DAC AUG 2024
MODULE- CORE JAVA

By-DEEPSHIKHA KURRE
PRN-240850120042

DAY 1

1.Part I- Introduction to Java

data type
variable
operators
control and conditional statement
array
method

2.Part II

oops
class and object
constructor
static and non static behavior
encapsulation
inheritance
polymorphism- method overloading,method overriding
abstraction-abstract class,interface
keyword-this,super,final
access specifier-private,public,default,protected

3.Part III

package
exception handling
collection- DS - java.util

array list
linked list
map

file handling- java.io
networking - java.net
Jdbc - java.sql
multithreading- java.lang

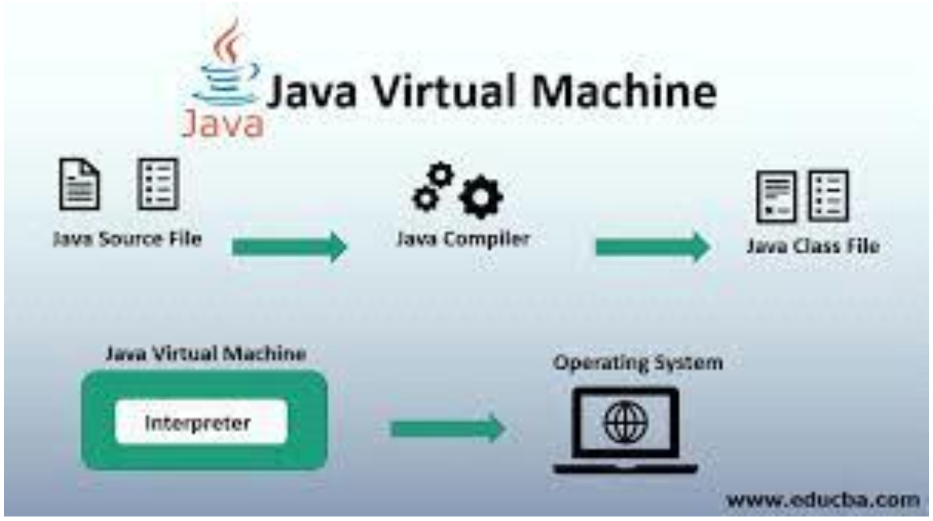
4.additional topics-
lambda
generics
Enum
stream API

1.Introduction to java- What is java- java is a high level ,object oriented programming language developed by james Gosling at sun microsystem in 1990.

2.Usage of java-
1) used to develop
 a) Desktop application
  application which runs on local machine
  swing
  java SE
 b) web application-1. application which runs on server. request,responce.
  2. Java EE- Spring,servlet,jsp,hibernate
  3. Enterprice application- EJB
 c) mobile application
  application which runs on electronic devices

 java ME

3.Component of java-



jdk- java development kit
- used to develop and run application
- javac,java,libraries
- jdk= jre + tool(java,javac)

jre- java runtime environment
- used to execute or run java application
- jre = jvm+set of libraries

jvm - java virtual machine, converting the .class file or bytecode to machine code

4.How java works- features of java-
1) simple
2) secure/robust
3) platform independence-write once run any where, develop in one os(windows) execute it on any os(Linux,Mac,windows).
4) portable
5) dynamic
6) multithreaded

Example- Hello.java

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

```
                System.out.println("HELLO WORLD ,THIS IS DEEPSHIKHA");
        }
}
```

5.Data type-

1) primitives=- Integer data-

```
                byte - 1
                  short = 2
                  int    =32
                 long  = 64
```

floating data=

```
                  float
                double
                char  - 2 byte
                boolean - 1bit
```

2) Non primitives/ reference data type=Array, String , Class

Example-
```
int[] mark = new int[5]
mark[0]= 20
mark[1]= 25
```

Example- Employee.java

```
class Employee
{
      int empid;
    String name;
    float sal;
}
 public class TestEmployee
{
    public static void main(String[] args)
    {
      Employee e1 = new Employee();

      e1.empid = 1001;
      e1.name = "nsnathan";
      e1.sal = 30000;
      System.out.println("First object");
      System.out.println(e1.empid);
      System.out.println(e1.name);
      System.out.println(e1.sal);

    System.out.println("second object");
      Employee e2 = new Employee();
      e2.empid = 1002;
      e2.name = "nathan";
      e2.sal = 40000;
      System.out.println(e2.empid);
      System.out.println(e2.name);
      System.out.println(e2.sal);
  }
```

6.Variable-

1) local variable- variable declared inside the method is called local variable.

