	(3>5) && (10>5)	True	Returns true if any of the operands is true
! a	NOT	! (3>= 5)	True	Complements the result of the expression

Bitwise Operators

Java's bitwise operators operate on individual bits of integer (int and long) values. If an operand is shorter than an int, it is promoted to int before doing the operations.

It helps to know how integers are represented in binary. For example the decimal number 3 is represented as 11 in binary and the decimal number 5 is represented as 101 in binary. Negative integers are store in two's complement form.

For example, -4 is 1111 1111 1111 1111 1111 1111 1111 1100.

The following table lists bitwise operators

Operator	Name	Example	Result	Description
a& b	And	3 & 5	1	1 if both bits are 1.
a|b	Or	3 | 5	7	1 if either bit is 1.
a ^ b	Xor	3 ^ 5	6	1 if both bits are different.
~a	Not	~3	-4	Inverts the bits.
n<<p	left shift	3 <<< 2	12	Shifts the bits of *n* left *p* positions. Zero bits are shifted into the low-order positions.
n>>p	Right shift	5 >> 2	1	Shifts the bits of *n* right *p* positions. If *n* is a 2's complement signed number, the sign bit is shifted into the high-order positions.
n>>>p	Right shift	-4 >>> 28	15	Shifts the bits of *n* right *p* positions. Zeros are shifted into the high-order positions.

Increment and Decrement Operators:

The ++ and the – are java's increment and decrement operators. The increment operator increases its operand by one. The decrement operator decreases its operand by one.

For example, this statement: x = x + 1 can be rewritten like this by use for the increment operator x++; Similarly, this statement x = x –1 is equivalent to x--;

Increment and decrement operators can be applied to all integers and floating point types. These operators are unique in that they can appear both in postfix form (x--, x++) and prefix form (--x, ++x), where they precede the operand.

In postfix form, the previous value is obtained for use in the expression, and then the operand is modified. For example:

x = 42;

y = ++x;

In this case, y is set to 43 as you would expect, because the increment occurs before x is assigned to y. thus, the line y=++x; is the equivalent of these two statements:

x = 42;

y = x++;

The value of x is obtained before the increment operator is executed, so the value of y is 42. Of course, in both cases x is set to 43. Here, the line y = x++; is the equivalent of these two statements:

y =x;

x = x + 1;

Example:

/* Demo increment and decrement operator */

```
class InDec
{
   public static void main (String args[])
   {
        int a = 1,b = 2,c,d;
        c = ++b;
        d = a++;
        c++;
        System.out.println("a = " +  a);
        System.out.println("b= " +  b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
    }
}
```

Assignment Operator:

Assignment operator is the most common operator almost used with all programming languages. It is represented by "=" symbol in Java which is used to assign a value to a variable lying to the left side of the assignment operator. But, If the value already exists in that variable then it will be overwritten by the assignment operator (=). This operator can also be used to assign the references to the objects.

Syntax: <variable> = <expression>;

Conditional Operator:

Java supports another conditional operator that is known as the **ternary** operator **"?:"**

It is basically used as an short hand for simple if..else

boolean expression ? operand1 : operand2;

The "**?:**" operator evaluates an expression which may also be an **operand** and returns **operand1** if the expression is **true**; otherwise returns **operand2,** if the expression is **false**. We can understand this thing with the help of a diagram shown as:

Expression? True value: false value

Example:

```
public class condiop
{
 public static void main(String[] args)
{
  String out;
  int  a = 6, b = 12;
  out = a==b ? "Yes":"No";
  System.out.println("Ans: "+out);
 }
}
```

Special Operators:

A. **Dot operator:** The dot operator (.) is used to access the instance variable and methods of class objects.

x.rno, x.name, x.average()

It is also used to access classes and sub-packages from a package.

B. **Instanceof Operator:** Java provides a run-time operator **instanceof** to compare a **class** and an **instance** of that class. The **instanceof** operator is defined to know about an object's **relationship** with a class. It evaluates to **true,** if the object or array is

an **instance** of the specified type otherwise it returns false. The **instanceof** operator can be used with the arrays and objects. It can't be used with primitive data types and values.

Syntax : object **instanceof** type

Operator precedence and associativity:

The precedence is used to determine how expression involving more than one operator is evaluated. The operator at the higher level of precedence is evaluated first. The operators of the same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as the associatively property of an operator.

Operator	Description	Association	Rank
. **( )** **[ ]**	Member selection Method call Array element reference	Left to right	1
- **++** **--** **!** **~**	Unary minus Increment Decrement Logical not One's complement	Right to left	2
***** / **%**	Multiplication Division Modulo division	Left to right	3
+ **-**	Addition Subtraction	Left to right	4
<< >> >>>	Left shift Right shift Right shift zero fill	Left to right	5
< <= > >=	Less than Less than or equals to Greater than Greater than or equals to	Left to right	6
== **!=**	Equality inequality	Left to right	7
&	Bitwise and	Left to right	8

^	Bitwise Xor		9
|	Bitwise or		10
&&	Logical and		11
||	Logical or		12
?:	Conditional operator		13
=	Assignment operator		14

13.Explain Control statements in JAVA:

A control structure determines the order in which statements are executed.

Decision Making and branching:

Decision making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.

If branching is based on a particular decision then that is known as decision making and branching.

Decision making and branching statement are classified into two types. They are

A. Conditional Statements.

B. Unconditional Statements.

A. Conditional Statements:

A programming language uses control statements to control the flow of execution of program based on certain conditions. There are two types of Conditional Statement.

1. If statement
2. Switch statement

2. Decision making with if statement:

It is basically a two-way decision statement.

Syn: if(test_expression)

statement;

It allows the computer to evaluate the **test_expression** first and then, depending on whether the value of the **test_expression** is true or false, it transfers the control to a particular statement.

Types of if statement:

a. Simple if

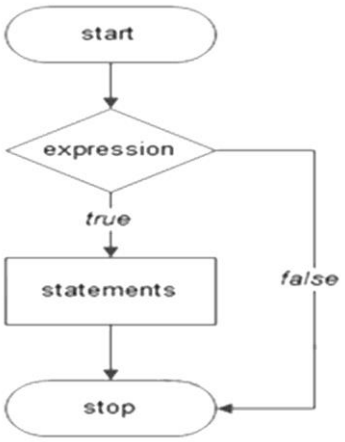
b. if …else

c. Nested if …else

d. Else if ladder

a. Simple if statement:

The `if` statement evaluates the test expression inside the parenthesis. If statement accepts Boolean values.



- If the test expression is evaluated to **true** (nonzero), statement(s) inside the body of `if`is executed.
- If the test expression is evaluated to **false** (0), statement(s) inside the body of `if` is skipped from execution.

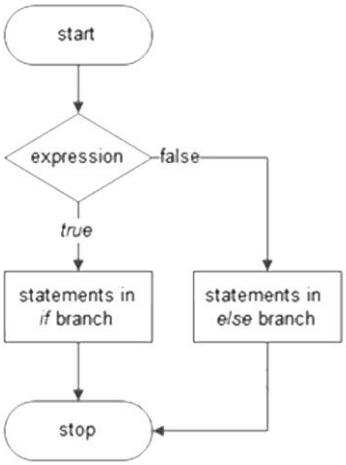
Syn: if(test_expression)

Statement;

Example: java program to demonstrate Simple if.

b. If_ else statement:

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.



Syntax: if (condition)

{

// Executes this block if condition is true

}

else

{

// Executes this block if condition is false }

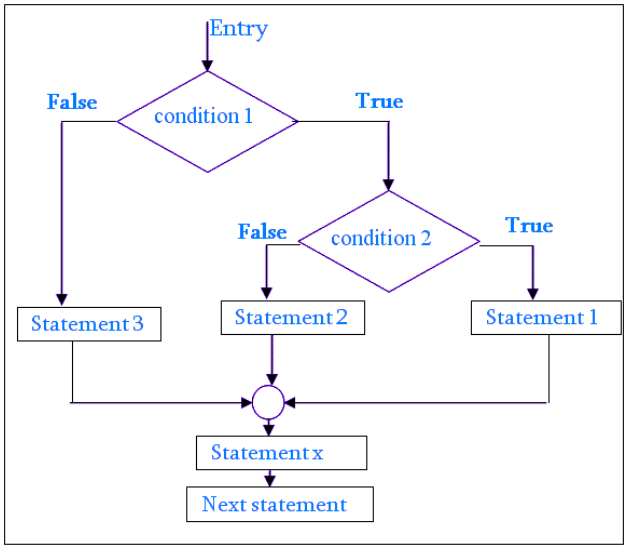
Example: java program to demonstrate if-else

c. Nested If_ else:

It's possible to have `if..else` statements inside a `if..else` statement in Java. It's called nested `if...e`

Syn: if(test_expression1)

```
{
if(test_expression2)
{
if(test_expression3)
        true statement;
else
        false statement;
}
else
        false statement;
}
else
        false statement;
statement x;
```

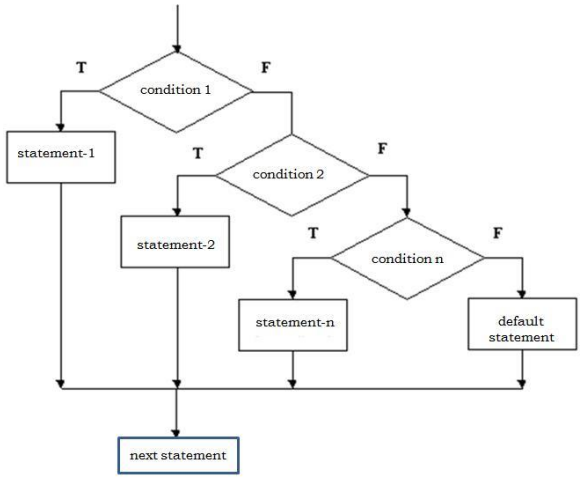


If the **test expression** is **false**, then **statement_3** will be executed; otherwise it continues to perform the second test. If the **test_condition2** is true, then **statement-1** will be executed, otherwise **statement-2** will be executed and then the control is transferred to the **statement-x**

Example: **java program to demonstrate nested if-else**

d. The else_if ladder:

There is another way of putting **if**s together when multi-path decision are involved, A multi-path decision is a chain of **if**s in which the statement associated with each else is an **if.** The general form is




```
Syn:  if(test_expression1)
            statement1;
      else if(test_expression2)
            statement 2;
            ……..
      ……..
      else
            default statement;
```

Whenever the condition is true, the associated statement will be executed and the remaining conditions will be bypassed. If none of the conditions are true then the else block default statements will execute.

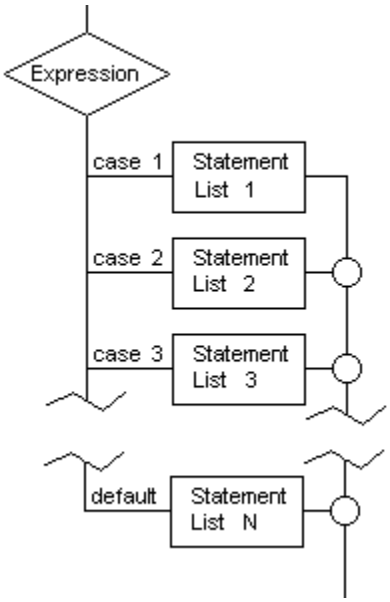
Example: java program to demonstrate else-if-else ladder

Switch statement:

The switch statement is a multi way decision statement. It tests the value from a given list of statements. The general form of the switch statement is as

Syn:

```
switch(variable)
{
case 1:                 //execute your code
        break;
case 2:                 //execute your code
        break;
.....................
case n:                 //execute your code
        break;
default:        //execute your code
}
```



After the end of each block it is necessary to insert a **break** statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each case onwards after matching the case block. If none of the case is true then the default statements will execute.

Example: java program to demonstrate switch case.

Decision Making and Looping

Loop statement:

Loop is a statement, which executes a sequence of statements several times until a condition is satisfied is called iteration. Iteration statements execute the same set of instructions until a termination condition is met. It consists of two segments, one known as the *body of the loop* and the other known as the *control statement*. The *control statement* tests certain conditions and executes the body of the loop.

Depending on the position of the control statement in the loop, the control structure may be classified either as ***entry controlled loop*** or as the ***exit controlled loop***.

In the ***entry controlled loop***, control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed.

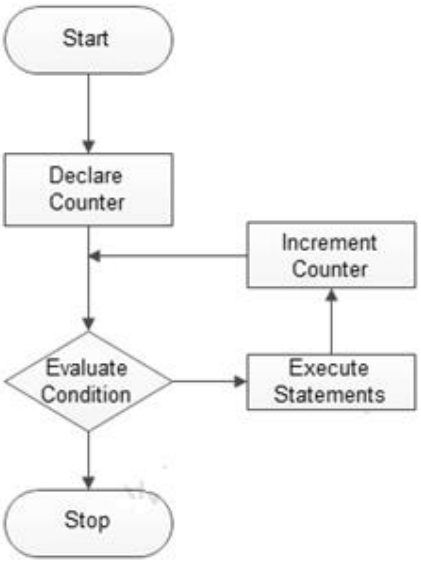
Example: For loop and While loop.

In the ***exit-controlled loop***, the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time.

Example: do while loop

A loop generally includes the following four steps:

1. Initialization.
2. Test_expression.
3. Body of the loop.
4. Increment or decrement



1. For loop: The **for** loop is *entry-controlled loop* that provides a short loop control structure. It is used when the final condition is known.

A for loop executes a statement (that is usually a block) as long as the boolean condition evaluates to true. A for loop is a combination of the three elements initialization statement, boolean expression and increment or decrement statement.

The general form is:

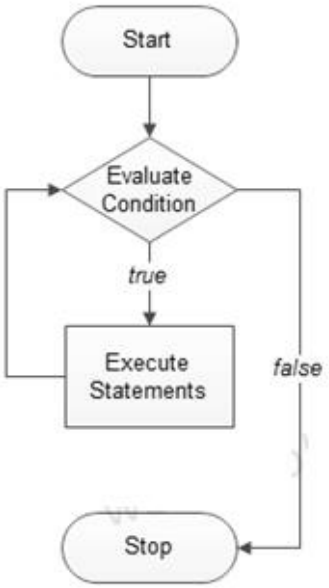
Syntax:

for(<initialization>;<condition>;<increment or decrement statement>)

{

<block of code>

}



The initialization block executes first before the loop starts. It is used to initialize the loop variable. The condition statement evaluates every time prior to when the statement (that is usually be a block) executes, if the condition is true then only the statement (that is usually a block) will execute. The increment or decrement statement executes every time after the statement (that is usually a block).

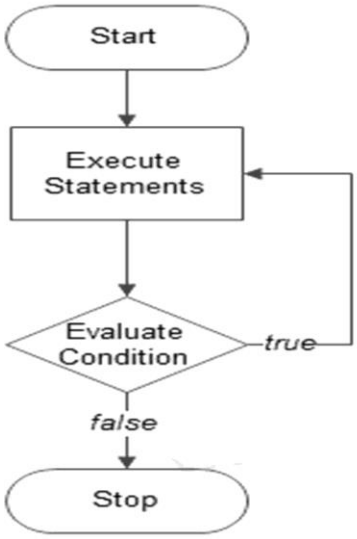
Example: java program to demonstrate For loop.

2. The while loop: The while loop is an entry controlled loop statement. The *test-condition* is tested and if the condition is *true*, then the body of the loop is executed. It can be also used when we don't know the final value.

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

Example: java program to demonstrate while loop.

3. do_while loop: The do while loop is an exit controlled loop statement. It executes all the statements with out testing the condition for the first time. From the second time onwards it executes all the statements after testing the condition as test condition will be placed at the end of the loop.



Syn:
```
initialization;
do
{
statement(s);
increment/ decrement;
}while( condition );
```

Example: java program to demonstrate do-while loop.

14. Explain about Un Conditional Statements (Jumping Statements) in JAVA:

In Java jump statements are mainly used to transfer control to another part of our program depending on the conditions. These statements are very useful from the programmer's view because these statements allow alteration of the flow of execution of the program. These statements can be used to jump directly to other statements, skip a specific statement and so on.

In Java we have the following three jump statements:

1. break (simple and labeled)
2. continue
3. return

The break statement:

If we want to go out of a loop then we use a break statement. If we use a break statement in a loop then execution will continue with the immediate next statement outside the loop. After a break, all the remaining statements in the loop are skipped. The break statement can be used in a while loop, for loop, do-while loop, and in a switch case.

```
Syntax :    if(condition)
                {
              Statements;
              break;
                }
```

Example: java program to demonstrate break statement.

The continue statement:

The continue keyword is used mainly with loops. When we do not want to execute some statements then we use a continue statement to skip those statements to execute. If the continue statement is confronted in the program then it will start the next iteration. It does not terminate the loop, it just skips some part of the loop. The execution again starts from the top of the loop. In some ways it is similar to the break statement.

```
Syntax :    if(condition)
              {          continue;          }
```

Example: java program to demonstrate contiue statement.

The return statement:

The return statement is the last jump statement. The return statement is used to end the execution of a specific method and then return a value. When we use a return statement in our program then it sends the program control to the method caller. The data type of the returned value should always be equal to the data type of the method's declared return value.

```
Syntax :    if(condition)
              {
            return;
}
```

Example: java program to demonstrate return statement.

15. Explain different INPUT and OUTPUT statements in JAVA.

In Java, there are four different ways for reading input from the user in the command line environment (console).

1.Using Buffered Reader Class

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

- The input is buffered for efficient reading.
- The wrapping code is hard to remember.

