Date :

Complete Notes On

Core Java

Instagram : @ curious_.programmer

Telegram : @ curious_coder

Written By : Damini Patil (CSE)

Date :

Index

Date :

Date :

Date :

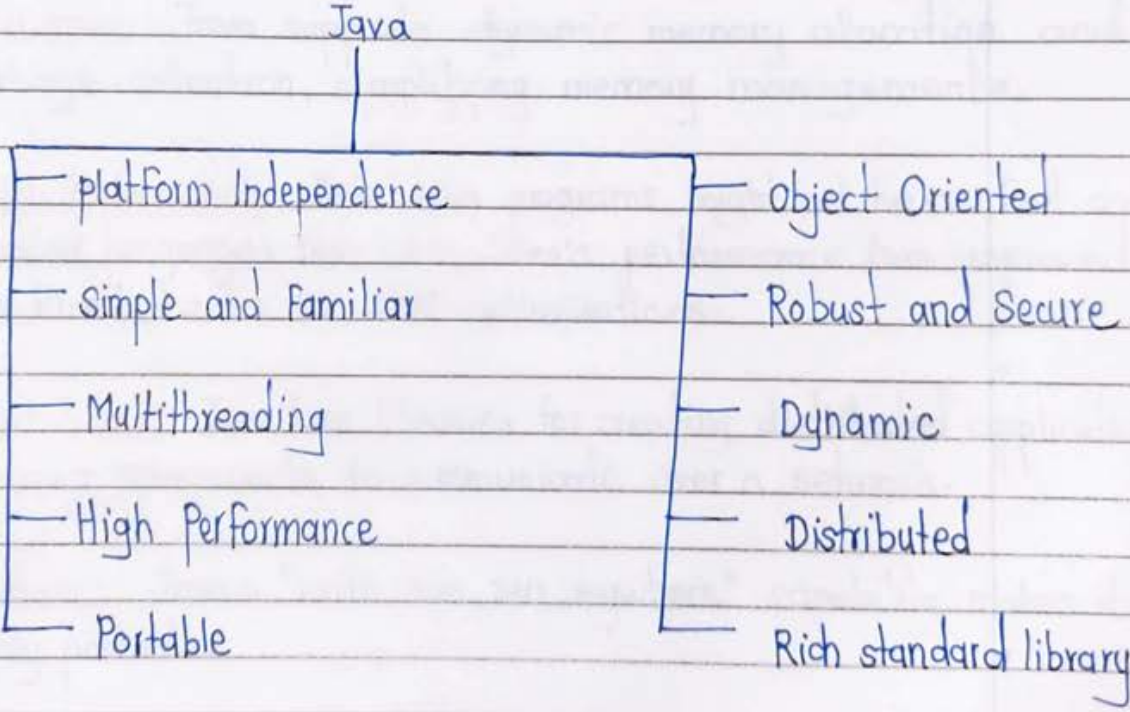
Date :

1. Introduction to Java

1.1 What is Java?

Java is a high-level, object-oriented programming language initially developed by Sun-Microsystems (Now owned by Oracle Corporation). It was designed to be platform-independent, meaning that java programs can run on any device or operating system that has a Java Virtual Machine (JVM) installed. This platform independence is achieved. This platform independence is achieved through a "write once, run anywhere" approach, where code written in java can be compiled into bytecode that is then executed on any system with a compatible JVM.

1.2 Features of Java:



Date :

1. Platform Independence : Java's bytecode can be executed on any platform with the appropriate JVM.

2. Object Oriented : Java follows the object-oriented programming paradigm, emphasizing encapsulation, inheritance, and polymorphism.

3. Simple and Familiar : Java's syntax is inspired by C++ and C, making it familiar to many programmers.

4. Robust and Secure : Java has feature like memory management, strong type checking, and exception handling to ensure robust and secure programs.

5. Multithreading : Java supports multithreading, allowing multiple tasks to be executed concurrently.

6. Dynamic : Java supports dynamic memory allocation and garbage collection, simplifying memory management.

7. High Performance : While Java programs might not be as fast as compiled languages like C++, Java's performance has improved over time, thanks to JVM optimizations.

8. Distributed : Java has libraries for creating distributed applications allowing components to communicate over a network.

9. Portable : Java's "write once, run anywhere" capability makes it highly portable.

10. Rich Standard library : Java provides a vast standard library

Date :

for various tasks, from data structures to network communication.

1.3 Applications of Java:

1. Web Applications: Java is commonly used for building web applications using frameworks, Javaserver faces (JSF) and servelets.

2. Mobile Applications: Java is used for developing Android applications.

3. Desktop Applications: Java Swing and JavaFX are used to create graphical user interfaces for desktop applications.

4. Embedded Systems: Java's portabillity makes it suitable for embedded systems and internet of Things (IoT) devices.

5. Enterprise Applications: Java EE (Java platform, Enterprise Edition) is used for building - large scale enterprise applicatio

6. Scientific and Research Applications: Java's flexibility and libraries makes it useful for scientific simulations and research projects.

1.4 Java Installation:

To install Java, you need to follow these general steps:

1. Download the appropriate Java. development kit (JDK) for

Date : _____

your operating system from the official Oracle Website or adopt Open JDK

2. Run the installer and follow the installations instructions.

3. Set the 'JAVA_HOME' environment variable to point to the JDK installation directory.

4. Update your system's 'PATH' environment variable to include the 'bin' directory within the JDK installation.

1.5 Java Program :

A simple Java program looks like this :

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

1.6 Internal Details : JVM, JRE, JDK:

1. JVM (Java Virtual Machine) :

JVM is a crucial part of the JAVA platform. It executes JAVA bytecode, providing platform independence, It translates bytecodes into machine code for the host system. It manages memory, handles garbage collection, and supports multithreading. JVM implementations exist

Date :

for various platforms.

2. JRE (Java Runtime Environment):

JRE is the environment required to run Java applications. It includes the JVM, class libraries, and other supporting files. JRE allows users to run Java applications without needing the development tools.

3. JDK (Java Development kit):

JDK is the software package that includes the tools and libraries necessary for developing JAVA applications. It includes the JRE, development tools like the JAVA compiler ('javac'), debugger, and other utilities.

In summary, Java is a versatile programming language known for its platform independence, object oriented nature, and extensive standard library. It finds applications in web, mobile, desktop, enterprise, and scientific domains. The Java ecosystem includes components like JVM, JRE, and JDK that collectively enable the creation and execution of JAVA programs.

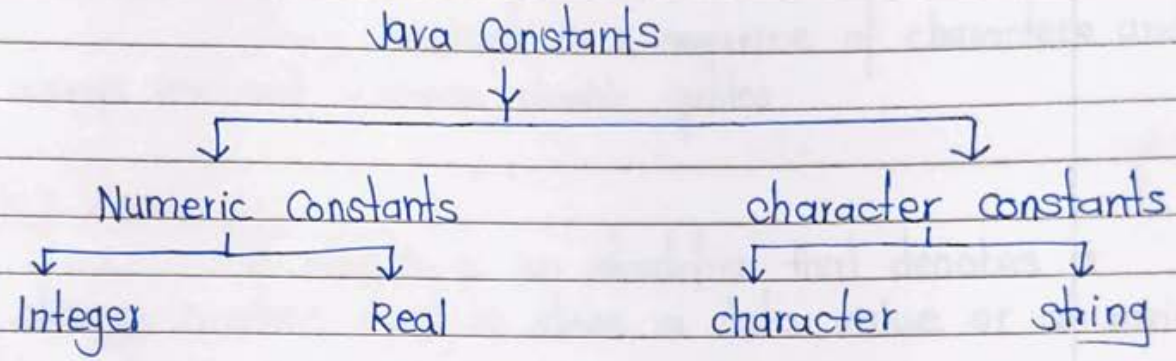
Date :

2. Constants, Variables & Data Types

*

2.1 Constants

In Java constants refer to fixed values that do-not change during the execution of a program. Java supports several types of constants.



1. Integer Constants:

An integer constant means the sequence of digits. There are three types of Integers,

i) Decimal integers consists of digits 0 to 9. Ex: 325, -125

ii) Octal integers consists of any combination of any digits from 0 to 7.
Ex: 027, 123

iii) Hexadecimal Integers - A sequence of digits proceeded by 0x or 0x. They may also include alphabets A to F or a to f.
Ex 0x4, 0x1F, 0xabcd

2. Real Constants:

Integer numbers are insufficient to represent quantities that vary continuously, such as distance, height

Date :

temperatures, prices and so on.

3. character Constant :

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

4. String constant

A string constant is a sequence of characters and always enclosed between double quotes.

2.2 Variables

A variable is an identifier that denotes a storage location used to store a data value or a variable is a basic unit of storage
There are three types of variables:

1) Local variable : Local variable declared inside the body of the method is called local variable.

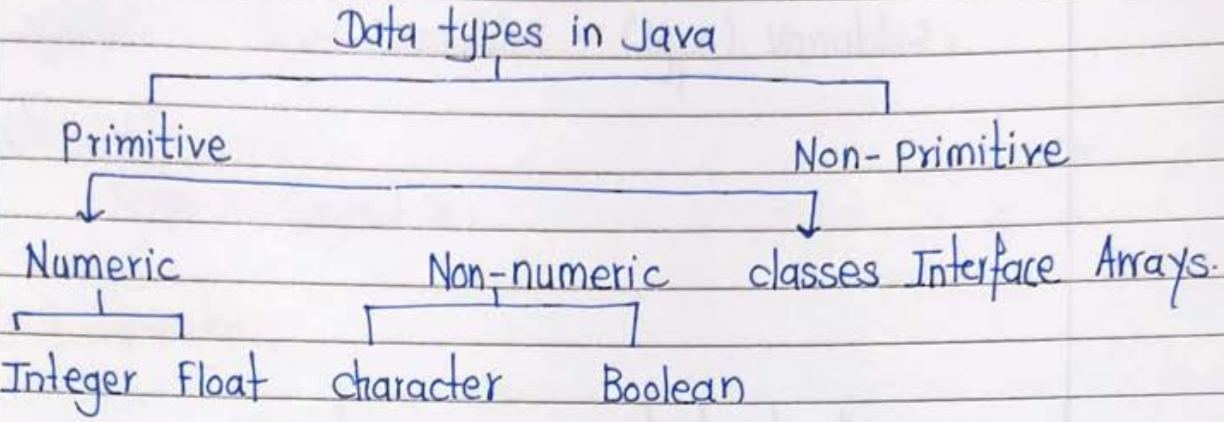
2) Instance variable: A variable declared inside the class but outside the body of the method is called Instance variable.