Example- VariableExample.java

```
class VariableExample
{
public static void main(String[] args)
{
    int id;
    String name;
    String desig;
    float sal;
    char grade;
    boolean ispromoted;
   id =1001;
   name = "nsnathan";
   desig = "faculty";
   sal = 30000;
   grade = 'A';
   ispromoted = true;
   System.out.println("employee id " + id);
   System.out.println("name   " +name);
   System.out.println(desig);
   System.out.println(grade);
   System.out.println(sal);
   System.out.println(ispromoted);
}}
```

2) instance variable- declared inside the class,outside the method, available in all method which inside the class.

3) static variable
Example-

```
main()
{
  int age = 20;
  f1()
  printf(age)
}
f1()
{
  printf(age);
}
```

Example- Adding.java

```
public class Adding
{
       public static void main(String[] args)
       {
             int num1 = 50;
             int num2 = 20;
             int sum = num1 +num2;
             System.out.println("sum of "+ num1 + " + "+num2+" =" + sum);
             if(num1>num2) {
                  System.out.println(num1 + "is greater");
             }
             else
                  System.out.println(num2 + "is greater");
```

}

DAY 2-

1.Operator-

1.1 Arithmetic=   +,-,*,/ %

1.2 Logical- &&, ||

Example-

```
class Cdac
{
        int lab = 30;
        int ccee = 20;
        if(lab>16 && ccee>16)
                result = "PASS";
        else
                result = “FAIL”;
}
```

1.3 Relational-

>
<
>=
<=
!=
==

1.4 Assignment-   +=,   -+

Example-

```
        a = 10;
        a = a+1;
        a +=1;
```

1.5 Increment and decrement-   ++, - -

1.6 Ternary- Example-

```
        int a = 8;
        int b = 10;
        String big  =(a>b)? a:b
        if(a>b)
            big = a;
        else
            big = b;
```

Example- OperatorExample.java

```
public class OperatorExample
{
    public static void main(String[] args)
    {   int a = 12;
//      int b = 5;
        //arithmetic

//      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);

        //relational
//      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);

        //logical && ||
//      int lab = 10;
//           int ccee = 20;
//           String result=null;
//          if(lab>16 && ccee>16)
//          {                 result = "pass";
            }else
                result = "fail";

            System.out.println(lab>16 && ccee>16);
            System.out.println(lab>16 || ccee>16);
             System.out.println(result);
             //increment and decrement
             int c = 0;
             int d = c + c++ + ++c + c;
             System.out.println(d);
      }
}
```

1.7 Control statement

if

if else

if else if

1.8 method()- Example-

```
import java.util.Scanner;
public class MethodExampleFindMax
{
      public static int findmax(int c,int d)
      {
             int max=0;
             if(c>d)
                 max = c;
             else
                      max = d;
```

```
            return max;
        }
        public static void main(String[] args)
        {
            Scanner s =new Scanner(System.in);
            int a;
            int b;
            a = s.nextInt();
            b = s.nextInt();
            int m=findmax(a,b);
            System.out.println("max "+ m);
        }
}
```

1.8.1. static method- if the method is static, we can call the method by the method name or can call the method by classname.method name.

Example- methodExample.java

```
class methodExample
{
        //user defined static method
        static void findmax()
        {
        syso("inside the findmax”);
        }
        //inbuilt static method
public static void main(String[] args)
      {
            syso("method example");
            methodExample.findmax();
        }
}
```

1.8.2. Non static method /instance method- To call non static method you require the object reference of the class.

Example-

```
class methodExample
{
      void findmax()
      {}
      public static void main()
      {
       methodExample m = new methodExample();
      m.findmax();
      }
}
```

1.9 How to read data from the user-

Example- ReadDataFromUser.java

```
import java.util.Scanner;
public class ReadDataFromUser
{
      public static char findGrade(int p )
      {
               char grade;
             if(p>=90)
                 grade = 'A';
             else if(p>=80 && p<90)
                 grade = 'B';
             else if(p>=60 && p<80)
                 grade = 'C';
             else
                 grade = 'D';
             return grade;
      }
public static void main(String[] args)
{
```

```
            Scanner s =new Scanner(System.in);
            System.out.println("enter the percentage");
            int per = s.nextInt();

             char g=findGrade(per);
            System.out.println("grade = "+g);
            System.out.println("end of main");
//how to read data from the user

      Scanner s =new Scanner(System.in);
       System.out.println("enter the empno"); int empno =s.nextInt();
       System.out.println("enter the sal"); float empsal = s.nextFloat();
       System.out.println("enter the name"); String name = s.next();
      System.out.println("empno = "+ empno); System.out.println("empsal =
" + empsal); System.out.println("empname = " + name );
      }
}
```

