CORE JAVA

SYLLABUS

Part-1

1) What is programming language?
2) What is JDK/JRE/JVM?
3) What is platform independent?
4) What is token and types of tokens?
5) What is variable?
6) What is data type?
7) Scope of variable
8) Typecast Operator
9) Operators
10) Methods
11) Dynamic read
12) Control statement
13) Loops

Part-2

14) Static context
15) Non static context
16) OOPS
17) Encapsulation
18) Relationship
19) Polymorphism

20) Abstraction
21) Interface

Part-3

22) Object class
23) String class
24) Arrays
25) Exception handling
26) Collection

What is java?

1) Java is programming language and computing platform.
2) Java is developed by James a. gosling in 1995.
3) And its java released by Sun Micro System.
4) And now taken by ORACLE.

Why java?

1) Most widely used programming language.
2) Java is used in real world application'
3) Easy to code.
4) Flexibility.
5) Java development is collaborative.

American Standard Code for Information Interchange

1) ASCII is used as a method to give all computers the same language, allowing them to share documents and files. ASCII is important because

the development gave computers a common language.

Programming language

1) A language of medium which is used to instruct. A computer to perform a specific task is known as language.
2) A language which is understandable by the computer.(In simple words)

Types of programming language

1) Machine level language.
2) Assembly level language.
3) High Level language.

Machine level language/Low level language

1) The language which is easily understandable by the machine.
2) I.e. it is processor understandable language.
3) E.g. 0 to 1 (instruction return only by o & 1)

Assembly level language

1) It is a one more level higher to the machine language.
2) i.e. it consist of the predefined word called as Mnemonics (SUB ADD MUL MOV)
3) E.g. instruction set 8086.

4) Also called as mid-level language.

Higher level language

There are two type of language.

1) Object oriented language

   E.g. Java, c++ etc...

2) Procedure oriented language.

   E.g. c etc...

Java

1) Java is a high level programming language.
2) mostly in used current IT industry.
3) It can be run on any platform.
4) It is very flexible.

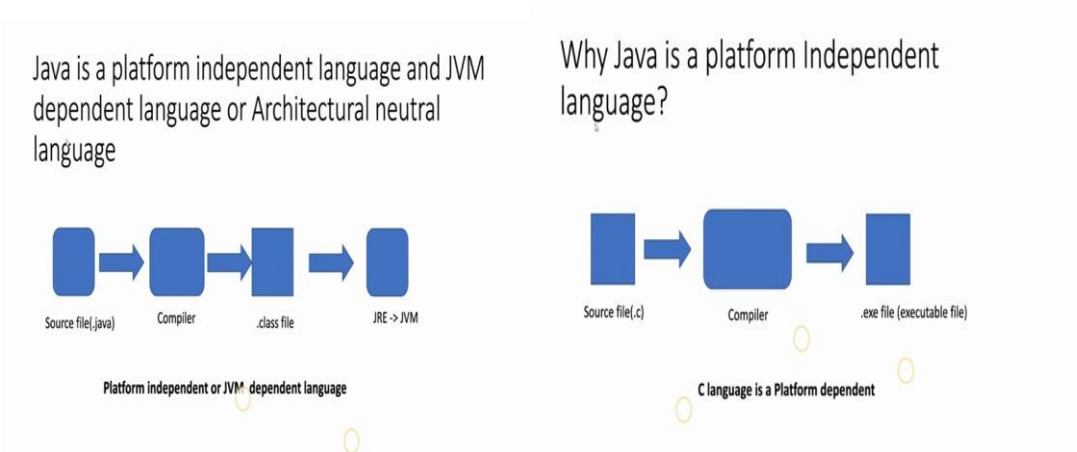
Unique features of java

1) Easy to learn.
2) High level language.
3) Object oriented language.
4) Highly case sensitive language.
5) Platform independent.
6) Class based language.
7) Multithreading language.
8) Thriving language.

What is platform independent?

1) Java compiler does not directly convert HLL to LLL language.

2) Instead of that it will convert into intermediate language called Byte code file.
3) The file is known as class file with extension .class
4) Once the class file ready it can be executed in any machine which has JVM (Java Virtual Machine)
5) This design makes java is platform dependent but jvm dependent language.



JDK (Java Development Kit)

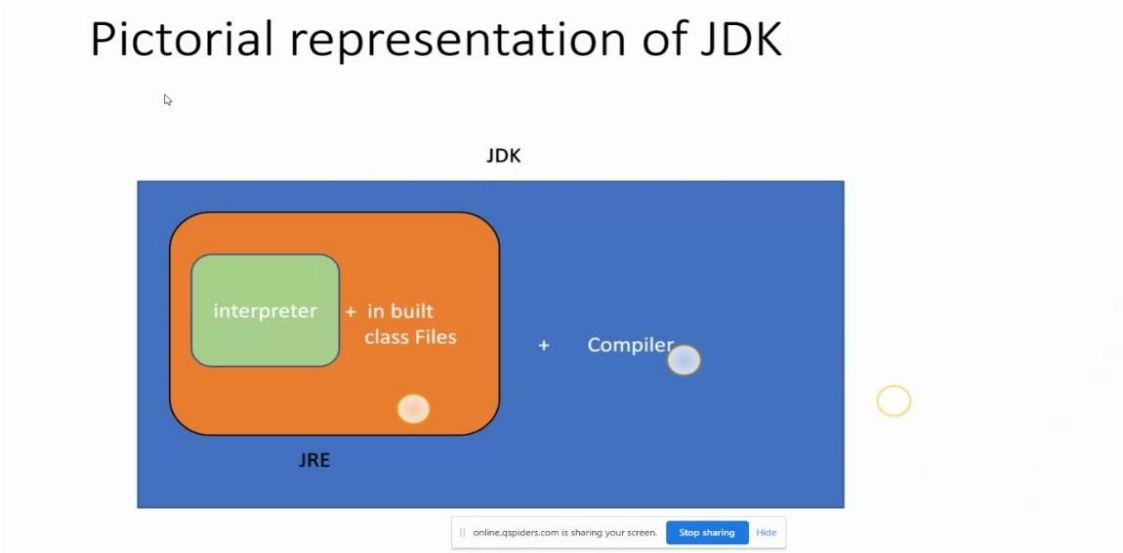
It is an important component developed by java community which consists of development tool such as java compiler as well as the java runtime environment (JRE) which is required for the execution on the java class file.

JRE (Java Runtime Environment)

1) JRE consist of java virtual machine (JVM) built in the class file.
2) And built in the class file or the library files are required for the execution of the java files.

JVM (Java Virtual Machine)

1) Responsible to convert byte code of instructions line by line into machine understandable language it will be executed.
2) In other words, it consists of an interpreter.



Source File

1) The file where the programmer writes the code is known as a source file.
2) The extension of the source file is .java.
3) The source file will not be shared with the clients or with the test engineer.

Class file

1) Once the source file is compiled then the generated file will be a class file.
2) The extension of the class file is .class.
3) The .class file contains the byte Code.

Intermediate Language

1) The language which is neither understandable by the programmer nor by the Processor or the machine is known as the Intermediate language.
2) Byte code is an intermediate language.

Compiler

1) The tool which is used to convert the programmer written code or the source file into machine understandable language is known as a compiler.
2) C compiler will convert source file to machine understandable language.
3) Where the java compiler will convert source file to the byte code or intermediate language.

Interpreter

1) The tool which will reads line by line and converts each and every line to machine understandable language is known as an interpreter.
2) Java interpreter will convert byte code to machine understandable code.

Note

1) Compilation is done by Compiler.
2) Execution is done by Interpreter.

Command prompt

1) It is the basic interface which is used to execute the java programs.

2) It is a command line User Interface. Where the commands are used for interfacing.
3) All the java programs are executed and compiled using Command prompt.
4) We use IDE for the oops concepts only

File Structure

1) In windows the hard disk can be divided into multiple portions and each portion is known as a Drive.
   Ex: C:, D:, E: F: etc....

Subdirectories or Subfolders

1) The directories which are present inside the other directory is called as the subdirectories.

User Home Directory

1) The folder that is created in the user name is called as user home directory.
2) Whenever we open command prompt by default we will be in home directory.

Directories

1) The directories are nothing but the folders which are used to store the data, documents, files, videos, songs etc..
2) Any name can be given to the directories.
3) It should not contain the space.

Working Directories

1) The folder which is currently use is known as Working Directories
2) Current working direction.

Path

1) It is used to travel from one directory to another directory.
2) There are 2 types of path:
    a. **Absolute path**
        The path which starts from drive name is known as absolute path.
        It is not dependent on working directory.

    b. **Relative path**
        The path which starts from directory name is known as relative path.
        It is always dependent on working directory.

Basic Commands of Command Prompt

Cd

1) Change Directory
2) This command is used to change the directory.

md/mkdir

1) Make Directory
2) This command is used to create a directory.

cls

1) Clear the screen.

2) This command is used to clear the screen.

dir

1) Directory.
2) This command is used to list all the files and directories present in the working directory.

Important Commands

javac

1) It is used to compile the java source file.
2) We should use source name as the input to this command.

   Ex: java fileName.java

java

1) It is used to execute the java class file.
2) We should use java class file name as an input this command.

   Ex: java filename

Note

Javac

It is a compiler (inbuilt compiler)

Java Types

Standard Edition (J2SE)Java

Used to develop stand-alone applications.

Ex: Excel, Adobe etc...

Java Enterprise Edition (J2EE)

Used to develop Web Applications (Client server applications).
Ex: Amazon, Swiggy, Flip kart etc.

Java Micro Edition (J2ME)

Used to develop Mobile based applications.
Ex: all mobile applications.

Structure of Java

```
class Class_Name
{
    public static void main(String[] args)
    {
        Statements;
    }
}
```

Class

1) A class is the blueprint from which individual objects are created. Class is a one which contains states (variables) and behavior (methods)
2) Every class in the java must have a file name and it is known as a class name.
3) Every class has a block and it is known as a class block.
4) Java instructions are always written inside the class only.

Members of the class

1) Inside class, we can create 3 members and they are
   - Variable
   - Methods
   - Initializers
2) Only inside the class block we can create these members.

Variables

A variable is a container which is used to store the data.

Methods

It is a block of instructions which is used to perform a task.

Initializers

Startup the instruction which are used to execute

NOTE

A class in java can be executed only if main method is created

FOR EXAMPLE

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

```
        System.out.println("hello world");
        }
    }
```

Note

1) We can create a class without main method,
2) But it is compile time successful and the class file is generated but we can't execute the class file.

```
Class Class_name
{
}
```

Syntax

The set of rules that are to be followed while writing the code By the programmer.

Compile time error

The error that is occurred during the compilation i.e. when the class files are not being generated.

Runtime error

The error occurred during runtime or during execution is known as runtime error.

Println

1) Syntax

```
System.out.println(data);
System.out.println();
```

2) println statement is used to print the data as well as to print the new line character.
3) We can use the println statement without passing any data also then it will just execute the new line character

Print

1) System.out.println(data);
2) Print statement is used to print the data.
3) We cannot use the print statement without passing any data if v use we will get compile time error

Tokens

1) The smallest unit of programming.
2) Used to compose the instructions
3) There are 3 tokens
    a. Keywords
    b. Identifiers
    c. Literals/data/values.

Keyword

1) A predefined word which the java compiler can understand is known as Keywords.
2) Every keyword in java is associated with a specific task.
3) A programmer can't change the meaning of the keyword.

Ex: we have 53 keywords. Class, public, static, void etc...

Rule

The keywords are written in lowercase.

Identifiers

1) The name given to the components of java by the programmer is known as identifiers.
2) Components are nothing but class, method, variables, interfaces/object etc...
3) The programmer should follow rules and conventions for identifiers.

Rules of identifiers

1) It should never start with a number.
2) It should not have special characters except underscore and dollar. (__ and $)
3) Character space is not allowed.
4) We can't use keywords as identifiers.

Conventions

The rules of coding on industrial standards to be followed by the programmer are known as conventions.

Note

1) Compiler does not validate the convention therefore; if conventions are not followed also we won't get any compile time error.
2) It is not compulsory but it is highly recommended to follow the conventions.

Convention for Class Name

Single Word

The first letter should be in upper case, and the remaining should be in lower case.
Ex: Addition, Statistics etc..

Multi Word

The first letter of each word should be in upper case and the remaining should be in lower case.
Ex: Addition Program, Statistics Program Of Semester etc..

Literals/ data/ values

1) The values or the data that we use in our programs are called as literals.
2) There are 2 types of literals
    a. Primitive value.
    b. Non Primitive values

Primitive values

Single value of data is called as a primitive value.
Ex: character, Boolean, number etc...

Character

Anything enclosed in single quotes are called as character.

Boolean

Which will return Boolean values i.e. true or false?

Number

Which consists of integer number (int) and floating number (float) or decimal values?

Non Primitive values

Multi value data are called as non-primitive data (group of data)

Ex: String, ClassName, Object Reference.

String Literal

Data Type	Default Value	Default size
Char	'\u0000'	2 byte
Boolean	false	1 bit
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

1) Anything enclosed with in the double quotes is called as a string.
2) The length can be anything and the main thing they are case sensitive.
3) Ex: "hello", "1", "welcome@123" etc..

Data Types

1) Data types specify the different sizes and values that can be stored in the variable.
2) Data types are used to create variables of a specific type.
   There are 2 types of data types:

Primitive data types

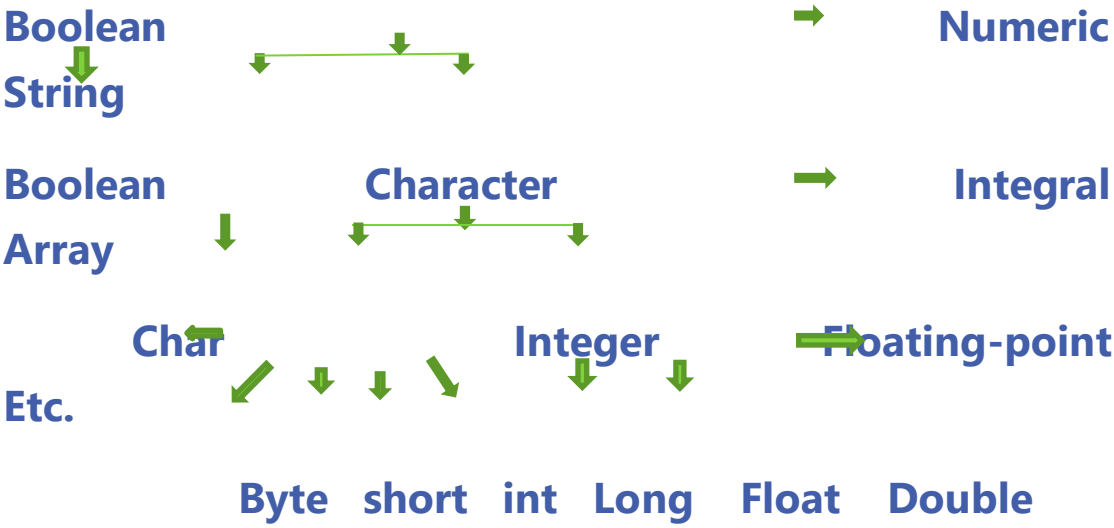
1) The primitive data types include Boolean, char, byte, short, int, long, float and double.
2) The data type which is used to create a variable to store primitive values such as numbers, characters, Boolean is known as primitive data type

Non-primitive data types

The non-primitive data types include Strings, Arrays etc.

Data types in Java

Primitive Non-Primitive



NOTE

Every class name in java is non-primitive data type.