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test {
    public static void main(String[] args) throws IOException
    {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String name = reader.readLine();
        System.out.println(name);
    }
}
```

Note: To read other types, we use functions like Integer.parseInt(), Double.parseDouble(). To read multiple values, we use split().

2. Using Scanner Class

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.

- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.
- The reading methods are not synchronized

```
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

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

```
    {
        Scanner in = new Scanner(System.in);
        String s = in.nextLine();
        System.out.println("You entered string " + s);
        int a = in.nextInt();
        System.out.println("You entered integer " + a);
        float b = in.nextFloat();
        System.out.println("You entered float " + b);
    }
}
```

3. Using Console Class

It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

Advantages:

- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.
- Does not work in non-interactive environment (such as in an IDE).

```
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample {
    public static void main(String[] args)
    {
      String name = System.console().readLine();
      System.out.println("You entered string " + name);
    }
}
```

4. Using Command line argument

Most used user input for competitive coding. The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into

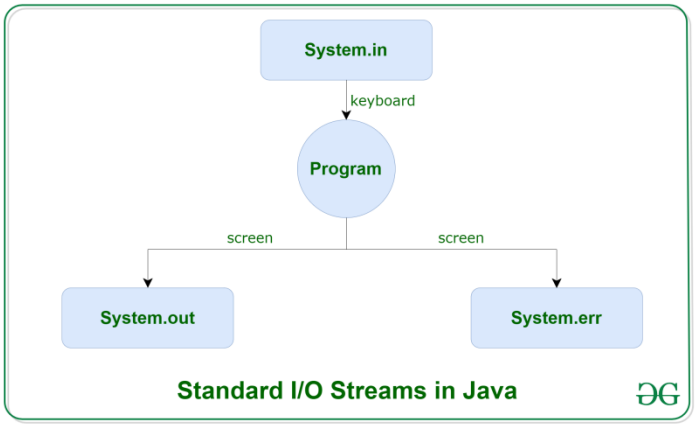
Integer. Similarly, for float and others during execution. The usage of args[] comes into existence in this input form. The passing of information takes place during the program run. The command line is given to args[]. These programs have to be run on cmd.

```
// Program to check for command line arguments
class Hello {
    public static void main(String[] args)
    {
        if (args.length > 0) {
            System.out.println("The command line arguments are:");
            for (String val : args)
                System.out.println(val);
        }
        else
            System.out.println("No command line " + "arguments found.");
    }
}
```

Output statement in Java:-

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.**

Java has to provide **default streams** 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:

- 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.

  Syntax:

  System.out.print(*parameter*);

  Example PROGRAMME:

- 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.

  Syntax:

  System.out.println(*parameter*);

  Example PROGRAMME:

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 above-mentioned functions to output the error data:

- print()
- println()

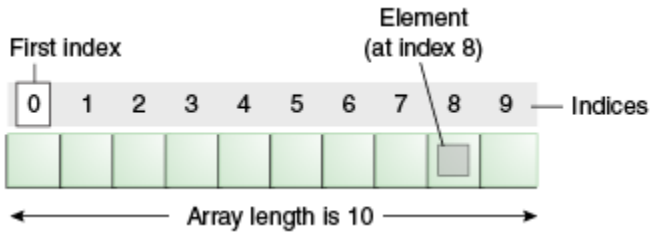
Example:

```
import java.io.*;
public class SimpleIO {
    public static void main(String args[])throws IOException
   {
       InputStreamReader inp = new InputStreamReader(System.in);
       System.out.println("Enter characters, " + " and '0' to quit.");
       char c;
       do {
         c = (char)inp.read();
         System.out.println(c);
       } while (c != '0');        }    }
```

16. Explain Arrays in JAVA:-

An array is a variable that can store multiple values of same data type that have contiguous memory locations where as an ordinary variable can store a single value at a time. 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 0 index.



Advantages:

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages:

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of an Array:

There are three types of array.

A. One or Single Dimensional Array
B. Two or Double Dimensional Array
C. Multi dimensional Array

A. One-Dimensional Array:

An Array in which list of elements are stored in continuous memory locations and accessed using only one subscript.

Declaration of one-dimensional array:

Syn: type array_name[ ];

(or)

type [ ] array_name;

Where, type is the data type of array elements; array_name is name of array variable. Remember, we do not enter the size of the arrays in the declaration.

Creation memory locations:

After declaring an array, we need to create it in the memory. Java allows us to create arrays using new operator only.

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

Example: a=new int[5];

The two statements may be combined as follows:

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

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

The total size of the array 5 X 4 =20 bytes, this is because each integer element requires 4 bytes for storage.

Initialization of one-dimensional array:

Assigning values to an array when it is created is known as initialization.

Syn: type array_name[ ]={list of elements separate by commas};

Ex: int a[ ]={5,8,2,6};

In the above example, four elements are stored in an array 'a'. The array elements are stored in contiguous memory locations. The array elements are read from 0 i.e. a[0] is assigned the value 5. Similarly a[1] is assigned the value 8, a[2] is 2 and a[3] is 6.

a[0]	a[1]	a[2]	a[3]
5	8	2	6

Subscripts can also be reflected in for loops that the array elements.

```
for (i=0;i<r;i++)
        {
                System.out.print(" "+a[i][j]);
        }
```

Array length: In java, we can obtain the length of the array **a** using **a.length**.

Example:
```
int n = a.length;
for (i=0;i<=n;i++)
   System.out.print(" "+a[i])
```

Example: Java Program to demonstrate - One Dimensional Array Program

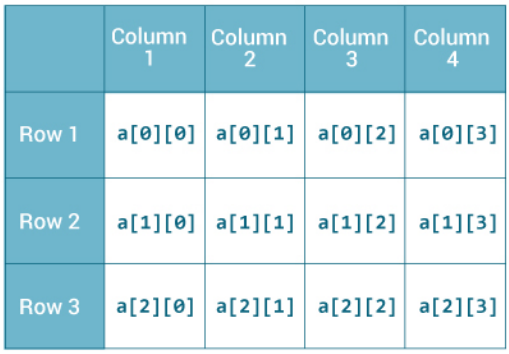
B. Two-Dimensional Array:

An Array in which list of elements are stored in some rows and columns and accessed using two subscripts.

Declaration of two-dimensional array:

Syn: type array_name [ ][ ];

Where, type is the data type of array elements; array_name is name of array variable. Remember, we do not enter the size of the arrays in the declaration.



Creation memory locations:

After declaring an array, we need to create it in the memory. Java allows us to create arrays using new operator only.

Syn: array_name= **new** type [row_size][col_size];

Example: a=new int a[5][3];

The two statements may be combined as follows:

Syn: type array_name[ ][ ] = new type[row_size][col_size];

Ex: int a[][]=new int[5][3];

The total size of the array 15 X 4 =60 bytes, this is because each integer element requires 4 bytes for storage.

Initialization of two-dimensional array:

Assigning values to an array created is known as initialization.

Syn: array_name [subscript1][subscript2]=value;

(or)

type array_name[ ][ ]={{list of elements in row},{list of elements in column}};

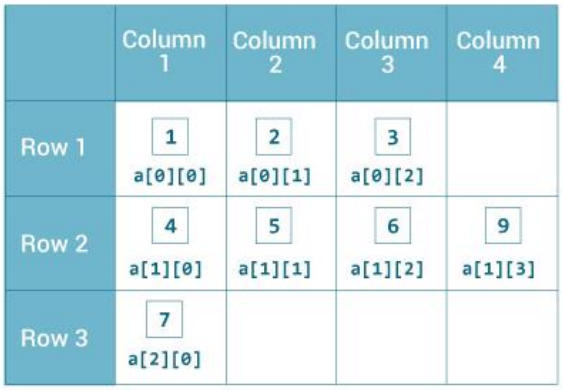
Example1: a[0][0]=5; a[0][1]=8;

int a[ ][ ]={{5,-7,2,1},{9,8,5,6},{-1,8,-3,4}};

In the above example, twelve elements are stored in an array 'a'. The array elements are stored in contiguous memory locations as rows and column format. The array elements are read from (0,0) i.e. a[0][0] is assigned the value 5. Similarly a[0][1] is assigned the value -7 and so on.

Subscripts can also be reflected in for loops that the array elements.

```
for (i=0;i<r;i++)        {
  for(j=0;j<c;j++)       {

System.out.print(" "+a[i][j]);
```



}

}

Example: Java Program to demonstrate - two Dimensional Array Program

C. Multi-Dimensional Array:

An Array with more than two dimensions are called multidimensional array. To create multidimensional array add as many pair of subscripts to array variable.

Syn: array_name [subscript1][subscript2] [subscript2].....;

Ex: int a[ ][ ][].....;

Each component of array a is an array in itself, and length of each rows is also different.

Example: Java Program to demonstrate - Multi Dimensional Array Program

Strings: Creating Strings, String Class Methods, String Comparison, Immutability of Strings Introduction to OOPs: Problems in Procedure Oriented Approach, Features of ObjectOriented 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 Inheritance: Inheritance, The keyword „super", The Protected Specifier, Types of Inheritance

17. Explain about Strings in JAVA.

String manipulation is the most common part of many java programs. Strings represent a sequence of characters.

In Java, strings are class object and implemented using two classes namely String and StringBuffer. A Java string is an instantiated of the String class. A Java string is not a character array and is not terminated with NULL.

Syntax: String string_name;

string_name = new String("string");

Example: String a;

a = new String("JAVA");

The two statements may be combined as follows:

String a = new String("JAVA");

It is possible to get the length of string using the length method of the String class.

int m = a. length()

String c = a + b;

String Arrays: We can also create and use arrays that contain strings

String a[ ]=new String[3];

Will create an array of size 3 to hold three string constants.

String Handling Methods:

The string class defines a number of methods that allow us to accomplish a variety of string manipulation tasks.

A String in Java is actually an object, which contain methods that can perform certain operations on strings.

1. **length()**: The length() method tells the length of the string. It returns count of total number of characters present in the String.

   Ex: String s2="whatsup";

```
System.out.println("string length is: "+s1.length());
```

Here, String length() function will return the length 5 for s1 and 7 for s2 respectively.

2. **compareTo():** The compareTo() method compares the given string with current string. It either returns positive number, negative number or 0.

   It is noticed that

   if s1 > s2, it returns a positive number

   if s1 < s2, it returns a negative number

   if s1 == s2, it returns 0

   Ex:
   ```
   String s1="hello";
   String s2="hello";
   String s3="hemlo";
   String s4="flag";
   System.out.println(s1.compareTo(s2)); // 0 because both are equal
   System.out.println(s1.compareTo(s3)); //-1 because "l" is only one time lower than "m"
   System.out.println(s1.compareTo(s4)); // 2 because "h" is 2 times greater than "f"
   ```

3. **concat() :** The concat() method combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string.

   Ex:
   ```
   String s1="hello";
   s1=s1.concat("how are you");
   System.out.println(s1);
   ```

   Output: The above code returns “hellohow are you”.

4. **toLowerCase()** : The toLowerCase() method converts all the characters of the String to lower case.

   Ex:
   ```
   String s1="HELLO HOW Are You?";
   String s1lower=s1.toLowerCase();
   System.out.println(s1lower);
   ```

   Output: The above code will return “hello how are you”.

5. **toUpper()** : The Java String toUpperCase() method converts all the characters of the String to upper case.

   Ex:
   ```
   String s1="hello how are you";
   String s1upper=s1.toUpperCase();
   System.out.println(s1upper);
   ```

   Output: The above code will return “HELLO HOW ARE YOU”.

6. **equals()** : The equals() method compares the two given strings on the basis of content of the string. If all the characters are matched, it returns true else it will return false.

   Ex:
   ```
   String s1="hello";
   String s2="hello";
   String s3="HELLO";
   System.out.println(s1.equals (s2));   // returns true
   System.out.println(s1.equals (s3));   // returns false
   ```

7. **equalsIgnoreCase():** This method compares two string on the basis of content but it does not check the case like equals() method. In this method, if the characters match, it returns true else false.

   Ex:
   ```
   String s1="hello";
   String s2="HELLO";
   String s3="hi";
   System.out.println(s1.equalsIgnoreCase(s2));   // returns true
   System.out.println(s1.equalsIgnoreCase(s3));   // returns false
   ```

8. **replace()** This method returns a string, replacing all the old characters or Char Sequence to new characters.

   Ex:
   ```
   String s1="hello how are you";
   String replaceString=s1.replace('h','t');
   System.out.println(replaceString);
   ```

   In the above code, it will replace all the occurrences of 'h' to 't'. Output to the above code will be "tello tow are you".

9. **charAt():** This method returns the character located at the String's specified index. The string indexes start from zero.

   Ex:
   ```
   String s = "Strings are immutable";
   char result = s.charAt(8);
   System.out.println(result);
   ```

   This will produce the following result:

   Output: a

10. **indexOf():** This method returns the index within this string of the first occurrence of the specified character or -1 if the character does not occur.

    Ex:
    ```
    String Str = new String("Welcome to the java programming");
    System.out.println(Str.indexOf( 'o' ));
    ```

    Output: Found Index :4

Method	Task performed
s2=s1.toLowerCase;	Converts the string s1 to all lowercase
s2=s1.toUpperCase;	Converts the string s1 to all uppercase
s2=s1.replace('x','y');	Replace all appearances of x with y
s2=s1.trim();	Remove white space s at the beginning and end of the string s1.
s1.equals(s2);	Return 'true' if s1 is equals to s2
s1.equalsIgnoreCase(s2);	Return 'true' if s1=s2, ignoring the case of characters
s1.lenght();	Gives the length of s1
s1.CharAt(n);	Gives nth character of s1
s1.comparTo(s2)	Return – if s1<s2 , + if s1>s2 and 0 if s1 =s2
s1.concat(s2);	Concatenates s1 and s2
s1.substring(n);	Gives substring starting from nth character
s1.substring(n,m);	Gives substring starting from nth character up to mth character.
s1.indexOf('x');	Gives the position of the first occurrence of 'x' in the string s1
s1.indexOf('x',n);	Gives the position of the 'x' that occurs after nth position in the string s1

18. What is Stringbuffer? Explain different stringbuffer handling methods in java.

StringBuffer in java is used to create modifiable String objects. 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 means that we can use StringBuffer to append, reverse, replace, concatenate and manipulate Strings or sequence of characters.

StringBuffer defines 3 constructors. They are:

a. StringBuffer():

Creates a StringBuffer with empty content and 16 reserved characters by default.

```
StringBuffer sb = new StringBuffer();
```

b. StringBuffer(int sizeOfBuffer):

Creates a StringBuffer with the passed argument as the size of the empty buffer.

```
StringBuffer sb = new StringBuffer(20);
```

c. StringBuffer(String string):

Creates a StringBuffer with the passed String as the initial content of the buffer. 16 contingent memory characters are pre-allocated, not including the buffer, for modification purposes.

```
StringBuffer sb = new StringBuffer("Hello World!");
```

String Buffer Handling Methods:

A StringBuffer in Java is actually an object, which contain methods that can perform certain operations on strings.

The following methods are some most commonly used methods of StringBuffer class.

1. **length():** This method returns the StringBuffer object's length.

   Ex:
   ```
   StringBuffer sb = new StringBuffer("Hello");
   int sbLength = sb.length();
   System.out.println("String Length of " + sb + " is " + sbLength);
   ```
   Output: String Length of Hello is 5

2. **capacity():** This method returns the capacity of the StringBuffer object.

   Ex:
   ```
   StringBuffer sb = new StringBuffer("Hello");
   int sbCapacity = sb.capacity();
   System.out.println("Capacity of " + sb + " is " + sbCapacity);
   ```
   Output: Capacity of Hello is 21

3. **append():** This method appends the specified argument string representation at the end of the existing String Buffer.

   Ex:
   ```
   StringBuffer sb = new StringBuffer("Hello ");
   sb.append("World ");
   sb.append(2017);
   System.out.println(sb);
   ```
   Output: Hello World 2017

4. **insert():** This method takes two parameters – the index integer value to insert a value and the value to be inserted. The index tells String Buffer where to insert the passed character sequence.

   Ex:
   ```
   StringBuffer sb = new StringBuffer("HelloWorld ");
   sb.insert(5, 2017);
   System.out.println(sb);
   ```
   Output: Hello World 2017

5. **s1.charAt(int index)**

   The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.

6. **s1.setCharAt(int index, char ch)**

   The character at the specified index of this string buffer is set to ch

7. **s1.toString()**

Converts to a string representing the data in this string buffer

8. **s1.deleteCharAt(int index)**

   Removes the characters at the specific index of the StringBuffer

9. **s1.replace(int start, int end, String str)**

   Replaces the characters in a substring of this StringBuffer with characters in the specified String.

10. **s1.setLength(int newLength)**

    Sets the length of this String buffer.

11. **reverse():** This method reverses the existing String or character sequence content in the buffer and returns it.

    Ex: StringBuffer sb = new StringBuffer("Hello World");

    System.out.println(sb.reverse());

    Output: dlroW olleH

Example: Java program by using vector class methods.

19. Explain the Difference between String class & StringBuffer class in JAVA:

- String objects are constants and immutable where as StringBuffer objects are not.
- StringBuffer Class supports growable and modifiable string where as String class supports constant strings.
- Strings once created we cannot modify them. Any such attempt will lead to the creation of new strings.Where as StingBuffer objects after creation also can be able to delete or append any characteres to it.
- String values are resolved at run time where as StringBuffer values are resolved at compile time.

20. What is Object Oriented Programming System? Explain its benefits.

Object-Oriented Programming (OOP) is the term used to describe a programming approach based on objects and classes. The object-oriented paradigm allows us to organise software as a collection of objects that consist of both data and behaviour. This is in contrast to conventional functional programming practice that only loosely connects data and behaviour.

Since the 1980s the word 'object' has appeared in relation to programming languages, with almost all languages developed since 1990 having object-oriented features. It is widely accepted that object-oriented programming is the most important and powerful way of creating software.

Benefits of OOP:

OOP offers several benefits to the program designer and the user. Object-orientation contributes to the solutions of many problem associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The benefits are:

- It is easy to model a real system as real objects are represented by programming objects in OOP. The objects are processed by their member data and functions. It is easy to analyze the user requirements.
- With the help of inheritance, we can reuse the existing class to derive a new class such that the redundant code is eliminated and the use of existing class is extended. This saves time and cost of program.
- In OOP, data can be made private to a class such that only member functions of the class can access the data.
- This principle of data hiding helps the programmer to build a secure program that cannot be invaded by code in other part of the program.
- With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
- Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
- It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.
- It is easy to partition the work in a project based on objects.
- Object-oriented systems can be easily upgraded from small to large system
- Message passing technique for communication between objects makes the interface descriptions with external system much simpler.

21. Explain the Applications of Object Oriented Programming Languages.

Main application areas of OOP are:

A. Client-Server Systems

Object-oriented Client-Server Systems provide the IT infrastructure, creating object-oriented Client-Server Internet (OCSI) applications.

B. Object Oriented Databases:

They are also called Object Database Management Systems (ODBMS). These databases store objects instead of data, such as real numbers and integers

C. Real-Time System Design:

Real time systems inherit complexities that makes difficult to build them. Object-oriented techniques make it easier to handle those complexities.

D. Simulation and Modeling System:

It's difficult to model complex systems due to the varying specification of variables. These are prevalent in medicine and in other areas of natural science, such as ecology, zoology, and agronomic systems

E. Hypertext and Hypermedia:

OOP also helps in laying out a framework for Hypertext. Basically, hypertext is similar to regular text as it can be stored, searched, and edited easily.