DAY 3-

1.static method- using method name or class name.method name
no need of object reference.
Example-

```
public class StaticMethodExample
{
      public static void findOddEven()
      {
            System.out.println("inside the oddeven method");
            int num = 30;
            if(num%2==0)
                  System.out.println("even");
            else
                  System.out.println("odd");
}
```

```
    public static void main(String[] args)
    {
        System.out.println("method example");
        findOddEven();
    }
}
```
2. Non static method- need object reference to make a call to the method.

Example-
```
public class NonstaticMethodExample
{
      public void findmax()
      {
          System.out.println("inside findmax");
      }
    public static void main(String[] args)
    {
        NonstaticMethodExample n = new NonstaticMethodExample();
        System.out.println("inside main");
        n.findmax();
    }
}
```
3. conditional statement-

3.1 switch case-   1.add

    2.delete
    3.update
    4.retrieve
    5.exit

Example- (A) - switch case-
```
user input ch= 3
do
{
switch(ch)
{
  case ch ==1:
  break;
  case ch == 2: syso(data deleted)

  default: syso(invaili input)
}
```

(B) while=
while(i!=5)

3.2 control and looping statements- break, continue, return
Example-(A)-

```
int i = 0;
while(i<10)
{
 i++
}
```

(B) for loop-

```
for(int i =0 ;i<10;i++)
{
//logic part
}
```

(C) do while loop-

```
i =0
do
{   i= i+1;
}while(i<5)
```

(D) ControlExampleFindOddEven.java

```
public class ControlExampleFindOddEven
{	// find  the given data is odd or even
	public static void main(String[] args)
	{	int num = 31;
		if(num%2 ==0)
			System.out.println("the given number is even");
		else
			System.out.println("the given number is odd");
	}
```

}
4.Type casting-
4.1) implicit casting(widening)

```
    int i = 20;
    float f = i;
```

4.2) explicit casting(narrowing)

```
float f = 40.5;
int i = (v int)f;
```

Example-

```
class jdbcmain
{
public static void main(String[] args)
{
    switch(ch)
   {
    case 1: jdbcManagement.create();
    delete();
    update();
    display();
}
class jdbcManagement
{
       public static void create()
       {       //logic part
       }
       public static void delete()
       {
       }
}
}
```

5. Reference data type-
5.1 Array- How to declare?, How to initialize?, How to retrieve?
int array, float array, char array, String array.

5.2 How to create array-   int[] mark = new int[5];

5.3 How to initialize-
```
mark[0]= 10;
mark[1]=20;
mark[2]=40;
mark[3]=50;
mark[4]=30;
```

Example- TO TAKE USER INPUT-

(A)
```
for(i=0;i<5;i++)
{
   mark[i]= sc.nextINt();
}
```
(B)`int[] mark= {10,20,30,40,50}`

5.4 How to retrieve-
```
for(int i =0;i<5;i++)
{
  syso(mark[i);
}
for(int ele    :mark)
{
   syso(ele);
}
```

DAY 4-

1.Classes and Objects-

1.1 Object- object is a real word entity, object has properties(data) and behavior (method). object is a instance of the class. object takes memory.

1.2 Class-
1) class is a template or blueprint
2) class is used to create object
3) it is a representation of object

Example- Student.java

```
class Student
{
  int sid;
   String name;
   Int age;
   Int mark;
   String cname;
}
```

Example 2- Mobilephone.java

```
class Mobilephone
{
       Int price;
       String model;
       String color;
}
```

Example3 - Cdacblr.java

```
class Cdacblr
{
       int course_id;
       String coursename;
       int noofstudent;
       int noofmodule;
       float fees;
public static void main()
{
       Cdacblr c1 = new Cdacblr();
       c1.course_Id = 1001;
       c1.coursename = "DAC";
       c1.no_of_student =200;
       c1.modules = 8;
}
```

1.3 Access specifier- A.public- within the class, within the package, outside the package.

B.private- within the class.