Note

(Lowest) Byte < Short < Int < Long < Float < Double (Highest)

Primitive variable

1) The variable which is used to store a primitive value such as numbers, character, boolean is known as a primitive variable
2) Syntax to create primitive variable
   Datatype variable=data;
3) EXAMPLE: int a=10;

Non primitive variable

1) The variable which is used to store a reference is known as Non-primitive variable.
2) It is also known as a reference variable
3) Syntax to create a non-primitive variable
   Non Primitive data type=identifier1;
4) EXAMPLE : String s "h=i";

Scope of a Variable

1) The visibility of a variable is known as the scope of the variable.
2) Based on the scope of the variable, we can categorize variables into 3 types
   a. Local variables
   b. Static variables
   c. Non-Static variables

Local Variable

The variable declared inside the method block or any other block except the class block is known as a local variables.

Characteristics of a local variable

1) We cannot use local variables without initialization. If we try to use the local variables without initialization then we will get compile time error.

2) The local variables will not be initialized with the default values.
3) The scope of the local variable is nested inside the block. Whenever It is declared hence, it cannot be used outside the block.

```
class LocalProgram
{
        Public static void main (string[ ] args)
        {
                int a = 10;
                System.out.println("the value of a: "+a);
        }
}
```

Type Casting

1) The process of converting one type of data to another type of data is known as type casting.
2) We have 2 types of type casting they are
    a. Primitive typecasting
        i. Widening
        ii. Narrowing
    b. Non Primitive typecasting
        i. Upcasting
        ii. Downcasting

Primitive type casting

The process of converting one primitive value into another primitive value is known as primitive type casting.

a. Widening/Implicitly/Automation

1) The process of converting smaller range of primitive data type into larger type of primitive data type is known as Widening.
2) In widening there is no data loss.
3) Since there is no data loss **compiler can implicitly perform widening** therefore it is also called as Auto Widening
4) For Example

```
public class Test
{
    public static void main(String[] args)
    {
      int i = 100;
      long l = i; //no explicit type casting required
      float f = l;  //no explicit type casting required
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
    }

}
```

b. Narrowing/Explicitly

1) The process of converting large range of primitive data type into smaller range of primitive data type is known as Narrowing.
2) In narrowing there is possibility of data loss.

3) Since there is a possibility of data loss the compiler does not do narrowing implicitly.
4) Therefore it can be done explicitly by the programmer with the help of **type cast operator.**

For example

```
class CastingDemo1
{
    public static void main(String[] args)
    {
      double d = 120.04;
      long l = (long)d;
      int i = (int)l;
      System.out.println("Double value "+d);
      System.out.println("Long value "+l);
      System.out.println("Int value "+i);
    }
}
```

Operators

They are predefined symbols which are used to perform specific task on the given data.
the data that is given as the input to the operator is known as an operand.

Based on the no of operands

There are 3 types of operators based on the number of operands

1) Unary operator]
    a. Accepts only one operand.
2) Binary operator

Accepts 2 operands.

3) Ternary operator

Accepts 3 operands.

Classification of Operators based on the task

1) Arithmetic operator (binary)
2) Bitwise operator
3) Conditional operator
4) Assignment operator (binary)
5) Logical operator
6) Increment/decrement operator
7) Relational operator
8) Miscellaneous operator

Operators

1) Arithmetic Operator (+ - * / %)

```
class Arithmatic
{
    public static void main(String[] args)
    {
        int a=10;
        int b=5;
        System.out.println(a+b);
        System.out.println(a-b);
        System.out.println(a/b);
        System.out.println(a%b);
        System.out.println(a*b);
    }
}
```

```
F:\10>javac Arithmatic.java

F:\10>java Arithmatic
15
5
2
0
50

F:\10>
```

2) Assignment Operator (=)

```
class Assignment
{
    public static void main(String[] args)
    {
        int a=10;
        int b=5;
        System.out.println(a+=b);
        System.out.println(a-=b);
        System.out.println(a/=b);
        System.out.println(a%=b);
        System.out.println(a*=b);
        System.out.println(a + b + "JAVA");
    }
}
```

```
F:\10>javac Assignment.java

F:\10>java Assignment
15
10
2
2
10
15JAVA

F:\10>
```

3) Relational Operators (>, <, >=, <=, ! =, ==)

```
class Relational
{
    public static void main(String[] args)
    {
        int a=10;
        int b=20;
        System.out.println(a!=b);
        System.out.println(a==b);
        System.out.println(a<b);
        System.out.println(a>b);
    }
}
```

```
F:\10>javac Relational.java

F:\10>java Relational
true
false
true
false

F:\10>
```

4) Increment/decrement operator

```
class IncrementDecrement
{
    public static void main(String[] args)
    {
        int a=10;
        int b=20;

        System.out.println(a++);
        System.out.println(b--);
        System.out.println(++a);
        System.out.println(--b);
        System.out.println(++a + ++a);
        System.out.println(--a + --b + a++ + b
    }
}
```

```
F:\10>javac IncrementDecrement.java

F:\10>java IncrementDecrement
10
20
12
18
27
60JAVA

F:\10>
```

5) Bitwise Operator (& ^ |)

6) Logical Operator (&&, ||, !)

7) Miscellaneous Operators (instance of, and typecast)

Conditional Operator

It is a ternary operator which consists of 3 operands.

Syntax

Operand1? Operand 2: Operand3;

Or

Condition? Statement 1: Statement 3;

Operation

1) The return type of operand 1 must be a Boolean.
2) If the condition is true statement 1 will get executed else statement 2 get executed.

Increment and Decrement operator

Increment operator (++)

1) Incrementing a value by one.
2) We have 2 types
    a. pre increment
    b. post increment

Decrementing Operator (--)

1) Decrementing the value by one.
2) We have 2 types
    a. pre decrement
    b. post decrement

Methods

1) A method is a block of instructions which is used to perform a specific task.
2) Whenever we call the method only then it will execute.
3) They are used to execute the instructions and pass the values.

Advantages of the methods

a. Reusability
b. easy modification

c. readability

Syntax

```
[accessmodifier][modifier]returnType
methodName([formal arguments])
{
    // Statements;
}
```

Parts of the method

Method Signature

The method signature consists of **method name** and **formal arguments**.

Method Declaration

The Method declaration consists of **access modifier**, **modifier**, and **return type** and **method signature**.

Method Definition/implementation

Method definition consists of the method declaration and the method body/ method block.

Terminologies

Modifiers

They are responsible to modify the characteristics of the members.

Ex: static, abstract, final, transient etc..

Access Modifiers

They are used to change the accessibility of the members and we have levels of access modifiers,

a. Private
b. Public
c. Default
d. protected

Access Modifiers/ Specifiers

Public

The method is accessible by all classes when we use public Modifiers in our application.

Private

When we use a private access Modifiers, the method is accessible only in the classes in which it is defined.

Protected

When we use protected access Modifiers, the method is accessible within the same package or subclasses in a different package.

Default

When we do not use any access Modifiers in the method declaration, Java uses default access Modifiers by default. It is visible from the same package only.

Terminologies:

Return Type

1) Return type is a data type which specifies what type of data is returned by the method after execution.
2) it is mandatory to specify what type of data returned by the method in the method declaration.
3) The method can have following return types
    a. Void
    b. primitive data types
    c. Non Primitive data types

Void

1) It is a type of return type
2) It is a key word
3) It is used when the method returns nothing.

Terminologies

Method name

1) The name given to the method is known as method name. Any name can be given to the method by following conventions.
2) The following are the method name conventions
    a. **Single word**

        All the letters in the word should be in lower case.

    b. **Multi word:**

        The first word should be in lower case and the second word first letter should be in upper case and the other letters should be in lower case.

Terminologies

Formal Arguments

A variable which is declared in method declaration is known as the formal arguments.

Important Concept

Called Method

The method which is called by the other method is called method.

Calling Method

The method which is calling the other method is known as the calling method.

Return statement

1) A method after the execution will return the data back to the caller with the help of the return statement.
2) Return is a keyword.
3) It is a control transfer statement.
4) When the return statement is executed, the execution of the methods terminated and control is transferred to the calling method.

Rule to use the return statement

1) The type specified as the return type should be same as the type of the value passed in the return statement.
2) The void will not return any data therefore when we declare the void as the return type in our method declaration then we should not use return statement in that method. If we use we will get the compile time error.

Types of methods

1) There are 2 types of methods:

a. No argument methods

The method which is not accepting any formal arguments is known as no argument method.

b. Parameterized methods

The method which has formal arguments is called as parameterized method.
They are used to accept data.

Rules to call a parameterized method

1) The number of the actual arguments should be same as the number of the formal arguments.
2) The number of data type of the corresponding actual argument should be same as the type of formal arguments.
3) Sequence of formal argument and actual argument should be same.
   If not the compiler tries implicit conversion; if not possible then we get compile time error.

Method call statement

1) The statement which is used to call a method is known as a method call statement.
2) We can call both the types of the method by using the method call statement.

Syntax

methodName([Actual arguments])
Actual arguments
The values that are passed into the method call statement are called as actual arguments.

Method call statement flow

1) Execution of calling method is paused.
2) Control is transferred to the called method.
3) Execution of the called method begins.
4) Once the execution of the called method is completed the control Is transferred back to the calling method.
5) Execution of the calling method resumes.

Main method and its purpose

1) The method which has the method name as main is called as the main method and main method is very important method.
2) The purpose of the main method are:
    a. To start the execution.
    b. To control the flow of execution.
    c. To end the execution.

Decision / Control Statement

The decision statement helps the programmer to skip the block of instructions from the execution if the condition is not satisfied.

Types of Decision Statements

1) if statement
2) if-else statement
3) if-else if ladder
4) switch

Syntax to create if statement

```
if (condition)
{
}
```

WORKFLOW

If the condition is satisfied then the instruction written inside the if block gets executed or normal flow of the execution continues (Instructions written inside the if block is skipped).

Syntax to create if-else statement:

```
if (condition)

{

}

Else

{

}
```

Workflow

If the condition is satisfied then the instruction written inside the if block gets executed if not

satisfied else block gets executed (any one of the blocks will be skipped based on a condition.)

Syntax to create if-else if statement:

```
if (condition)
{
}
else if (condition)
{
}
else if (condition)
{
}
else
{
}
```

Work Flow

If the condition is satisfied then the instruction written inside the if block gets executed if not satisfied, the condition is checked in the else if block from top to bottom order and if the condition is satisfied in any of the else if block then, only that else if block is gets executed if not satisfied else block gets executed remaining blocks are skipped

Syntax to create switch block

```
switch(value / variable / expression )
{
        case value / expression :
        {
        statement;
```

```
}
[break ;]
case value / expression :
{
statement;
}
[break ;]
default :
{
statement ;
}
[break ;]
}
```

Workflow

1) The value / variable / expression passed in the switch get compared with the value passed in the case from top to bottom order.
2) If any of a case is satisfied, the case block is executed and all the blocks present below get executed.
3) If no case is satisfied then the default block gets executed.
4) For a case we can use a break statement which is optional.

Note

1) For a switch we can't pass long, float, double, Boolean.
2) For a case we can't pass a variable.

Break

1) Break is a keyword; it is a control transfer statement.
2) break is used inside the switch and loop block
3) When the break statement is executed control is transferred outside the block.

Loop Statement

1) Loop statement helps the programmer to execute the set of instructions repeatedly
2) In java we have different types of loop statements, they are:
    a. while loop
    b. do-while loop
    c. for loop
    d. for each / advanced for / enhanced for
    e. nested loop

Syntax to create while loop

```
while(condition)
{
      Statement to be repeated ;
}
```

Workflow

CASE 1: When the condition is true

1) The loop continues.
2) Control executes the statement which belongs to the loop.

3) After execution once the loop block ends, control goes back to the condition and the entire process will be repeated till the condition becomes false.

CASE 2: When the condition is false

1) The loop is stopped i.e. repetition is stopped.
2) The loop block will not get executed.
3) The control comes outside the loop to the next statement.

Syntax to create do-while loop

```
Do
{
      //statement;
}
while(condition) ;
```

Workflow

CASE 1: When the condition is true

1) Control goes to the loop block directly, execute the instructions.
2) Then control goes to the condition, if the condition is true the control goes back to the do block.

CASE 2: When the condition is false

1) Control goes to the loop block directly, execute the instructions.
2) Then control goes to the condition, if the condition is false the loop is stops and control goes to the next statement.

Difference Between While and Do-While Loop

WHILE	DO - WHILE
First the condition is checked, if the condition is true then the loop block gets executed.	In do while, first the loop block gets executed and then the condition is checked.
The minimum iteration can be zero.	The minimum iteration is one.

EXAMPLE :
Int a = 5 , b = 10 , count = 0;
while(a>b)
{
count ++;
S.o.pln("value of b is "+b);
}
S.o.pln("Iteration " + count);
OUTPUT : Iteration 0
	EXAMPLE :
	Int a = 5 , b = 10,count=0 ;
	do
	{
	count ++;
	S.o.pln("value of b is "+b);
	}
	while(a>b);
	S.o.pln("Iteration " + count);
	OUTPUT : value of b is 10
	Iteration 1

Syntax to create for loop

```
for(initialization ; condition ; update)
{
// statement to be repeated
}
```

Workflow

1) control go to the initialization part.
2) Then it will go to the condition part.
3) If the condition is true then it will enter inside the loop blocks
4) Once the execution of instruction inside the loop block is completed control will go to the update segment.
5) Then it will go back to the condition. Step1,2,3,4 will continue until the condition become false.

Note

1) All the three segments are optional (Initialization, condition, update).
2) If the condition is not provided, by default it is considered as true.

Nested Loop

1) Writing a loop statement inside another loop statement is known as a nested loop statement.
2) In a nested loop, the inner loop is executed completely for each and every iteration of the outer loop.

Static

1) Static is a keyword.
2) It is a modifier.
3) Any member of a class is prefixed with a static modifier then it is known as a static member of a class.
4) Static members are also known as class members.

Note

1) Static members can be prefixed only for class members (members declared in a class).

Static Members

2) Static method
3) Static variable
4) Static initializers

Java Runtime Memory

1) To execute the java program a portion of memory in RAM is allocated for JRE.
2) In that portion of memory allocated , we have different range of memory, hence they are classified as follows,

    1. Method area
    2. Class static area
    3. Stack area
    4. Heap area

Method Area

All the method blocks will be stored in a method area (Instruction of the methods).

Class Static Area

1) For every class there is a dedicated block of memory is created in the class static area (static pool).
2) The static members of the class will be allocated inside the memory created for the class.

Stack Area

1) The stack area is used for the execution of instructions.
2) For every method that is under execution a block of memory is created in this stack area which is known as a frame.
3) Once the execution of a method is completed the frame is removed.

Heap Area

1) In a heap area, a block of memory is created for the instance of a class (Object).
2) Every block of memory is created with the help of reference.
3) All the non-static members of a class will be allocated inside this block of memory.
4) Therefore we can access the non-static member with the help of reference.

Static Method

1) A method prefixed with a static modifier is known as the static method.

Characteristics

1) Static method block is stored in the method area and reference of the static method is stored inside the class static area (static pool).
2) We can use the static method with or without creating an object of the class.
3) We can use the static method with the help of the class name.
4) A static method of the class can be used in any class with the help of a class name.

Static Context

1) The block which belongs to the static method and multi-line static initializer is known as static context.
2) Inside a static context, we can use the static members of the same class directly by using its name.
3) Inside a static context, we can't use the non-static members of the same or different class directly by using its name or by using its class name.
4) this keyword is not allowed inside the static context.

Static Variable

1) Variable declared in a class block and prefixed with static modifier is known as static variable.

Characteristics

1) It is a member of the class.
2) It will be assigned a default value.
3) Memory will be allocated inside the class static area.
4) It is global in nature; it can be used within the class as well as in different classes.
5) We can use a static variable with the help of the class name as well as with the help of object reference.
6) We can access the static variable from different classes directly with the help of the class name.