3) static variable: A variable that is declared as static is called static variable
It cannot be local.
It can create a single copy of static variable and share it among all the instances of the class.

Date :

2.3 Data types

In Java, every variable has a data type. Data type specifies the size and type of data that can be shared.



2.4 Arrays in Java:

Array is a collection of homogeneous or similar data types. elements. Arrays are commonly used to store multiple values in a single variable rather than declaring multiple variables for each value. All arrays are dynamically allocated.

Array declaration

- int [] num;
- string [] cars;
- int [] nums = {1,2,3,4};

Instantiating Array

- Var_name = new type [size];

Date :

Ex: nums = new int [20] ;

2.5 Type Casting

Type casting refers to changing an entity of one data type into another.

syntax: Type variable1 = (type)_Variable2 ;

Ex: int x = 60 ;
byte n = (byte) x ;

2.6 Operators

Operators are classified into following categories:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment operators
5. Increment & Decrement
6. Conditional Operators

- **Arithmetic Operators**

1. Addition (+) : Adds two operands togethers
ex: a+b , where a=1, b=2 , a+b =3

2. Substraction (-) : Subtract the second operand from the first operand.
ex: a-b where a=3 , b=1 , a-b = 2

3. Multiplication (*) : Multiplies one operand by another operand
ex: a*b where a=1 , b=2 , a*b = 2

Date : _____

4. Division (/) : Divides the first operand by second operand and returns the quotients.
ex: a/b , where a=20, b=10, a/b = 2.0

5. Modulus (%) : Returns remainder after dividing the first operand by second operand.
ex: a=6, b=2 remainder is 0.

Relational Operators:

When comparison of two quantities is performed depending on their relation certain decisions are made.

1. Equal to (==) : This operator is used to check whether the two given operands are equal or not.

2. Not equal to (!=) : This operator is used to check whether two given operands are equal or not. If not equal, it returns True.

3. Greater Than (>) : This checks whether the first operand is greater than second or not.

4. Less than (<) : This checks whether the first operand is less than second or not.

5. Greater than or equal to (>=) : This checks whether the first operand is greaterthan or equal to second operand or not.

6. Less than or equal to (<=) : This checks whether the first operand is less than or equal to second operand.

Date :

Logical Operators:

1. Logical AND (&&) : Returns true when both conditions under consideration are satisfied.

2. Logical OR (||) : Returns true when one of the two conditions under considerations are true.

3. Logical NOT (!) : Returns true when condition under the consideration is not satisfied.

Assignment Operators :

Assignment Operators are used to assign the value of an expression to a variable (=)

Increment and decrement :

Java has increment (++) which adds one and decrement (--) operator which substracts one. These operators has following forms.

i) Postfix Increment (a++)
ii) prefix Increment (++a)
iii) postfix decrement (x--)
iv) prefix decrement (--x)

Conditional Operators (Ternary operators)

- expr1 ? expr2 : expr3

It works in three steps:

Date :

1) First exp1 is evaluated for result.
2) If it is true the expression exp2 is evaluated for result.
3) If exp1 is false, exp3 is evaluated.

2.7 keywords:

Java keywords are also known as reserved words. keywords are special words that have specific meaning and purpose within Java language. They are reserved and cannot be used as variable names or identifiers. keywords play a crucial role in defining the structure and behaviour of programs. keywords are like building block that allow us to create conditional statements loops, functions, classes, handles errors and perform other important operations.

List of Java keywords:

short	do	class	break
continue	default	float	super
boolean	double	public	import
Protected	transient	static	throw
case	native	throws	while
this	return	finally	interface
abstract	new	long	try
if	implements	private	extends
for	int	else	package
void	switch	catch	char
final	byte	instance of	synchronized

Date :

3. Flow control in Java

3.1 Decision making statement :

Decision-making statement allows you to execute different code blocks based on certain conditions. In Java, you have two main decision making statements:

if statement
if-else statement
else-if statement (Ladder)

if statement

The 'if' statement is used to execute a block of code if a given condition is true. It has the following syntax:

```
if (condition) {
    //code to execute if the condition is true
}
```

Example:

```
int number = 10;
if (number > 0) {
    System.out.println(" Number is positive");
}
```

if-else statement :

The 'if-else' statement extends the 'if' statement by allowing you to execute different blocks of code based on

Date :

whether a conditions is true or false.

```
if (condition) {
    // code to execute if the condition is true
} else {
    // code to execute if the condition is false
}
```

Example:

```
int num = -5;
if (num > 0) {
    System.out.println("Number is positive");
} else {
    System.out.println("Number is non-positive");
}
```

else-if Ladder:

You can also use an 'else-if' ladder to test multiple conditions sequentially.

```
if (condition1) {
    // code for condition1
} else if (condition2) {
    // code for condition2
} else if (condition3) {
    // code for condition3
} else {
    // code if none of the conditions are true
}
```

Date :

Example :

```
int score = 85;
if (score >= 90){
  System.out.println("Grade : A");
}else if (score >= 80){
  System.out.println("Grade : B");
}else if (score >= 70){
  System.out.println("Grade : C");
}else{
  System.out.println("Grade : F");
}
```

3.2 Switch statement :

The 'switch' statement in Java is used to evaluate an expression against a series of constant values. It provides an efficient way to perform different actions based on the value of the expression. Here's the syntax of the 'switch' statement :

```
switch(expression){
  case value1:
      //code to execute if expression equals value1
    break;
  case value2:
     // code to execute if expression equals value2
    break;
  case valueN:
    // code to execute if expression equals valueN
    break;
  default : // code to execute if none of the cases match expr.
  }
```

Date :

Example:

```
int day = 3;
string dayName;

switch(day){
  case 1:
    dayName = "Monday";
    break;
  Case 2:
    dayName = "Tuesday";
    break;
  Case 3:
    dayName = "Wednesday";
    breack;
  default:
    dayName = "Invalid day";
}
System.out.println("Day:" +dayName);
```

3.3 Loops in Java:

Loops allow you to repeatedlly execute a block of a java code. Java supports three types of loops:

1) For loop

2) While loop

3) do-while loop

Date :

For loop:

The 'for' loop is used for iterating a specific numbers of times.

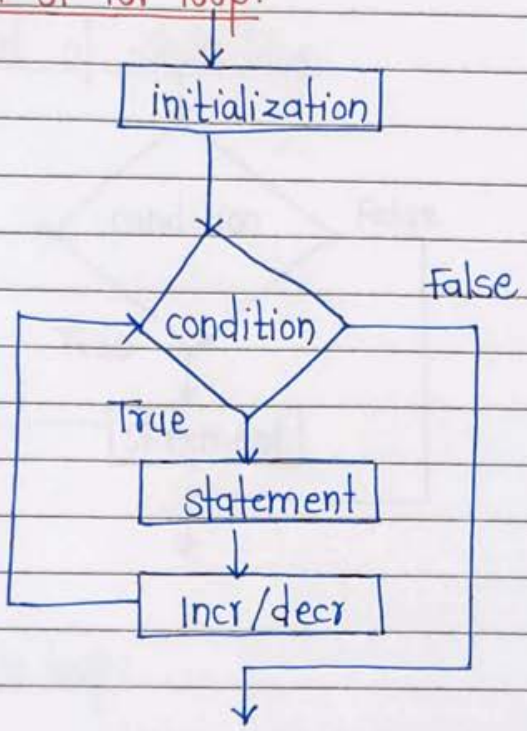
syntax:

```
for(initialization; condition; update){
   //code to execute in each iteration
}
```

Example:

```
For (int i=1; i<=5; i++){
   System.out.println ("Iteration" +i);
}
```

Flowchart of for loop:



While loop:

The 'While' loop repeatedly executes a block of code as long as a given condition is true.