C.default- within the class, within the package.
D.protected

1st TEST OF PART 1

What is the default value of an int variable in Java? a) 0
b) 0.0
c) null
d) NaN

Which of the following is a valid declaration of a float variable in Java? a) float f = 1.0;
b) float f = 1.0f;
c) float f = (float) 1.0;
d) Both b and c

What will be the output of the following code snippet?

java
Copy code
```
int x = 10;
int y = 5;
System.out.println(x % y);
```
a) 0
b) 5
c) 2
d) 10

Which control statement is used to exit a loop in Java? a) continue
b) break
c) return
d) exit

What is the purpose of the default keyword in a switch statement? a) To define a default case

b) To specify the default value of a variable
c) To terminate the switch statement
d) To override the default method in an interface

How can you define a method in Java that does not return any value? a) void methodName()
b) methodName() {}
c) methodName() : void
d) void: methodName()

What is the correct syntax to declare a one-dimensional array in Java? a) int[] arr;
b) int arr[];
c) int arr;
d) Both a and b

Which loop is guaranteed to execute at least once? a) for loop
b) while loop
c) do-while loop
d) foreach loop

What is the scope of a local variable in Java? a) Entire class
b) Entire method
c) Entire program
d) The block in which it is defined

Which of the following is NOT a valid data type in Java? a) int
b) float
c) boolean
d) string

What will be the result of the following expression in Java?

```
int x = 5;
x += 10;
System.out.println(x);
```

a) 15
b) 5
c) 10
d) 20

What is the default value of a boolean variable in Java? a) true
b) false
c) 0
d) null

Which of the following statements is used to skip the current iteration of a loop in Java? a) break
b) continue
c) exit
d) return

How do you declare an array with 10 elements in Java? a) int[] arr = new int[10];
b) int arr[] = new int[10];
c) int arr[10];
d) Both a and b

What is the output of the following code snippet?

java
Copy code
```
int[] arr = {1, 2, 3, 4, 5};
System.out.println(arr[2]);
```
a) 1
b) 2
c) 3
d) 4

What is the range of values for a byte data type in Java? a) -128 to 127
b) -32768 to 32767

c) -2147483648 to 2147483647
d) 0 to 255

Which method signature is correct for defining a method that accepts two integers and returns their sum? a) int add(int a, int b)
b) void add(int a, int b)
c) int add(int a, int)
d) int add(int, int)

In Java, which keyword is used to create an instance of a class? a) new
b) create
c) class
d) instantiate

What will be the output of the following code snippet?int i = 10;
```
if (i > 5) {
    System.out.println("Greater");
} else {
    System.out.println("Lesser");
}
```
a) Greater
b) Lesser
c) Nothing
d) Compilation error

Which loop will iterate over the elements of an array? a) for loop
b) while loop
c) do-while loop
d) foreach loop

What will be the output of the following code snippet?
```
public class Main {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        System.out.println(numbers[4]);
    }
}
```
a) 1
b) 2
c) 4
d) 5

Consider the following code. What will be the result?
```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        if (x > y) {
            System.out.println("x is greater");
        } else {
            System.out.println("y is greater");
        }
    }
}
```
a) x is greater
b) y is greater
c) Nothing
d) Compilation error

What will be the output of this code snippet?
```
public class Main {
    public static void main(String[] args) {
        int num = 10;
        System.out.println(num / 3);
        System.out.println(num % 3);
    }
}
```
a) 3 and 1
b) 3 and 0

c) 3 and 3
d) 3 and 2

What will the following code snippet print?

```
public class Main {
    public static void main(String[] args) {
        int[] array = new int[5];
        array[0] = 10;
        array[1] = 20;
        array[2] = 30;
        array[3] = 40;
        array[4] = 50;
        System.out.println(array[1] + array[3]);
    }
}
```

a) 30
b) 50
c) 60
d) 70

What will be the output of the following code snippet?

```
public class Main {
    public static void main(String[] args) {
        int i = 0;
        while (i < 3) {
            System.out.println("Value: " + i);
            i++;
        }
    }
}
```

a) Value: 0
Value: 1
Value: 2

b) Value: 1
Value: 2
Value: 3
c) Value: 0
d) Nothing

DAY 5-

Polymorphism-Polymorphism is one of the key principles of Object-Oriented Programming (OOP) in Java. It allows a single action to behave differently based on the object on which it is being performed. In Java, polymorphism is achieved through method overloading and method overriding.