Note

1) If static variable and local variable are in the same name then we can differentiate static variable with the help of class name

Static Initializers

We have two types of static initializers. They are,

1. Single line static initializer
2. Multi-line static initializer

Single Line Static Initializer

Syntax to create single line static initializers

```
static data type variable = value / expression
;
```

MULTILINE STATIC INITIALIZER

Syntax to create multi line static initializers

```
static
{
      Statements ;
}
```

Characteristics

1) Static initializers get executed implicitly during the loading process of the class.
2) A class can have more than one static initializer they execute top to bottom order.

Purpose Of Static Initializer

1) Static initializers are used to execute the startup instructions.
2) As the static blocks get executed before the actual execution of the main method.

Class loading Process

1) A block is created for a class in the static pool; it can be accessed with the help of a class name.
2) All the method definitions are loaded in the method area and if the method is static then the reference of that method is stored inside the class block (class static area).
3) If the class has any static variables they are loaded in the class static area with a default value.
4) If the class has any static initializers they are executed from top to bottom order.
5) The loading process of the class is completed, then JVM will call the main method of the initial loading class.

Note

1) JVM will call only the main method of the initial loaded class.

Object

2) Any substance which has existed in the real world is known as an object.
3) Every object will have attributes and behaviors.

Object in Java

1) According to object-oriented programming, the object is a block of memory created in the heap area during the runtime; it represents a real-world object.
2) A real-world object consists of attributes and behavior.

3) Attributes are represented with the help of non-static variables.
4) Behaviors are represented with the help of non-static methods,

Class

1) According to real-world situations before constructing an object blueprint of the object must be designed, it provides the specification of the real-world object.
2) Similarly in object-oriented programming before creating an object the blueprint of the object must be designed which provides the specification of the object, this is done with the help of class.

Definition of Class

1) It is a user-defined non-primitive data type; it represents the blueprint of the real-world object.
2) The class provides the specification of real-world objects.

Note

1) We can create any number of objects for a class, it is known as an instance of a class.

Steps To Create an Object

1) STEP 1

Create a class or use an existing class if already created.

2) STEP 2
   Instantiation

Instantiation

1) The process of creating an object is known as instantiation.

Syntax to create an object

```
new className();
```

New

1) new is a keyword.
2) It is a unary operator.
3) It is used to create a block of memory inside a heap area during runtime.
4) Once the object is created it returns the reference of an object.
5) EXAMPLE
   Step1 Designing a class
   class Employee { String ename ; Int eId; }
   Step 2 Instantiation
   new Employee();

Non Primitive Data Type

1) Every class name in java is a non-primitive data type.

2) Non-primitive data types are used to create a non-primitive variable to store the reference of an object.
3) EXAMPLE

```
class Employee
{ String ename ; int eid ; }
Employee e = new Employee();
S.o.pln(e); // ox1
```

Non Static

1) Any member declared in a class and not prefixed with a static modifier is known as a non-static member of a class.
2) Non-static members belong to an instance of a class. Hence it is also known as an instance member or object member.
3) The memory for the non-static variable is allocated inside the heap area (instance of a class).
4) We can create any number of instances for a class.
5) Non-static members will be allocated in every instance of a class. NON

Static Members

1) Non-static variable
2) Non-static method
3) Non-static initializers
4) Constructors

Non Static Variable

1) A variable declared inside a class block and not prefixed with a static modifier is known as a non-static variable.

Characteristics

1) We can't use the non-static variable without creating an object.
2) We can only use the non-static variable with the help of object reference.
3) Non-static variables are assigned with default during the object loading process in.
4) Multiple copies of non-static variables will be created (once for every object).

Non Static Method

A method declared in a class block and not prefixed with a static modifier is known as a non-static method.

Characteristics

1) A method block will get stored inside the method area and a reference of the method is stored inside the instance of a class [object].
2) We can't call the non-static method of a class without creating an instance of a class[object].
3) We can't access the non-static method directly with the help of class names.

4) The non-static method can't be accessed directly with their names inside the static context.

Non Static Context

1) The block which belongs to the non-static method and multi-line non-static initializer is known as non-static context.
2) Inside a non-static context, we can use static and non-static members of the same class directly by using its name.

Non Static Initializers

1) Non-static initializers will execute during the loading process of an object.
2) Non-static initializers will execute once for every instance of a class created. [object created].

Purpose of Non Static Initializers

Non-static initializers are used to execute the startup instructions for an object

Types of Non Static Initializers

1) Single line non-static initializer
2) Multi-line non-static initializer

Single Line Non Static Initializer

Syntax to create single line non static initializers

datatype variable = value / reference

Multi Line Non Static Initializer

Syntax to create multi line non static initializers

```
{
   // statements ;
}
```

Note

1) All the Non-static initializers will execute from top to bottom order for every object creation.

This

1) It is a keyword.
2) It is a non-static variable it holds the reference of a current executing object.

Uses of This

1) Used to access the members of the current object.
2) It is used to give the reference of the current object.
3) Reference of a current object can be passed from the method using the 'this' keyword.
4) Calling a constructor of the same class is achieved with the help of this call statement.

Constructor

Constructor is a special type of non-static method whose name is the same as the class name but it does not have a return type.

Syntax to create the constructor

A programmer can define a constructor by using the following syntax

```
[access_modifier]className([Formal_Arguments])
{
   // initialization;
}
```

Constructor Body

1) A constructor body will have the following things
    a. Load instructions added by the compiler during compile time.
    b. Non static initializers of the class.
    c. Programmer written instructions.

Purpose of the Constructor

1) During the execution of the constructor
2) Non-static members of the class will be loaded into the object.
3) If there is a non-static initializer in the class they start executing from top to bottom order.
4) Programmer written instruction of the constructor gets executed.

Note

1) If the programmer fails to create a constructor then the compiler will add a default constructor.

Classification of Constructor

1) Constructors can be classified into two types based on the formal argument,
    a. No argument constructor
    b. Parameterized constructor

No Argument Constructor

1) A constructor which doesn't have a formal argument is known as a no-argument constructor.

Syntax to create no argument constructor

```
[access modifier] className()
{
    //code;
 }
```

Note

If the programmer fails to create a constructor then the compiler implicitly adds a no-argument constructor only.

Parameterized Constructor

The constructor which has a formal argument is known as parameterized constructor.

Purpose of the Parameterized Constructor

Parameterized constructors are used to initialize the variables (non-static) by accepting the data from the constructor in the object creation statement.

Constructor Overloading

1) If a class is having more than one constructor it is known as constructor overloading

Rule

1) The signature of the constructor must be different.

Constructor Chaining

1) A constructor calling another constructor is known as constructor chaining.
2) In java, we can achieve constructor chaining by using two ways
    a. this() (this call statement)
    b. super() (super call statement)

This()

1) It is used to call the constructor of the same class from another constructor.

Rule

1) this() can be used only inside the constructor
2) It should always be the first statement in the constructor.

3) The recursive call to the constructor is not allowed (Calling by itself).
4) If a class has n constructors we can use this statement in n-1 constructors only(at least a constructor should be without this()

Note

1) If the constructor has this() statement then the compiler doesn't add load instruction & non-static initializers into the constructor body.

Note

If the programmer fails to create a constructor then the compiler implicitly adds a no-argument constructor only

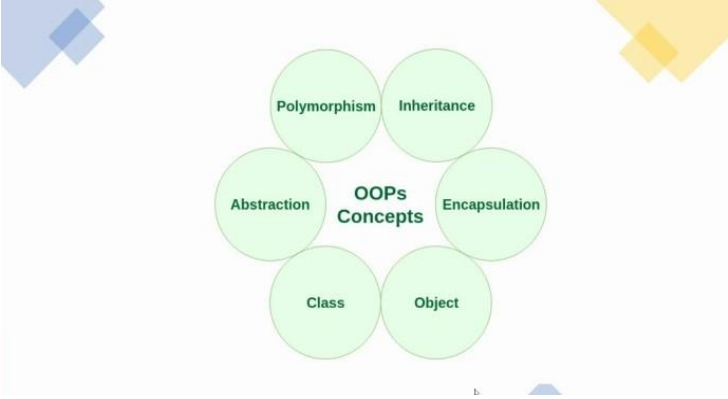
Loading Process of an Object

1) a new keyword will create a block of memory in a heap area
2) Constructor is called.
3) During the execution of the constructor,
    a. All the non-static members of the class are loaded into the object.
    b. If there are non-static initializers they are executed from top to bottom order
    c. Programmer written instruction of the constructors will be executed.
4) The execution of the constructor is completed
5) The object is created successfully.

6) The reference of an object is returned by the new keyword
7) These steps are repeated for every object creation.

PRINCIPLE OF Oops

1) Object-oriented programming has the following principles
    a. Encapsulation
    b. Inheritance
    c. Polymorphism
    d. Abstraction



Encapsulation

1) The process of binding the state (attributes/fields) and behavior of an object together is known as encapsulation.
2) We can achieve encapsulation in java with the help of the class; class has both state and behavior of an object.

Advantage of Encapsulation

1) By using encapsulation we can achieve data hiding.

Data Hiding

1) It is a process of restricting the direct access of data members of an object and provides indirect secured access of data members via methods of the same object is known as data hiding.
2) Data hiding helps to verify and validate the data before storing and modifying it.

Steps To Achieve Data Hiding

1) STEP1
    a. Prefix data members of a class with the private modifier.
2) STEP2
    a. Design a getter and setter method.

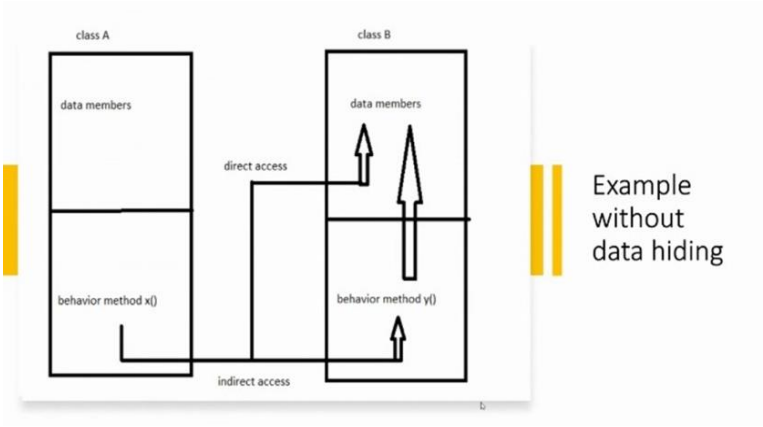
Private Modifier

1) private is an access modifier.
2) private is a class-level modifier
3) If the members of the class are prefixed with a private modifier then we can access that member only within the class.

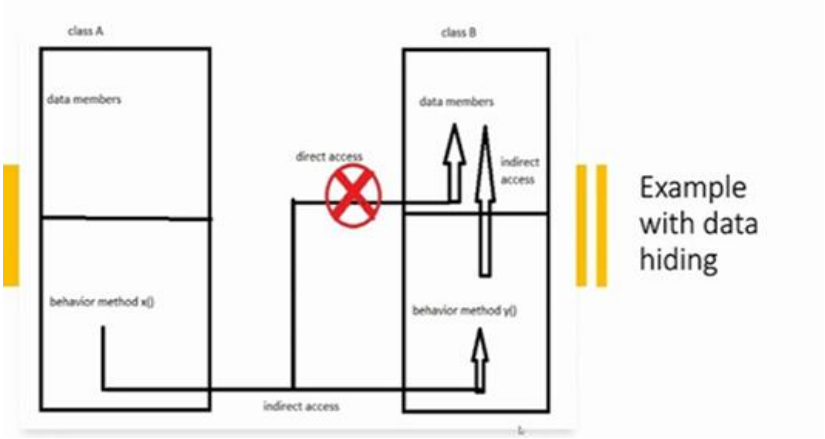
Note

1) Data hiding can be achieved with the help of a private modifier.

Example without Data Hiding



Example with Data Hiding



Getter Method

1) The getter method is used to fetch the data.
2) The return type of the getter method is the type of the hidden value.

Setter Method

1) The setter method is used to update or modify the data.

2) The return type of the setter method is always void.

Note

1) The validation and verification can be done in this method before storing the data and before reading private data members.

Note

1) If you want to make your hidden data member- only readable then create only the getter method.
2) If you want to make your hidden data member- only modifiable then create only the setter method.
3) If you want to make your hidden data member both readable and modifiable then create both getter and setter methods.
4) If you want to make your hidden data member neither readable and nor modifiable then don't create a getter and setter method.

Advantages

1) Provides security to the data members.
2) We can verify and validate the data before modifying it.
3) We can make the data member of the class to ➔ Only readable ➔ Only modifiable ➔ Both

readable and modifiable ➔ Neither readable nor modifiable

What is relationship?

1) The connection (Association) between two object is known as the relationship.
2) Types of relationship?
    a. Has-a relationship
    b. Is-a relationship

Has-a relationship

1) If one object is dependent on another object is known as has-a relationship
    a. Aggregation
    b. composition

What Is Composition?

1) The dependency between two object such that one object cannot exist without another object is known as composition.
2) EXAMPLE Car-Engine, Human-Oxygen, etc....

```
class Engine
{
}
class Car
{
        Engine e ;
```

}

Note

1) We can achieve a Has A Relationship in java by placing the reference variable of the dependent class inside another class.
2) The instance of a dependent object can be created in two ways
    a. Early instantiation
    b. Late / Lazy instantiation

Early Instantiation

1) If the instance of a dependent object is created implicitly it is known as early instantiation.
2) This design can be achieved with the help of an initializer.

STEPS TO ACHIEVE EARLY INSTANTIATION

1) STEP1: Create a dependent class.
2) STEP2: Create another class and place the reference of the dependent object variable inside the class.
3) STEP3: Create a constructor for the class which also accepts the dependent type object.
4) STEP4: Create the object for a class so the object of a dependent object is also created.

```
class Engine
{
        int eno;
```

```
        Engine(int eno)
        {
            this.eno=eno;
        }
    }
5) Class Car
    {
        String carColor;
        Engine e;
        Car(Engine e,String carColor)
        {
            this.e=e;
            this.carColor=carColor;
        }
    }
```

Aggregation

1) The dependency between two objects such that one object can exist without the other is known as aggregation.
2) EXAMPLE Cab-Ola, Train-Online ticket booking, Bus-Passenger, etc...

Lazy / Late Instantiation

1) In this design, the instance of the dependent object is created only when it is required (It is not implicitly created).
2) We can achieve this design with the help of a method; it can be called a helper.

Steps To Achieve Late/Lazy Instantiation

1) STEP1: Create a dependent class.
2) STEP 2: Create another class and define a parameterized method that will accept the reference of the dependent object and inside that method initialize the dependent object.
3) STEP3: Create the object for a class and call a method by passing dependent type object reference so that we can achieve a late/lazy instantiation.

Is-A RELATIONSHIP

1) The relationship between two objects which is similar to the parent and child relation is known as the Is-A relationship
2) In an Is-A relationship, the child object will acquire all properties of the parent object, and the child object will have its own extra properties.
3) In an Is-A relationship, we can achieve generalization and specialization.

Note1

1) Parents are called generalized.
2) Children are called specialized.

Note2

1) Private members, constructors, and initializers are not inherited to the child class.

Example

1) With the help of the child class reference type, we can use the members of the parent's class as well as the child.
2) With the help of parent class reference, we can use only the members of a parent but not the child class.

Parent Class

1) The parent class is also known as a superclass or base class.

Child Class

1) The child class is also known as a subclass or derived class.

Note

Is-A relationship is achieved with the help of inheritance.

Inheritance

1) The process of one class acquiring all the properties and behavior from the other class is called inheritance.
2) In java, we can achieve inheritance with the help of
    a. extends keyword
    b. implements keyword

Extends Keyword

1) Extends keyword is used to achieve inheritance between two classes.