F. Neural Networking and Parallel Programming:

It addresses the problem of prediction and approximation of complex time-varying systems. Firstly, the entire time-varying process is split into several time intervals or slots.

G. Office Automation Systems:

These include formal as well as informal electronic systems primarily concerned with information sharing and communication to and from people inside as well as outside the organization. Like Email ,Word processing etc..

H. CIM/CAD/CAM Systems:

OOP can also be used in manufacturing and design applications as it allows people to reduce the effort involved.

I. AI Expert Systems:

These are computer applications which are developed to solve complex problems pertaining to a specific domain, which is at a level far beyond the reach of a human brain.

21. Explain the Basic concepts of OOPs.

Object Oriented Programming (OOP) is a programming model where programs are organized around objects and data rather than action and logic.

OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

The software is divided into a number of small units called objects. The data and functions are built around these objects.

The data of the objects can be accessed only by the functions associated with that object.The functions of one object can access the functions of another object.

Java supports the following fundamental concepts −

A. Class

B. Objects
C. Abstraction
D. Encapsulation
E. Polymorphism
F. Inheritance
G. Message Passing
H. Dynamic binding
I. Data hiding

A. Class:

In OOP languages it is mandatory to create a class for representing data.

A class is a blueprint from which individual objects are created. A class contains variables for storing data and functions to perform operations on the data. A class will not occupy any memory space and hence it is only a logical representation of data. To create a class, you simply use the keyword "class" followed by the class name.

For example, if you had a class called “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc. Its properties (data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.

B. Object:

Objects are the basic run-time entities of an object oriented system. They may represent a person, a place or any item that the program must handle.

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object contains both the data and the function, which operates on the data.

Objects have states, behaviours and identifier.

Example: A dog has states - color, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

A class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class that is called an object.

When an object is created using the new operator, memory is allocated for the class in the heap, the object is called an instance and its starting address will be stored in the object in stack memory.

When an object is created without the new operator, memory will not be allocated in the heap, in other words an instance will not be created and the object in the stack contains the value **null**.

When an object contains null, then it is not possible to access the members of the class using that object.

C. Abstraction:

An abstraction is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. Abstraction lets you focus on what the object does instead of how it does it. Abstraction provides you a generalized view of your classes or objects by providing relevant information. Abstraction is the process of hiding the working style of an object, and showing the information of an object in an understandable manner.

For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

D. Encapsulation

Wrapping up a data member and a method together into a single unit (in other words class) is called Encapsulation.

Encapsulation means hiding the internal details of an object, in other words how an object does something. It prevents clients from seeing it's inside view, where the behaviour of the abstraction is implemented. It is a technique used to protect the information in an object from another object. Hide the data for security such as making the variables private, and expose the property to access the private data that will be public.

So, when you access the property you can validate the data and set it.

Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object. Encapsulation is like your bag in which you can keep your pen, book etcetera. It means this is the property of encapsulating members and functions.

```
class Bag
{
book;
 pen;
ReadBook();
}
```

E. Polymorphism

Polymorphism means one name, many forms. One function behaves in different forms. Polymorphism refers to the ability of a variable, object or function to take on multiple forms.

F. Inheritance

Inheritance is a mechanism in which one class acquires the property of another class. It's creating a parent-child relationship between two classes. With inheritance, we can reuse the fields and

methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.

- In Java, when an "Is-A" relationship exists between two classes we use Inheritance
- The parent class is termed super class and the inherited class is the sub class
- The keyword "extend" is used by the sub class to inherit the features of super class

Inheritance is important since it leads to reusability of code

G. Dynamic Binding:

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of execution.

H. Message passing:

An object oriented program consists of a set of objects that communicate with each other. The process of programming in an object oriented language is:

- Create classes that define objects.
- Creating objects from class definitions.
- Establishing communication among objects.

22. Explain the Difference between Procedural Oriented Programming and Object-Oriented Programming.

Procedural Oriented Programming	**Object-Oriented Programming**
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.

Procedural Oriented Programming	Object-Oriented Programming
In procedural programming, there is no concept of data hiding and inheritance.	In object-oriented programming, the concept of data hiding and inheritance is used.
In procedural programming, the function is more important than the data.	In object-oriented programming, data is more important than function.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Procedural programming uses the concept of procedure abstraction.	Object-oriented programming uses the concept of data abstraction.
Code reusability absent in procedural programming,	Code reusability present in object-oriented programming.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

23. What is class? Explain the building blocks of a class.

A class is user defined data type with A a blueprint or prototype that defines the variables and the methods (functions) common to all objects of a certain kind. If once the class is defined then we can create any number of objects of that type. These objects are called as instances of classes. A class binds the data and its associated methods together and hides them.

Syntax:
```
class class_name
{
field declarations;
method declarations;
}
```

Ex:
```
class student
{
int x, y;
void getdata(int a, int b)
{
        x = a;
        y = b;                  }                   }
```

Java Class Building Blocks:

A Java class can contain the following building blocks:

1. Fields
2. Constructors
3. Methods
4. Nested Classes

1. **Fields** are variables (data) that are local to the class, or instances (objects) of that class.
2. **Constructors** are methods that initialize an instance of the class. Constructors often sets the values of fields in the given instance
3. **Methods** are operations that the class or instances of that class can perform. For instance, a method may perform an operation on input parameters, or change the value of fields kept internally in the object etc.
4. **Nested classes** are Java classes that are defined inside another class.

Not all Java classes have fields, constructors and methods. Sometimes you have classes that only contain fields (data), and sometimes you have classes that only contain methods (operations). It depends on what the Java class is supposed to do.

Fields declaration:

It is a place where we can declare the type and scope of the variables that are to be used in the program. These variables are called instance variables because they are created whenever and object of the class in instantiated.

Syntax: data_type variable1, variable2, …. , variable n;

Ex: int l, b;

Methods declaration:

Methods are used to manipulate the data that is present in the class. Methods are declared inside the body of the class but immediately after the declaration of variables.

Syntax:
```
type method_name(arguments list)
{
body of the method;          }
```

Ex:
```
void getdata(int a, int b)
{
        x = a;
        y = b;
}
```

Here The **type** specifies the type of value that the method would return. The **method name** is a valid identifier. The **argument list** contains the variables and their types. The **body of the method** describes the operations to be performed on the data.

24. What is Object? How to create objects in JAVA.

A class provides the blueprints for objects. So basically, an object is created from a class. The declaration of an object is similar to the declaration of a variable. But to declare a variable we need primitive data types like int or float. To declare an object we a non-primitive data types such as classes or interfaces.

Creating an object is also called as instantiating an object. In java objects are created using a new operator.

There are three steps when creating an object from a class −

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Syntax: class_name object_name;
Object_name = new default constructor();

Ex: student s; // declare the object
s = new student(); // instantiate the object

Here The first statement declares a variable to hold the object's address. The statement assigns the object's address to the variable.

Accessing class members:

Each object containing its own set of variables. All variables must be assigned values before they are used. We cannot access the instance variable and the method directly, when we are in outside the class. Instance variables and methods are accessed via created objects with the **.** operator.

Accessing variables of a class:

Syntax: object_name.variable

Ex: s. a =3;

Accessing methods of a class:

Syntax: object_name.method_name(parameter list);

Ex: s. getdata(2, 3);

Memory allocation for objects:

Memory space for objects is allocated when they are declared and not when the class is specified. The member functions are created and placed in the memory only once when they are defined as a part of a class.

Example1: Java Program to illustrate the use of Rectangle class which has length and width data members.

Example2: Java Program to demonstrate the working of a banking-system where we deposit and withdraw amount from our account.

25. What is constructor? Explain different types of Constructors in Java:

In Java, A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. It is called when an instance of the object is created, and memory is allocated for the object. It is a special type of method which is used to initialize the object.

It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Every time an object is created using new() keyword, at least one constructor is called. It calls a default constructor.

Rules for creating Java constructor:

1. Constructor name must be the same as its class name.
2. A Constructor must have no explicit return type.
3. A Java constructor cannot be abstract, static, final, and synchronized.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default constructor:

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type. you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

```
Ex:     Square()
        {
            ---------
            ---------
        }
```

Example: java program to demonstrate constructor statement.

Default constructor.java

Parameterized Constructor:

A constructor which has a specific number of parameters is called a parameterized constructor. The parameterized constructor is used to provide different values to the distinct objects. However, you can provide the same values also.

Ex: Square(int x, int y)

{ --------- -------- }

Ex: Program for demonstrating the concept of Parameterised constructor.

```
public class MyClass
{
int x;
public MyClass(int y)
{
x = y;
}
}
Class ParameterisedEx
{
public static void main(String[] args)
{
        MyClass myObj = new MyClass(5);
        System.out.println(myObj.x);
}
}
```

26. Explain the Static Keyword in JAVA.

Static keyword can be used with class, variable, method and block. Static members belong to the class instead of a specific instance, this means if you make a member static, you can access it without object. Static members are common for all the instances(objects) of the class.

The static can be used as:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block

4. Nested class

1) Java static variable:

If you declare any variable as static, it is known as a static variable. The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

The static variable gets memory only once in the class area at the time of class loading.

Example: Java Program to demonstrate the use of static variable

2) Java static method:

If you apply static keyword with any method, it is known as static method. A static method belongs to the class rather than the object of a class. A static method can be invoked without the need for creating an instance of a class. A static method can access static data member and can change the value of it.

Example: Java Program to demonstrate the use of a static method.

Restrictions for the static method:

There are two main restrictions for the static method. They are:

1. The static method cannot use non static data member or call non-static method directly.
2. This and super cannot be used in static context.

27. Explain Nesting of Methods in JAVA.

When a method in java calls another method in the same class, it is called Nesting of methods.

Enter length, breadth and height as input. After that we first call the volume method. From volume method we call area method and from area method we call perimeter method. Hence we get perimeter, area and volume of cuboids as output.

Here is the source code of the Java Program to Show the Nesting of Methods. The Java program is successfully compiled and run on a Windows system.

Example: java program to demonstrate nested methods statement.

A nested method must have access to the main method declarations, as well as what can access the main method. A nested method can be called by the code in the main method that follows the declaration of the nested method, or by other nested methods of the main method. The principle is recursive: A nested method can itself have nested methods.

28. What is Inheritance? Explain its types.

The mechanism of deriving a new class from an old class is called inheritance. The old class is referred to as the **base class** and the new class is called the **derived class**. This concept supports the reusability feature of java.

The derived class inherits some or all the properties from the base class. A class can also inherit properties from more than one class or more than one level.

The keyword **extends** indicates that the properties of the super class are extended to the subclass. Now the subclass will contain both subclass and super class properties in it. The meaning of extends is to increase the functionality.

Defining super class:

The class whose properties are used by another class is known as super or base or parent class. A subclass is defined as follows:

Syn: **class** super class name

{

data members;

member functions;

}

Defining a subclass:

The class that extends the features of super class is known as sub or derived or child class. A subclass is defined as follows:

Syn: **class** subclass name **extends** superclass name

{

data members;

member functions;

}

A sub class can access only public or protected members of the super class.

Example:

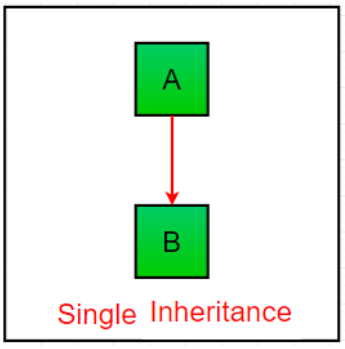
Types of inheritance in java:

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.

A. Single Inheritance:

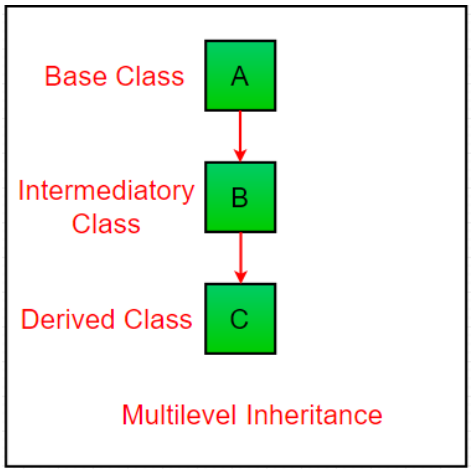
In single inheritance, subclass inherits the features of one super class. In image below, the class A serves as a base class for the derived class B.



Example: **java program to demonstrate single inheritance.**

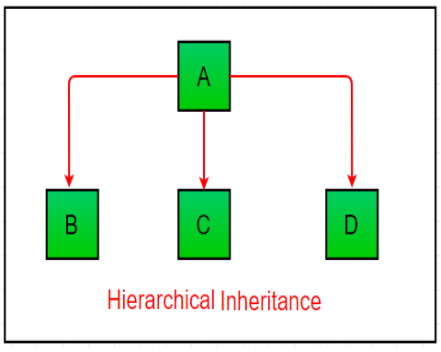
B. Multi Level Inheritance:

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class(called intermediate base class) for the derived class C.



Example: java program to demonstrate

```
class Animal
{
void eat(){System.out.println("eating...");
}
}
class Dog extends Animal
{
void bark(){System.out.println("barking...");
}
}
class BabyDog extends Dog
{
void weep()
{
System.out.println("weeping...");
}
}
class MultilevelInheritance
{
public static void main(String args[])
{
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}
}
```



C. Hierarchical inheritance:

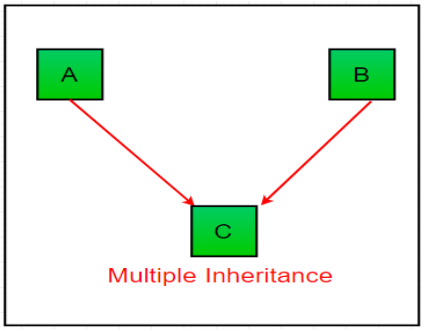
In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.

Example: java program to demonstrate Hierarchical inheritance

D. Multiple Inheritance:

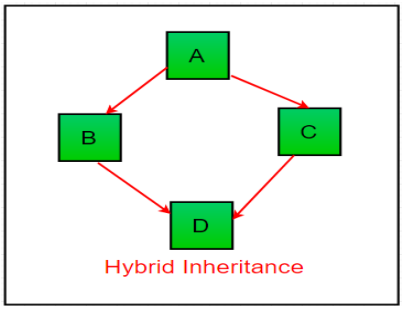
Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class.

Please note that Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B.



E. Hybrid Inheritance(Through Interfaces) :

It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



29. Why multiple inheritance is not supported in java?

The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

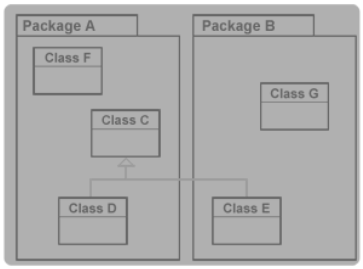
Example: java program to demonstrate multiple inheritance

```
class A
{
void msg(){System.out.println("Hello");
}
}
class B
```

```
{
void msg(){System.out.println("Welcome");
}
}
class C extends A,B
{
//suppose if it were
public static void main(String args[])
{
C obj=new C();
obj.msg();              //Now which msg() method would be invoked? }}
```

30. **Explain different Visibility Controls available in JAVA.**

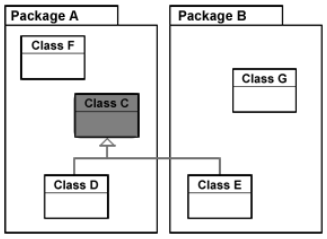
How a member can be accessed is determined by the access specifier that modifies its declaraion. Java supplies a rich set of access specifiers. Some acpects of access control are related mostly to inheritance or packages.



Java's access specifiers are public, private and protected. Java also defines a default access level.

Public:

When a member of a class is modified by the public specifier then that member can be accesed by any other code in your program.

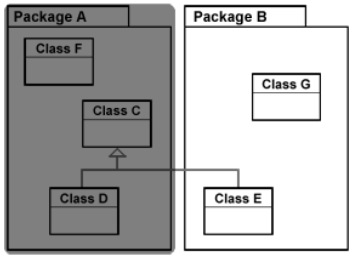


private

When a member of a class is specified as private then that member can only be accessed by other members of its class.

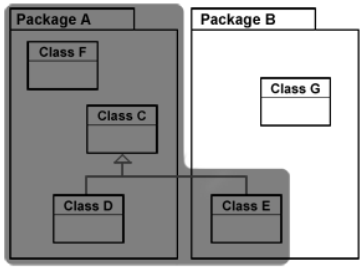
Default access specifier

When no access specifier is used then by default the member of a class is public within it's own package but cannot be accessed outside of its package.



Protected:

When a member of a class is specified as protected it is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature.

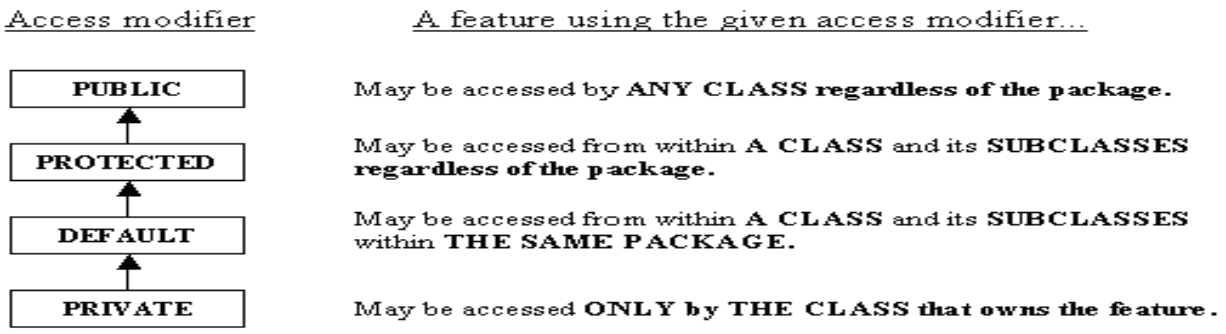


This access is provided even to subclasses that reside in a

different package from the class that owns the protected feature.