Types of Polymorphism-
1.Compile-Time Polymorphism (Static Binding)
2. Run-Time Polymorphism (Dynamic Binding)

1. Compile-Time Polymorphism (Method Overloading)
Method overloading occurs when two or more methods in the same class have the same name but different parameters (type, number, or both).

Example of Method Overloading

```
class Calculator {
    // Method with two integer parameters
    public int add(int a, int b) {
        return a + b;
    }
    // Method with three integer parameters
    public int add(int a, int b, int c) {
        return a + b + c;
    }
    // Method with two double parameters
    public double add(double a, double b) {
        return a + b;
    }
}
public class OverloadingExample {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Addition of two integers: " + calc.add(10, 20));
        System.out.println("Addition of three integers: " + calc.add(10, 20,
30));
        System.out.println("Addition of two doubles: " + calc.add(10.5, 20.5));
    }
}
```

2. Run-Time Polymorphism (Method Overriding)-

Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its parent class.

Rules for Method Overriding:

The method in the child class must have the same name, return type, and parameters as the method in the parent class.

The @Override annotation is optional but recommended.

Access modifiers in the child class must not reduce the visibility of the method.

Example of Method Overriding-

```
class Animal {
    public void sound() {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("Dog barks");
    }
}
class Cat extends Animal {
    @Override
    public void sound() {
        System.out.println("Cat meows");
    }
}
public class OverridingExample {
    public static void main(String[] args) {
        Animal myAnimal;

        myAnimal = new Dog();  // Dog-specific behavior
        myAnimal.sound();

        myAnimal = new Cat();  // Cat-specific behavior
        myAnimal.sound();
    }
}
```

Polymorphism with Interfaces-

Polymorphism can also be achieved through interfaces, where multiple classes implement the same interface, and the behavior is decided at runtime.

Example-

```
interface Shape {
    void draw();
}
class Circle implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a Circle");
    }
}
class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a Rectangle");
    }
}
```

```
public class InterfacePolymorphismExample {
    public static void main(String[] args) {
        Shape shape;
        shape = new Circle();
        shape.draw();
        shape = new Rectangle();
        shape.draw();
    }
}
```

Advantages of Polymorphism

1.Code Reusability.

2.Flexibility.

DAY 6-

1.1.Encapsulation- Process of bundling data and the method which operates on the data together is called encapsulation.

Example 1 - Student.java

```
import java.util.Scanner;
public class Student
{
         private int id;
         private String name;
         public int getId() {
                  return id;
         }
         public String getName() {
                  return name;
         }
         public void setId(int id) {
                  this.id = id;
         }
         public void setName(String name) {
                  this.name = name;
         }
         public void display()
```

```
    {
        System.out.println(id+name);
    }
    public void readstudent()
    {
        Scanner s = new Scanner(System.in);
        id = s.nextInt();
        name = s.next();
    }
}
public class TestSTudent
{
    public static void main(String[] args)
    {
        Student s = new Student();
        s.setId(1001);
        s.setName("nathan");
        System.out.println(s.getId() +" "+ s.getName()) ;
    }
}
```

1.2 Inheritance- Inheritance a process of acquiring the properties(data) and the behavior(method) from one class to the other class.creating new class from the existing class so that the new class get the properties and the behavior of existing class.creating sub class object accessing the
sub class features(properties and behavior) and super class features(properties and behavior) this is called inheritance.

1.3 Advantage- time, code reusability ,achieving runtime polymorphism.
Syntax-

```
class A
{
}
class B extend A
```

```
{
}
```

Example-

```
public class A
{
    int a;
    void printA()
    {
         System.out.println("A class method "+a);
    }
}
 class B extends A
{
    int b;
    void printB()
    {
         System.out.println("B class method "+b);
    }
}
 class C extends B
 {
     int c;
     void printC()
     {
         System.out.println("B class method "+c);
     }
 }
 class Testinheritance
 {
          public static void main(String[] args)
          {
                   B b = new B();
                   b.b = 20;
                   b.printB();
                   b.a = 10;
```

```
                b.printA();

                C c = new C();
                c.a = 10;
                c.b = 20;
                c.c = 30;

                c.printA();
                c.printB();
                c.printC();
        }
 }
```