Example:

```
class A
{
    int i=10;

}

class B extends A
{
    int j=20;

    public static void main(String[] args)
    {
        B b = new B();
        System.out.println(b.i);
        System.out.println(b.j);
    }
}
```

Output:
10
20

Example:

```
class A
{
    int i=10;

}

class B extends A
{
    int j=20;

    public static void main(String[] args)
    {
        A a = new A();
        System.out.println(a.i);
        System.out.println(a.j);
    }
}
```

```
B.java:15: error: cannot find symbol
                System.out.println(a.j);
                                   ^
  symbol:   variable j
  location: variable a of type A
1 error
```

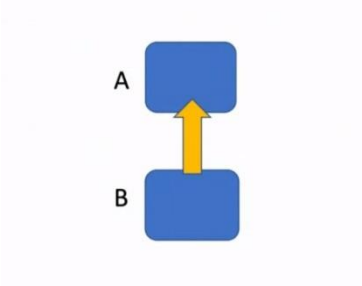
Types of Inheritance

1) Single Level Inheritance
2) Multilevel Inheritance
3) Hierarchical Inheritance

4) Multiple Inheritance
5) Hybrid Inheritance

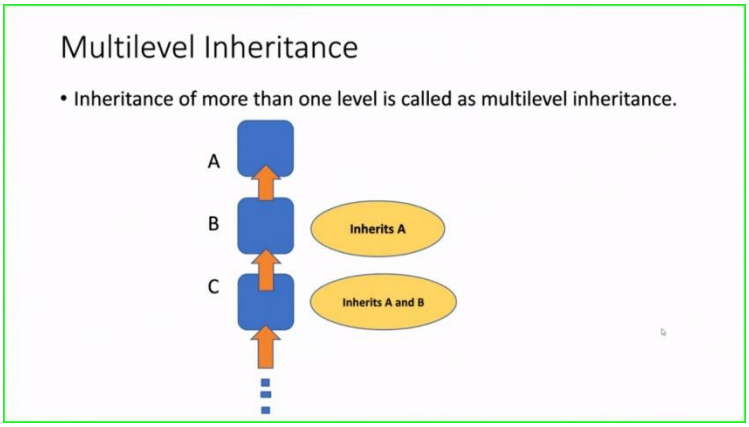
Single level inheritance

1) One parent class is having one child class is known as single level inheritance



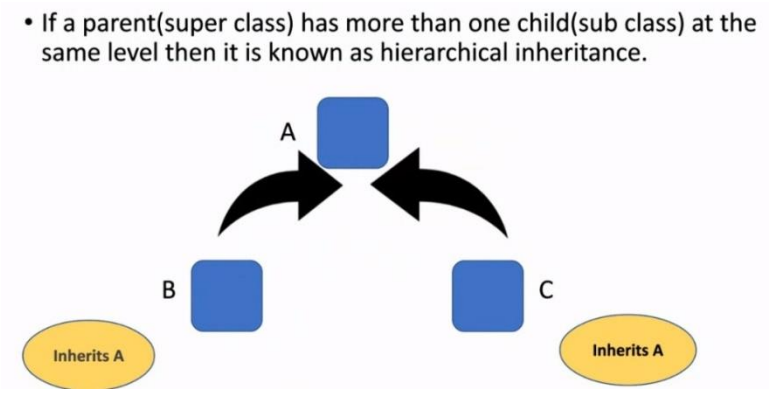
Multilevel Inheritance

1) One superclass is having one subclass and subclass is having another one sub class is known as multilevel inheritance



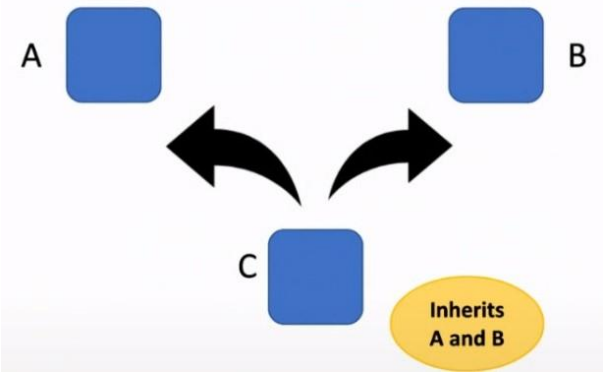
Hierarchical Inheritance

1) One parent class is having more than one child is known as Hierarchical inheritance



Multiple Inheritances

1) One child is having more than one parent is known as multiple inheritance



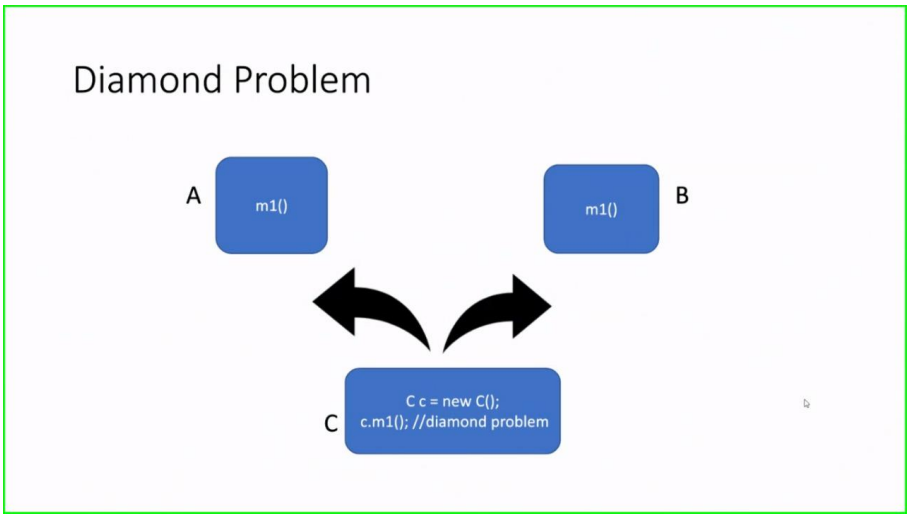
Note

1) Multiple inheritances have a problem known as the diamond problem.
2) Because of the diamond problem, we can't achieve multiple inheritances with the help of class.
3) In java, we can achieve multiple inheritances with the help of an interface.

Diamond Problem

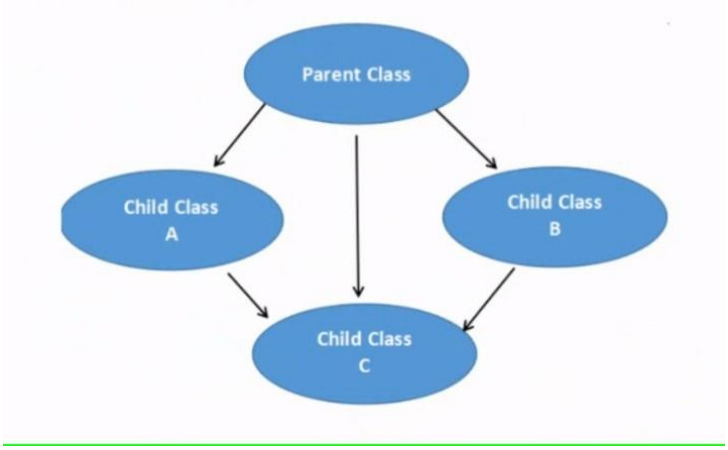
1) Assume that there are two classes A and B.

2) Both are having the method with the same signature.
3) If class C inherits A and B then these two methods are inherited to C.
4) Now an ambiguity arises when we try to call the superclass method with the help of subclass reference.
5) This problem is known as the diamond problem.



Hybrid Inheritance

1) Its combination of more than one inheritance is known as hybrid inheritance



Polymorphism

1) Polymorphism is derived from two different Greek words 'Poly' means Numerous 'Morphs' means form which means numerous forms.
2) Polymorphism is the ability of an object to exhibit more than one form with the same name.
3) For Understanding :
    a. One name -----> Multiple forms
    b. One variable name ------> Different values
    c. Onemethodname----->Different behavior

Types of Polymorphism

1) In java, we have two types of polymorphism,
    a. Compile-time polymorphism
    b. Runtime Polymorphism

Compile-Time Polymorphism

1) If the binding is achieved at the compile-time and the same behavior is executed it is known as compile-time polymorphism.
2) It is also said to be static polymorphism.

Note

1) Binding means an association of method call to themethod definition. It is achieved by
    a. Method overloading
    b. constructor overloading
    c. Variable shadowing
    d. Method shadowing

e. Operator overloading (does not support in java)

Method Overloading

1) If more than one method is created with the same name but different formal arguments in the same class are known as method overloading.
2) EXAMPLE java.lang.Math; abs(double d) abs(float f) abs(int i) abs(long l) These are some of the overloaded methods (methods with the same name but different formal arguments) implemented in java. lang.Math class.

Constructor Overloading

A class having more than one constructor with different formal arguments is known as constructor overloading.

Runtime Polymorphism

1) If the binding is achieved at the runtime then it is known as runtime polymorphism.
2) It is also known as dynamic binding.
3) It is achieved by method overriding,

Method Overriding

1) If the subclass and superclass have non static methods with the same signature, it is known as method overriding.

Rule To Achieve Method Overriding

1) Is-A relationship is mandatory.
2) It is applicable only for no static methods.
3) The signature of the subclass method and superclass method should be the same.
4) The return type of the subclass and superclass method should be the same until the 1.4 version but, from the 1.5 version, covariant return type in the overriding method is acceptable (subclass return type should be the same or child to the parent class return type.).
5) EXAMPLE

```
class Parent
{
public void test()
{
System.out.println("From parent");
}
}
Class Child extends Parent
{
@override public void test()
{
System.out.println("From child");
```

```
}
public static void main(String[] args)
{
Parent p = new Child();
p.test();
}
}
```

EXAMPLE

```
Child c = new Child();
c.test();                // from child Parent
p = c; p.test();         //from child Internal runtime
```

object is a child so child test() will get executed, it does not depend on the reference type.

Note

1) **Variable overriding is not applicable.**

Method Shadowing

1) **If a subclass and superclass have the static method with the same signature, it is known as method shadowing.**
2) **Which method implementation gets executed, depending on what? In method shadowing binding is done at compile-time, hence it is compile-time polymorphism.**

3) The execution of the method depends on the reference type and does not depend on the type of object created.

Note

1) The return type should be the same or it should be a covariant return type.
2) Access modifier should be same or higher visibility than superclass method.
3) Method shadowing is applicable only for the static method.
4) It is compiled time polymorphism
5) Execution of implemented method depends on the reference type of an object.
6) Example:

```
class Parent
{
Public static void m1(){
 System.out.println("From m1() of Parent");
}
} class Child extends Parent
{
Public static void m1()
{
System.out.println("From m1() of Child");
}
}
class Shadowing
```

```
{
public static void main(String[] args)
{
Parent p = new Parent();
p.m1();//From Parent
Child c = new Child();
c.m1();//From ChildParent
p1 = c;
p1.m1();//From Parent
}
}
```

Variable Shadowing

1) If the superclass and subclass have variables with the same name then it is known as variable shadowing.

Which Variable Is Used, Depending On What?

1) In variable shadowing binding is done at compile-time, hence it is a compile-time polymorphism. The Variable used depends on the reference type and does not depend on the type of object created.

Note

1) It is applicable for both static and non-static variables.
2) It is a compile-time polymorphism.

3) Variable usage depends on the type of reference and does not depend on the type of object created.

4) Example

```
Static Variable class A
{
static int i = 20;
}
class B extends A
{
static int i = 40;
}
class VariableShadowing
{
public static void main(String[] args)
{
A a = new A();
System.out.println(a.i);//20
B b = new B();
System.out.println(b.i);//40
a=b; System.out.println(a.i);//20
}
}
```

Abstraction

1) It is a design process of hiding the implementation and showing only the

functionality (only declaration) to the user is known as abstraction.

How to Achieve Abstraction in Java?

1) In java, we can achieve abstraction with the help of abstract classes and interfaces.
2) We can provide implementation to the abstract component with the help of inheritance and method overriding.

ABSTRACT MODIFIER

1) The abstract is a modifier, it is a keyword.
2) It is applicable for methods and classes.

Abstract Method

1) A method that is prefixed with an abstract modifier is known as the abstract method.
2) This is also said to be an incomplete method.
3) The abstract method doesn't have a body (it has the only declaration)

Syntax to create abstract method

```
abstract[accessmodifier]returnType
methodName([For_Arg]);
```

Note

1) Only child class of that class is responsible for giving implementation to the abstract method.

Abstract Class

1) If the class is prefixed with an abstract modifier then it is known as abstract class. We can't create the object (INSTANCE) for an abstract class.

Note

1) We can't instantiate an abstract class
2) An abstract class can have both abstract and concrete methods.
3) If a class has at least one abstract method either declared or inherited but not overridden it is mandatory to make that class an abstract class.
4) EXAMPLE

```
abstract class Atm
{
        abstract public double withdrawal();
        abstract public void getBalance();
        abstract public void deposit();
}               // hiding implementation by providing
only functionality
Atm a = new Atm() // CTE
```

Note

1) Only subclass of Atm is responsible for giving implementation to the methods declared in an Atm class.

Implementation of abstract method

1) If a class extends abstract class then it should give implementation to all the abstract method of the superclass.
2) If inheriting class doesn't like to give implementation to the abstract method of superclass then it is mandatory to make subclass as an abstract class.
3) If a subclass is also becoming an abstract class then the next level child class is responsible to give implementation to the abstract methods.

Steps To Implement Abstract Method:

1) STEP1
   Create a class.
2) STEP2
   Inherit the abstract class/ component.
3) STEP3
   Override the abstract method inherited (Provide implementation to the inherited abstract method).

Concrete Class

1) The class which is not prefixed with an abstract modifier and doesn't have any abstract method, either declared or inherited is known as a concrete class.

Note

1) In java, we can create objects only for the concrete class.

Concrete Method

1) The method which gives implementation to the abstract method is known as the concrete method
2) Example

```
abstract class What Sapp
{
        abstract public void send();
}
class Application extends WhatsApa
{
        public void send()
        {
        System.out.println("Send()  method  is implemented");
        }
}
```

Creating object and calling the abstract method :

```
WhatsApp w = new Application();
w.send()// Send() method is implemented
```

Interfaces

1) It is a component in java which is used to achieve 100 % abstraction with multiple implementations.

Syntax to create an interface

```
[Access Modifier] interface InterfaceName
{
    // declare members
}
```

When an interface is compiled we get a class file with an extension .class only

EXAMPLE

```
interface Demo
{
}
```

The Demo is an interface

What all are the members the can be declared in an interface?

MEMBERS	CLASS	INTERFACE
Static variables	Yes	Yes, but only final static variables
Non-static variables	Yes	No
Static methods	Yes	Yes, From JDK 1.8 v. NOTE: They are by default public in nature
Non-static methods	Yes	Yes but we can have only abstract non-static methods NOTE : Non-static methods are by default • public • abstract
Constructors	Yes	No
Initializers (Static & non-static block)	Yes	No

Note

1) In interface, all the members are by default have public access modifier.
2) Example

```
interface Demo1
{
        Int a;          //CTE: Variable a is by default public, static,final. A final variable must be initialized.
}
interface Demo2
{
        public void test();          // CTE
        {
        }
}
```

Why do we need an interface?

1) To achieve 100% abstraction. Concrete non-static methods are not allowed.
2) To achieve multiple inheritances.

What all the members are not inherited from an interface?