Date :

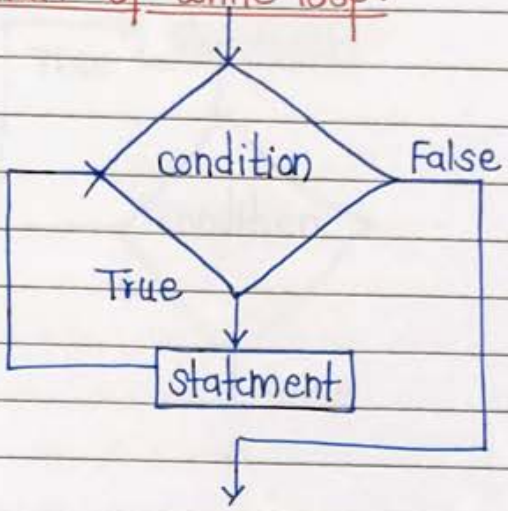
syntax:

```
While(condition){
 // code to execute as long as the condition is true.
}
```

Example :

```
int count = 0;
while(count<5){
  system.out.println("count: "+count);
  count++;
}
```

Flowchart of while loop:



do-while loop:

The 'do-while' loop is similar to the 'while' loop, but it ensures that the code block is executed at least once before checking the condition.

syntax:

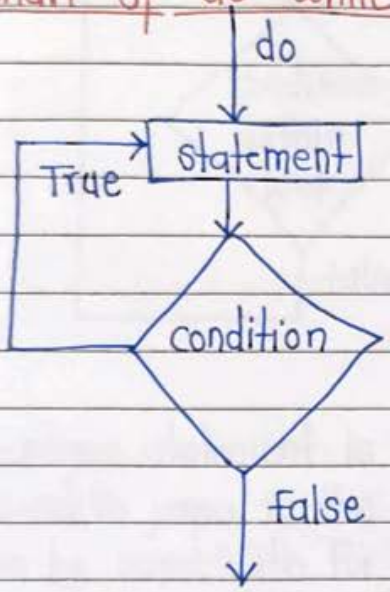
Date :

```
do{
 // code to execute at least once
}while(condition);
```

Example:

```
int x = 1;
do{
  System.out.println("value of x: "+x);
  x++;
}while(x<=5);
```

Flowchart of do-while loop:



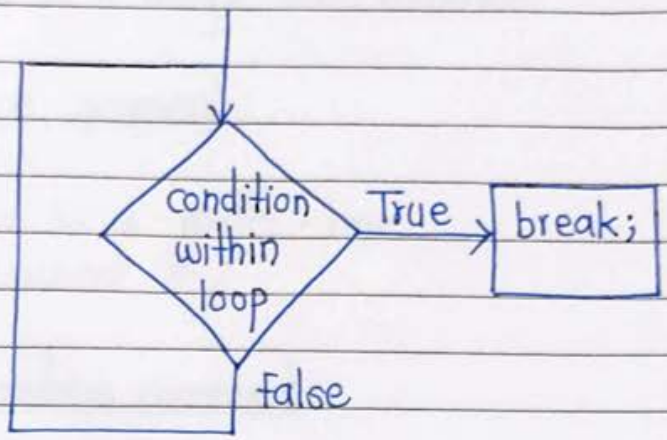
3.4 Break and continue statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Example:

```
for (int i = 1; i<=10; i++) {
  if (i == 6) {
    break; //Exit the loop when i equals 6
  }
  system.out.println("value of i: " +i);
}
```

Flowchart of Break statement



The continue statement is used to loop control structure when you need to jump to the next iteration of the loop immediatly. It can be used with for loop or while loop. The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

Example

```
for (int i =1; i<=10; i++) {
  if (i==5) {
    continue; // It will skip the rest statement
  }
  System.out.println(i);
}
```

3.5 Comments in Java :

Comments are non-executable statements that provide explainations within the code. They are used to improve code readability and understanding. Java supports three types of comments.

Single-Line Comment :

```
// This is a single line comment
```

Multi-line comment :

```
/* This is a multi-line
   comment */
```

Documentation comment :

```
/**
 * This is a documentation comment.
 * It is used for generating documentation.
*/
```

4. Objects and Classes

Object:

An object in java is the physical as well as a logical entity. An entity that has state and behaviour is known as an object.

Ex: chair, bike, pen, etc.

An object is the instance of a class. An object has three characteristics.

- state: Represents data of an object
- Behaviour: Represents behaviour (functionality) of an object.
- Identify: It is used internally by JVM to identify each object uniquely.

Class:

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is the logical entity. Class is a user defined blueprint or prototype from which objects are created.

A class in Java contains:

- fields
- methods
- constructors
- Blocks
- Nested class and interface

Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

Date :

Instance variable in Java:

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in java:

In Java, a method is like a function which is used to expose the behaviour of an object

Advantage of Method:

- code reusability
- code Optimization

New keyword in Java:

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object and Class Example : main within the class :

Here, we are creating a main() method inside the class

```
class Student{
int id;
string name;
public static void main (string args[]){
 Student st = new Student ();
```

Date :

```
System.out.println(s1.id);
System.out.println(s1.name);
}
}
```

Output :

```
0
null
```

Object and Class Example : main outside the class

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

3 ways to initialize object :

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

Date :

1) Object and class Example: Initialization through reference

Initializing an object means storing data into the objec
Let's see a simple example.

```
class Student{
int id;
String name;
}
class TestStudent2{
 public static void main (String args[]){
   Student s1 = new Student();
   s1.id = 101;
   s1.name = "Sonoo";
   System.out.println (s1.id+ " "+s1.name); //printing member
 }
}
```

output:

```
101 Sonoo
```

2) Object and class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state(data) of the objects by invoking the displayInformation() method.

```
class Student{
int rollno;
```

```
String name;
void insertRecord(int r, String n){
  roll_no = r;
  name = n;
}
void displayInformation(){
  System.out.println(rollno + " " + name);}
}
class TestStudent4{
  public static void main(String args[]){
    Student s1 = new Student();
    Student s2 = new Student();
    s1.insertRecord(111, "Karan");
    s2.insertRecord(222, "Aryan");
    s1.displayInformation();
    s2.displayInformation();
  }
}
```

Output:

```
111 Karan
222 Aryan
```

Date :

5. Constructor in Java

5.1 Constructor :

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class.

Rules For creating Java constructor :

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

5.2 Constructor Types :

There are two types of constructor in Java.

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

Java default constructor :

A constructor is called "Default constructor" when it doesn't have any parameter.

Syntax:

```
<class_name>(){}
```

Example:

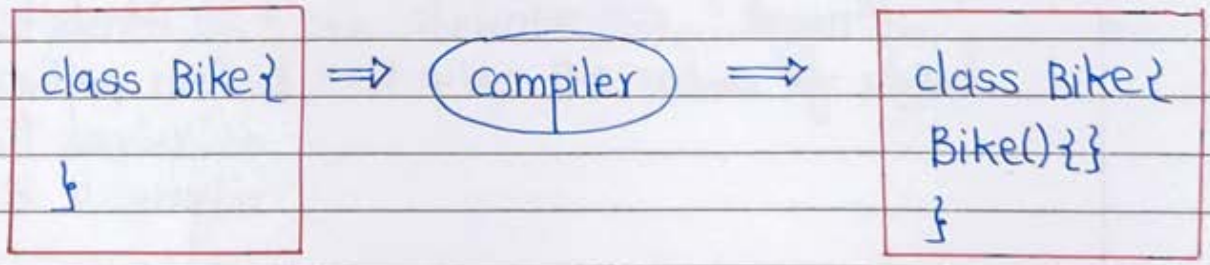
```
// Java program to create and call a default constructor
class Bike1{
// creating a default constructor
Bike1(){ System.out.println("Bike is created");}
// main method
public static void main(String args[]){
// calling a default constructor.
Bike1 b = new Bike1();
}
}
```

Output:

```
Bike is created.
```

Rule:

If there is no constructor in a class, compiler automatically creates a default constructor.



The default constructor is used to provide the default values to the object like 0, null, etc..., depending on the type.

Date

Java parameterized constructor:

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

Example:

```
// Java program to demonstrate the use of the parameterized constructor.
class Student4 {
  int id;
  String name;
 // creating a parameterized constructor
  Student4(int i, String n) {
  id = i;
  name = n;
  }
 // method to display the values
  void display() { System.out.println(id+" "+name);}
  public static void main (String args []) {
 // creating objects and passing values
   Student4 s1 = new Student4 (111, "Karan");
   Student4 s2 = new Student4 (222, "Aryan");
 // calling method to display the values of object
   s1.display();
   s2.display();
  }
 }
```

Output:

```
111 Karan
222 Aryan
```

Date :

5.3 Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

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

Example of constructor overloading :

```
public class Student {
    string name;
    int age;
    // constructor with one parameter
    public Student (string name){
        this.name = name;
    }
    // Constructor with two parameters
    public Student (string name, int age){
        this.name = name;
        this.age = age;
    }
}
```

5.4 static keyword :

The static keyword in java is used for the memory management mainly. We can apply static keyword.

Date :

with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

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

Java static variable:

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

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

Java static Method:

- A static method belongs to the class rather than object of a class
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Java static block:

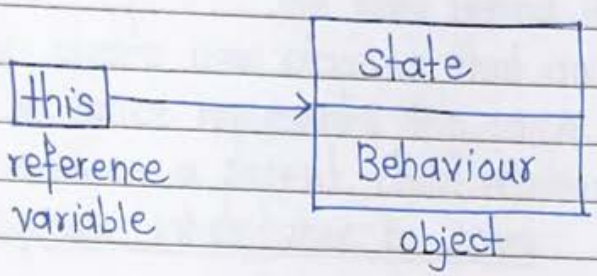
- Is used to initialize the static data member.

Date :

- It is executed before the main method at the time of class boarding.

5.5 this keyword in Java :

There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the curren object.



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

Example:

```
Public class Person{
  string name;
  public Person(string name){
    this.name = name; // "this.name" refers to instance variable
  }
  Public void printname(){
    System.out.println("Name:" + this.name);}}
```

Date :

6 Inheritance

6.1 What is inheritance ?

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

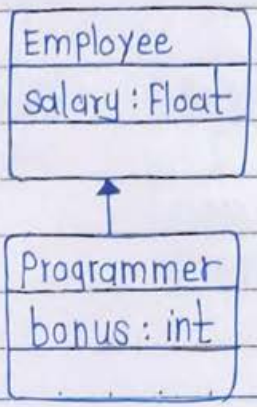
Why use inheritance in Java :

- For method Overriding (so runtime polymorphism can be archieved)
- For code reusability.