The following table lists the visibility modifiers and their visibility areas.

	private	No Modifier	protected	public
Same class	Yes	Yes	Yes	Yes
Same package sub class	No	Yes	Yes	Yes
Same package Non-sub class	No	Yes	Yes	Yes
Different package sub class	No	No	Yes	Yes
Different package Non-sub class	No	No	No	Yes

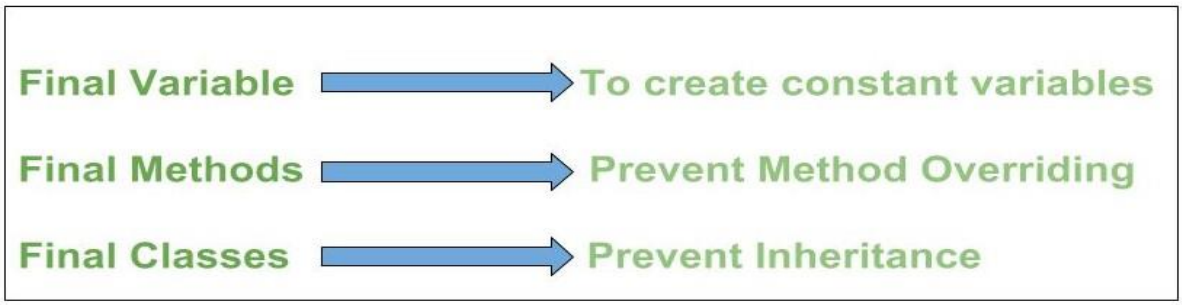


31. Explain about Final Keyword in Java.

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

1. variable
2. method
3. class

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



1) Java final variable:

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

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

Example: java program to demonstrate final variable

2) Java final method: When a method is declared with *final* keyword, it is called a final method. A final method cannot be overridden. The Object class does this—a number of its methods are final. We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes. The following fragment illustrates final keyword with a method:

Example: java program to demonstrate final method

If you make any method as final, you cannot override it.

3) **Java final class:** When a class is declared with *final* keyword, it is called a final class. A final class cannot be extended(inherited).

There are two uses of a final class :

1. One is definitely to prevent inheritance, as final classes cannot be extended.

   For example, all Wrapper Classes like Integer,Float etc. are final classes. We cannot extend them.

```
final class A
{
    // methods and fields
}
class B extends A                    // The following class is illegal.
{
   // COMPILE-ERROR! Can't subclass A
}
```

2. The other use of final with classes is to create an immutable class like the predefined String class. You cannot make a class immutable without making it final.

   Immutable class means that once an object is created, we cannot change its content. In Java, all the wrapper classes (like String, Boolean, Byte, Short) and String class is immutable.

Example: java program to demonstrate final class

If you make any class as final, you cannot extend it.

UNIT – III

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 Data Types, 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 Exception Handling: Errors in Java Program, Exceptions, throws Clause, throw Clause, Types of Exceptions, Re – throwing an Exception.

---

32. What is Polymorphism?

One of the important concepts in Object Oriented Programming (OOP) is polymorphism which means that a single action can be performed in different ways. It is derived from the Greek words: poly and morphs meaning many and forms. The different forms exist when they are related through inheritance.

Polymorphism in Java creates a single method render() that behaves according to different shapes. A real-life example of polymorphism would be a woman who is a Vice President of a company, daughter, mother, sister, or wife.

33. Explain Method Overloading in Java.

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different as number of input parameters or type of input parameters or order of input parameters. If we have to perform only one operation, having same name of the methods increases the readability of the program. Overloading is related to compile time (or static) polymorphism.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Three ways to overload a method:

In order to overload a method, the argument lists of the methods must differ in either of these:

1. Number of parameters.

Example: add(int, int)

add(int, int, int)

2. Data type of parameters.

Example: add(int, int)

add(int, float)

3. Sequence of Data type of parameters.

Example: add(int, float)

add(float, int)

Invalid case of method overloading:

If two methods have same name, same parameters and have different return type, then this is not a valid method overloading. This will throw compilation error.

int add(int, int)

float add(int, int)

Points to Note:

1. Static Polymorphism is also known as compile time binding or early binding.

2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

Example 1: Overloading – Different Number of parameters in argument list

Example 2: Overloading – Difference in data type of parameters

Example3: Overloading – Sequence of data type of arguments

Example: Java program to demonstrate working of method overloading in Java.

```
public class Sum
{
public int sum(int x, int y)        {    return (x + y);  }
public int sum(int x, int y, int z)   {  return (x + y + z);  }
public double sum(double x, double y)          {  return (x + y); }
}
class MethodOverloadEx
{
 public static void main(String args[])
{
Sum s = new Sum();
System.out.println(s.sum(10, 20));
System.out.println(s.sum(10, 20, 30));
```

```
System.out.println(s.sum(10.5, 20.5));      }     }
```

35. Explain Method overriding in JAVA.

Defining a method in the sub class that has the same name, same arguments and same return type as a method in the super class. When method is called, the method defined in the subclass is invoked and executed instead of the one in the super class. This is known as overriding.

Usage of Java Method Overriding:

- Method overriding is used to provide the specific implementation of a method which is already provided by its super class.
- Method overriding is used for runtime polymorphism.

Rules for Java Method Overriding:

- The method must have the same name as in the parent class.
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Example: Program for implementing Method Overriding.

```
class Vehicle
{
void run()                          //defining a method
{
System.out.println("Vehicle is running");
}
}
class Bike2 extends Vehicle         //Creating a child class
{
void run()                          //defining the same method as in the parent class
{
System.out.println("Bike is running safely");
}
public static void main(String args[])
{
Bike2 obj = new Bike2();            //creating object
obj.run();                          //calling method      }     }
```

36. Explain the Difference between method overloading and method overriding in java.

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2)	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3)	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4)	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5)	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

	Ex:
	class OverloadingExample
	{
	static int add(**int** a,**int** b)
	{
	return a+b;
	}
	static int add(**int** a,**int** b,**int** c)
	{
	return a+b+c;
	}
	}
		Ex:
		class Animal
		{
		void eat()
		{
		System.out.println("eating...");
		}
		}
		class Dog **extends** Animal
		{
		void eat()
		{
		System.out.println("eating bread...");}}

37. Explain the Types of Polymorphism in Java

Object-Oriented Programming focuses on four basic concepts i.e. abstraction, encapsulation, inheritance, and polymorphism. Polymorphism is the ability to process objects differently on the basis of their class and data types. As discussed, that polymorphism is of different types, for it to cover different types of data. Nd polymorphism has properties and methods.

There are two types of **polymorphism in Java**: **compile time polymorphism** and **run time polymorphism in java**. This **java polymorphism** is also referred to as static polymorphisms and dynamic polymorphisms.

1. Static polymorphism (or compile-time polymorphism)

Like most of the other OOP programming languages, Java polymorphism allows the incorporation of multiple methods within a class. The methods use the same name but the parameter varies. This represents the static polymorphism. This polymorphism is resolved during the compiler time and is achieved through the method overloading. (Compile-time Polymorphism) Static Polymorphism in Java **decides which method to execute during compile time**.

Example of static polymorphism

One of the ways by which Java supports static polymorphism is method overloading. An example showing the case of method overloading in static polymorphism is shown below:

Example:

```
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));
  }  }
```

2. Dynamic Polymorphism (or run time polymorphism in Java)

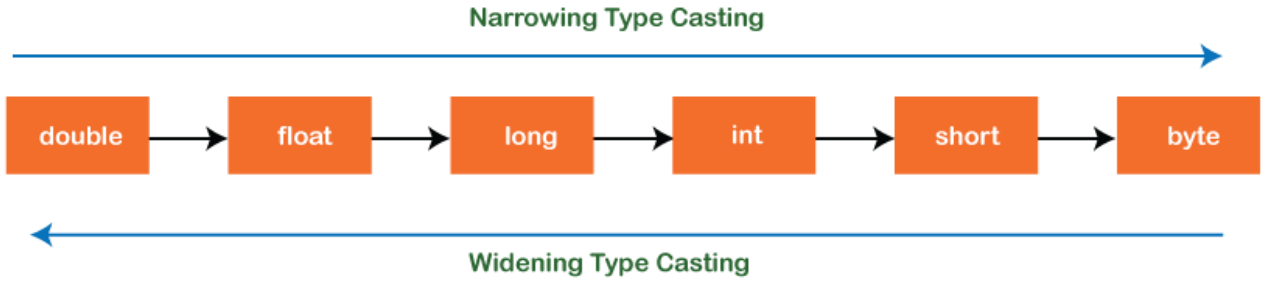
In this form of **polymorphism in java**, the compiler doesn't determine the method to be executed. It's the Java Virtual Machine (JVM) that performs the process at the run time. Dynamic polymorphism in Java refers to the process when a call to an overridden process is resolved at the run time. The reference variable of a superclass calls the overridden method. As the name dynamic connotes, dynamic polymorphism happens among different classes as opposed to static polymorphism. Dynamic polymorphism facilitates the overriding of methods in Java which is core for run-time polymorphism.

Example of Dynamic polymorphism (or run time)

```
class Bike{
 void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("walking safely with 30km");}
    public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

38. Explain about Type Casting in Java.

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. In this section, we will discuss **type casting** and **its types** with proper examples.



Type Casting in Java

Type casting

Convert a value from one data type to another data type is known as **type casting**.

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.

1. **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

   For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

   Example

```
public class WideningTypeCastingExample
{
public static void main(String[] args)
{
int x = 7;
//automatically converts the integer type into long type
long y = x;
//automatically converts the long type into float type
float z = y;
System.out.println("Before conversion, int value "+x);
System.out.println("After conversion, long value "+y);
System.out.println("After conversion, float value "+z);
}
}
```

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.

1. **double -> float -> long -> int -> char -> short -> byte**

Example

```
public class NarrowingTypeCastingExample
{
public static void main(String args[])
{
double d = 166.66;
//converting double data type into long data type
long l = (long)d;
//converting long data type into int data type
int i = (int)l;
System.out.println("Before conversion: "+d);
//fractional part lost
System.out.println("After conversion into long type: "+l);
//fractional part lost
System.out.println("After conversion into int type: "+i);
}
}
```

39. Explain about Object class in Java

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.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:

Object obj=getObject();//we don't know what object will be returned from this method

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

The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

40. What is Abstraction? Explain the Abstract Keyword In Java.

Abstract is a non-access modifier in java applicable for classes, methods but not variables. It is used to achieve abstraction which is one of the pillar of Object Oriented programming (OOP) . Following are different contexts where abstract can be used in Java.

1. Abstract classes

2. Abstract methods

1. Abstract classes:

The class which is having partial implementation(i.e. not all methods present in the class have method definition). To declare a class abstract, use this general form :

```
abstract class class-name
{
   //body of class
}
```

Due to their partial implementation, we cannot instantiate abstract classes. Any subclass of an abstract class must either implement all of the abstract methods in the super-class, or be declared abstract itself. Some of the predefined classes in java are abstract. They depends on their sub-classes to provide complete implementation.

For example, java.lang.Number is a abstract class.

2. Abstract methods:

Sometimes, we require just method declaration in super-classes. This can be achieve by specifying the **abstract** type modifier. These methods are sometimes referred to as *subclasser responsibility* because they have no implementation specified in the super-class. Thus, a subclass must override them to provide method definition.

To declare an abstract method, use this general form:

```
abstract type method-name(parameter-list);
```

As you can see, no method body is present. Any concrete class(i.e. class without abstract keyword) that extends an abstract class must overrides all the abstract methods of the class.

Any class that contains one or more abstract methods must also be declared abstract. Consider the following java program, that illustrate the use of *abstract* keyword with classes and methods.

Example: A java program to demonstrate the use of abstract keyword.

```
abstract class A              // abstract with class
{
abstract void m1();           // abstract with method. it has no body
void m2()                     // concrete methods are still allowed in abstract classes
{
System.out.println("This is a concrete method.");
}
```

```
}
class B extends A                    // concrete class B
{
void m1()              // class B must override m1() method otherwise, compile- time
{                          exception will be thrown
System.out.println("B's implementation of m2.");
}
}
class AbstractDemo
{
public static void main(String args[])
{
B b = new B();
b.m1();
b.m2();
}
}
```

41. Explain about Super Keyword in Java.

The super keyword in java programming language refers to the superclass of the class where the super keyword is currently being used.

The super keyword as a standalone statement is used to call the constructor of the superclass in the base class.

The syntax for using super to call the super class constructor

```
class subclass extends superclass
{
subclass(args)
{
super(args);
        //Statements;
}
.....
}
```

When used as a standalone statement to call the superclass constructor the super should be the first statement within the subclass constructor.

The syntax to call method of super class using super

super.<method_Name>(args) ;

This kind of use of the super keyword is only necessary when we need to call a method that is overridden in this base class in order to specify that the method should be called on the superclass.

Example: A java program to demonstrate the use of super keyword.

```
class Superclass
{
        int x;
        Superclass(int a)               {       x=a;    }
        void display()    {   System.out.println("Super x=",+x);   }
}
class Sub extends Superclass
{
        int y;
        Sub(int a,int b)    {   super(a);   y=b;   }
        void display()
        {
                System.out.println("Super x=",+x);
                System.out.println("Sub y=",+y);
        }
}
Class Override
{
public static void main(String args[])
        {
                Sub s1=new Sub(100,200);
                S1.display();
        }
}
```

Note that the method display () defined in the subclass is invoked.

42. What is Interface? Explain.

An interface is basically a kind of class. The difference is that interfaces define only abstract methods and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constants.

```
interface interface_name
{
        final Variable declaration;
        abstact Method declaration;
}
```

Here, **interface** is the keyword and interface_name is any valid java identifier. Variables are declared as follows:

Static final type variable_Name = value;

Note that all variables are declared as constants. Method declaration will contain only a list of methods without anybody statement.

return_type method_name (parameter_list);

Example:

```
interface Area
        {
  final static float PI=3.14f;
  float compute(float x,float y);
        }
```

Extending interfaces:

Interfaces can also be extended. An interface can be *sub interfaced* from other interfaces. The new *sub interface* will inherit all the members of the *super interface* in the manner similar to subclasses. This is achieved using the keyword **extends** as…

```
interface name2 extends name1
{
        Body of name2;
}
```

We can put all the constants in one interface and the methods in the other. This will enable us to use the constant in classes where the methods are not required.

```
interface A
{
  final static float PI=3.14f;
```

```
}
interface Area extends A
{
  float compute(float x,float y);
}
```

Implementing interfaces:

implements is a keyword used to give implementation for the interface methods in its sub class. Interfaces are used as super classes whose properties are inherited classes.

Syn: **class** classname **implements** interfacename1, interfacename2

```
{
Body of the class
}
```

Here, the class class_name implements the interface interface_name.

Example:

```
class Rectangle implements Area
{
public float compute(float x,float y)
{
return(x*y);
}
}
```

Example: Java program for implementing interfaces.

Any number of dissimilar classes can implement an interface. However, to implement the methods we need to refer to the class objects as types of the interface rather than types of their respective classes. Note that if a class that implements an interface does not implement all the methods of the interface, then the class becomes an abstract class and cannot be instantiated.

Accessing interface variables:

Interfaces can be used to declare a set of constants that can be used in different classes. The interfaces do not contain methods. The constant values will be available to any class that implements the interface. The values can be used in any method, as part of any variable declaration.

```
interface A
{
        final static int m=10;
        final static int n=50;
}
class B implements A
{
        int x=m;
        void methodB(int size)
        {
                ……….
                if(size<n)
                ……….
        }
}
```

Example: Implementing multiple inheritance.

Eample: Implementing hybrid inheritance.

43. What is Package?

One of the main important features of object oriented programming is the reuse of the program code already created. One way of achieving this is by extending the classes (inheritance) and implementing the interfaces. This is limited to reuse in the classes with in a program only.

If we need to use classes from other programs without physically copying them into the program this can be achieved in java by using packages.

What is a package?

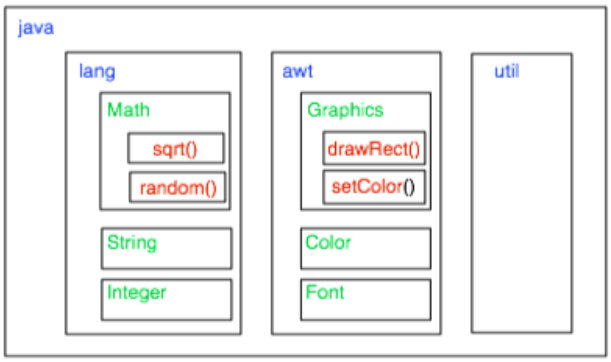
Package is nothing but collection of classes and interfaces. A package that is contained in another package is sometimes called a "sub-package."

By organizing our classes into packages we can achieve the following benefits.

The following is the partial graphical representation of the levels of nesting in the java package, its sub-packages, the classes in those sub-packages, and the subroutines in those classes.

Benefits:

1. The classes contained in the packages of other programs **can be easily reused**.
2. In packages, classes can be **unique.** Two classes in two different packages can have the same name.
3. Packages provide a way to **hide classes**.
4. Packages also provide a way for **separating "design" from coding**. First we can design classes and then we can develop the java code.



Subroutines nested in classes nested in two layers of packages.
The full name of sqrt() is java.lang.Math.sqrt()

44. Explain different types of packages in JAVA.

Java packages are classified into two types: They are:

1. Java API (Application Programming Interface) Packages
2. User defined Packages.

1. Java Application Programming Interface (API) Packages:

An API is a set of methods and classes that provides an interface for the users. Java API provides a large number of classes grouped into different packages according to functionality. The packages are organized in a hierarchical structure.

Some of the Most commonly used Java API packages are:

A. java.lang:

Package that contains essential Java classes, including numerics, strings, objects, compiler, runtime, security, and threads. This is the only package that is automatically imported into every Java program.

B. java.io:

Package that provides classes to manage input and output streams to read data from and write data to files, strings, and other sources.

C. java.util:

Package that contains miscellaneous utility classes, including generic data structures, bit sets, time, date, string manipulation, random number generation, system properties, notification, and enumeration of data structures.

D. java.net:

Package that provides classes for network support, including URLs, TCP sockets, UDP sockets, IP addresses, and a binary-to-text converter.

E. java.awt:

Package that provides an integrated set of classes to manage user interface components such as windows, dialog boxes, buttons, checkboxes, lists, menus, scrollbars, and text fields. (AWT = Abstract Window Toolkit)

F. java.awt.image:

Package that provides classes for managing image data, including color models, cropping, color filtering, setting pixel values, and grabbing snapshots.

G. java.awt.peer:

Package that connects AWT components to their platform-specific implementations (such as Motif widgets or Microsoft Windows controls).

H. java.applet:

Package that enables the creation of applets through the Applet class. It also provides several interfaces that connect an applet to its document and to resources for playing audio.

Using API packages:

There are two ways to access classes stored in a package. They are:

1. The first approach: This approach uses the fully qualified class name. The fully qualified class name consists of java package name and class name separated by dots representing different levels.

Process:

- Use the package name in which the required class is contained.
- Use the class name in which required method in contained.

  Syn: java.package name.classname;

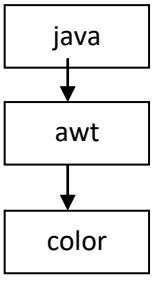
  Ex: java.awt.Color;

  In the above example awt is the package of java package and color is the class of awt package.

2. The second approach:

This approach uses the import statement. This statement must be written at the top of the program. It consists of import keyword, package name and class name along with dot operator.

Process:



- Use the import keyword.
- Use the package name in which the required class is contained.
- Use the class name in which required method in contained.

  Syn: import java.package name.classname;

Ex1: import java.awt.Color;

This statement imports the class color into the program.

Ex2: import java.awt.*;

This statement imports all classes of java.awt package into the program.

Naming convention for packages in JAVA:

Packages can be named using the standard java naming rules.

1. All package names begin with lower case letters.
2. All class names begin with upper case letters.
3. All methods begin with lower case letters.

Syn: java.package_name.class_name.method_name;

Ex: java.lang.Math.sqrt(x);

45. Explain how to create and use User defined packages in JAVA:

Packages that are designed and created by the user are called user defined packages.

Creating a package:

To create a package, we must first declare a package using the **package** keyword. Then we must define a class in it.

Declaring a package:

Syn:
```
package package_name;
public class class_name
{
        …….. body ……..
}
```

Ex:
```
package student;
public class sum
{
        public int add(int x, int y)
        {
                return(x + y);
        }
}
}
```

Steps for creating packages:

1. Decide the name of the package.
2. Create a subdirectory with this name under the directory where main source files are stored.
3. Declare the package at the beginning of a file using the following syntax

   package packagename;
4. The name of the subdirectory must be the same name of the package exactly.
5. Define the class that is to be put in the package and declare it **public**.