1) Only static methods of an interface are not inherited to both class and Interface.

```
Interface Demo3
{
        public  static void main(String[ ] a)
        {
```

```
            System.out.println("Hello World..!!!");
        }
    }
    interface Demo1
    {
        Int a;//CTE: Variable a is by default public, static, final. A final variable must be initialized.
    }
    interface Demo2
    {
        public void test()// CTE
        {
        }
    }
    Demo3
    {
        public static interface void main(String[] a)
        {
            System.out.println("Hello World..!!!");
        }
    }
```

Inheritance With Respect To Interface

1) An Interface can inherit any number of interfaces with the help of an extended keyword.
2) EXAMPLE

```
interface I1
{
```

```
}
interface I2 extends I1
{
}
```

Note

1) The interface which is inheriting an interface should not give implementation to the abstract methods.
2) EXAMPLE 1
   Interface I1 have 3 methods
   2 - non static (t1() , t2())
   1 - static (t3()) ●
3) Interface I2 have 3 methods
   2 - inherited non static method (t1() , t2())
   1 - declared non static methods (t4())
4) EXAMPLE 2:
   Interface can inherit multiple interfaces at a time

Note

1) With respect to interface there is no diamond problem. The reason,
2) They don't have constructors.
3) Non-static methods are abstract (do not have implementation)
4) Static methods are not inherited.

Inheritance of an Interface by the Class

1) Class can inherit an interface with the help of implements keywords.
2) Class can inherit more than one interface
3) Class can inherit a class and an interface at a time.

Note

1) If a class inherits an interface then it should give implementation to the abstract non-static methods of an interface.
2) If the class is not ready to give implementation to the abstract methods of an interface then it is mandatory to make that class an abstract class.
3) The next level of child class is responsible for giving implementation to the rest of the abstract methods of an interface.
4) EXAMPLE 1 :
5) Class inheriting an interface
   Interface I1 have 3 methods
   2 - non static (t1() , t2())
   1 - static (t3())
6) Class c1 have 3 methods
   2 - inherited and implemented non static method (t1() , t2())
   1 - declared non static methods (demo())

Class Inheriting Multiple Interfaces

1) The diamond problem is solved by implementing multiple interfaces by the class at a time.

Reason

1) Non-static methods are not implemented in an interface.
2) Static methods are not inherited.
3) EXAMPLE 3: Class can inherit interface and class at a time

Rule

1) Use the extends first and then implements.

Note

1) Class can inherit multiple interfaces but it can't inherit multiple classes at a time.
2) Interface can't inherit a class. Interface can't inherit from the class

Reason

1) Class has concrete non-static methods, So class can't be a parent to the interface

Up casting & down casting

Non Primitive Type-Casting (Derived Type Casting)

1) The process of converting one reference type into another reference type is known as non-primitive or derived typecasting.

Rules To Achieve Non Primitive Type Casting:

1) We can convert one reference type into another reference type only if it satisfies the following condition, There must be an Is-A relation (Parent and child) exist between two references.

If The Class/Interface Has A Common Child.

1) Fruit can be converted to Apple and Apple can be converted to fruit as well as Vegetables can be converted to brinjal and brinjal can be converted to Vegetable.
2) But Fruit and apple can't be converted to Vegetable and brinjal as well as Vegetables and brinjal can't be converted to Fruit and Apple.
3) Fruit can be converted to Apple as well as Mango and Mango and Apple can be converted into Fruit.
4) But Apple can't be converted into Mango as well as Mango can't be converted into Apple.

Types of Non-Primitive or Derived Type Casting:

1) Non-primitive type casting can be further classified into two types,
    a. Upcasting
    b. Downcasting

Upcasting

1) The process of storing the child class object into a parent class reference type is known as upcasting

Note

1) The upcasting is implicitly done by the compiler.
2) It is also known as auto upcasting.
3) Upcasting can also be done explicitly with the help of a typecast operator.
4) Once the reference is upcasted we can't access the members of the child.

Why Do We Need Upcasting?

1) It is used to achieve generalization.
2) It helps to create a generalized container so that the reference of any type of child object can be stored.

Disadvantage

1) There is only one disadvantage of up casting that is, once the reference is up casted its child members can't be used.

Note

1) In order to overcome this problem, we should go for down casting.

Downcasting

1) The process of converting a parent (superclass) reference type to a child (subclass) reference type is known as down casting.

NOTE

1) Downcasting is not implicitly done by the compiler.
2) It should be done explicitly by the programmer with the help of a typecast operator.

Why Do We Need A Downcasting?

1) If the reference is upcasted, we can't use the members of a subclass.
2) To use the members of a subclass we need to downcast the reference to a subclass.

Class cast exception

1) It is a Runtime Exception.
2) It is a problem that occurs during runtime while downcasting.

When And Why Do We Get A Class cast exception?

1) When we try to convert a reference to a specific type(class), and the object doesn't have an instance of that type then we get Class Cast Exception.

EXAMPLE:

```
Case 1:
Child c = (Child)new Parent();
//ClassCastException
Case 2
Parent p=new Parent( );
```

```
Child c=(Child)p;
//ClassCastExceptio
```

Instance of Operator

1) It is a binary operator
2) It is used to test if an object is of the given type.
3) The return type of this operator is boolean.
4) If the specified object is of a given type then this operator will return true else it returns false.

Syntax to use instanceOf operator

```
(Object_Ref) instanceof (type)
```

Class

1) Object class is defined in java.lang package.
2) Object class is a supermost parent class for all the classes in java.
3) In object class there are 11 non static methods.
4) One no argument constructor is there

1) public String toString()
2) public boolean equals(Object o)
3) public int hashCode()
4) protected Object clone() throws CloneNotSupportedException
5) protected void finalize()
6) final public void wait() throws InterruptedException

7) final public void wait(long l) throws InterruptedException
8) final public void wait(long l , int i) throws InterruptedException
9) final public void notify() throws InterruptedException
10) final public void notifyAll() throws InterruptedException
11) final public void getClass()

toString()

1) toString() method returns String.
2) toString() implementation of Object class returns the reference of an object in the String format.

Return Format

ClassName@HexaDecimal

EXAMPLE :

```
class Demo
{
	public static void main(String[] args)
	{
		Demo d = new Demo();
		System.out.println(d);//d.toString()Demo@2f92e0f4
	}
}
```

Note

1) Java doesn't provide the real address of an object.
2) Whenever programmer tries to print the reference variable toString() is implicitly called.

Purpose Of Overriding tostring()

1) We override toString() method to return state of an object instead of returning reference of an object.

EXAMPLE

```
class Circle
{
        Int radius;
        Circle(int radius)
        {
                this.radius = radius ;
        }

        @Override
        public String toString()
        {
                return "radius : "+radius;
        }
        public static void main(String[] args)
        {
                Circle c = new Circle(5);
```

```
            System.out.println(c)                    //
        c.toString()---radius :5
        }
    }
```

equals(Object)

1) The return type of equals(Object) method is boolean.
2) To equals(Object) method we can pass reference of any object
3) The java.lang.Object class implementation of equals(Object) method is used to compare the reference of two objects.

EXAMPLE 1

```
class Book
{
    String bname;
    Book(String bname)
    {
        this.bname = bname;
    }
}
```

Note

1) If the reference is same == operator will return true else it returns false.

2) The equals(Object) method is similar to == operator.

PURPOSE OF OVERRIDING Equals(Object)

1) We can override to equals(Object) method to compare the state of an two Objects instead of comparing reference of two Objects.

NOTE

1) If equals(Object) method is not overridden it compares the reference of two objects similar to == operator.
2) If equals(Object) method is overridden it compares the state of two objects, in such case comparing the reference of two objects is possible only by == operator.

Design Tip

1) In equals method compare the state of an current(this) object with the passed object by downcasting the passed object.

EXAMPLE

```
class Book
{
        String bname ;
        Book(String bname)
```

```
    {
        this.bname = bname ;
    }
    @Override
    public boolean equals(Object o)
    {
        Book b = (Book)o;
        if(this.bname.equals(b.bname))
        return  true;
        else
        return false;
    }
}
```

hashcode()

1) The return type of hashCode() method is int.
2) The java.lang.Object implementation of hashCode() method is used to give the unique integer number for every object created.
3) The unique number generated based on the reference of an object.

PURPOSE OF OVERRIDING Hashcode()

1) If the equals(Object) method is overridden , then it is necessary to override the hashCode() method.

Design tip

1) 1hashCode() method should return an integer number based on the state of an object.

EXAMPLE 1

```
class Pen
{
	double price ;
	Pen(double price)
	{
		this.price = price ;
	}
	@Override
	public int hashCode()
	{
		int hc = (int)price;
		return hc ;
	}
}

class Book
{
	int bid ;
	double price ;
	Book(int bid , double price)
	{
		this.bid = bid ;
		this.price = price;
	}
	@Override
	public int hashCode()
	{
```

```
            int hc1 = bid ;
            double hc2 = price ;
            int hc = hc1+(int)hc2;
            return hc ;
        }
    }
```

EXAMPLE 1

```
class Book
{
        String bname;
        Book(String bname)
        {
            this.bname = bname;
        }
}
```

1) In the above two cases it is clear that,
2) If the hashcode for two object is same, equals(Object) method will return true.
3) If the hashcode for two object is different, equals(Object) method will return false.

Wrapper Classes

1) The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

2) The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Auto boxing

1) The automatic conversion of primitive data type into its corresponding wrapper class is known as auto boxing.
2) for example:
3) byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects

Wrapper class Example:

```
class WrapperExample1
{
    static void main(String args[])
    {
        int a=20;
        Integer i=Integer.valueOf(a);

        Integer j=a
        System.out.println(a+" "+i+" "+j);
    }
```

```
}
```

1) The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.
2) It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

Example1

```
class WrapperPrimitiveDemo1
{
	public static void main(String[] args)
	{
		Integer a = new Integer(12);
		System.out.println("Old value = "+a);
		xyz(a);
		System.out.println("New Value = "+a);
	}
	private static void xyz(Integer a)
	{
		a = a + 10;
	}
}
```

Value() Method

1) This method is used for converting numeric object into a primitive data type.
2) For example we can convert a Integer object to int or Double object to float type.
3) The value() method is available for each primitive type and syntax are given below.

Example2

```
class NumberDemo1
{
        public static void main(String[] args)
         {
                Double d1 = new Double("4.2365");
                byte b = d1.byteValue();
                short s = d1.shortValue();
                int i = d1.intValue();
                long l = d1.longValue();
                float f = d1.floatValue();
                double d = d1.doubleValue();

                System.out.println("byte : " + b);
                System.out.println("short : " + s);
                System.out.println("int : " + i);
                System.out.println("long : " + l);
                System.out.println(": " + f);
                System.out.println("double : " + d1);
        }
}
```

In this example, we are using toString method to get string representation of Integer object.

```
public class IntegerClassDemo1
{
    public static void main(String args[])
    {
        int a = 95;
        Integer x = new Integer(a);

System.out.println("toString(a)="+Integer.toStrin
g(a));
    }
}
```

valueOf()

1) The valueOf() method is used to get an Integer object representing the specified int value.
2) It takes a single int type argument and returns an Integer instance.
3) Syntax of the method is given below
4) public static Integer valueOf(int b)

```
class IntegerClassDemo1
{
    public static void main(String args[])
    {
        int a = 95;
        Integer x = Integer.valueOf(a);
```

```
                System.out.println("valueOf(a) = " + x);
        }
    }
```

ARRAY

1) Array is a continuous block of memory that is used to store multiple values or data.

CHARACTERISTIC OF AN ARRAY

1) The size of an array must be defined at the time of declaration.
2) Once declared, the size of an array can't be modified.
3) Hence array is known as fixed size.
4) In an array, we can access the elements with the help of an index or subscript. It is an integer number that starts from 0 and ends at the length of the array-1.
5) In an array, we can store only homogeneous type values. It is also known as the homogeneous collection of an object.

NOTE

1) In java array is an object

To declare an array

1) datatype[ ] variable; (or) datatype variable[ ];

EXAMPLE

1) int a[] --- single dimensional array reference variable of int type
2) float f[] --- single dimensional array reference

variable of float type.

3) String s[] --- single dimensional array reference variable of String type.

Syntax to instantiate an array

```
new datatype[ size ];
```

Example

```
new int[5];
```

ox1

0
0
0
0
0

Note

1) Once the array is instantiated it is assigned with a default value.

Adding Elements

1) We can add an element into an array with the help of array index.

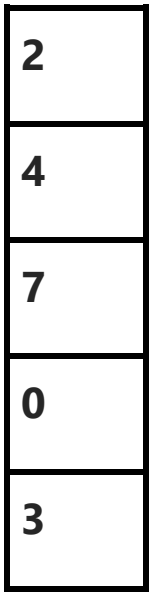
Syntax to add an element into an array

```
array_ref_variable[ index ] = value ;
```

Example

```
int[] arr = new int[5] ; //declaration
arr[0] = 2;
arr[1] = 4;
arr[2] = 7;
arr[3] = 0;
arr[4] = 3;
```

ox1---------->arr



Accessing Elements

1) We can access an element from an array with the help of array reference variable and index.

Syntax to access an elements from an array

```
array_ref_variable[ index ] ;
```

Example

```
System.out.println(arr[0]); // 2

System.out.println(arr[2]); // 7

System.out.println(arr[4]); // 3

System.out.println(arr[5]);
//ArrayIndexOutOfBoundsException
```

Declaring, Instantiating, Initializing an Array in A Single Line

1) We can also declare instantiate and initialize an array by using single line.

Syntax

1) datatype[] arr_ref_var = { element1 , element2 , element3 , etc... } ;

Example

```
int arr[] = { 1 , 2 , 3 , 4 , 5 };
```

EXAMPLE 1:

```
package com.java.arrays;
class Bag
{
        public static void main(String[] args)
            {
                String[] bag = new String[3];
                bag[0]="Apple";
                bag[1]="mango";
                bag[2]="Orange";
```

```
                System.out.println(bag[0]);
                System.out.println(bag[1]);
                System.out.println(bag[2]);
            }
    }
```

Example 2:

```
    package com.java.arrays;
    class EvenNumbers
    {
        public static void main(String[] args)
        {
            int[] arr = {3,45,2,3,4,88,76,41,12};
            for(int i=0; i<arr.length;i++)
            {
                if(arr[i]%2==0)
                System.out.print(arr[i]+" ");
            }
         }
     }
```

Example 3:

```
        package com.java.arrays;
        class Greatest
        {
            public static void main(String[] args)
            {
                Student student[] = new
                Student[5];
                student[0] = new
                Student("Smith",123);
```

```
                student[1]          =          new
                Student("Alan",124);
                student[2]          =          new
                Student("Sheela",12);
                student[3]          =          new
                Student("Laila",1);
                student[4]          =          new
                Student("Mala",56);
                int greatest = -1;
                for(int i=0;i<student.length;i++)
                {
                    if(student[i].sid>greatest)
                    greatest=student[i].sid;
                }
                    System.out.println(greatest+"
                is the greatest id of all");
            }
        }
```

Example 4.1:

```
        package com.java.arrays;
        class Student implements Comparable
        {
            String sname;
            int sid;
            public Student(String sname, int sid)
            {
                super();
                this.sname = sname;
                this.sid = sid;
            }
            @Override
```

```
    public String toString()
    {
        return sname;
    }
    @Override
    public boolean equals(Object obj)
    {
        Student s=(Student)obj;
        if(this.sname.equals(s.sname)    &&
        this.sid==s.sid)
            return true;
        else
            return false;
    }
    @Override
    public int hashCode()
    {
        int hc1 = sname.hashCode();
        int hc2 = sid;
        return hc1+hc2;
    }
    @Override
    public int compareTo(Object o)
    {
        Student s = (Student)o;
        return
        this.sname.compareTo(s.sname);
    }
}
```

Example 4.2:

```
package com.java.arrays;
```

```
class School
{
	public static void main(String[] args)
	{
		Student s1 = new Student("Smith",123);
		Student s2 = new Student("Alan",124);
		Student s3 = new Student("Sheela",12);
		Student s4 = new Student("Laila",1);
		Student s5 = new Student("Mala",56);
		Student student[] = {s1,s2,s3,s4,s5};
		System.out.println(student[4].sname);//
	Mala
		System.out.println(student[1].sid);//124
	}
}
```

Example 4.3:

```
	package com.java.arrays;
	class SearchStudent
	{
		public static void main(String[] args)
		{
			Student student[] = new
		Student[5];
			student[0] = new
			Student("Smith",123);
			student[1] = new
			Student("Alan",124);
			student[2] = new
			Student("Sheela",12);
			student[3] = new
			Student("Laila",1);
```

```
                student[4] = new Student("Mala",56); ji
                for(int i=0; i<student.length;i++)
                {
                    if(student[i].sid==2)
                    {
                        System.out.println("Found");
                        return;
                    }
                }
                    System.out.println("Not found");
            }
        }
```

To Sort an Array Using Built In Methods

java.util.Arrays

1) In java.util.Arrays class we have a static sort method which can accept an array and sort in ascending order.
2) The sort method uses compareTo(Object) method of comparable interface for sorting.