Syntax:

```
class Subclass-name extends Superclass-name
{
  // methods and fields
}
```

Example:



Date :

Example :

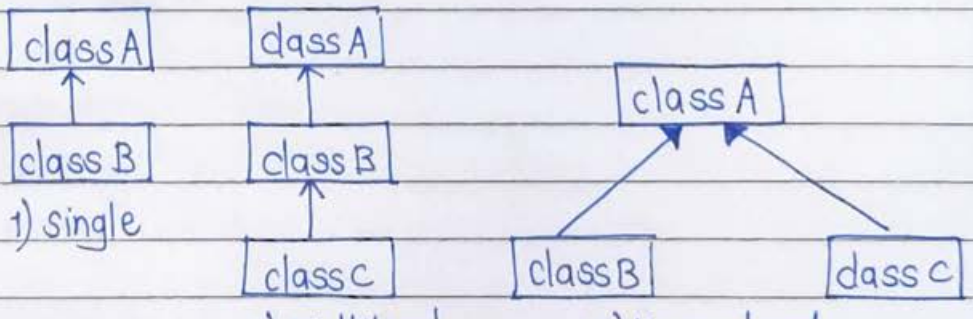
```
class Employee{
  float salary =40000;
}
class Programmer extends Employee{
int bonus =10000;
public static void main(string args[]){
  Programmer P =new Programmer();
   System.out.println("programmer salary is:"+p.salary);
   System.out.println(" Bonus of programmer is:"+p.bonus);
}
}
```

Output:

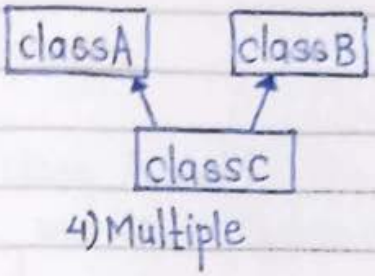
```
programmer salary is : 40000.0
Bonus of programmer is : 10000
```

6.2 Types of inheritance in Java :

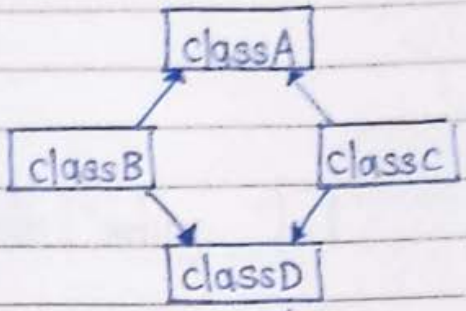
On the basis of class, there can be three types of inheritance in java : single, multilevel and hierarchical. In Java programming, multiple and hybrid inheritance is support through interface only.



Multiple Inheritance is not supported in Java through class



4) Multiple



5) Hybrid

Single Inheritance Example:

When a class inherits another class, it is known as a single inheritance.

```
class Animal{
 void eat(){system.out.println("eating...");}
}
 class dog extends Animal{
  void bark(){system.out.println("barking...");}
}
 class TestInheritance{
 public static void main(String args[]){
  Dog d = new Dog();
    d.bark();
    d.eat();
  }}
```

Output:

```
barking....
eating....
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is

Date :

known as multilevel inheritance.

Example:

```
class Animal {
  void eat() { System.out.println("eating...");}
}
 class dog extends Animal {
 void bark() { system.out.println("barking...");}
 }
 class BabyDog extends dog {
 void sweep() { system.out.println("weeping...");}
 }
 class inheritance2 {
  public static void main (string args[]){
   BabyDog d = new BabyDog();
    d.weep();
    d.bark();
    d.eat();
  }
 }
```

Output:

```
weeping....
barking....
eating....
```

Hierarchical Inheritance Example:

When two or more classes inherits a single class, it is known as hierarchical inheritance. Ex: Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
class Animal {
 void eat() { system.out.println("eating...");}
}
class Dog extends Animal{
  void bark() { system.out.println("barking..");}
}
class Cat extends Animal{
  void meow() { System.out.println("meowing..");}
}
class TestInheritance{
  public static void main (string args[]){
  Cat c = new Cat();
   c.meow();
   c.eat();
  }}
```

Output:

```
meowing...
eating...
```

6.3 Super keyword

The 'super' keyword in Java is used to refer to the superclass and its members from within the subclass. It is often used to access overriden methods, constructor chaining, and to differentiate between superclass, subclass members with the same time.

example:

```
class Animal {
   void eat() {
```

Date :

```
        System.out.println("Animal is eating.");
    }
}
class Dog extends Animal{
    void eat(){
        System.out.println("Dog is eating.");
    }
void eatsomething(){
    Super.eat(); //calls the eat() method of the superclass
    }
}
```

6.4 Aggregation in Java

Aggregation is a form of association between classes where one class represents the part of or container (the 'whole' class) while another class represents the part of contained objects. Aggregation is typically expressed as a "has-a" relationship, where one class contains objects of another class as its instance variables.

Example of aggregation:

```
class Address{
    String street;
    String city;
    Address(String street, String city){
        this.street = street;
        this.city = city;
    }
}
class Person{
```

```
string name;
Address address;

Person(string name, Address address){
  this.name = name;
  this.address = address;
}
}
```

Date :

7. Polymorphism

7.1 Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as Method overloading. If we have to perform only one operation, having same of the methods increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as (int, int) for two parameters, and b(int, int, int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method beacause its name differs. So we perform method overloading to figure out the program quickly.

Advantages of method Overloading :

Method overloading increases the readability of the program.

Different ways to overload the method :

There are two ways to overload the method in java :

1. By changing number of arguments
2. By changing the data type

Method Overloading : changing no. of arguments

In this example, we have created two methods

first add() method performs addition of two numbers and second add() method performs addition of three numbers.

```
class Adder{
static int add (int a, int b){return a+b;}
static int add(int a, int b, int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
  System.out.println(Adder.add(11,11));
  System.out.println(Adder.add(11,11,11));
}}
```

output:

22
33

Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Date :

output:
22
24.9

7.2 Method Overriding in Java:

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java. In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding:

- Method Overiding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method Overriding is used for runtime polymorphism.

Rules for Java Method Overriding:

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

Example of Method Overriding:

```
// Java program to illustrate the use of Java Method Overriding
// creating a parent class
class Vehicle {
    // defining a method
```

Date :

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

Output:

```
Bike is running safely.
```

7.3 Final keyword in Java :

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

1. Variable
2. Method
3. class

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

7.4 Runtime polymorphism :

Polymorphism in Java is a concept by which

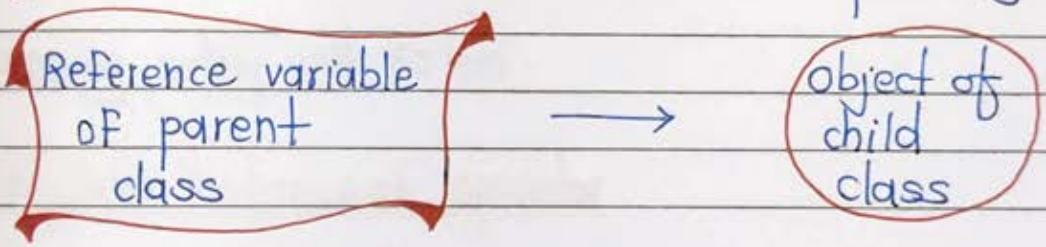
Date : _____

we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: Poly and morphs. The word "poly" means many and "morph" means forms.

Runtime polymorphism or Dynamic Method dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile time.
In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being reffered to by the reference variable.

Upcasting :

If the reference variable of parent class refers to the object of child class, it is known as upcasting.




```
class A{}
class B extends A{}

A a=new B(); //upcasting
```

For upcasting, we can use the reference varible of class type or an interface type. For example:

```
interface I{}
```

Date :

```
class A{}
class B extends A implements I{}
```

Java Runtime Polymorphism Example:

```
class Bike{
  void Run(){system.out.println("running");}
}
class splendor extends Bike{
  void run(){system.out.println("running safely with 60km");}
public static void main(String args[]){
  Bike b = new Splendor();
  b.run();
  }
}
```

Output:

running safely with 60 km.

7.5 Java instanceof operator

The java instanceof operators is used to test whether the object is an instance of the specified type(class or subclass or interface)

The instanceof in Java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false

Date :

example:

```
class Simple1{
public static void main(String args[]){
Simple1 s = new Simple1();
System.out.println(s instance-of simple1);
}
}
```

output:

True.

Date :

8. Abstraction

8.1 What is abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in Java.

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java :

A class which is declared as abstract is known as an abstract class. It can be have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated. An abstract class must be delared with an abstract keyword. It can have abstract and non-abstract methods. It cannot be instantiated. It can have constructors and static method also. It can have final methods which will force the subclass not to change the body of the method.

Example:

```
abstract class A{}
```

Abstrat Method in Java:

A abstract method is a method which is declared as abstract and does not have implementation.

Example:

Date :

abstract void printstatus(); // no method body and abstract.

Example of abstract class having abstract method

```
abstract class Bike{
  abstract void run();
  }
class Honda4 extends Bike{
 void run(){ system.out.println("running safely");}
 public static void main (string args[]){
  Bike obj = new Honda4();
  obj.run();
  }
 }
```

8.2 Interface :

An interface in Java is a blueprint of a class. It has static constants. The inheritance in java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in java.

Java interface also represents the Is-A relationship.
It cannot be instantiated just like a abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.

Date :

- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Interface Syntax:

```
interface <interface_name>{
  // declare constant fields
  // declare methods that abstract
  // by default.
}
```

Java interface Example

```
interface printable{
 void print();
}
class AG implements printable{
 public void print(){ system.out.println("Hello"); }

 public static void main(string args[]){
   AG obj = new AG();
   obj.print();
  }
}
```

Output:

```
Hello
```

Date :

8.3 Difference between abstract class and interface:

Abstract class	Interface
Abstract class have abstract and non-abstract method	Interface can have only abstract methods.
Doesn't support multiple Inheritance.	Supports Multiple inheritance.
Can provide the implementation of interface.	Can't provide implementation of abstract class.
Abstract class can have class members like private, protected, etc	Members of java interface are public by default.