```
package arithmetic
public class Addition
{
        public int sum(int x,int y)
        {       return(x+y);    }    }
```

6. Save the source file by class name **Addition.java**
7. Switch to the subdirectory created earlier and compile the source file. When completed, the package would contain **Addition.class** files of the source file.

- Remember that the **.class** file must be located in a directory that has the same name as the package
- Java also supports the concept of package hierarchy. This is done by specifying multiple names in a package statement, separated by dots.

Ex: package firstPackage.secondPackage;

Using or accessing a Package:

The import statement is used to use a package. The general form of using a package is:

Syn: **import** package_name.classname;

Ex: **import** arithmetic.Addition;

Here **arithmetic** is the name of the package and **Addition** is the name of the class in it.

```
import arithmetic.Addition;
class PackageEx
{
        public static void main(String args[])
        {
```

```
                    Addition s = new Addition ();
                    int n=s.sum(5,6);
                    System.out.println("Sum="+n);
                }
            }
```

Adding a class to a package:

We also can add new class to the packages. Suppose we want to add another class **Sub** to **arithmetic** package. Then follow these steps:

1. Define the class and make it public.
2. Place the package statement before the class definition as follows:

```
package arithmetic;
public class Subtract
{
    public int sub(int x, int y)
    {
        return(x-y);
    }
}
```

3. Store this as **Subtract.java** file under the directory **arithmetic**.
4. Compile **Subtract.java** file. This will create a **Subtract.class** file and place it in the directory **arithmetic**.
5. Now the package **arithmetic** will contain both the classes **Addition** and **Subract**.
6. Statement like **import arithmetic.*** can import the definitions of classes Addition and Subtract into the program.

Hiding classes:

When we import package using asterisk (*), all public classes are imported. If we want to hide a particular class we must not use the class access specifier as public in the class definition.

In the below example the **package Mypack** contains two class **first class** is **public** class and the other class **second** is not a public so it is hidden from other classes.

The statement **import Mypack.*;** will import only the definition of class first but not class second as it is hidden.

Ex:

```
package Mypack;
public class first          // public
{
        body of X;
}
class second                // hidden
{
        body of Y
}
```

3. Static import:

Static import is another feature eliminates the need of qualifying a static member with the class name. The static import declaration is similar to that of import. We can use the import statement to import class from packages and use them without qualifying the package. Similarly, we can use the static import statement to import static members from classes and use them without qualifying the class name.

Syn: import static package_name.class_name.staticmember_name;

```
import static java.lang.Math.*;
public class Mathop
{
        public void circle(double r)
        {
                double area = PI * r * r;
                System.out.println("Area=" + area);
        }
        public static void main(String args[])
        {
                Mathop obj = new Mathop();
                obj.circle(2,3);
        }
}
```

Example: Java program for creating an arithmetic package.

package name: arithmetic

class names: Sum.java

Subtract.java

Mulitply.java

Divide.java

Sum.java:-

```
package arithmatic;
public class Sum
{
int x,y,z;
public Sum(int a,int b)
{
x=a;
y=b;
}
public void caliculatesum()
{
z=x+y;
        System.out.println("the sum is "+z);
}
}
```

Subtract.java:-

```
package arithmatic;
public class Subtract
{
int x,y,z;
public Subtract(int a,int b)
{
x=a;
y=b;
}
public void caliculatesub()
{
```

```
z=x-y;
System.out.println("thesubtraction value is "+z);
}
}
```

Mulitply.java:-

```
package arithmatic;
public class Multiply
{
int x,y,z;
public Multiply(int a,int b)

{
x=a;
y=b;
}
public void caliculatemul()
{
z=x*y;
System.out.println("the multiplied result is "+z);
}
}
```

Divide.java:-

```
package arithmatic;
public class Divide
{
int x,y,z;
public Divide(int a,int b)
{
x=a;
y=b;
}
public void caliculatediv()
{
```

```
z=x/y;
System.out.println("the division result is "+z);
}
}
```

The above sum.java
subtract.java
multiply.java
divide.java

are stored in a directory (name) arithmetic. Why because the package name is **arithmetic** and complile individually. so compiler creates **.class** files.

The above classes are imported into **Packdemo.java** program.

Packdemo.java:-

```
import arithmatic.Sum;
import arithmatic.Subtract;
import arithmatic.Mulitply;
import arithmatic.Divide;
class Packdemo
{
public static void main(String args[])
{
Sum s=new Sum(25,35);
s.caliculate();
Subtract u=new Subtract();
u.caliculate();
Mulitply m=new Mulitply();
m.caliculte();
Divide d=new Divide();
d.caliculate();
}
}
```

EXCEPTION HANDLING

46. WHAT IS EXCEPTION HANDLING? EXPLAIN ITS TYPES.

Errors in Programming:

An error may produce incorrect output or terminate the program execution.It is therefore important to detect and manage properly all the possible error condition in the program.

Errors may classified into 2 types. 1) Compile time errors.

2) Runtime Errors.

COMPILE TIME ERRORS:-

All the syntactical errors will be detect and display by the java compiler. These errors are known as compile time errors. Whenever the compiler displays an error, it will not create the **.class** file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program. Java compiler does a nice job of telling us where the errors are in the program. Most of the compile-time errors are due to typing mistakes.

The most common problems are.

- Missing semicolon at the end of any statement.
- Missing class or method brackets.
- Missing spellings of identifiers and keywords.
- Use of undeclared variables.
- Bad reference to object.
- Use of = in the place of = = operator.

Ex:

```
Class CompileError                                  ----------> Missing class brackets
Public static void main(String args[]               ----------> missing method brackets.
{
  int a=10,b=20                                     ----------> missing semicolon
      c=a/b;                                        ----------> undeclared variable c
System.out.println("the sum is"+c);
        }
      }
```

RUNTIME ERRORS:-

Some times a program may compile successfully then java compiler creates ".class file". But may not run properly such programs may produce incorrect result due to wrong logic or may terminate program due to errors such errors are runtime errors.

Some common types of run time errors are.

- Dividing an integer by Zero.
- Accessing an element that is out of the bounds of an array.
- Passing a parameter that is not in a valid type.

When such type of errors encounter then java default exception handler display an exception type.

Ex:- Class RuntimeError

```
{
Public static void main(String args[])
    {
    Int a=10,b=0,c;
    C=a/b;  ——————————————→ Division by zero (b=0) condition;
    System.out.println("the result is"+c);
    }
}
```

47. How to handle Exceptions in JAVA.

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. When an error occurs within a method, the method creates an object called an *exception object*, contains information about the error, including its type and the state of the program. Creating an exception object and handing it to the runtime system is called *throwing an exception*.

If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program. If we want the program to continue with execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known an Exception handling.

The Exception Handling mechanism performs the following task:

- Find the problem (Hit the Exception-**try**)
- Inform that an error has occur(**Throw** the exception)
- Receive that error information(**catch** the exception)
- Take creative action (Handle the exception)

The exception handling code basically consist of two segments.

- To detect errors and throw that exception.

- To catch exception and to take appropriate action

We can implement java Exception Handling by using the keywords: **try, catch, throw, throws, finally**

Syntax of Exception Handling Code:

The basic concepts of exception handling are throwing an exception and catching it.

```
try
{
        Statements;    // generate an exception
}
catch (Exception-Type e)
{
        Statement;              // process the exception
}
```

Java uses a keyword **try** to preface a block of code that is likely to cause an error condition and throw an exception. A catch block defined by the keyword **catch** catches the exception thrown by thee try block and handles it appropriately. The catch block is added immediately after the try block.

Some common exception types are:

Exception type	Description
ArithmeticException	Caused by mathematical errors such as Division By zero
ArrayIndexOutOfBoundsException	Caused by bad array index.
ArrayStoreException	Caused by when a program try to store the wrong type of data in an array
FileNotFoundException	Caused by it is an attempting to cause an non existing file
I/OException	Causedby general input/output failures.

Try keyword:-

The try block can have one or more statements that could generates an exception. If anyone statement generates an exception the remaining statements in the try block are skipped and execution jumps to the catch block that is placed to the next of try block.

Catch keyword:-

A try block to be followed by at least one catch block. Each catch block is like a little method that makes one and only one argument of a particular exception type.

Ex: Write a java program to handle arithmetic Exception

```
import java.util.*;
class Arithmetic
{
public static void main(String args[])
{
int a,b,c;
Scanner s=new Scanner(System.in);
a=s.nextInt();
b=s.nextInt();
try
{
c=a/b;                                  //causes error due division by zero
System.out.println("the remainder  is "+c);
}
catch(ArithmeticException ae)
{
System.out.println("you are given ramainder value zero.");
}
}
}
```

Finally:

java supports another exception handling statement known as finally. That can be used to handle an exception that is not caught by any of the previous catch statement. That means finally block can be used to handle any type of exception generated within try block.

The finally block is executed regardless of whether or not an exception is thrown.

```
try { .... }
catch() { ..... }
finally { ..... }
```

Ex: Write a java program by using finally keywoard.

```
import java.util.*;
class Arithmatic
{
public static void main(String args[])
{
int a,b,c;
Scanner s=new Scanner(System.in);
a=s.nextInt();
b=s.nextInt();
try
{
c=a/b;                                    //causes error due division by zero
System.out.println("the remainder  is "+c);
}
catch(ArithmeticException ae)
{
System.out.println("you are given ramainder value zero.so division by zero exception is raised");
}
finally
{
System.out.println("this is final block statements and executed at last ");
}
}
}
```

While you are using the throw keyword the flow of execution stop immediately after the throws statement the remaining sub statements inside the try block after the throw keyword are not executed.

Ex: Write a java program to handle the user defined exception (marks out of bounds Exception)

Throws:

Throws is a keyword used in a method definition to declare exception to be thrown. So we need not consider about the try,catch,finally blocks

Syntax:

```
Void methodname() throws Exception type
{
  Statements
}
```

Ex:

Throwing our own exceptions:

We can throw our own exceptions using the keyword **throw** as follows:

throw new throwable_subclass;

Example: throw new ArithmeticException();

```
import java.io.*;
import java.lang.Exception;
class MyException extends Exception
{
        MyException(String message)
        {
                Super(message);
        }
}
Class TestMyException
{
        public static void main(String args[])throws Exception
        {
                InputStreamReader instr= new InputStreamReader(System.in);
                BufferedReader br=new Buffered(instr);
                int m;
                System.out.println(:Enter marks:");
                try
                {
                        m=Integer.parseInt(br.readLine());
                        If(m<0 || m>100)
                        {
                                Throw new MyException("Invalid Marks!");    }     }
```

Streams: Stream, Creating a File using FileOutputStream, Reading Data from a File uingFileInputStream, 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 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

---

48. What is file? Explain different file handling methods in JAVA.

Java uses variables and arrays to store data inside the program due to this.

- The data is lost when the program terminates or a variable goes out of the scope i.e, Here storage is temporary.
- It is difficult when data is large.

So we can overcome this problem by storing data in secondary storage devices in two ways.

1. By using files
2. By using databases.

Files are used for long term storage of large amounts of data, even after the program that created the data terminates. But in files data can be manipulated by un authorized users. Files concept never provides security. Hence it is not recommended.

Data base provides more security by giving username and passwords. So majority of them will use database concept.

File:- A File is a collection of related records placed in a specific area on the disk. Data that is stored in files is often called persistent data. A record is a collection of several fields and the field is a collection of characters. Each character in a computer character set is represented as a pattern of 1's and 0's.

Creation, Updating, Managing data etc., using file is known as 'File Processing'.

To deal with files we must use some predefined classes and interfaces which contains in java.io.package.

We have to **import java.io.*;**

File Class: - java.io. package supports for Input & Output operations on files. This package provides a class known as File Class to creating Files. File Object is used to manipulate or obtain the information associated with a disk file, such as permission, data, directory path and soon.

The following constructors are used to create a File Object.

File(String directorypath)

File(String directorypath,String filename)

File(File object,String filename)

Methods: -

1. boolean canRead()
2. boolean canWrite()
3. boolean exists()
4. String getName()
5. String getParent()
6. String getPath()
7. boolean isDirecotry()
8. boolean isFile()
9. long length()
10. long lastModified( )
11. boolean isHidden()
12. String getAbsolutePath()
13. boolean isAbsolute( )
14. String [ ]list()

Ex: Write a java program by using all file handling methods

```
import java.io.*;
class FileDemo
{
public static void main(String arg[])
{
File f=new File("D:/giri/hai.java");
System.out.println(f.canRead()); //true
System.out.println(f.canWrite()); //true
System.out.println(f.exists()); //true
System.out.println(f.getName()); //hai.java
System.out.println(f.getParent()); //D:\giri
System.out.println(f.getPath());
//D:\giri\hai.java
System.out.println(f.isDirectory()); //false
System.out.println(f.isFile()); //true
```

```
System.out.println(f.length()); //210
System.out.println(f.lastModified()); //1165345688000
System.out.println(f.isHidden()); //false
System.out.println(f.getAbsolutePath()); //D:\giri\hai.java
System.out.println(f.isAbsolute()); //true
}
}
```

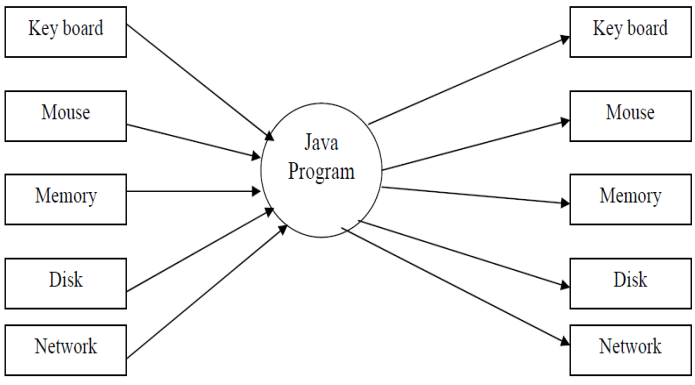
Ex: Write a program to display all files in the given directory

```
import java.io.*;
class All
{
public static void main(String arg[])
{
File f=new File("D:/ramesh");
String str[]=null;
if(f.isDirectory())
str=f.list();
for(int i=0;i<str.length;i++)
System.out.println(str[i]);
}
}
```

49. What is Stream? Explain different stream objects used by JAVA.

A stream is path traveled by data in program. An input stream sends data from a source into a program, and an output stream sends data out of a program to a destination.

When a file is opened, an object is created and a stream is associated with the object. Three stream objects are created by the java system automatically when the program execution begins.



They are
1. System.in
2. System.out
3. System.err

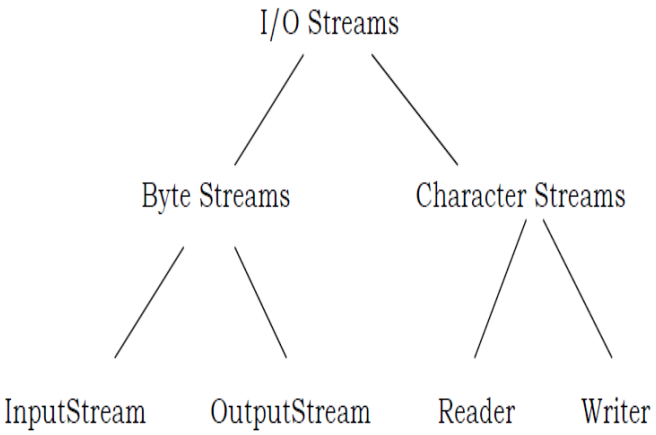
System.in: It is a standard input stream object. It enables a program to input bytes from the keyboard

System.out: It is a standard output stream object. It enables a program to output data to the screen.

System.err: It is a standard error stream object, It enables a program to output error message to the screen.

I/O Stream Hierarchy:

Java 2 defines two types of Streams called Byte Streams and Character Streams



1. **Byte Streams:** It carry integer values that range from 0 to 255. All the types of data can be expressed in a byte format, including numerical data, executable programs, Internet communication and the byte code produced by Java Virtual Machine.
2. **Character Streams:** Character streams are a specialized type of byte stream that only handles textual data. They are distinguished from byte streams because Java's character set supports Unicode, a standard that includes many more characters(0 to 65,535)

The character and byte streams are all sequential access streams; that is, they can read a file from a point to point, say start to finish.

50. Explain different file streams in JAVA.

FileInputStream:- File Input Stream is derived from Input Stream and used to read bytes of data from a file.

Constructors of FileInputStream are

FileInputStream(String filepath)

FileInputStream(File fileobj)

Here, filepath is the full path name of the file.

Fileobj is a File object that describes the file.

FileOutputStream:- FileOutputStream is derived from OutputStream and used to send bytes of data into a file.

Constructors of FileOutputStream are

FileOutputStream(String filepath)

FileOutputStream(File fileobj)

FileOutputStream(String filepath, boolean append)

Here, filepath is the full path name of the file

Fileobj is a File object that describes the file

If append is true, then the file is in append mode

Reading from a file and writing to a file using Input/Output byte streams.