Therefore the sort method can sort an array only if,

1) All the elements in the array belongs to comparable type.
2) All the elements in the array must be of same

type(Homogeneous).

Note

1) If any one of the above condition is not followed we get, ClassCastException.

Comparable Interface

1) Comparable interface is a functional interface(Only one abstract method), since it has only one abstract method.

Method Declaration

1) public abstract int compareTo(Object o)

Steps To Make An Object Comparable Type

STEP 1:

The class must implements java.lang.Comparable interface.

STEP 2:

Override compareTo(Object o) method.

compareTo(Object)

1) It is defined in Comparable interface.
2) This method is used to compare the current object with passed object.
3) The return type is int.

Function

1) If the value of current object is greater than the passed object it will return positive integer.
2) If the value of current object is lesser then the passed object it will return negative integer.
3) If the value of current object is equal to the passed object it will return zero.

Note

1) compareTo() method is called only when the object is of comparable type.

Example 4.4:

```
package com.java.arrays;
import java.util.Arrays;
public class Sorting {
	public static void main(String[] args) {
		int[] arr = {3,45,2,3,4,88,76,41,12};
		Arrays.sort(arr);
		for(int i=0; i<arr.length;i++)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

Example 4.5

```
package com.java.arrays;
import java.util.Arrays;
public class SortStudent {
	public static void main(String[] args) {
		Student student[] = new Student[5];
		student[0] = new Student("Smith",123);
		student[1] = new Student("Alan",124);
		student[2] = new Student("Sheela",12);
		student[3] = new Student("Laila",1);
		student[4] = new Student("Mala",56);
		Arrays.sort(student);
		for(int i=0;i<student.length;i++)
		{

	System.out.print(student[i].sname+" ");
		}
	}
}
```

STRING LITERAL

1) Anything enclosed within the double quote " " in java is considered as String literal.

Characteristics of String Literal

1) When a string literal is used in a java program, an instance of java.lang.String class is created inside a String pool.
2) For the given String literal, If the instance of a String is already present, then new instance is not created instead the reference of a existing instance is given.

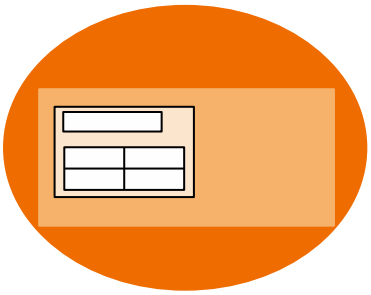
STRING

1) The string is literal (data). It is a group of characters that is enclosed within the double quote "".
2) It is Non-primitive data.
3) In java, we can store a string by creating instances of the following classes.
   a. java.lang.String
   b. java.lang.StringBuffer
   c. java.lang.StringBuilder
4) In java, whenever we create a string compiler implicitly create an instance for java.lang.string in string pool area/string constant pool (SCP).

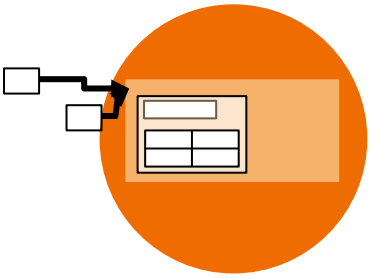
EXAMPLE1

```
Class Demo
{
      public static void main(String[] args)
      {
            System.out.println("Hello");//String@01
            23
            System.out.println("Hello");//String@01
```

```
                23
                }
        }
}
```



```
Class Demo
{
        public static void main(String[] args)
        {
                String s1, s2;
                s1 = “Hello”;
                s2 = “Hello”;
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s1==s2);//true
                System.out.println(s1.equals(s2));//true
        }
}
```



java.lang.String

1) String is a inbuilt class defined in a java.lang package.
2) It is a final class.
3) inherits java.lang.Object
4) a String class toString(), equals(), hashCode() methods of java.lang.Object class are overridden.

It implements

1) Comparable
2) Serializable
3) CharSequence

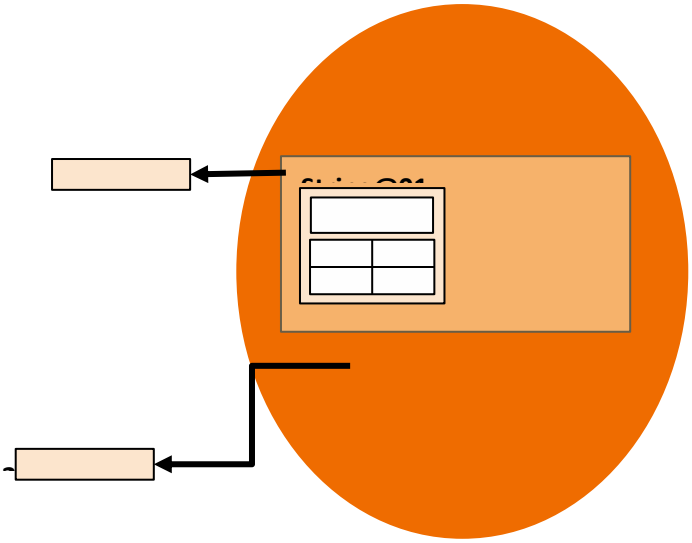
CONSTRUCTORS

CONSTRUCTORS	DESCRIPTION
String()	Creates an empty string object
String(String literals)	Creates string object by initializing with string literals
String(char[] ch)	Creates String by converting character array into string
String(byte[] b)	Creates String by converting byte array into string
String(StringBuffer sb)	Creates String by converting StringBuffer to string

String(StringBuilder sb)	Creates String by converting StringBuilder to string

EXAMPLE3

```
Class Demo
{
        public static void main(String[] args)
        {
                String s1, s2;
                s1 = “Hello”;
                s2 = new String (“Hello”);
                System.out.println(s1);
                System.out.println(s2);
                System.out.println    ls(s2));//true
                System.out.println(s1.hashCode()=
        =s2.hashCode());//true
        }
}
```



IMPORTANT METHODS :

RETUR		
char	charAt(int index)	Returns character of the specified index from the string
int	indexOf(char ch)	Return the index of the character specified if not return -1
int	indexOf(char ch, int Start_Index)	Return the index of the character specified by searching from specified index if not return -1
int	indexOf(charSequence str)	Return the index of the specified string index if not return -1
int	indexOf(charSequence str,int Start_Index)	Return the index of the specified string by searching from specified index if not returns -1
int	lastIndexOf(char ch)	Returns the index of the character which is occurred at last in the original String
int	length()	Returns length of the string

boolean	equals(Object o)	Compares states of a two strings

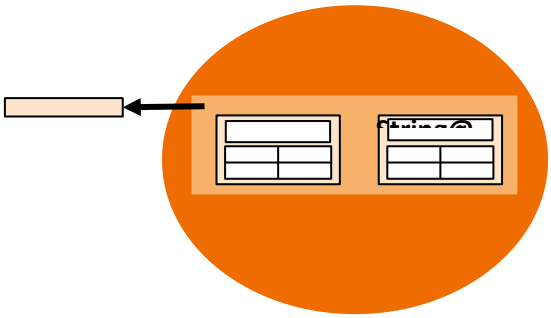
boolean	equalsIgnoreCase(String s)	Compares two strings by ignoring its case
boolean	contains(String str)	Returns true if specified String is present else it returns false
boolean	isEmpty()	Returns true if string is empty else return false
char[]	toCharArray()	Converts the specified string into character array
string[]	split(String str)	Break the specified string into multiple string and returns String array
byte[]	getBytes()	Converts the specified string to byte value and returns byte array

Characteristic

1. Instance of String class is immutable in nature. (Once the object is created then the state is not modified)
2. If we try to manipulate (Modify) the state/data then new object is created and reference is given

EXAMPLE1

```
Class Demo
{
    public static void main(String[] args)
    {
        String s1, s2;
        s1 = "Hello";
        s1.toUpperCase();
        System.out.println(s1);
    }
}
```



DISADVANTAGES

Immutability, because for every modification separate object is getting created in a memory, it reduces the performance.

NOTE

To overcome the disadvantage of String class we can go for StringBuffer and StringBuilder

StringBuffer

java.lang.StringBuffer

1. It is a inbuilt class defined in java.lang package.
2. It is a final class.

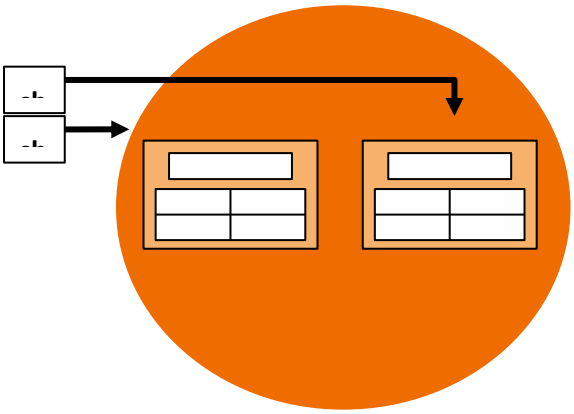
3. It helps to create mutable instance of String.
4. StringBuffer does not have SCP.
5. It inherits java.lang.Object class.
6. In StringBuffer equals(), hashcode() methods of java.lang.Object class are not overridden.
7. It implements :
    a. Serializable
    b. CharSequence

CONSTRUCTORS	DESCRIPTION
StringBuffer()	Creates empty String with initial capacity 16
StringBuffer(String str)	Creates string buffer with the specified string

EXAMPLE1

```
Class Dem0
{
    public static void main(String[] args)
    {
    StringBuffer sb1 = new StringBuffer(“Hello”);
    StringBuffer sb2 = new StringBuffer(“Hello”);
    System.out.println(sb1);
    System.out.println(sb2);
    System.out.println(sb1==sb2);//false here
```

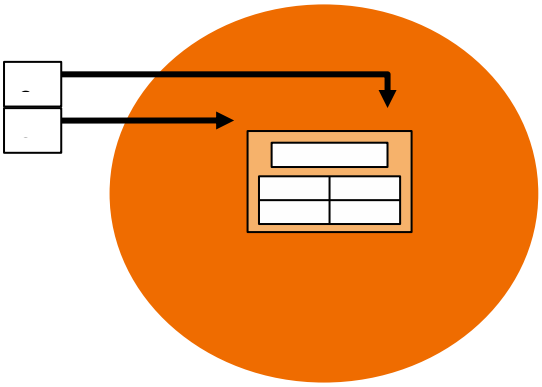
```
also ref is          //comparing
        System.out.println(sb1.equals(sb2));//false
here also          //ref is comparing
        }
}
```



EXAMPLE : 2

```
Class Demo
{
        public static void main(String[] args)
        {
                StringBuffer sb1, sb2;
                sb1 = new StringBuffer("Hello");
                sb2 = sb1;
                System.out.println(sb1);
                System.out.println(sb2);
          System.out.println(sb1==sb2);//true
```

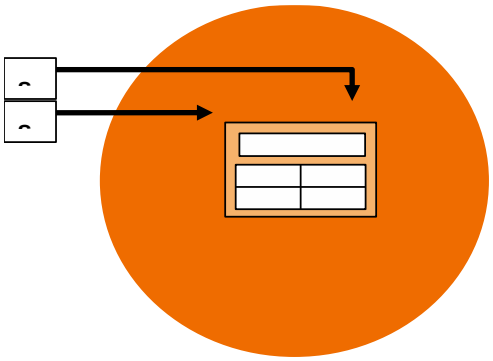
```
        System.out.println(sb1.equals(sb2));//true
    }
}
```



EXAMPLE : 3

```
Class Demo
{
    public static void main(String[] args)
    {
        StringBuffer sb1, sb2;
        sb1 = new StringBuffer("Hello");
        sb2 = sb1;
        System.out.println(sb1);//Hello
        System.out.println(sb2);//Hello
        sb1.append(" World");
        System.out.println(sb1);//Hello World
        System.out.println(sb2);//Hello World
      System.out.println(sb1==sb2);//true
```

```
System.out.println(sb1.equals(sb2));//true
    }
}
```



RETURN TYPE	METHOD NAME	DESCRIPTION
int	capacity()	Returns current capacity.
int	length()	Returns length of the string.
char	charAt(int index)	Returns the character of the specified index.
StringBuffer	append(String s)	Join strings. (Overloaded method).
StringBuffer	insert(int index, String s)	Insert a specified string into original String of specified index. (Overloaded method)
StringBuffer	delete(int begin, int end)	Delete String from specified beginning index to end-1 index.

StringBuffer	deleteCharAt(int index)	Delete the character present in the specified index.
StringBuffer	reverse()	Reverse the string literal
StringBuffer	setLength(int length)	Only specified length in a string is exist remaining get removed.
StringBuffer	substring(int begin)	Returns the substring from the specified beginning index
StringBuffer	substring(int begin, int end)	Returns substring from specified beginning index to end-1 index.
StringBuffer	replace(int begin,int end,String s)	Replace a specified string from the beginning index to end-1 index
void	trimToSize()	Removes the unused capacity or set the capacity till length of the string

void	setCharAt(int index, char new char)	Replace the new character in a string of specified index.
void	ensureCapacity(int capacity)	Sets capacity for storing a string.

CHARACTERISTICS

It is Mutable.

NOTE

String constant pool is not applicable to String Buffer.

DISADVANTAGES

1. Multiple thread can't execute the StringBuffer object simultaneously because all the methods are synchronized. So, Execution time is more. In order to overcome this problem we will go for StringBuilder.

NOTE

The characteristics of StringBuffer and StringBuilder are same

DIFFERENCE BETWEEN StringBuffer AND StringBuilder:

STRING BUFFER	STRING BUILDER

All the method present in StringBuffer is synchronized.	All the method present in StringBuilder is non synchronized.
At a time only one thread is allowed to access String Buffer object. Hence it is Thread safe.	At a time multiple thread is allowed to access String Builder object. Hence it is Not Thread safe.
Threads are required to wait to operate a stringBuffer object. Hence Relatively performance is low .	Threads are not required to wait to operate a StringBuilder object. Hence Relatively performance is high.
Less efficient than StringBuilder	Efficiency is high compared to StringBuffer
Introduced in 1.0 v	Introduced in 1.5v

Exception

The exception is a problem that occurs during the execution of a program (Runtime). When an exception occurs, the execution of the program stops abruptly (Unexpected stop).

Note

Every exception in java is a class of 'Throwable type'

Note

1. Every exception is occurred because of a statement.
2. A statement will throw an exception during the abnormal situation.

STATEMENT	EXCEPTION
a/b	ArithmeticException
reference.member	NullPointerException
(ClassName)referenc nie	ClassCastException
array_ref[index]	ArrayIndexOutOfBoundsException
string.charAt(index)	StringIndexOutOfBoundsException
string.substring(Index)	StringIndexOutOfBoundsException

Checked Exception

1. The compiler-aware exception is known as the checked exception. i.e., the Compiler knows the statement responsible for abnormal situations (Exception). Therefore the compiler forces the

programmer to either handle or declares the exception. If it is not done we will get an unreported compile-time error.

2. Example: FileNotFoundException

Unchecked Exception

1) The compiler-unaware exception is known as the unchecked exception. i.e., the Compiler doesn't know the statements which are responsible for abnormal situations (Exception). Hence, the compiler will not force the programmer either to handle or declare the exception.
2) Example: ArithmeticException

Throw Able

Throwable class is defined in java.lang package.

Exception Handling

Exception handling is a mechanism used in java that is used to continue the normal flow of execution when the exception occurred during runtime.

How to Handle the Exception?

In java, we can handle the exception with the help of a try-catch block.

Syntax to Use Try-Catch Block