Date :

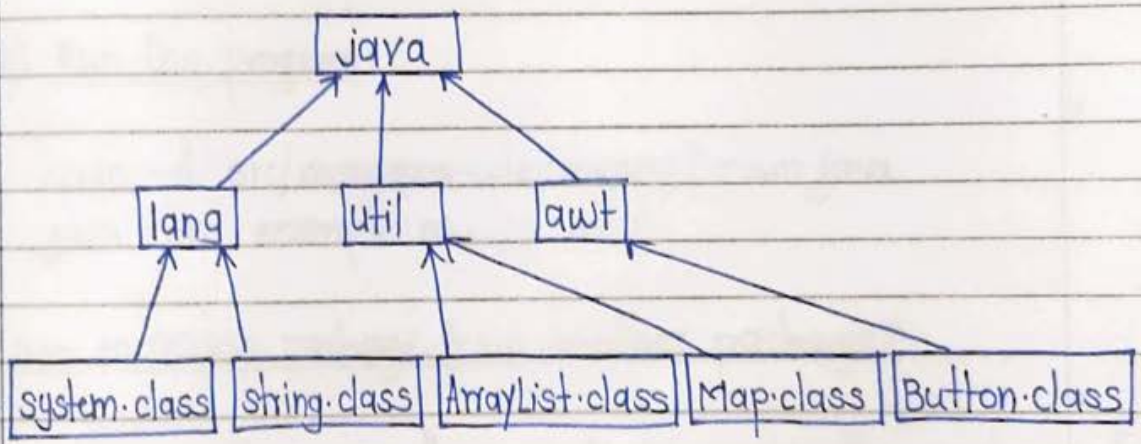
9 · Java Encapsulation

9.1 Java package:

A java package is a group of similar types of classes, interfaces, subpackages. Packages in java can be categorized in two form, built-in packages and user-defined packages. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql, etc.

Advantages of Java package:

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision



Creation of Java packages:

1) Package name - 'com.example.myapp'.

2) Create directory structure:
mkdir -p src/com/example/myapp

Date :

3) Write a class:

```
package com.example.myapp;
  public class Myclass{
    public void display(){
      System.out.println("Hello from myclass!");
  }
}
```

4) Compile the code:

```
javac -d .src/com/example/myapp/*.java
```

5) Run the program:

```
Java com.example.myapp.Main
```

6) Run the program:

```
Javac -d. src/com/example/myapp/Main.java
java com.example.myapp.main.
```

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname
3. fully qualified name.

Date :

9.2 Access Modifiers in Java.

The access modifiers in java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifer on it.
There are four types of Java access modifiers:

1) private:

Private access modifier is accessible only within the class

Example:

```
class A{
 private int data=40;
 private void msg(){ system.out.println("Hello java");}
 }
 public class simple{
  public static void main(string args[]){
   A obj = new A();
   System.out.println(obj.data);
   obj.msg();
  }
 }
```

2) Default:

If you don't use any modifier, it is created as default by default. The default modifier is accessible

only within package.

3) Protected:

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

4) Public:

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

9.3 Encapsulation in Java:

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example a capsule which is mixed of several medicines.

Advantages of Encapsulation

By providing only a setter or getter method, you can make the class read-only or write-only. In otherwords, you can skip the getter and setter methods.

It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve data hiding in Java beacause other class will not be able to access the data through the private data members.
The encapsulates class is easy to test, so it is better for unit testing.

Date :

10. Java arrays

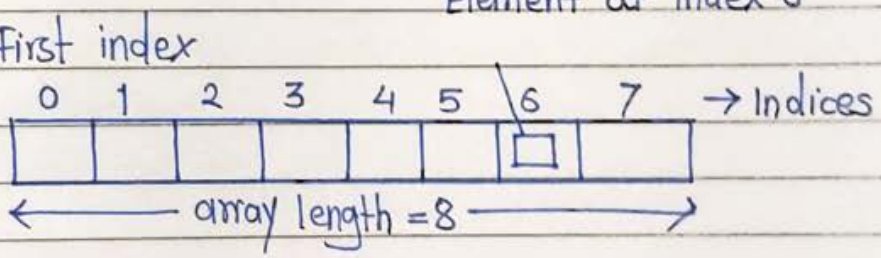
10.1 Single Dimension array

Normally, an array is a collection of similar type of elements which has contiguous memory location.

Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are ~~sorted~~ stored in a contiguous memory allocation. It is a data structure where we store similar elements. We can store only a fixed set of elements in a java array.

Array in java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index.

Unlike C/C++, we get the length of the array using the length member. In C/C++, we need to use the sizeof operator.



Single dimensional array syntax:

```
dataType[] arr;
```

Instantiation:

```
arrayRefVar = new datatype[size];
```

Date :

Example:

```
// Java program to illustrate how to declare, instantiate, initialize
// and traverse the Java array
class Testarray {
public static void main(String args[]){
int a[] = new int[5]; //declaration and instantiation
 a[0] =10; // initialization
 a[1] =20;
 a[2] =70;
 a[3] =40;
 a[4] =50;
 // traversing array
 for(int i=0 ; i<a.length; i++)
 System.out.println(a[i]); //length is the property of array.
 }}
```

Output:

```
10
20
70
40
50
```

10.2 Multidimensional array

In such case, data is stored in row and column based index (also known as matrix form).

Syntax:

```
dataType[][] arrayRefVar;
```

Date :

Instantiation:

```
int[][] arr = new int[3][3]; // 3 rows and 3 column.
```

Example to initialize Multidimensional Array in Java:

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of multidimensional array:

```
// Java program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
// declaring and initializing 2D array
int arr[][] = {{1,2,3},{2,4,5},{4,4,5}};
// printing 2D array
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
  System.out.println(arr[i][j]+" ");
 }
System.out.println();
}
}}
```

Date :

Output :

```
1 2 3
2 4 5
4 4 5
```

10.3 cloning an Array in Java :

Since, Java array implements the cloneable interface we can create the clone of the Java array. If we create the clone of a single-dimensional array, it creates the deep copy of the java array. It means, it will copy the actual value. But, if we create the clone of a multidimensional array, it creates the shallow copy of the Java array which means it copies the references.

```
// Java program to clone the array
class Testarray1{
 public static void main (String args[]){
 int arr [] ={33,3,4,5};
  System.out.println(" printing original array:");
 for(int i:arr)
  System. out.println(i);
  system.out.println(" printing clone of the array:");
  int carr[] = arr.clone();
  for (int i: carr)
   System.out.println(i);
   System.out.println ("Are both equal?");
   System. out.println ( arr == carr);
}}
```

Date :

Output:

printing original array:
33
3
4
5
printing clone of the array:
33
3
4
5
Are both equal?
false

Date :

11. Object and Math Class

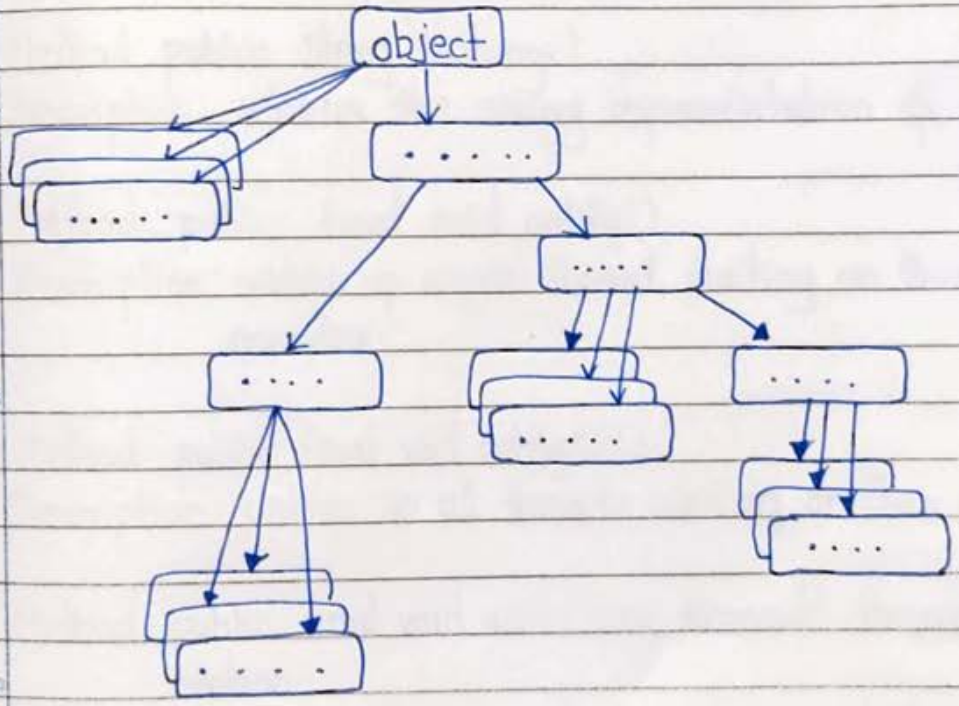
11.1 Method of Object Class:

The object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java. The object class is beneficial if you want to refer any object whose type you don't know.

syntax:

```
Object obj = getobject();
```

The object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified, etc.



The object class provides many methods. They are given as

Date :

Follows:

Method: public final class getClass()
Description: return the class class object this object. The Class class can further be used to get the meta-data of this class.

Method: public int hashCode()
Description: returns the hashcode number for this object.

Method: public boolean equals(object obj)
Description: Compares the given object to this object.

Method: protected object clone() throws CloneNotSupportedException
Description: creates and returns the exact copy (clone) of this object.

Method: public String toString()
Description: returns the string representation of this object.

Method: public final void notify()
Description: wakes up single thread, waiting on this object's monitor.

Method: public final void notifyAll()
Description: wakes up all threads, waiting on this object's monitor

Method: public final void wait(long timeout) throws InterruptedException
Description: causes the current thread to wait for the specified milliseconds, untill another thread notifies (invokes notify()

Date :

or notifyAll() method)

Method: public final void wait() throws InterruptedException
Description: causes the current thread to wait, untill another thread notifies (invokes notify())

Method: public void finalize() throws throwable
Description: is invoked by the garbage collector before object is being garbage collected.