```
import java.io.*;
public class f1
{
public static void main(String arg[])
{
try
{
FileInputStream fis=new FileInputStream("New.java");
FileOutputStream fos=new FileOutputStream("abc.txt");
int k;
while((k=fis.read())!=-1)
fos.write(k);
fos.close();
fis.close();
}
catch(IOException e)
{
System.out.println("An error occurred in IO Operation "+e);
}
}
}
```

The above program can be rewritten with Reader/Writer classes. Just replace FileInputStream with FileReader and FileOutputStream with FileWriter

```
import java.io.*;
public class f1
{
public static void main(String arg[])
{
try
{
FileReader fr=new FileReader("New.java");
```

```
FileWriter fw=new FileWriter("ab.txt");
int k;
while((k=fr.read())!=-1)
fw.write(k);
fr.close();
fw.close();
}
catch(IOException e)
{
System.out.println("An error occurred in IO Operation "+e);
}
}
}
```

51. Explain about Character Streams in JAVA.

Character Stream classes are used for processing the data in the form of characters. This class defines two important abstract classes called Reader and Writer.

Reader: - It is an abstract class that defines Java's model of streaming character input. The system is reading the data in the form of characters from the physical device using Reader. The methods that are provided by the Reader class are

a) void close( ): Closes the input source. Further read attempts will generate an IOException.

b) int read( ): Returns an integer representation of the next available character for the invoking stream. -1 is returned when the end of the is encountered.

c) int read(char buf[]): Attempts to read up to buf.length characters into buffer and returns the actual number of characters that were successfully read. -1 is returned when the end of file is encountered.

d) int read(char buf[ ],int off, int num): Attempts to read up to num characters into buffer starting at buffer[off], returning the number of characters successfully read. -1 is returned when the end of file is encountered.

e) long skip(long numchars): Skips over numchars characters of input, returning the number of characters actually skipped.

Writer: - It is an abstract class that defines java's model of streaming character output. The system is sending the data in the form of characters to the physical device using Writer. The methods that are provided by the Writer class are

a) void close( ): Closes the output stream. Further write attempts will generate an IOException

b) void write(int ch): Writes a single character to the invoking output stream

c) void write(char buf[]): Writes a complete array of characters

```
import java.io.*;
public class ww
{
public static void main(String arg[]) throws IOException
{
DataInputStream dis=new DataInputStream(System.in);
System.out.print("enter first number ");
String str=dis.readLine();
int fn=Integer.parseInt(str);
System.out.print("enter seconf number ");
str=dis.readLine();
double sn=Double.parseDouble(str);
System.out.println("You entered: "+fn+"and"+sn);
System.out.println("Their sum is "+(fn+sn));
dis.close();
}
}
```

Threads In JAVA

52. Explain about multitasking in JAVA.

The ability to execute several programs simultaneously is known as Multi tasking. They are two types of Multi tasking.

- Process based Multi tasking
- Thread based Multi tasking

Process based Multi tasking : -

A process is a program that is executing. A process based multitasking is a feature that allows a computer to run 2 or more programs simultaneously.

Eg:- It enables to run the java complier at the same time that you are using notepad.

Thread based Multi tasking :-

A thread is a small part of a program that has a single flow of control. A thread based multi tasking is a single program that can perform 2 or more tasks simultaneously.

Eg:- In MS Word at the time of entering the data the spelling and grammar facility can check the data at the same time.

Difference between processing based Multi tasking and thread based Multi tasking:-

Process based Multi tasking	Thread based Multi tasking
An executing program is called a process.	A thread is a small part of a process.
Every process has its separate address space.	All the threads of a process share the same address space cooperatively as that of a process.
In process based multitasking a process or a program is the smallest unit.	In thread based multitasking a thread is the smallest unit.
In process based multitasking two or more processes and programs can be run concurrently.	In thread based multitasking two or more threads can be run concurrently.
Process based multitasking requires more overhead.	Thread based multitasking requires less overhead.
Process to Process communication is expensive and limited.	Communication between two threads is less expensive as compared to process.
Context switching from one process to another process is expensive.	Context switching from one thread to another thread is less expensive as compared to process.
Process are also called heavyweight task.	Thread are also called lightweight task.
It has slower data rate multi-tasking.	It has faster data rate multi-tasking.
Process-based multitasking is not under the control of Java.	Thread-based multitasking is under the control of Java.

53. What is Thread? Explain its types.

Any application can have multiple processes (instances). Each of this process can be assigned either as a single thread or multiple threads.

What is Single Thread?

A single thread is basically a lightweight and the smallest unit of processing. Java uses threads by using a "Thread Class".

There are two types of thread – **user thread and daemon thread**

When an application first begins, user thread is created. Post that, we can create many user threads and daemon threads. Demon threads are used when we want to clean the application and are used in the background.

Single Thread Example:

```
public class SingleThreadEx
{
    public static void main(String args[])
        {
         System.out.println("Single Thread example");
    }
}
```

Advantages of single thread:

Reduces overhead in the application as single thread execute in the system Also, it reduces the maintenance cost of the application.

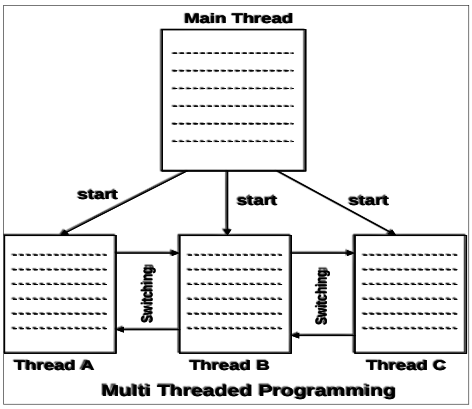
Multithreading:-

A thread is similar to a program that has a single flow of control. A unique property of java is its support for multithreading. Java enables us to use multiple flows of control in developing programs. Each flow of control may be thought of as a separate tiny program known as a thread that runs in parallel to others. A program that contains multiple flows of control known as Multithreaded Program.

The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently.

Hence it is also known as **Concurrency** in Java. Each such part of a program called thread. This multitasking is done, when multiple processes share common resources like CPU, memory, etc.

Each thread runs parallel to each other. Threads don't allocate separate memory area. Hence it saves memory. Also, context switching between threads takes less time.



1. Multithreading is a powerful programming tool that makes java distinctly different from other programming language.
2. It enables programmers to do multiple things at one time.
3. They can divide a long program into threads and execute them in parallel.
4. Threads are extensively used in java-enabled browsers such HotJava.

Example of Multi thread:

```
public class MultiThreadEx implements Runnable
        {
        public static void main(String[] args)
        {
                Thread MultiThreadEx1 = new Thread("Giri");
                Thread MultiThreadEx2 = new Thread("Babu");
                MultiThreadEx1.start();
                MultiThreadEx2.start();
                System.out.println("Thread names are following:");
                System.out.println(MultiThreadEx1.getName());
                System.out.println(MultiThreadEx2.getName());
        }
        @Override
        public void run()
                {
                }
        }
```

Advantages of multithread:

The users are not blocked because threads are independent, and we can perform multiple operations at times

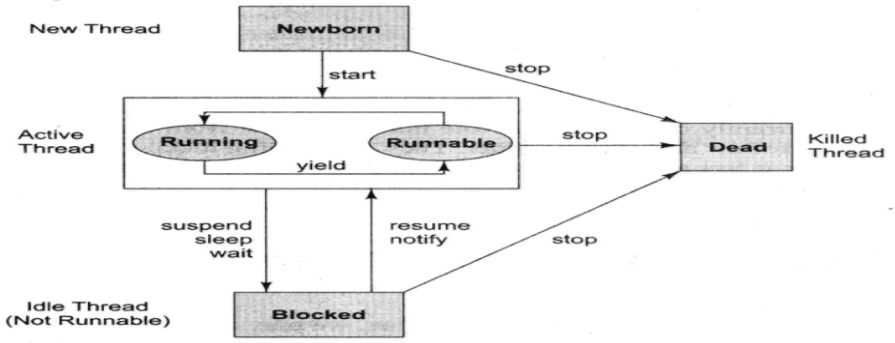
As such the threads are independent, the other threads won't get affected if one thread meets an exception.

54. Explain the Life Cycle Of A Thread.

During the life of a thread, there are many states it can enter.

They are:

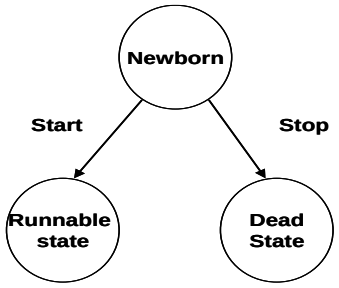
1. Newborn state.
2. Runnable state
3. Running state.
4. Blocked state.
5. Dead state.



New born state:-

In this phase, the thread is created using class "Thread class". When we create a thread object the thread is born and is said to be New born state. The thread is not at all schedule for running at that state we can so any of the following things.

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

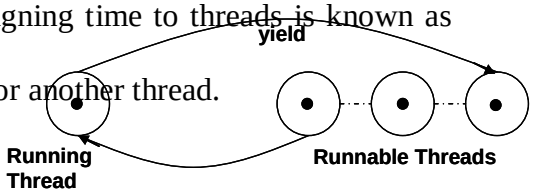


Scheduling a newborn thread

Runnable state:-

The runnable state states that the thread is ready for execution and is waiting for the availability of the processor. That is the runnable thread has join in the queue of the threads are waiting for execution. If all threads have equal priority then they are executed in round robins fashion (first-come, first-serve manner). This process of assigning time to threads is known as time-slicing.

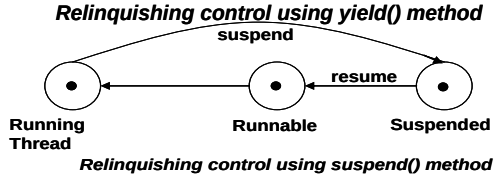
Yield means that the current thread is willing to give for another thread.



Relinquishing control using yield() method

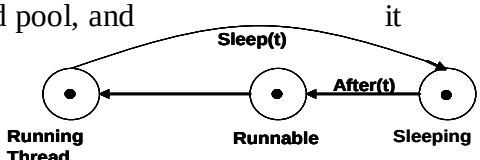
Running state: -

Running means that the processor has given its time to the thread for its execution the thread runs until it out of control on its own or it is pointed by a high priority thread. When the thread starts executing, then the state is changed to "running" state. The scheduler selects one thread from the thread pool, and it starts executing in the application.



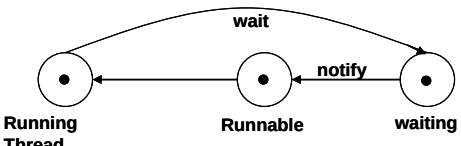
Relinquishing control using suspend() method

The running thread break its control is one of the following situations.



Relinquishing control using sleep() method

a. It has been using suspended by using suspend () method and can retrieve by using resume () method.
b. The thread has been made of sleep when the time period of using a method sleep (time) is out and again reenters the runnable state has this time period is elapsed.



Relinquishing control using wait() method

c. It has been told to wait until some events occur this is done by using the wait () method. The thread can be again go to runnable state when it call notify () method.

Blocked State:-

A thread is said to be a blocked state when it is prevented entering to the runnable state and subsequently running state. This happens when the thread is suspended or waiting or sleeping in order to satisfy certain requirements. A blocked thread is consider not runnable but not dead and there for the qualified to run again.

Dead State:-

This is the state when the thread is terminated. Every thread has a life cycle a thread ends its life when it has completed its run method **Or** however we can kill it by sending this stop() method.

55. How to create Threads in JAVA?

Threads are implemented in the form of objects that contain a method called **run()**. The **run()** method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the thread's behavior can be implemented.

```
public void run()
{
        Statements for thread
}
```

The **run()** method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating it with the help of another thread method called **start()**.

A new thread can be created in two ways:

i. **By using extending the thread class.:**
   Define a class that extends **Thread** class and override its **run()** method with the code required by the thread.

ii. **By using implementing Runnable interface:**
   Define a class that implements Runnable interface. The **Runnable** interface has only one method, **run()** that is to be defined in the method with the code to be executed by the thread.

1. Extending the thread class:

We can make our class runnable as thread by extending the class **java.lang.Thread**. This gives us access to all thread methods directly.

1. Declare the class as extending the **Thread** class.

```
Class MyThread extends Thread
{
        . . . . . .
}
```

2. Implement the **run()** method that is responsible for executing the sequence of code that the thread will execute.

```
public void run ()
{
        . . . . . .
}
```

3. Create a thread object and call the **start()** method to initiate the thread execution.

```
classname objectname=new classname();
Objectname.start();                    // invoking run() method
```

Ex Program:

Write a java program to create 3 threads. Even, Odd, Natural numbers up to 20 numbers.

Multithread.java:

```
class Even extends Thread
        {
public void run()
        {
        for(int i=2;i<=20;i+=2)
                {
        System.out.println("even number"+i);
                }
        }
}
class Odd extends Thread
        {
public void run()
          {
        for(int i=1;i<=20;i+=2)
                {
        System.out.println("odd number"+i);
                }
       }
}
class Natural extends Thread
        {
public void run()
        {
        for(int i=1;i<=20;i++)
                {
        System.out.println("natural number"+i);
                }
       }
}
class Multithread
        {
public static void main(String args[])
        {
Even e=new Even();
Odd o=new Odd();
Natural n=new Natural();
e.start();
o.start();
n.start();
}  }
```

Note that the output from the threads is not sequential. They do not follow any specific order. They are running independently of one another and each executes whenever it has a chance.

56. How to implement threads using Runnable Interface:-

We can implement thread programs using Runnable interface contains all the methods of the threads start(),strop(),run() without body implementation.

We have to implement these methods in our class to implement a thread class using Runnable interface.

It includes the following steps.

1. Implement our classes using Runnable interface.

```
class classname implements Runnable
    {
    Statements;
    }
```

2. Provide body to the run() method.

```
public void run()
    {
    Statements;
    }
```

3. Create an object of our class that must be instantiated with thread class. So create a thread object and pass that object to the thread class constructor.

```
class name objectname = new classname();
```

4. Then call the start method using thread object.

```
thread object name.start();
```

Ex Program: Write a java program to create threads using Runnable interface.

Runnablethread.java:

```
import java.lang.Thread;
class X implements Runnable                    // Step 1
{
public void run()                              // Step 2
{
for(int i=1;i<=10;i++)
{
System.out.println("Thread X: i=" + i);
}
System.out.println("Exit from X");
}
}
Class Runnablethread
{
public static void main(String args[])
{
X r=new X();
Thread threadX=new Thread(r);                  // Step 3
threadX.start();                               // Step 4
System.out.println("End of the main thread");
}
}
```

57. Explain about Thread Priority JAVA.

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

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

ThreadName.setPriority(intNumber)

The **intNumber** is an integer value to which the thread's priority is set. The Thread class defines several priority constants:

MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10

The **intNumber** may assume one of these constants or any value between 1 to 10. Note that the default setting is NORM_PRIORITY.

Whenever multiple threads are ready for execution, the Java system chooses the highest priority thread and executes it.

It stops running at the end of run().

It is made to sleep using sleep().

It is told to wait using wait().

Ex Program: Write a java program to create threads using Various Thread Priorities.

```
class A extends Thread
{
public void run()
{
System.out.println("A started");
for(int i=1;i<5;i++)
{
System.out.println("Thread A: i=" + i);
}
System.out.println("Exit from A");
}
}
class B extends Thread
{
public void run()
{
System.out.println("B started");
for(int j=1;j<5;j++)
{
System.out.println("Thread B: j=" + j);
}
System.out.println("Exit from B");
}
}
class C extends Thread
{
public void run()
{
System.out.println("C started");
for(int k=1;k<5;k++)
{
System.out.println("Thread C: k=" + k);
}
System.out.println("Exit from C");
}
}
Class ThreadPriority
```

```
{
public static void main(String args[])
{
A tA =new A();
B tB =new B();
C tC =new C();
tA.setPriority(Thread.MIN_PRIORITY);
tB.setPriority(tA.getPriority()+1);
tC.setPriority(Thread.MAX_PRIORITY);
System.out.println("Start thread A");
tA.start();
System.out.println("Start thread B");
tB.start();
System.out.println("Start thread C");
tC.start();
System.out.println("End of main thread");
}
}
```

58. Explain about Thread Synchronization.

Synchronization generally means sharing data between multiple or threads. One thread may try to read a record from a file while another thread is still writing to the same file. Depending on the situation, we may get strange results. Java enables us to overcome this problem using a technique known as synchronization.

The keyword synchronized helps to solve such problems by keeping a watch on such locations. Key to Synchronization is the concept of monitor. A monitor is an object that is used as a mutual exclusion clock. One thread can own monitor at a given time.

When a thread acquires a clock it is said to enter the monitor. All other threads attempting to enter the clocked monitor will be suspended until the first theread (running thread) exist the monitor.

For example, the method that will read information from a file and the method that will update the same file may be declared as synchronized.

Example:

```
Synchronized void update()
{
        ……
}
```

Example: Java program for Thread Synchronization

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
class Table
{
synchronized void printTable(int n)
{
//synchronized method
for(int i=1;i<=5;i++)
{
System.out.println(n*i);
```

```
Try
{
Thread.sleep(400);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
}
class MyThread1 extends Thread
{
Table t;
MyThread1(Table t)
{
this.t=t;
}
public void run()
{
t.printTable(5);
}
}
class MyThread2 extends Thread
{
Table t;
MyThread2(Table t)
{
this.t=t;
}
public void run()
{
t.printTable(100);
}
}
public class TestSynchronization2
{
public static void main(String args[])
{
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

UNIT – V

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

Java Database Connectivity: Database Servers, Database Clients, JDBC (Java Database Connectivity), Working with Oracle Database, working with MySQL Database, Stages in a JDBC Program, Registering the Driver, connecting to a Database, Preparing SQL Statements, Using jdbc–odbc Bridge Driver to Connect to Oracle Database, Retrieving Data from MySQL Database, Retrieving Data from MS Access Database, Stored Procedures and Callable Statements, Types of Result Set

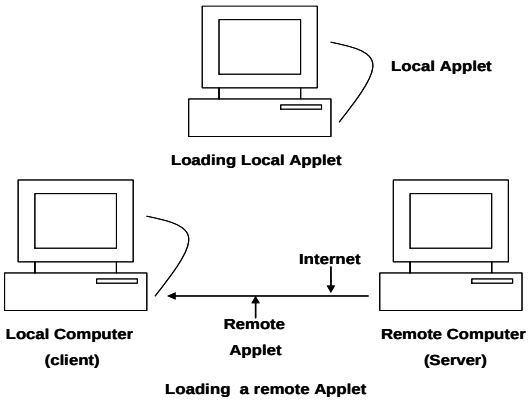
---

59. What is Applet? How to develop applets in JAVA.

Applet programs are small programs that are primarily used in Internet programming. These programs are either developed in local systems or in remote systems and are executed by either a java compatible "Web browser" or "Applet viewer". Like any application program applets can perform arithmetic operations, display graphics, play sounds, accept user input, create animations etc..

Applets can develop in two ways.

- We can write our own applets and embed them into web pages
- We can download an applet from a remote computer and then embed it into a web page.



Types Of Applets:-

Applets are two types: 1) Local Applets

2) Remote Applets

A. Local Applets:

An applet developed locally and stored in a local system is known as 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 and run the specified applet.

B. Remote Applets:

An applet developed by someone else and stored on a remote computer is known as Remote Applet. If our system is connected to the Internet, we download the remote applet onto

our system via the Internet and run it. In order to locate the remote applet, we must know the applet's address on the Web. This address is known as Uniform Resource Locator (URL) and must be specified in the applet's HTML document as the value of the CODEBASE attribute.

60. How Applets Differ From Applications?

Both applets and stand-alone applications are java programs. There is significant difference between them. Applets are not full-featured application programs.

a. Applets do not use the **main()** method for initiating the execution of the code. Applets, when loaded, automatically call certain methods of **Applet** class to start and execute the applet code.
b. Unlike Stand-applications, applets cannot be run independently. They are run from inside a web page using **HTML** tag.
c. Applets are restricted from using **libraries** from other languages such as **C** or **C++.**
d. Applets cannot read from or write to the files the local computer.
e. Applets cannot communicate with other server on the network.
f. Applets cannot run any program from the local computer.

Advantages of applets:

1. When we need something dynamic to be included in the display of a web page.
2. When we require some flash outputs. For example, applets that produce sounds, animations or some special effects would be useful when display certain pages.
3. When we want to create a program and make it available on the internet foe us by other on their computers.

Steps for developing and testing applets:

1. Building an applet code (**.java** file)
2. Creating an executable applet(**.class** file)
3. Designing a web page using HTML tags.
4. Preparing **<APPLET>** tag.
5. Incorporating **<APPLET>** tag into the web page.
6. Creating HTML file.
7. Testing the applet code.

61. How to build an applet code in JAVA (.java file)?

It is essential that our applet code uses the services of two classes, namely **Applet** and **Graphics** from the Java class library. The Applet class which is contained in the **java.applet** package provides life and behavior of the applet through its methods such as **init(), start()** and **paint().** Unlike the applications, where Java calls the **main()** method directly to initiate the

execution of the program, when an applet is loaded, Java automatically calls a series of **Applet** class methods for starting running and stopping the applet code. The Applet class therefore maintains the *lifecycle* of an applet.

The **paint()** method of the applet class, when it is called, actually displays the result of the applet code on the screen. The output may be text, graphics or sound. The **paint()** method, which requires a Graphics objects as an argument,

public void paint(Graphics g)

This requires that the applet code imports the java.awt package that contains the Graphics class. All output operations of an applet are performed using the methods defined in the Graphics class.

```
import java.awt.*;
import java.applet.*;
Public class appletclassname extends Applet
{
        . . . .
        Public void paint(Graphs g)
{
        . . .      // Applet operations code
}
        . . .
. . .
}
```

The **appletclassname** is the main class for the applet. When the applet is loaded, Java creates an instance of this class, and then a series of **Applet** class methods are called on that instance to execute the code.

```
Import java.awt.*;
Import java.applet.*;
Public class HelloJava extends Applet
{
Public void paint(Graphs g)
{
g.drawSring("Hello Giribabu", 10,100);
}
}
```

62. Explain the Life Cycle Of An Applet.

The life cycle of an applet can be specified by 4 states.

- Born state – init()
- Running state – start()
- Idle state – stop()
- Dead state – destroy()

Initialized state:**[init() method]**

The life cycle of an applet is begin on that time when the applet is first loaded into the browser and called the init() method. The init() method is called only one time in the life cycle on an applet. After the initialization of the init() method user can interact with the Applet and mostly applet contains the init() method

- The init() method is basically called to read the PARAM tag in the html file.
- Initialization of variables and
- Initialization of the objects like image, sound file.

Syn: public void init()

```
{
        Statements;
        ----
}
```

Running State: [Start () method]:

The start method of an applet is called automatically after the initialization method init(). This method may be called multiples time when the Applet needs to be started or restarted.

For Example if the user wants to return to the Applet, in this situation the start Method() of an Applet will be called by the web browser and the user will be back on the applet.

Syn: public void start()

```
{
        Statements;
}
```

Idle State:[Stop () method]:

An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do by calling The stop() method.

There is only miner difference between the start() method and stop () method. the stop() method is called by the web browser when the user leaves one applet to go another applet and the start() method is called when the user wants to go back into the first program or Applet.

Syn: public void stop()

```
{
        Statements;
}
```

Dead State:[destroy() method]:

An applet is said to be dead when it is removed from memory. This occurs automatically by invoking the **destroy ()** method when we quit the browser. Like initialization, destroying stage occurs only once in the applet's life cycle. If the applet has created any resources, like threads, we may override the **destroy ()** method to clean up these resources.

```
Public void destroy()
{
        . . . .
}
```

Display State:[paint() method]: Applet moves to the **display** state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. Almost every applet will have a **paint()** method. Like other methods in the life cycle, the default version of **paint()** method does absolutely nothing.

Syn: public void paint(Graphics g)

```
{
Display statements;
}
```

Preparing The Applets:

Before writing and executing applets into our system either java Applet viewer or a java-enabled web browser must be properly installed.

The following steps involved in developing applets.

- Building an applet code (**.java** file **)**
- Creating an executable applet (**.class** file)
- Designing a web page using HTML tags.
- Preparing < APPLET > tag and incorporate into the web page.
- Creating HTML file.

- Testing and Executing the Applet code by Applet Viewer or Java Enabled Web browser.

63. How to create and Execute Applets in JAVA?

All applets are subclasses of **Applet**. All applets must **import java.applet;** and **import java.awt;**

Applet package provides all necessary support for applet execution, such as starting and stopping. It also provides methods such as **init(), start(), stop( ), and destroy( )** that load and display images, and load and play audio clips etc. Output to your applet's is handled with The **paint()** method of the **applet** display the results of the applet code on the screen. The output may be text, graphics, images, sounds etc.

Syntax: Import java.awt.*;

```
Import java.applet.*;
public class myapplet extends Applet
    {
    public void init()      {       //initialization          }
    public void start()     {       //start or resume execution          }
    public void stop()      {       //suspends execution          }
    public void destroy()   {       //shut down activity          }
    public void paint(Graphics g) {     //display the contents          }     }
```

Where the **Graphics** class contain methods in the **java.awt** package perform all the output operations.

Ex:-

```
import java.applet.*;
mport java.awt.*;
public class Myapplet extends Applet
   {
        public void paint(Graphics g)
        {
        g.drawString("hello gud mornig", 50,50);
        }
    }
```

Here **drawsting()** method in a **Graphics class** contain **arguments** namely **output string** , **X-axis** position(in pixels) , **Y-axis** postion (in pixels),.

11. Designing a web page: A web page is basically made up of text and HTML tags that can be interpreted by a web browser or an applet viewer. Like java source code, it can be prepared using

any ASCII text editor. A web page is also known as HTML page or HTML document. Web pages are stored using a file extension **.html**. HTML files should be stored in the same directory as the compiled code of the applets.

A Web page is marked by an opening HTML tag **<HTML>** and a closing HTML tag **</HTML>** and is divided into the following three major sections:

1. Comment section (Optional)
2. Head section (Optional)
3. Body section

Comment section: It include comments that tell us what is going on the web page. A comment line begins with a **<!** and ends with a **>.** Web browers will ignore the text enclosed between them. Note that comments are optional and can be included anywhere in the web page.

```
<!
        This page includes a welcome title..... .. . .
>
```

Head section: The head section is defined with a starting <HEAD> tag and a closing tag </HEAD> tag. This section usually contains a title for the web page

```
<HEAD>
        <TITLE> WELCOME TO JAVA     </TITLE>
</HEAD>
```

Body section: This section contains the entire information about the webpage and its behavior. We can set up many options to indicate how our page must appear on the screen.

```
<BODY>
<CENTER>
        <H1> WELCOME TO THE APPLET PROGRAMMING </H1>
</CENTER>                                   <BR>
        <APPLET>            ----------      </APPLET>
</BODY>
```

Note that <CENTER> tag makes sure that the text is centered and <H1> tag causes the text to be of the largest size. We may use other heading tags <H2> to <H6> to reduce the size of letters in the text.

12. Creating Applet Tag:

To run applet once an applet has been compiled, it is included in an HTML file using the APPLET tag. let's, you must create an HTML file with the **applet tag** to specify the applet byte code file, the applet viewing are dimension (width and height), and other associated parameters.

The syntax of the <applet> tag is:

<applet code=classfilename.class

width=applet_viewing_width_in_pixels

height=aplet_viewing_height_in_pixels

[archieve=archivefile]

[codebase=applet_url]

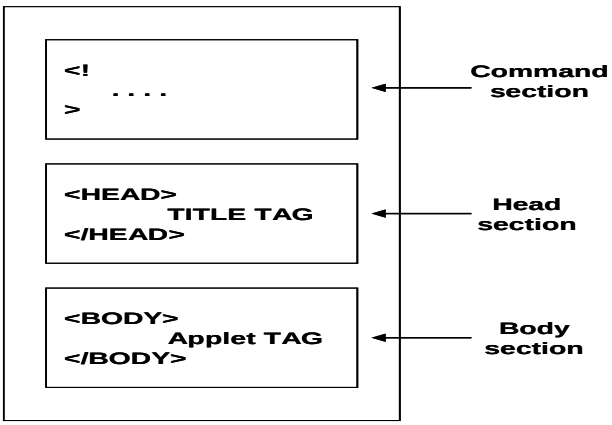
[vspace=vertical_margin]

[hspace=horizontal_margion]

[align=applet_alignment]

[alt=alternative_text] >

</applet>



The code, width, and height are required attributes; all others are optional.

- **Archive**: use this attribute to instruct the browser to load an archive file that contains all the class files needed to run the applet.
- **Code base**: this attribute is used if your applet is located in a different directory from the HTML page, you must specify the applet_url for the browser to load the applet. This attribute enables you to load the class from anywhere on the internet.
- **Vspace and Hspace**: These two attributes specify the size of the blank margin to leave around the applet vertically and horizontally in pixels.
- **Align**: This attribute specifies how the applet will be aligned in browser. One of nine values is used : left, right, top, texttop, middle, absmiddle, baseline, bottom, and absbottom.
- **Alt:** This attribute specifies the text to be displayed incase the browser cannot run java.

Ex: < applet code=”myapplet” width=200 height=300 ></ applet >

This applet tag that will run a applet called **myapplet** in window that is **200** pixels **wide** and **60** pixels **hight**.

Example program:-

Myapplet.java

```
import java.applet.*;
import java.awt.*;
public class Myapplet extends Applet
{
   String str;
   public void init()
   {
         str = "This is my first applet";
   }
   public void paint(Graphics g)
   {
         g.drawString(str, 50,50);
   }
}
```

Sample.html

```
<HTML>
<BODY>
   <applet code="Myapplet" height="200" width="200">      </applet>
</BODY>
</HTML>
```

Executing The Applet:-

To run an applet we required one of the following tools.

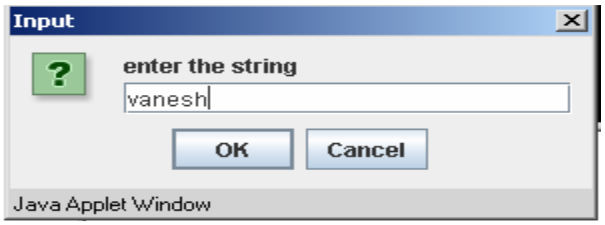
a. **Applet viewer** is a command line program to run Java applets. It is included in the SDK. It helps you to test an applet an **applet** before you run it in a browser.
b. An applet is a special type of application that's included as a part of an HTML page and run within a web browser. Then the browser executes that code and displays the output..

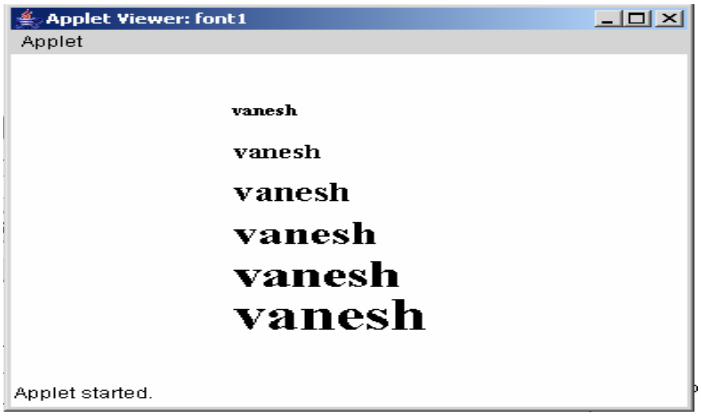
Example:- After building the program, run the applet and the applet viewer as shown below. When we run the applet viewer it will display the window as shown below.

Example-2:

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
    /*<applet code=font1 width=400 height=400></applet>*/
public class font1 extends JApplet
{
String s1=JOptionPane.showInputDialog("enter the string ");
public void paint(Graphics g)
{
int k=12,y=50;
for(int i=0;i<s1.length();i++)
{
g.setFont(new Font("Times New Roman",Font.BOLD,k));
g.drawString(s1,125,y);
y=y+35;
k=k+5;
}
}
```

Output:





14. Passing parameters to applets:

Each Parameter is passed to applets in NAME and VALUE pairs in <PARAM> tags between the opening and closing APPLET tags.

Syn: <param name=param_name1 value=param_value1>

<param name=param_name2 value=param_value2>

……

<param name=param_name n value=param-valuen>

Ex: <PARAM NAME = text VALUE = "Programming With Java"

1. Include appropriate <PARAM…> tags in the HTML document.
2. Provide code in the applet to parse these parameters.

Parameters are passed on an applet when it is loaded. We can define the **init()** method in the applet to get hold of the parameters defined in the **<PARAM>** tags. This is done using the **getParameter** method of the java.applet.Applet class., which takes one string argument representing the name of the parameter and returns a string containing the value of the that parameter.

To setup and handle parameters we need two things.

1) Include appropriate parameters in the html document.
2) Provide code in the applet to pass these parameters to an applet when it is loaded.

This code defines inside the **init()** method. That is done by using the **getParameter() method.** This takes one string argument representing the **name** of the parameter and returns a string containing the **value** of the parameter.

Syn:- getParameter("parameter name");

String s=getParameter("color");

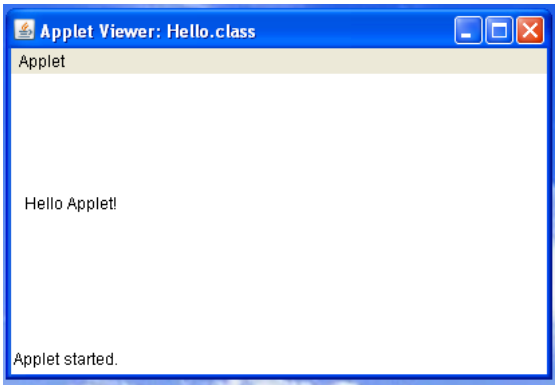
Ex: program for taking input parameter from user.

```
import java.awt.*;
import java.applet.*;
public class Hello extends Applet
{
  String s;
  public void init()
  {
```

```
    s=getParameter("X");
    if (s==null)
      s="Java";
    s="Hello "+s;
   }
   public void paint(Graphics g)
   {
    g.drawString(s,10,100);
   }
}
```

Let us create HTML file that contains this applet. Web page that passes a parameter whose NAME is string and whose VALUE is "**Applet**" to the applet **Hello**.

```
<HTML>
  <HEAD>
    <TITLE> WELCOME TO JAVA APPLETS </TITLE>
  </HEAD>
  <BODY>
    <APPLET CODE=Hello.class
          WIDTH=400
          HEIGHT=200>
      <PARAM NAME="X"
        VALUE="Applet!">
    </APPLET>
  </BODY>
</HTML>
```



Save this file as Hello.html and then run the applet using appletviewer as follows:

appletviewer Hello.html

Aligning the Display:

We can align the output of the applet using the ALIGN attribute. This attribute can have one of the nine **LEFT, RIGHT, TOP, TEXT TOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM,** and **ABSBOTTOM.**

```
<HTML>
<HEAD>
<TITLE> SUM OF TWO NUMBERS
```

```
        </TITLE>
    </HEAD>
    <BODY>
        <APPLET CODE=Num.classWIDTH=300  HEIGHT=300 ALIGN =RIGHT>
        </APPLET>
    </BODY>
    </HTML>
```

Displaying Numerical values: We can display numerical values by first converting them into string and then using the **drawstring()** method of **Graphics** class. We can do this easily by calling the **valueOf()** method of **String** class.

Step 1: Type and save the program (.java file)

```
import java.awt.*;
import java.applet.*;
public class Num extends Applet
{
public void paint(Graphics g)
{
int a=10;
int b=20;
int sum=a+b;
String s="SUM="+
String.valueOf(sum);
g.drawString(s,100,100);
}
}
```

Step 2: Compile the applet (.class file)

Step 3: Write a html document (.html file)

Num.html file

```
<HTML>
<HEAD>
    <TITLE> SUM OF TWO NUMBERS </TITLE>
</HEAD>
<BODY>
    <APPLET CODE=Num.class WIDTH=300 HEIGHT=300> </APPLET>
```

```
</BODY>
</HTML>
```

Step 4: Use the appletviewer to display the results

Example: Java program for arithmetic calculations using applets

```
import javax.swing.*;
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*<applet code="Appletcaliculator.class" width=300 height=200></applet>*/
public class Appletcaliculator extends Applet implements ActionListener
{
Label l1,l2,l3;
TextField t1,t2,t3;
Button b1,b2,b3,b4;
public void init()
{
l1=new Label("enter the first value: ");
l2=new Label("enter the second value: ");
l3=new Label("result: ");
t1=new TextField(15);
t2=new TextField(15);
t3=new TextField(15);
b1=new Button("Add");
b2=new Button("Sub");
b3=new Button("Mul");
b4=new Button("Div");
add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(t3);
add(b1);
add(b2);
```

```
add(b3);
add(b4);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
String s1=t1.getText();
String s2=t2.getText();
int x=Integer.parseInt(s1);
int y=Integer.parseInt(s2);
if(e.getSource()==b1)
{
int z=x+y;
t3.setText(String.valueOf(z));
}
if(e.getSource()==b2)
{
int z=x-y;
t3.setText(String.valueOf(z));
}
if(e.getSource()==b3)
{
int z=x*y;
t3.setText(String.valueOf(z));
}
if(e.getSource()==b4)
{
int z=x/y;
t3.setText(String.valueOf(z));
}
} }
```

Q:Explain Database Servers?