```
try
{
    // Statements;
```

```
}
catch(declare one variable of Throwable type)
{
        // Statements;
}
```

try{ }

1) The statements which are responsible for exceptions should be written inside the try block.
2) When an exception occurs,
    a. Execution of try block is stopped.
    b. A Throwable type object is created.
    c. The reference of throwable type object created is passed to the catch block.

catch() { }

1. The catch block is used to catch the throwable type reference thrown by the try block.
2. If it catches, We say the exception is handled. Statements inside the locatch block are get executed and the normal flow of the program will continue.
3. If it doesn't catch, we say the exception is not handled. Statements written inside the catch block are not executed and the program is terminated.

EXCEPTION OCCURS BUT NOT	EXCEPTION OCCURS AND

HANDLED	HANDLED

System.out.println("Main Begins");
try
{
int c = 5/0;
}
catch(NullPointerException e)
{
System.out.println("Divisible by zero is not possible");
}
System.out.println("Main End");
CONSOLE:
Main Begins
Exception in thread "main" java.lang.ArithmeticException: / by zero
	System.out.println("Main Begins");
	try
	{
	int c = 5/0;
	}
	catch(ArithmeticException e)
	{
	System.out.println("Divisible by zero is not possible");
	}
	System.out.println("Main End");
	CONSOLE:
	Main Begins
	Divisible by zero is not possible
	Main End

try with multiple catch

try block can be associated with more than one catch blocks

Syntax

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

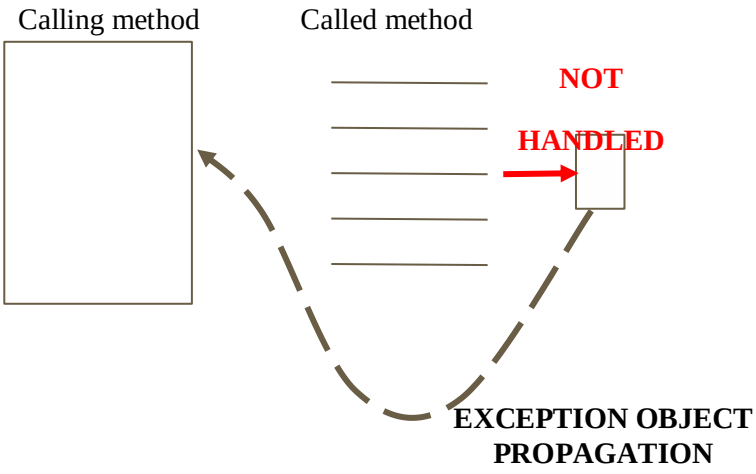
Example

CASE 1	CASE 2

try
{
10/0;
}
catch(Exception e)
{
}
catch(ArithmeticException e)
{
}
	try
	{
	10/0;
	}
	catch(ArithmeticException e)
	{
	}
	catch(Exception e)
	{
	}

Exception Object Propagation

The movement of exception from called method to calling method when it is not handled is known as Exception object propagation.



Case 1: Exception Occurred And Handled By The Calling Method

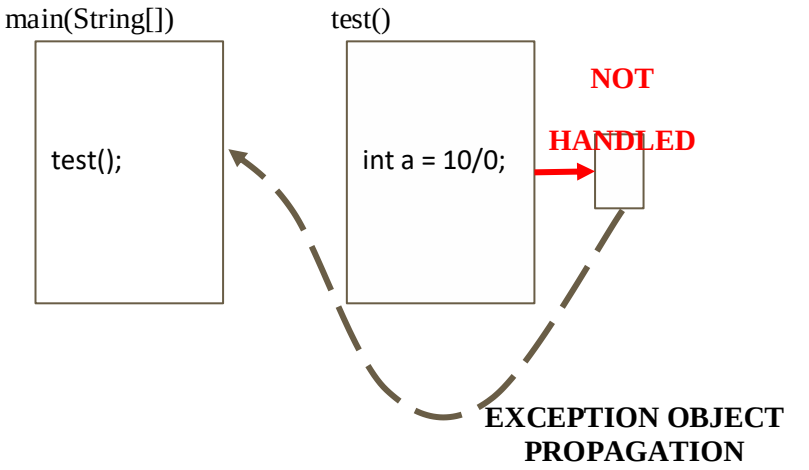
```
class Case1
{
        public static void main(String[] args)
        {
            try
            {
                test();
                }
                catch(ArithmeticException e)
                {
                System.out.println("Exception is handled by the calling method");
```

```
            }
        }
        static void test()
        {
            Int a = 10/0;
        }
}
```

Case 2: Exception Occurred And Not Handled By The Calling Method:

```
class Case1
{
        public static void main(String[] args)
        {
            test();
        }
        static void test()
        {
            Int a = 10/0;
        }
}
```

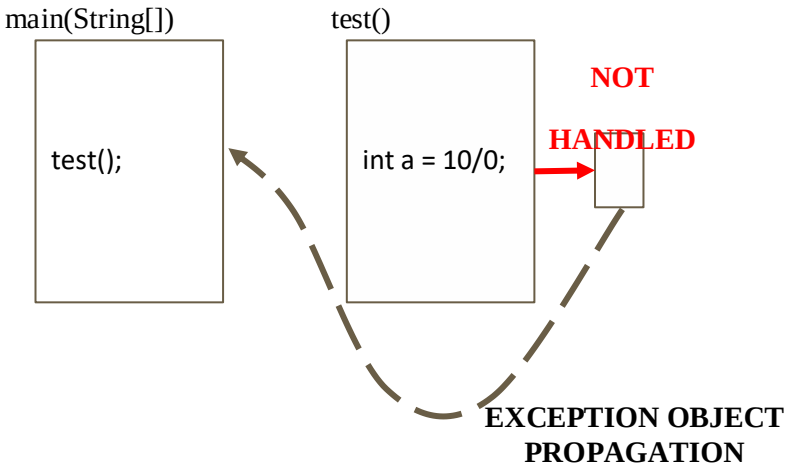
```
Exception in thread "main" java.lang.ArithmeticException: / byzero
        at Case1.test(Case1.java: 8)
        at Case1.main(Case1.java: 4)
```

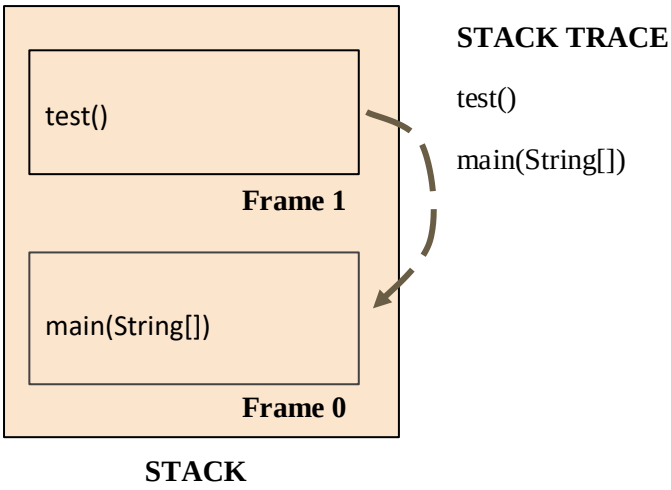


Stack Trace

1) It provides the order in which the exception is occurred and flown from top of the stack to the bottom of the stack.
2) It contains fully classified name of exception, reason for the exception, method name and line number.

EXAMPLE :





Checked Exception

1) The compiler-aware exception is known as checked exception.
2) If any statements are responsible for checked exception it is mandatory to either declare or handle the exception otherwise, we will get unreported compile time error.
3) Compiler will force the programmer to either declare or handle the checked exception during compile time.

Throws

1) It is a keyword.
2) It is used to declare an exception to be thrown to the caller.

Note

Throws keyword should be used in the method declaration statement.

Syntax

```
[modifier] return_type methodName([formal arg])
throws excep1, excep2,...
    {
        //statements;
    }
```

Note

throws keyword does not handle the exception

Throw

1) It is a keyword.
2) It is used to throw an exception manually.
3) By using throw we can throw checked exception, unchecked exception. It is mainly used to throw the custom exception.

Syntax

1) throw exception ;
2) throw new CustomException("String");

Example

```
class ThrowDemo
{
    public static void main(String[] args)
    {
        int a = 15;
        int b = 10;
      if(a>b)
      throw                                new
      ArithmeticException("manually
```

```
                thrown");
                else
                System.out.println("No exception");
            }
        }
```

Output

```
Exception in thread main java.lang.ArithmeticException: manually thrown
```

Note

If we don't handle the manually thrown exception then we will get unreported exception.

Finally { }

1) It is a block which is used along with try-catch block.
2) It will execute even the exception is not handled

Syntax

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

Note

We can also use finally block with try block alone.

Final Vs Finally Vs Finalize

final	finally{ }	finalize()
final is a modifier	finally is a block	finalize is a method
It is applicable to class, method, variable. final class can't be inherited, final variable value can't be changed , final method can't be Overridden	Instruction written inside the 'finally block' will be executed even if the exception isn't handled.	finalize method is used to perform cleanup processing just before the object is garbage collected

Why do we need collection Framework in java?

1) To store multiple objects or group of objects together we can generally use arrays. But arrays has some limitations

Limitations of Array

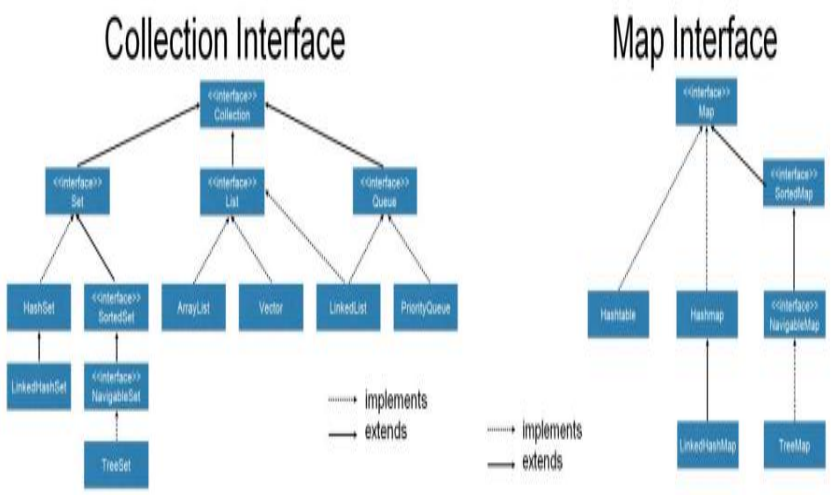
1) The size of the array is fixed; we cannot reduce or increase dynamically during the execution of the program.
2) Array is a collection of homogeneous elements.
3) Array manipulation such as
    a. Removing an element from an array.
    b. Adding the element in between the array etc…
4) Requires complex logic to solve.
5) Therefore, to avoid the limitations of the array we can store the group of objects or elements using different data structures such as :
    a. List
    b. Set
    c. Queue
    d. maps / dictionaries

Definition

1) Collection Framework is set of classes and interfaces (hierarchies), which provide mechanism to store group of objects (elements) together.
2) It also provides mechanism to perform actions such as :
3) (CRUD - operations)
    a. create and add an element
    b. access the elements
    c. remove / delete the elements
    d. search elements
    e. update elements
    f. sort the elements

Note

1) 2 important hierarchies of collection frame Work is :
2) Collection Framework Page 1
   a)Collection hierarchy
   b) Map hierarchy



Collection Framework Page 2
Collection Framework Page 3

Collection **interface**

1) Collection is an interface defined in java.util. package
2) Collection interface provides the mechanism to store group of objects (elements) together.
3) All the elements in the collection are stored in the form of Objects. (i.e. only Non-primitive is allowed)
4) Which helps the programmer to perform the following task?

a)add an element into the collection
b) search an element in the collection
c)remove an element from the collection
d) access the elements present in the collection

Methods of Collection interface:

Collection Framework Page 4

Purpose	return type	method signature
Add an element		1) add(Object) 2) addAll(Collection)
search an element		1) contains(Object) 2) containsAll(Collection)
remove element		1) remove(Object) 2) removeAll(Collection) 3) retainAll(Collection) 4) clear()
Access elements	Iterator	1) iterator( ) 2) We can access elements with the help of for each loop.
Miscellaneous		1) size() 2) isEmpty() 3) toArray() 4) hashCode() 5) equals()

List interface

1) List is an interface defined in java.util.package.
2) List is a sub interface of Collection interface. Therefore, it has all the methods inherited from Collection interface.

Methods Of List Interface :

Purpose	return type	method signature
Add an element		1) add(Object) 2) addAll(Collection) 3) add(int index , Object) 4) addAll(int index , Object)
search an element		1) contains(Object) 2) containsAll(Collection)
remove element		1) indexOf(Object) 2) remove(Object) 3) removeAll(Collection) 4) retainAll(Collection) 5) clear() 6) remove(int index)
Access elements	Iterator	1) iterator( ) 2) listIterator() 3) get(int index) 4) Note : we can access with the help of for each loop.

Miscellaneous		6) size() 7) isEmpty() 8) toArray() 9) hashCode() 10) equals()

What is list? What is it Characteristics?

1) List is a collection of objects.
2) List is an ordered collection of Objects. (It maintains the insertion order of elements)
3) List has indexing; therefore, we can insert, remove or access elements with the help of index.
4) List can have duplicate objects.

ArrayList

1) It is a Concrete implementing class of List interface.
2) The characteristics of ArrayList are same as List.
3) ArrayList is defined in java.util package.

Note

1) All the abstract methods of List and Collection interface are implemented.
2) We can create an instance of ArrayList

Constructors

ArrayList( )	1) creates an empty ArrayList Object
	1) ArrayList (collection) creates an ArrayList Object, and copies all the elements present in the given collection into the ArrayList created.

Note

1) toString method is Overridden such that it returns a view of all the elements present in the array list.
2) [element1, element2,]….

Note

1) The elements in the ArrayList can be accessed in the following ways :

a. get method
b. iterator
c. ListIterator
d. for each loop

1. get(index) :

1. get method is used to access the elements present in the ArrayList. 2. it accepts index as an argument.

3. It returns the Object type.

Example refer : workspace/collections/src/arraylist/get 2. To access the elements of ArrayList using iterator.

1. iterator() is method which belongs to Iterable Interface.

2.

iterator() method creates an Iterator type object and returns the reference.

3. iterator() method returns the reference in Iterator (interface) type

java.util.Iterator :

Collection Framework Page 8

java.util.Iterator :

Iterator interface has 2 important methods to perform iteration.

INDEX

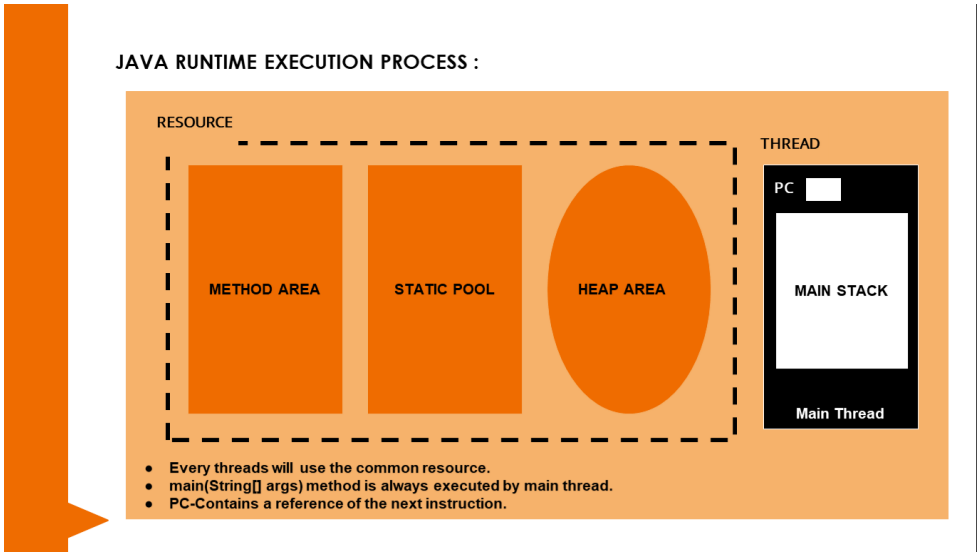
2) Thread
3) Life cycle of a thread
4) Ways to creating a thread
5) Thread class
6) Important methods

Thread

1) Thread is lightweight sub process, smallest unit of a processing which is used to execute the program.
2) It uses a shared memory area.
3) Every thread will have components which is required for execution.
4) Example: Stack, Program Counter, etc….

Java process consist of

1) Every process will have default thread for execution, known as Main thread.
2) Every process will have a memory to store the resource (Code segment, data segment).
3) Code segment - instructions to be stored.
4) Data segment - data's are stored.



Main Thread

1) The execution of a main thread always starts from main(String[] args) method and ends at main(String[] args) Method.
2) Attributes of a thread :
    a. Name
    b. Thread ID
    c. Priority, etc,.

Thread Class

1) In java, we have a class that defines a thread that class is known as Thread class present in java.lang.package.
2) In thread class name, id, priority, state, etc. Of a thread is defined.
3) Thread class has a built in methods to perform actions on thread.

4) Thread class is an encapsulated class, all the attributes are made private, and we can access them through getter and setter methods.

Declaration of Thread Class

1) Thread class declared in java.lang package.
2) public class Thread extends Object implements Runnable

Constructors Of Thread Class

Thread()	Creates a thread
Thread(String)	Creates a thread, initializing with name
Thread(Runnable)	Creates a thread, initializing with instance of Runnable
Thread(Runnable, String)	Create a thread, along initializing with instance of Runnable and Name
Thread(ThreadGroup group, String name))	Allocates a new thread object

Return type	Method signature	Function
Thread	currentThread()	Returns the reference of the currently executing thread
boolean	interrupted()	Tests whether the current thread has been interrupted.
void	sleep(long millis)	Cause the currently executing thread to sleep for a specified time.
void	sleep(long millis, int nanos)	Cause the currently executing thread to sleep for a specified time.
void	yield()	A hint to the processor that current thread is willing to yield its current use of a processor.

Return type	Method signature	Function
String	getName()	Returns this thread name
long	getId()	Returns this identifier of a thread
int	getPriority()	Returns this thread's priority
ThreadGroup	getThreadGroup()	Returns thread group to which this thread belongs to
Thread.state	getState()	Returns the state of this thread
void	interrupt()	Interrupts this thread
boolean	isAlive()	Tests if this thread is alive
boolean	isDaemon()	Tests if this thread is daemon
boolean	isInterrupted()	Tests whether this thread is interrupted

Return type	Method signature	Function
void	join()	Waits for this thread to die
void	join(long millis)	Waits at most millis millisecond for this thread to die
void	join(long millis, int nanos)	Waits at most millis millisecond plus nanosecond for this thread to die
void	resume()	This method exist for solely for use with suspend()
void	run()	The execution of thread we created is starts from run() method and eds at run().
void	setDaemon(boolean on)	Make this thread as a daemon
void	setPriority(int newPriority)	Changes the priority of this thread
void	start()	Once the start method is called the thread is ready to execute
void	stop()	
void	suspend()	

To Access the Properties of Main Thread

1) Thread.currentThread()
2) getName()
3) getPriority()

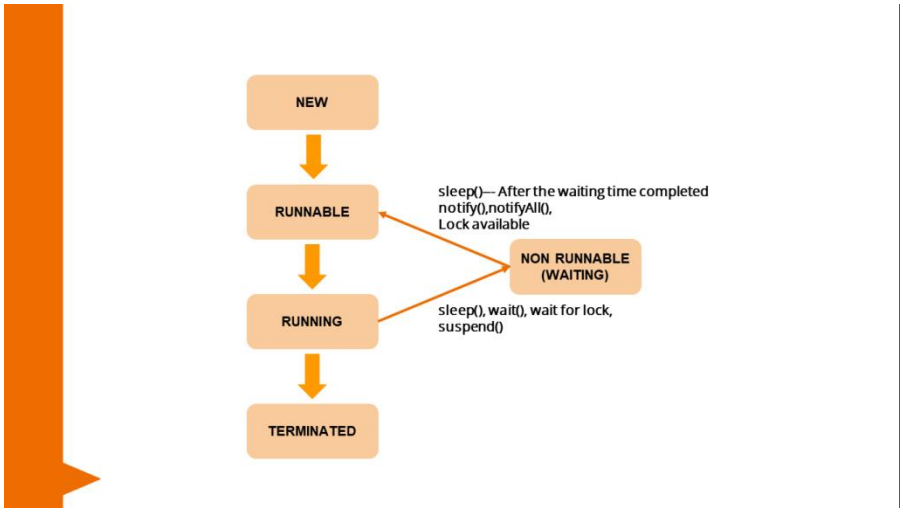
EXAMPLE