11.2 Object cloning:

The object cloning is a way to create exact copy of an object. The clone() method of object class is used to clone an object. The java.lang.cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement cloneable interface, clone() method generates cloneNotSupportedException.

The clone() Method is defined in the object class. Syntax of the clone() method is as follows:

protected Object clone() throws CloneNotSupportedException

Why use clone() Method?

The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we object cloning.

Date :

Advantage of object cloning:

- You don't need to write lengthy and repetitive codes, just use an abstract class with a 4-or 5-line long clone() method.
- It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project.
- clone() is the fastest way to copy array.

Example of clone() method (object cloning):

```
class Student18 implements cloneable{
int rollNo;
String name;

Student18 (int rollNo, String name){
this.rollNo = rollNo;
this.name = name;
}

public object clone() throws cloneNotSupportedException{
return super.clone();
}
public static void main(String args[]){
try{

Student18 s1 = new Student18 (101, "amit");

Student18 s2 = (Student18)s1.clone();
System.out.println(s1.rollNo+" "+s1.name);
System.out.println(s2.rollNo+" "+s2.name);
```

```
}
 catch (cloneNotsupportedException c){}
 }
}
```

11.3 Java Math class

Java math class provide sereral methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(labs(), etc.

Unlike some of strictMath class numeric methods, all implementations of the equivalent function of Math class can't define to return the bit-for-bit same results

Example:

```
public class Javamath
 {
  public static void main (string[] args
    {
      double x = 28;
      double y = 4;
      //return the maximum of two members
      System.out.println ("Maximum number of x andy is:"
        + Math.max (x, y));
      //return the square root of y
      System.out.println ("square root of y is: "+Math.sqrt(y));

 }
}
```

Date :

11.4 Basic Math Methods

The java.lang.Math class contains various methods for performing basic numeric operations such as the logarith, cube root, and trignometric functions, etc. The various java math methods are as follows:

Method	Description
1) Math.abs()	It will return the absolute value of the given value.
2) Math.max()	It returns the Largest of two values.
3) Math.min()	It is used to return the smallest of two values.
4) Math.round()	It is used to round of the decimal numbers to the nearest value.
5) Math.sqrt()	It is used to return the square root of a number.
6) Math.cbrt()	It is used to return the cube root of a number.
7) Math.pow()	It returns the value of first argument raised to the power to second argument.
8) Math.Signum()	It is used to find the sign of a given value.

Date :

Method	Description
9) Math.ceil()	It is used to find the smallest integer value that is greater than or equal to the argument or mathematical integer.
10) Math.CopySign()	It is used to find the absolute value of first argument along with sign specified to an second argument.
11) Math.nextAfter()	It is used to return the floating point number adjacent to the first argument in the direction of second argument.
12) Math.nextdown()	It returns the floating point value adjacent to d in the direction of negative infinity.
13) Math.floor()	It is used to find the largest integer value which is less than or equal to argument
14) Math.random()	It returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
15) Math.rint()	It returns the double value that is closest to the given argument and equal to the mathematical integer.
16) Math.hypot()	It returns $\sqrt{(x^2+y^2)}$ without intermediate overflow or underflow.

Date :

Method	Description
17) Math.ulp()	It returns the size of an ulp of the argument.
18) Math.getExponent()	It is used to return the unbiased exponent used in the representation of a value.
19) Math.addExact()	It is used to return the sum of its arguments, throwing an exception if the result overflows an int or long.
20) Math.subtractExact()	It returns the difference of the arguments, throwing an exception if the result overflows an int.
21) Math.multiplyExact()	It is used to return the product of the arguments, throwing an exception if the result overflows an int or long.
22) Math.incrementExact()	It returns the argument incremented by one, throwing an exception if the result overflows an int.
23) Math.decrementExact()	It is used to return the argument decremented by one, throwing an exception the result overflows an int or long.
24) Math.toIntExact()	It returns the value of the long argument, throwing and exception if the value overflows an int.

Date :

11.5 Logarithmic Math Methods:

Method	Description
1) Math.log()	It returns the natural logarithm of a double value.
2) Math.log10()	It is used to return the base 10 logarithm of a double value.
3) Math.log1P()	It returns the natural logarithm of the sum of the argument and 1.
4) Math.exp()	It returns E raised to power of a double value, where E is Euler's number and it is approximately equal to 2.71828.
5) Math.expm()	It is used to calculate the power of E and subtract one from it.

11.6 Trigonometric Math Methods:

Method	Description
1) Math.sin()	It is used to return the trigonometric sine value of a Given double value.
2) Math.cos()	It is used to return the trigonometric cosine value of a given double value.

Date :

Method	Description
3) Math.tan()	It is used to the trigonometric tagent of value
4) Math.asin()	It is used to return the trigonometric Arc sine value of a given double value.
5) Math.acos()	It is used to return the trigonometric Arc cosine value of a given double value.
6) Math.atan()	It is used to return the trigonometric Arc Tangent value of a given double value.

11.7 Wrapper class in Java.

The Wrapper class in Java provides the mechanism to convert primitive into object and object into primitive. Java is an object oriented programming language, so we need to deal with objects many times like in collections, Serialization, Synchronization, etc

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long

Date :

11.8 Autoboxing and Unboxing in Java:

Autoboxing : The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, 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.

```
public class WrapperExample1{
 public static void main (String args[]){
  int a =20;
 integer i = Integer.valueof(a);
 integer j = a; //autoboxing

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

output:
20 20 20

Unboxing: The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. 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 primtives.

```
public class WrapperExample2{
  public static void main (String args[]){
    Integer a = new Integer(3);
    int i = a.intValue();
    int j = a;
    System.out.println (a+" "+i+" "+j);
}}
```

Output:
3 3 3

Date :

12. Recursion

12.1 Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method. It makes the code compact but complex to understand.

Syntax:

```
returntype methodname() {
// code to be executed
methodname(); // calling same method
}
```

Java Recursion Example 1 : InFinite times

```
public class RecursionExample1 {
 static void p(){
  System.out.println("hello");
  p();
 }
public static void main(String[] args){
 p();
}
}
```

Output:

```
hello
hello
.....
Java.lang.stackOverFlowError
```

Date :

12.2 Call by value and call by Reference in Java :

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java:

```
class Operation{
 int data = 50;
 void change (int data){
  data = data + 100; // changes will be in the local variable only
 }
 public static void main (string args[]){
   Operation op = new Operation();

 System.out.println("before change" + op.data);
 op.change(500);
 System.out.println("after change" + op.data);
 }
}
```

Date :

13 · Java String

13·1 Introduction

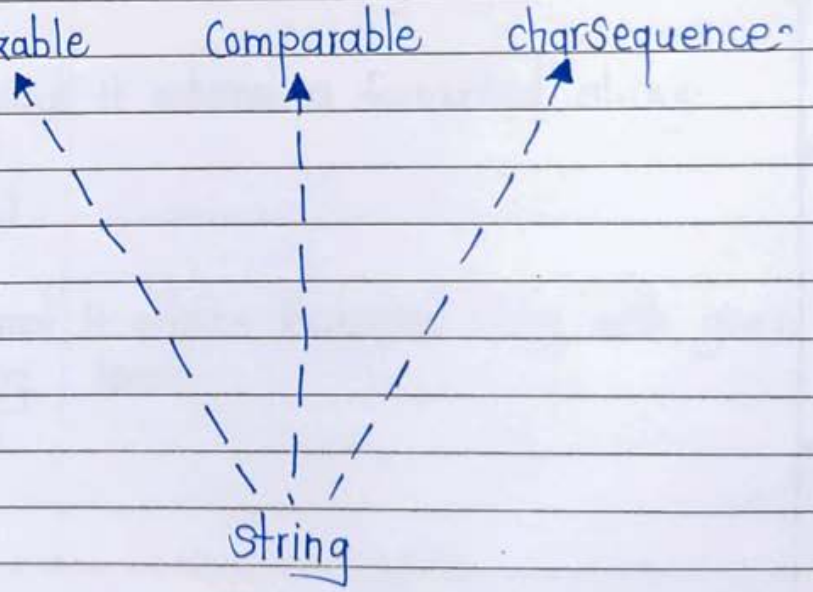
In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. for example:

```
char[] ch = {'j', 'a', 'v', 'a', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e'};
String s = new String(ch);
```

is same as :

```
String s = 'Javalanguage';
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc. The java.lang.string class implements Serializable, Comparable, charSequence interfaces.



Date :

String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

There are two ways to create string object:

1. By string literal
2. By new keyword

13.2 Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Method	Description
1) char charAt (int index)	It returns char value for the particular index
2) int length()	It returns string length.
3) static string format (string format, object... args)	It returns a formatted string.
4) static string format (Locale l, string format, object.. args)	It returns formatted string with given locale.

Date : _____

Method	Description
5) String substring (int beginIndex)	It returns substring for given begin index.
6) String substring (int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7) boolean contains (charSequence s)	It returns true or false after matching the sequence of char value.
8) static String join (charSequence delimiter, char-Sequence .... elements)	It returns a joined string.
9) String intern()	It returns an interned string.
10) int indexOF (int ch)	It returns the specified char value index.
11) StringtoLowerCase()	It returns a string in lowercase.
12) StringtoUpperCase()	It returns a string in uppercase.
13) String trim()	It removes beginning and ending spaces of this string.

Date : _____

13.3 Java StringBuffer class

Java stringBuffer class is used to create mutable (modifiable) string objects. The StringBuffer class in Java is the same as String class except it is mutable. i.e it can be changed. Java stringBuffer class is thread-safe. i.e multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Constructor	Description
stringBuffer()	It creates an empty string buffer with the initial capacity of 16.
stringBuffer(string str)	It creates a string buffer with the specified string.
stringBuffer(int capacity)	It creates an empty string buffer with the specified capacity as length.