A database server runs a database management system and provides database services to clients. The server manages data access and retrieval and completes clients' requests. A database server is a server which uses a database application that provides database services to other computer programs or to computers, as defined by the client–server model.

Database Management System (DBMSs) frequently provide database-server functionality, and some database management systems (such as MySQL) rely exclusively on the client–server model for database access. Users access a database server either through a "front end" running on the user's computer – which displays requested data – or through the "back end", which runs on the server and handles tasks such as data analysis and storage.

Ex:

- Oracle RDBMS
- IBM DB2
- Microsoft SQL Server
- MySQL
- FileMaker
- Microsoft Access
- PostgreSQL
- DB2
- phpMyAdmin
- SQL Developer

Q: Write a short note on Database Clients

A client system is that who sends the request to the server system and the server system has to response the request as result. A client system is managed by users whereas the server system is managed by a computer expert. No need of a computer expert in client system. The client system is a system on which the results are prepared and displayed whereas the server system is a system in which the information is prepared for the client.

Q: Explain about JDBC.

JDBC:

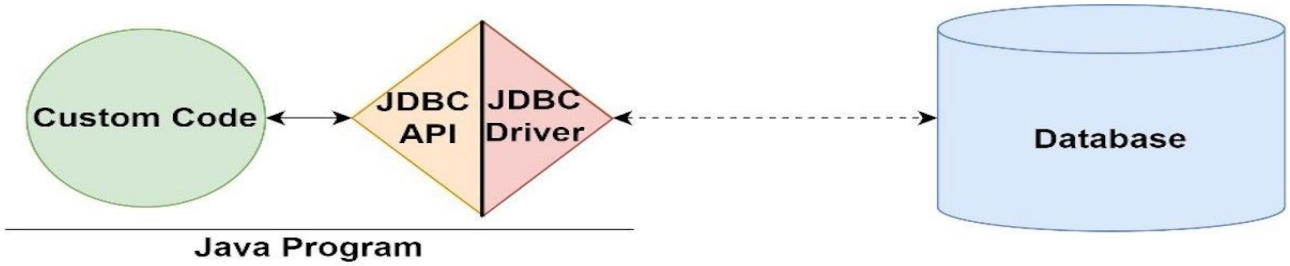
JDBC (Java Database Connectivity) is the Java API that manages connecting to a database, issuing queries and commands, and handling result sets obtained from the database. Released as part of JDK 1.1 in 1997, JDBC was one of the earliest libraries developed for the Java language.

JDBC acts as a bridge from your code to the database, as shown in Figure 1.

JDBC's architecture

The JDBC interface consists of two layers:

1. The JDBC API supports communication between the Java application and the JDBC manager.
2. The JDBC driver supports communication between the JDBC manager and the database driver.



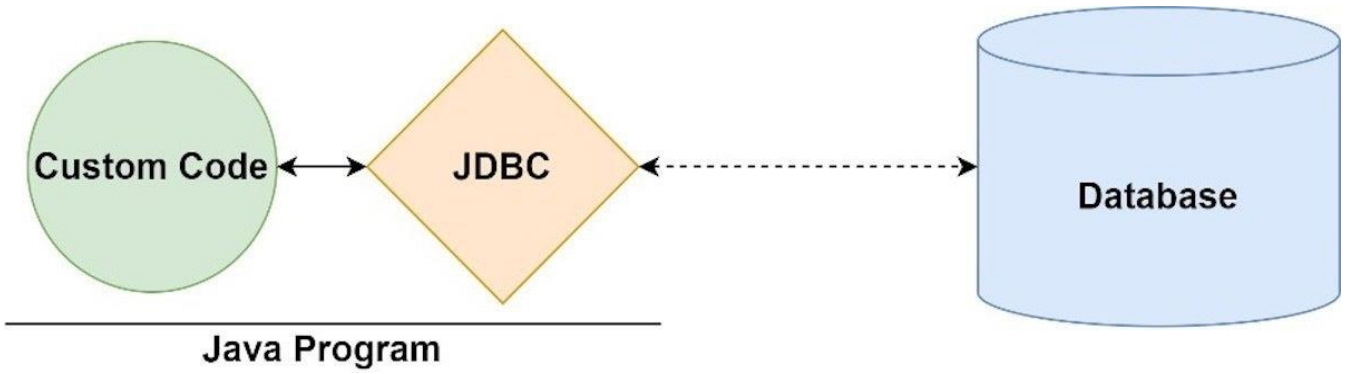
JDBC drivers

As an application programmer, you don't need to immediately be concerned with the implementation of the driver you use, so long as it is secure and official. However, it is useful to be aware that there are four JDBC driver types:

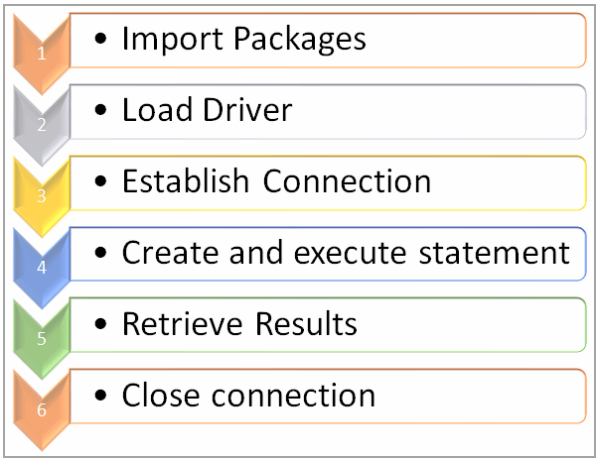
1. **JDBC-ODBC bridge driver:** A thin Java layer that uses an ODBC driver under the hood.
2. **Native API driver:** Provides an interface from Java to the native database client.
3. **Middleware driver:** A universal interface ("middleware") between Java and the RDBMS's vendor-specific protocol.
4. **Pure Java driver:** A driver that implements the vendor-specific protocol directly in Java.

Q: What are different Stages in a JDBC Program? Explain

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:



1. Import the Packages
2. Load the drivers using the forName() method
3. Register the drivers using DriverManager
4. Establish a connection using the Connection class object
5. Create a statement
6. Execute the query
7. Close the connections



Step 1: Import the Packages

Step 2: Loading the drivers

In order to begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of two ways mentioned below as follows:

2-A Class.forName()

Here we load the driver's class file into memory at the runtime. No need of using new or create objects. The following example uses Class.forName() to load the Oracle driver as shown below as follows:

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

2-B DriverManager.registerDriver()

DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. The following example uses DriverManager.registerDriver()to register the Oracle driver as shown below:

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

Step 3: Establish a connection *using* the *Connection class object*

After loading the driver, establish connections as shown below as follows:

Connection con = DriverManager.getConnection(url,user,password)

- **user: U**sername from which your SQL command prompt can be accessed.
- **password:** password from which the SQL command prompt can be accessed.
- **con:** It is a reference to the Connection interface.
- **Url**: Uniform Resource Locator which is created as shown below:

String url = " jdbc:oracle:thin:@localhost:1521:xe"

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where a database is stored, 1521 is the port number and xe is the service provider. All 3 parameters above are of String type and are to be declared by the programmer before calling the function. Use of this can be referred to from the final code.

Step 4: Create a statement

Once a connection is established you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

Use of JDBC Statement is as follows:

Statement st = con.createStatement();

Note: *Here, con is a reference to Connection interface used in previous step .*

Step 5: Execute the query

Now comes the most important part i.e executing the query. The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

- The query for updating/inserting a table in a database.
- The query for retrieving data.

The executeQuery() method of the **Statement interface** is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.
The executeUpdate(sql query) method of the Statement interface is used to execute queries of updating/inserting.

Step 6: Closing the connections

So finally, we have sent the data to the specified location and now we are on the verge of completing our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is shown below as follows:

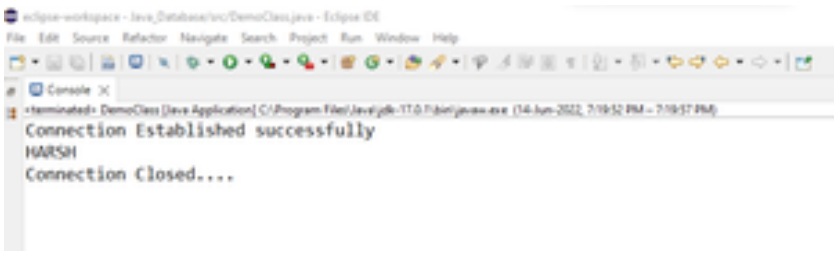
con.close();

Q: Explain briefly Retrieving Data from MySQL Database.

```
import java.io.*;
import java.sql.*;
 class GFG {
   public static void main(String[] args) throws Exception
   {
      String url  = "jdbc:mysql://localhost:3306/studinfo"; // table details
      String username = "sgcsrc";
      String password = "12345";
      String query = "select *from students";
      Class.forName( "com.mysql.cj.jdbc.Driver");
      Connection con = DriverManager.getConnection(url, username, password);
      System.out.println( "Connection Established successfully");
```

```
        Statement st = con.createStatement();
        ResultSet rs   = st.executeQuery(query);
        rs.next();
        String name   = rs.getString("name");
        System.out.println(name);
        st.close();
        con.close();
        System.out.println("Connection Closed....");
    }
}
```

Output:



name of the student is retrieved from database

Q: Explain briefly Retrieving Data from Oracle Database.

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

1. **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name
3. **Username:** The default username for the oracle database is **system**.
4. **Password:** It is the password given by the user at the time of installing the oracle database.

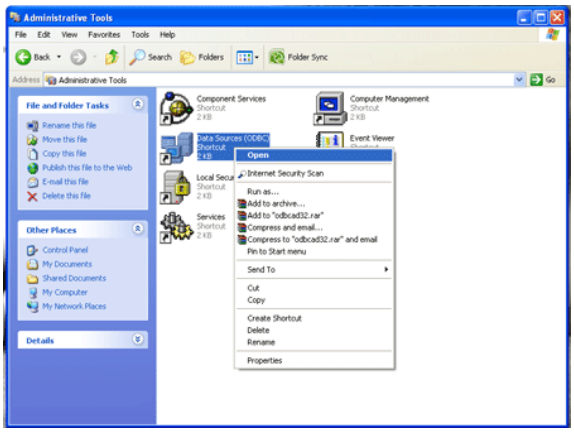
```
import java.sql.*;
class OracleCon
{
public static void main(String args[])
{
        try
        {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con;
```

```
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","
oracle");
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}
catch(Exception e)
{
 System.out.println(e);
}

}
}
```

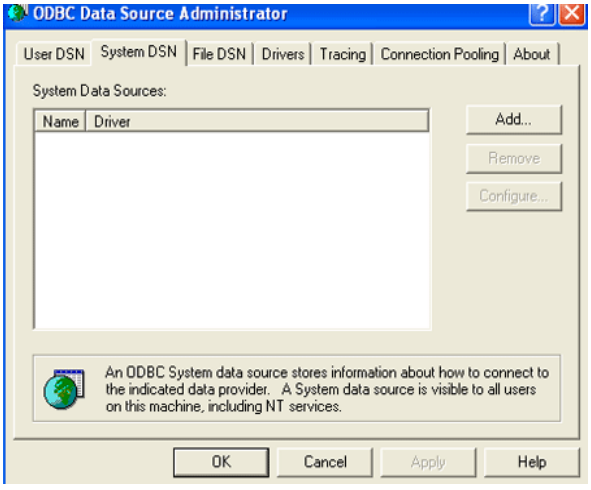
Q: Explain briefly Retrieving Data from MS Access Database.

Different step for creating DSN (datasource name) and adding it to our database.

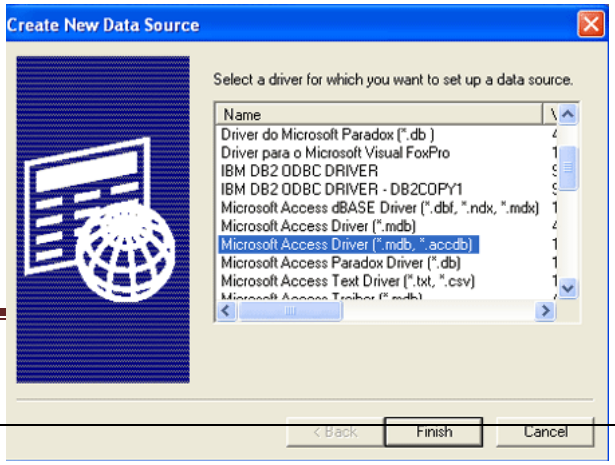
- Go to the Control Panel and select the Administrative Tool
- Select and open the Data Sources (ODBC)



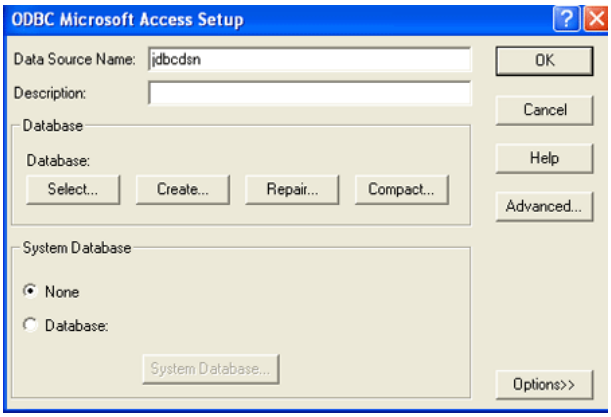
- A window, ODBC Administrator, will open then select the System DSN menu and click Add button



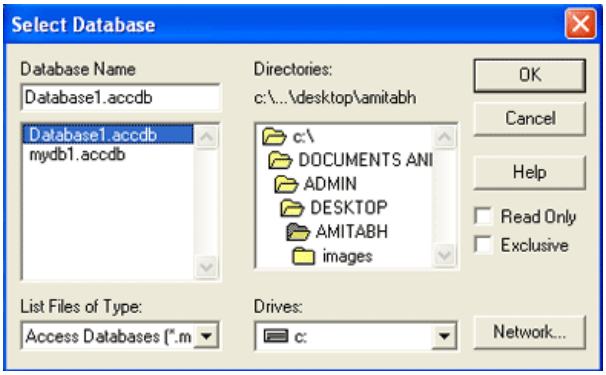
- A new window, Create a new Data Source will open then select the Microsoft Access Driver, as shown below



- After this a new window ODBC Microsoft Access Setup will open then write the DSN and select the Select button



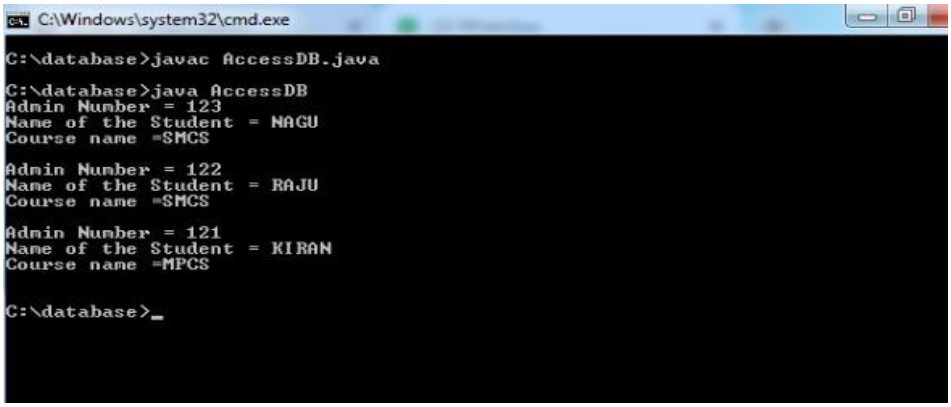
- A new window Select Database will open select your database and click on OK button



Now we are ready to create any table and insert the values in that table.
Now write the code for retrieving data from the above database

```
import java.sql.*;

class AccessDB

{

        public static void main(String args[])

        {

                try

                {

                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

                Connection con=DriverManager.getConnection("jdbc:odbc:","","");

                Statement st=con.createStatement();

                ResultSet rs=st.executeQuery("select * from studinfo");
```

```
                while(rs.next())
                {
                System.out.print("Admin Number = "+rs.getString(1));
                System.out.print("\nName of the Student = "+rs.getString(2));
                System.out.print("\nCourse name ="+rs.getString(3)+"\n\n");
                }
                }
                catch(Exception e){System.out.print(e);}
        }
}
```



Q: What is DSN?

Data Source Name is a name given to the database to identify it in the Java program. The DSN is linked with the actual location of the database.

Q: What is stored Procedure?

A stored procedure represents a set of statements that is stored and executed at database server sending the results to the client.

Q: What is the use of the callable statement?

Callable statement is useful to call stored procedure and functions which run at a database server and get the results into the client.

Q: Explain different types of ResultSet Types

In default, we can iterate the data/values in ResultSet which have returned as an output of the executed SQL statement in the forward direction. We can iterate the values in other directions using Scrollable ResultSet. We can specify the type and concurrency of ResultSet while creating Statement, PreparedStatement, and CallableStatement objects.

There are 3 types in ResultSet. They are:

1. **TYPE_FORWARD_ONLY:** It is the default option, where the cursor moves from start to end i.e. in the forward direction.
2. **TYPE_SCROLL_INSENSITIVE:** In this type, it will make the cursor to move in both forward and backward directions. If we make any changes in the data while iterating the stored data it won't update in the dataset if anyone changes the

data in DB. Because the dataset has the data from the time the SQL query returns the Data.

3. **TYPE_SCROLL_SENSITIVE:** It is similar to TYPE_SCROLL_INSENSITIVE, the difference is if anyone updates the data after the SQL Query has returned the data, while iterating it will reflect the changes to the dataset.