```
class ThreadPropertyDemo
{
    public static void main(String[] args)
    {
        System.out.println(Thread.currentT
        hread().getName()); //main
        System.out.println(Thread.currentT
        hread().getId());
    }
}
```

LIFE CYCLE OF A THREAD

Stages of threads

1) Born (New)
2) Ready (Runnable)
3) Executing (Running)
4) Waiting (Non runnable)
5) Dead (Terminated)



NEW

1) Thread object is created.

RUNNABLE:

1) By calling start() method thread goes to runnable stage.

RUNNING

1) When Thread Scheduler selects the thread for execution we say thread is under execution.

TERMINATED

1) Once the execution of a thread is successfully completed it goes to dead stage.
2) A thread can also be stopped when forcefully send to dead stage.

WAIT STAGE

1) A thread which is under execution if interrupted with the help of methods such as sleep (), wait ().
2) A thread can also goes to wait stage if the required resource is not available
3) Java has a beautiful mechanism to create a multiple threads with the help of Thread class.

Why programmer should create a multiple thread?

1) To achieve parallelism
2) To reduce a execution time
3) Efficient use of resource

Creating a thread by extending Thread class

1) Create a class by extending java.lang.Thread class
2) Override the run() Method

Executing a thread:

1) Inside a main method of any class create an object for the class that extends Thread and overrides runnable.

2) Call the start method by using object reference of that instantiated class.

EXAMPLE

```
class MyThread
{
    @Override
    public void run()
    {
        System.out.println(currentThread().
        getName()+" is executing run()");
        System.out.println("Priority is
        "+currentThread().getPriority());
    }
    public static void main(String[] args)
    {
        MyThread t = new MyThread();
        t.start();
        System.out.println(currentThread().
        getName()+" is executing main()");
        System.out.println("Priorityofmain
        "+currentThread().getPriority());
    }
}
```

Steps To Create A Thread By Implementing Runnable Interface

STEP 1:

1) Create a class and implement runnable interface

EXAMPLE :

```
class MyThread implements Runnable
{
    @Override
    public void run()
    {
        // Execution instruction for custom created thread
    }
}
```

STEP 2:

Instantiate the Thread class by passing Runnable type Object as an argument.

EXAMPLE :

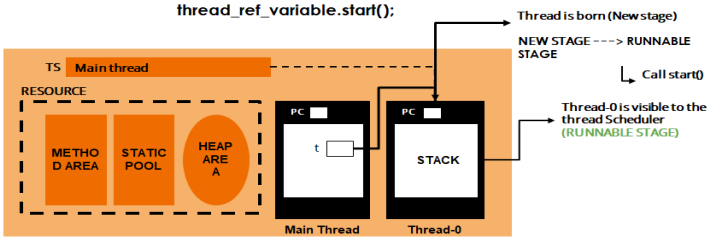
```
Thread t = new Thread(new MyThread());
```

STEP 3:

Move the newly created thread from new stage to ready/runnable stage by calling start() method.

EXAMPLE :

thread_ref_variable.start();

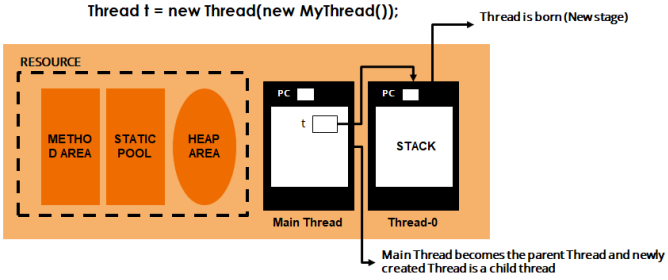


STEP 2:

Instantiate the Thread class by passing Runnable type Object as an argument.

EXAMPLE :

Thread t = new Thread(new MyThread());

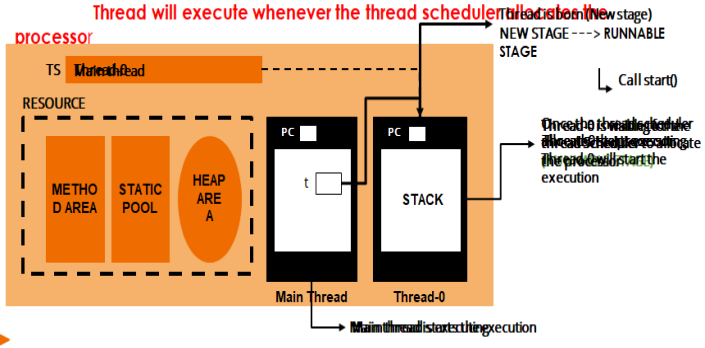


STEP 4:

Executing the thread (Move thread to runnable stage to running stage).

NOTE :

Thread will execute whenever the thread scheduler will assign the processor



Interview Question

1) what is binary language

2) explain briefly levels of languages
3) what is mnemonic
4) What is the use of assembler in assembly level language?
5) can a computer \machine can understand all the characters such as alphabets, special characters, decimal number
6) What is high level language?
7) Example of high level languages.
8) Instruction written in high level language can be converted to machine level language by _____.
9) What is ascii code? Why do we need it?
10) Machine level language also known as.
11) Assembly level also known as.
12) 0represent_____and1represents_____
___
13) What is programming language.
14) Difference between compiler and interpreter.
15) Difference between platform independent and platform dependent.
16) What are all the components of java?
17) What is byte code?
18) Between JDK, JRE, JVM.
19) What is source file? Extension of source file.
20) What is class file? Extension of source file.

1) Features of java?
2) Why java is not pure object oriented language?
    a. Because of it support primitive data type like int byte long
    b. Which are not an object
3) Tell us something about just-in-time (jit) compiler?
    a. Just in time is a component of java runtime environment
    b. Used to improve the performance of the java application at run time
4) What do you understand by "write once and compile anywhere" in java?
    a. Once we write the source code converted into byte codes this byte code run on any operating system like Linux, mac.
5) Pointers are used in c\c++. Why java does not makes use of pointers?
    a. It does not need them because we adding the pointer in java would undermine the security and robustness and make the language more complex
6) What if i don't write main method in program?
    a. Then we can't execute the program but compilation is successful
7) What are tokens? Types of tokens?
    a. Smallest unit off programming.
    b. It is used to compose the instruction

c. Types-keyword, identifier, literals

8) What are keywords? How many keywords are there in java?
9) What is identifier? Rules of identifiers?
10) What are the conventions of class name?
11) What are literals? Types of literals?
12) What is variable? Types of variable?
13) What is scope of a variable?
14) What is local variable? Characteristics of local variable?
15) What is data type? Type's pf data type?
16) What is primitive data type? List out types of primitive data type?
17) What is type casting? Types of type casting?
18) What is widening\implicit type casting?
19) What is narrowing\explicit type casting?
20) What is type cast operator? Use of type cast operator?

1) What is the java ternary operator? Explain with example.
2) What is the difference between the prefix and postfix increment/decrement operators?
3) What is the java modulus operator? Explain with a code sample?
    a. Return the reminder of the two numbers after division

4) What is the difference between logical and operator and the logical or operator?
    a. In logical and operator if both operand are non-zero then condition is true
    b. In logical OR operator if both operand are non-zero then condition is true
5) What are compound assignment operators?
6) What is the difference between ++x and x++ under increment operators?
7) What is the difference between --x and x--under increment operators?
8) Write a complete program explaining the increment and decrement operators in java?
9) What is unary operator? list out unary operators?
10) What is binary operator? list out binary operator?

1) What is method?
2) Syntax to create method?
3) What is method signature?
4) What is method definition?
5) What is method declaration?
6) Explain method terminologies?
7) What is the use of methods?
8) What are access modifiers\access specifiers? Explain briefly?

9) What are modifiers\non-access modifiers?
10) What is return type? list out following return types/
11) What about formal arguments?
12) Explain about actual arguments?
13) What is method call statement? Syntax to create method call statement?
14) Between calling and called method?
15) Explain method call statement flow?
16) Explain about return statement?
17) Rules to access formal argument and actual argument?
18) Types on method?
19) Difference between no argument method and parameterized method?
20) What is main method? Purpose of main method?
21) Can method have return type _____?
22) Rules to define method name?

1) What do you know about java?
2) what are the supported platforms by java programming langue
    a. java standard edition
    b. java enterprise edition

c. **java micro edition**

3) List some java keywords (unlike c, c++ keywords)?
4) List any five features of java?
5) Why is java architectural neutral?
6) How java enabled high performance?
7) Why java is considered dynamic?
8) List two java ide's?
9) What do you mean by object?
10) Define class?
11) What kind of variables a class can consist of?
12) what is a local variable
13) what is a instance variable
14) what is a class variable
15) What is singleton class?
16) What do you mean by constructor?
17) List the three steps for creating an object for a class?
18) What is the default value of byte data type in java?
19) What is the default value of float and double data type in java?
20) When a byte data type is?
21) What is a static variable?
22) What do you mean by access modifier?
23) What is protected access modifier?
24) What do you mean by synchronized non access modifier?

25) According to java operator precedence, which operator is considered to be with highest precedence?
26) Variables used in a switch statement can be used with which data types?
27) When parent () method can be used?
28) Why string class is considered immutable?
29) Why string buffer is called mutable?
30) What is the difference between string buffer and string builder class?

1) How to implements /achieve oops in java?
2) Difference between class and objects?
3) What is mean by instantiation?
4) Initialization? List out different ways?
5) Why constructor is a special method?
6) Difference between overloading and overriding? With example?
7) Inheritance? Types?
8) Which inheritance type is not allowed? Give explanation?
9) Work of this keyword with example?
10) What is super keyword with example?
11) Difference between constructor calling and constructor chaining with example?
12) Difference between this and super keyword?

13) Write a program to show constructor calling and chaining in the same program?
14) What is package? Important of package?
15) Concept of data encapsulation with example?
16) Data hiding?
17) Abstract method?
18) What is abstract class? What is not allowed for a abstract class?
19) When to go for abstract method and abstract class?
20) What is interface?
21) Difference between class and interface?
22) What is abstraction? Explain how to achieve abstraction?
23) What is polymorphism?
24) What is object class? Why it's the super most class?
25) What is final keyword?
26) What are strings? Why string is mutable?
27) What is exception?
28) To handle exception?
29) What are arrays?
30) What are collections?
31) What is collection hierarchy? What are all the methods in collection hierarchy?
32) What is map hierarchy? what are all the methods in map hierarchy?
33) What is thread? what is thread life cycle?

34) What is multithreading?

35) What is data-inconsistency?

36) What is starvation?

37) What is deadlock?

38) What is interring thread communication?