13.4 Java stringBuilder class:

Java StringBuider class is used to create mutable (modifiable) string objects. The Java StringBuilder class is same as stringBuffer class except that it is non-synchronized. It is availdble since JDK 1.5

1) StringBuilder append() method

Date :

2) StringBuilder insert() method
3) StringBuilder replace() method
4) StringBuilder delete() method
5) StringBuilder reverse() method
6) StringBuilder capacity() method
7) StringBuilder ensureCapacity() method

Example:

```
class StringBuilderExample{
  public static void main(String args[]){
   StringBuilder sb = new StringBuilder("Hello");
   sb.append("Java");
   System.out.println(sb);
  }
}
```

13.5 Java toString():

If you want to represent any object as a string, toString() method comes into existence. The toString() method returns the string representation of the object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depending on your implementation.

Advantages of Java toString() method:

By overriding the toString() method of the object class, we can return values of the object, so we don't need

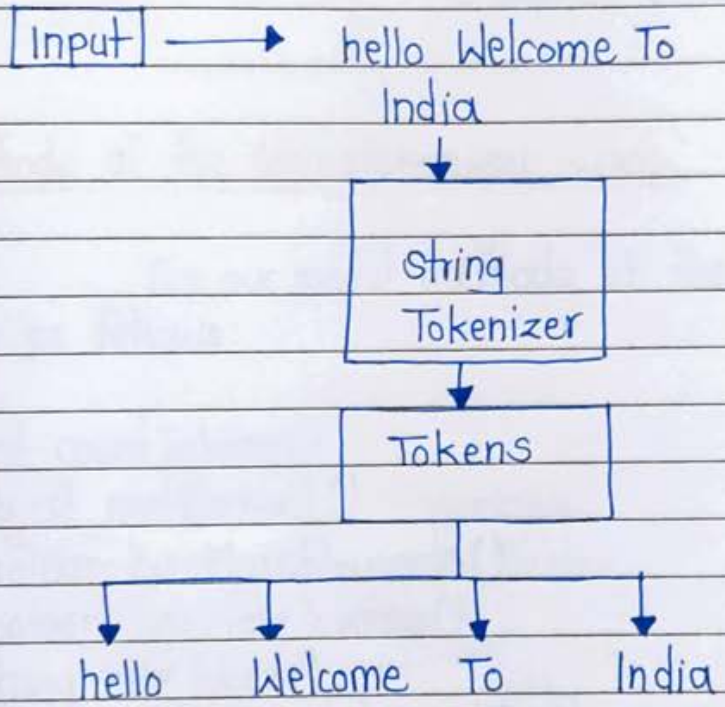
Date :

to write much code.

13.6 String Tokenizer in Java:

The java.util.StringTokenizer class allows you to break a string into tokens. It is simple way to break a string. It is a legacy class of Java. It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We. In the StringTokenizer class, the delimiters can be provided at the time of creation or more one by one to the tokens.

Example of String Tokenizer class in Java



Constructors of the StringTokenizer class:

There are 3 constructors defined in the String-Tokenizer class.

Date :

Constructor	Description
StringTokenizer(string str)	It creates stringTokenizer with specified string.
stringTokenizer(string str, string delim)	It creates stringTokenizer with specified string and delimiter.
StringTokenizer(string str, string delim, boolean returnvalue)	It creates stringTokenizer with specified string, delimiter and returnvalue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to seprate tokens.

Methods of the StringTokenizer Class :

The six useful methods of the StringTokenizer class are as follows:

1) int countTokens()
2) Object nextElement()
3) boolean hasMoreElements()
4) boolean hasMoreTokens()
5) String nextToken()
6) String nextToken(string delim)

Example of StringTokenizer class :

```
import java.util.StringTokenizer;
```

Date :

```
public class Simple{
  public static void main(String args[]){
    StringTokenizer st = new StringTokenizer("my name is
    yadnyesh"," ");
    while(st.hasMoreTokens()){
        System.out.println(st.nextToken());
    }
  }
}
```

Output:

```
my
name
is
yadnyesh
```

Date :

14. Java Exception Handling and Multi-Threading

14.1 Multithreading

Java provides built in support for multithreaded programming concept. A multithreaded program contains two or more parts that can run concurrently.
Each part of program is called a thread and each thread defines a seprate path of execution.
Thus, multithreading is form of multitasking.
It is also similar to dividing a task into subtasks and assigning them differently for execution independently and simultaneously.

There are two types of Multitasking:

1. Process based:

A process is a program that is executing. Thus, a process -based multitasking is the feature that allows the computer to run two or more programs concurrently.
In the process based multitasking, a program is a smallest unit of code that can be dispatched by the scheduler.

2. Thread based:

In a thread based multitasking environment, the thread is the smallest unit of dispatchable code.
That means a single program can perform two or more task

Date :

simultaneously.

The threads are light weight processes. They share the same address space and cooperatively share the same heavy weight process

LifeCycle of a Thread :

The thread can enter into many states during its lifetime
The five types of states are:

i) Newborn state

ii) Runnable

iii) Running state

iv) Blocked state

v) Dead state

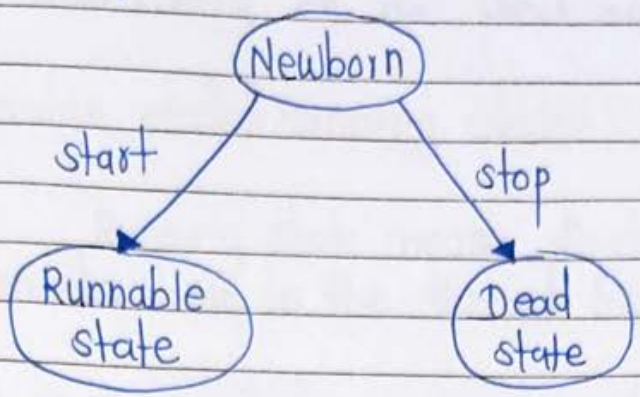
The thread can be in one state from there five states:

Newborn state:

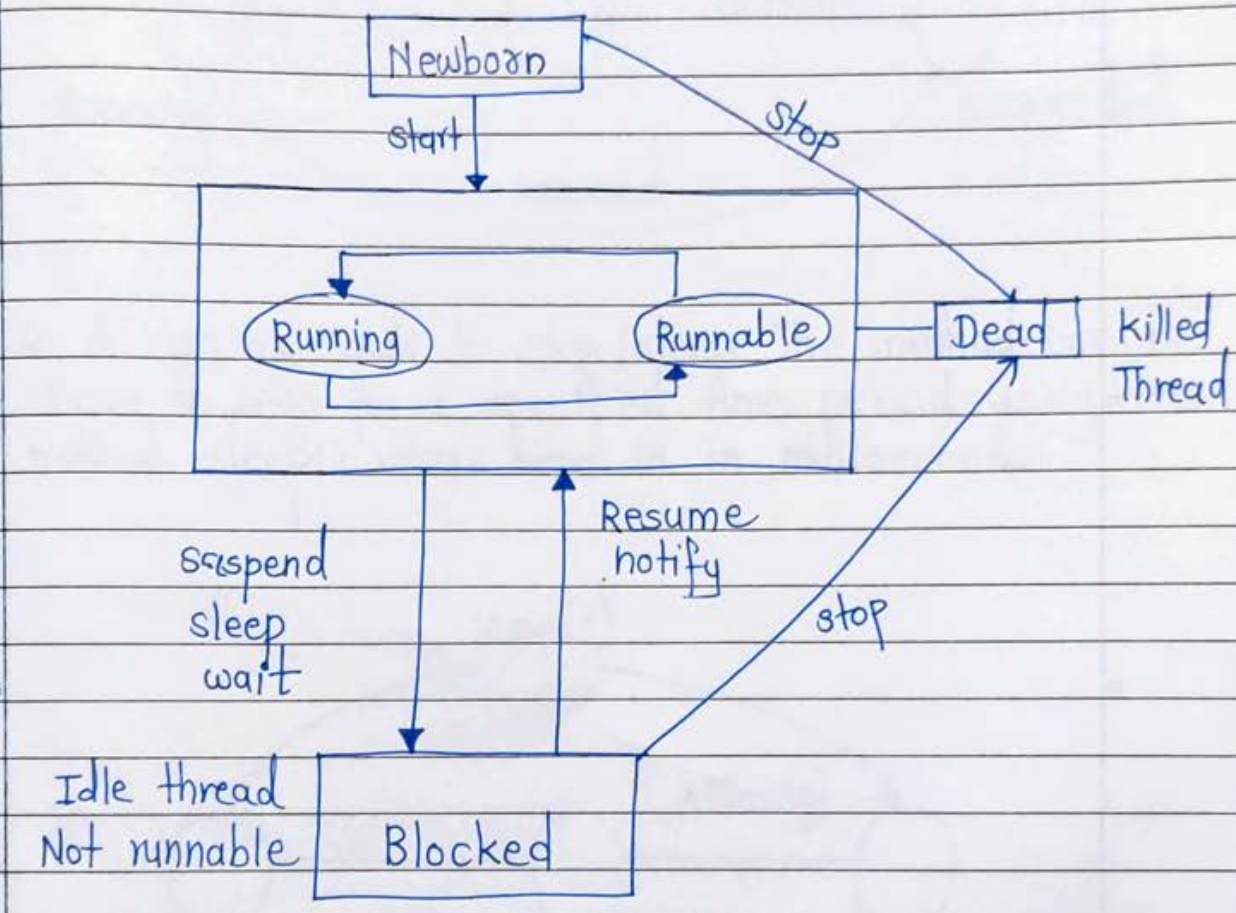
When a thread object is created, the thread is born and called as thread is in Newborn state.
When the thread is in newborn states if it can be either scheduled for running using start() method or kill it

Date : _____

using stop methods.



• Five states of thread •



Runnable state:

The runnable state of a thread means that

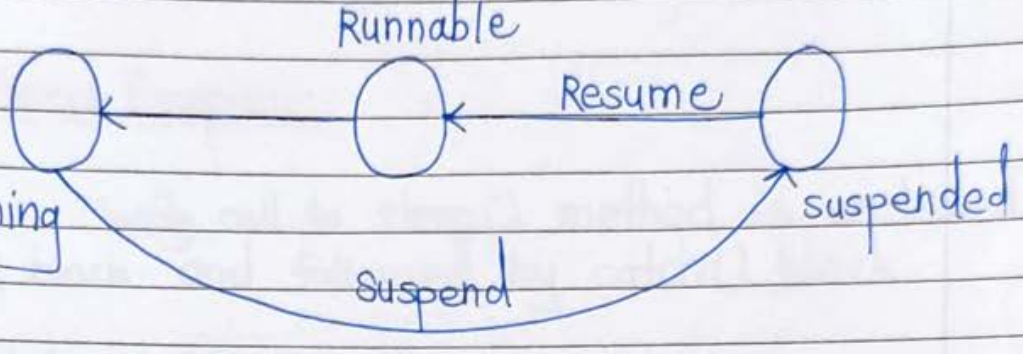
Date :

the thread is ready for execution and is waiting for the availability of the processor.

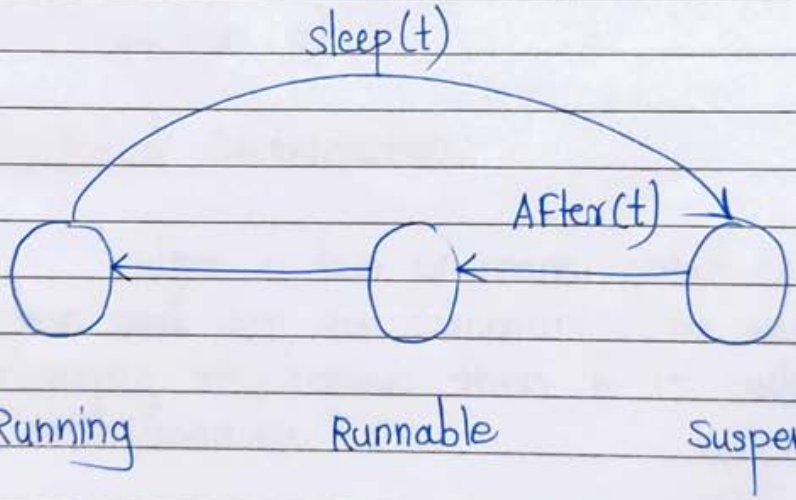
Runnable state/ Running state:

Running state means that the processor has given its time to the thread for its execution.

1. It is suspended using suspended method.



2. It can be made to stop/sleep. The method can be made to sleep for a specified time period using the method sleep() where time is in milliseconds.



Date :

Blocked state:

It is prevented from entering into the runnable state and subsequently the running state then it is said to be blocked state.

Dead state:

As every thread has a life cycle, a running threads ends its safe when it has completed executing its run() method.

14.3 Thread Exceptions:

Mostly call to sleep() method is enclosed in try block and followed by catch() block.

1. Catch (ThreadDeath e)

2. Catch (InterruptedException e)

3. Catch (IllegalArgumentException e)

4. Catch (Exception e)

14.4 InterThread Communication:

When a two or more thread exists in an application and they are communicating each other by exchanging information then it is called as inter thread communication.

Date :

Following three methods are used:

1) wait()

2) notify()

3) notifyAll()

14.5 Deadlock

When two threads have a circular dependency on a pair of synchronized objects, then there is a deadlock.

When two or more threads are waiting to gain control of resource.

Due to some reasons, the condition on which the waiting threads rely on to gain control does not happen.

The result is called as Deadlock.

The code below illustrates deadlock

```
Thread A
    synchronized Method2() {
            .... synchronized Method1(){
        }
    }
    Thread B
    {
        synchronized Method1(){
            synchronized Method2() {
}}
```

Date :

14.6 Errors and Exceptions :

Errors are wrongs or mistakes that can make a program go wrong.
Errors may be logical or may be typing mistakes.

An exception is an abnormal condition that arises in a code sequence at run time.

Types of Errors :

1) Compile time error

2) Run time error

Compile time errors

Java compilers detect all syntax error and display it. So these errors are known as compile time error

The common compile time errors :

1) missing semicolon

2) missing brackets in classes and methods

3) Misplelling of identifiers and keywords.

4) missing double quotes in strings

5) Bad reference to objects.

Date :

Runtime error

After successfully compiling a program, class file is created but may not run properly like programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow.

Exceptions:

An exception is a condition that is caused by a run-time error in a program.

Java keywords to manage exception:

1) try
2) catch
3) throw
4) throws
5) Finally

Exception handling Mechanism

The try keyword is used for the block of codes that causes an error condition and throws an exception. The catch block is added immediately after the key block.

```
try
{
   statement;
}
```

Date :

```
catch (Exception e)
{
   statement;
}
   :
finally {

}
```

19.7 Builtin Exception

The standard java.lang. package defined several built in exceptions.
There are two types:

1) Unchecked Runtime Exceptions
2) checked Runtime Exceptions

Following list is Unchecked Runtime exceptions:

Exception	Description
1) Arithmetic exception	Divide by zero
2) ArrayIndexOutofBounds Exception	Array index out of Bounds
3) class cast Exception	Invalid cast
4) Illegal Argument Exception	Illiegal argument invoked.

Following are checked Runtime. Exception

Exception	Description
1) classNotFound Exception	class not found
2) IllegalAccessExeception	Access to class denied
3) InstatiationException	Attempt to create an object of an abstract class
4) NosuchFieldException	A requested field does not exist.

14:8 Throwing our own exceptions

Our own exception can be thrown by using the keyword throw as Follow:

```
throw new Throwable_subclass;
```

Date :

15. Streams & Files

15.1 Introduction to streams:

A stream can be a contiguous group of data or a channel through which data travels from one point to another input stream receives data from source into a program and output stream sends data to a destination from a program. A stream provides a consistent, object-oriented interface between the program and input/output devices. In Java a stream is considered as a path for data movement between source and destination.

1. System.out (output stream)
It takes data from program and sends it to destination.
It is used to display output on screen
It is defined as—

```
public static final PrintStream out
```

2. System.in (Input stream)
It extracts or reacts data from source file() and sends it to a program.
It is used for reading of characters of data.
It is defined as

```
public static final Inputstream in
```

3. System.err (Error stream)
This is a error stream By default this is the user's console
It is defined as—

```
public static final Printstream err
```

Date :

15.2 Stream Classes and Methods

The io package or Java.io package supports Java's basic I/O System including file.

1. Byte Stream class
> It provides support for handling I/O operation on bytes.
> The byte streams have Input stream and output stream classes at the top of hierarchy.

2. character stream class
> It provides support for managing I/O operations on characters.

Byte stream classes:

Byte stream classes for creating and manipulating streams. Byte stream classes also used for redding and writing bytes in a file. The streams are undirectional.
It provides two types of Byte stream classes:

1) Input stream
2) Output stream

Input stream classes

Used to read 8-bit bytes include a super class known Input-stream and number of sub-classes for supporting various input related functions.
It is an abstract class, so we can't create instance of this class. The inputstream class declares methods for performing input function such as:

i) Reading bytes
ii) classing streams.

iii) making positions in streams.
iv) skipping ahead in a stream.
v) Finding the number of bytes in a stream.

Outstream/Outstream classes:

Outstream class is also an abstract class, Therefore we can't instatiate it.
The outputstream includes methods that are designed to perform following task:
i) Writing bytes
ii) closing bytes
iii) Flushing streams

15.3 File class:

The java.io package includes a class known as File class that provides support for creating files and directories. The class includes several constructors for instantiating the file object.
This class contains methods for supporting following:

1) Creating a file
2) Opening a file
3) closing a file
4) Deleting a file
5) Getting name of a file
6) Getting size of a file
7) Renaming a file
8) cheaking whether file is writable and readable

Let's see fileInputStream and fileOutputstream.

Date :

15.4 FileInputStream

This class is used to read input from a file in the form of a stream. This class provides methods for reading from a file. This class has three constructors:

1. FileInputStream(string Filename) throws FileNotFoundException
- creates an input stream that we can use to read bytes from a file.
- File name is full path of file.

2. FileInputStream(file File) throws FileNotFoundException
- creates an Input stream that we can use to read bytes from a file.

Example:

```
import java.io.*;
class FileInputDemo{
  public static void main(String args()) throws Exception
{
 int size;
 InputStream f = new FileInputStream(args[0]);
 System.out.println("Bytes available to read :" +(size =
   F.available());
 char str[] = new char[200];
 for(int i=0; i<size; i++)
 {
   str[i] = ((char).f.read());
   System.out.println(str[i]);
 }
```

Date :

```
System.out.println(" ");
f.close();
}
}
```

15.5 FileOutputstream :

This class is used to write output to a file stream
Three constructors:

1. FileOutputStream(string Filename) throws FileNotFoundException
- creates an outputstream that can be used to write bytes to a file.

2. FileOutputStream(string Filename, Boolean flag) throws File NotFoundException.
- Creates an outputstream that we can used to write bytes to a file.

3. FileOutputstream (Filename) throws FileNotFound Exception
- creates an outputstream that we can used to write bytes to a file.

Example:

```
import java.io.*;
class Demo{
    public static void main (string args[]) throws IO-
    Exception
{
    byte b[] = new byte [100];
```

```
System.out.println("Enter text");
int bytes = system.in.read(b);
FileOutputStream fo = new FileoutputStream("sample.txt");
fo.write(b, 0, bytes);
System.out.println("Data written");
fo.close();
}
}
```