1.4 Types of inheritance-

1.4.1 is a relation

1.4.2 has a relation

1.5 Varieties of inheritance-

Example-

```
class Bank
{
        int aid;
        String aname;
        float balance;
        public Bank(int aid, String aname, float balance)
        {
                this.aid = aid;
                this.aname = aname;
                this.balance = balance;
        }
        void dispBankDetails()
        {
                System.out.print(aid+aname+balance+ " ");
        }
        void withdraw(int amt)
        {
                balance =balance -amt;
                System.out.println(balance);
        }
}
class Savings extends Bank
{      float ri;

          Savings(int aid,String aname,float balance,float ri)
          {
                      super(aid,aname,balance);
                      this.ri = ri;
          }
         void printsavingDetails()
         {
                 super.dispBankDetails();
                 System.out.println(ri);
         }
          void findInterest()
          {
                   float interest = balance*ri;
                   System.out.println(interest);
          }
}
class Current extends Bank
{
        int overduefees;
        public Current(int aid, String aname, float balance,int overduefees)
        {
                super(aid, aname, balance);
                this.overduefees = overduefees;
        }
        void printCurrentDetails()
        {
                super.dispBankDetails();
                System.out.println(overduefees);
        }
          void findoverdraft(int amt)
          {
                if(balance<amt)
                        balance= balance-500;
                System.out.println(balance);
          }
}
public class InheritanceMain
{
        public static void main(String[] args)
        {
                Savings s = new Savings(1001,"nsnathan",40000,0.02f);
                s.printsavingDetails();
                s.withdraw(2000);
                s.findInterest();
                System.out.println("-------------------------");
                Current c = new Current(2001,"shan",20000,500);
                c.printCurrentDetails();
                c.findoverdraft(3000);
        }
}
```

DAY 7-

1. Constructor behavior during inheritance-

Example-

```
class A
{
  int a;
  A(int a)
  {   this.a = a; }
  }
  void dispA(){   }
```

```
class B extends A
{
   int b;
  B(int a,int b )
  { super(a)
this .b = b}
}
main()
{
  B b = new B(10,20)
  b.dispA();
}
```

2. Polymorphism- poly = many, morphism = form(method). More than one method can have same name with different functionality.

2.1 Method overloading- in single class, single class has two different method with same name. method name can be same at least the no of parameters and the type of parameters should be different.

Example-

```
class soring
{
static void sort(int[] n)
{ - for sorting integer array
}
static void sort(float[] f)
{ - for sorting float array
}
static void sort(String  s)
{ - for
sorting string array
}
}
main()
{
  String[]  = {'c++',"java",'python'}
  sortSting(subject)
}
class A
{
   void findtax(){}
   void dispEmp();
}
class B extends A
{
  void findtax(){}
}
```

3. Method overriding- During inheritance, method name can be same the no of parameters ,type of parameters also be same.

Example1 -

```
class A
        {
        void print(String name){ }
        }
class B extends A
        {
        void print(String name);
                {       super.print();
                }
        }
public class  C
{
        public static void main(String[] args)
        B b = new B();
        b.print();
}}
```

Example 2-

```
Bank
{
void findinterest()
{
    interest = balance*0.02f;
}
}
class sbi extends Bank
{
void findinterest()
{
    interest = balance*0.03f;
}
}
class icici extends Bank
{
void findinterest()
{
    interest = balance*0.01f;
}
}
public static void main()
{
  sbi s = new sbi();
  s.findInterest()
  Icici i = new Icici();
  i.Interest():
}
```

2nd TEST OF PART 2

1. What is a class in Java?
a) A blueprint for creating objects
b) A data type
c) A method
d) An instance of an object

2. Which of the following is not an access modifier in Java?
a) public

b) protected
c) internal
d) private

3. What does a constructor do in Java?
a) Initializes an object
b) Defines a method
c) Deletes an object
d) Copies an object

4. Can a constructor be private in Java?
a) Yes
b) No

5. What is the purpose of the static keyword in Java?
a) To create methods and variables that belong to the class rather than an instance of the class
b) To make methods non-overridable
c) To prevent inheritance
d) To define abstract methods

6. Which of the following is true about static methods?
a) They can access both static and non-static variables
b) They cannot access non-static variables directly
c) They cannot be inherited
d) They can only be used in abstract classes

7. What is encapsulation in Java?
a) Hiding data and methods inside an object
b) Inheritance of classes
c) Using static methods
d) Overloading constructors

8. Which of the following is a correct way to achieve encapsulation?
a) Making fields private and providing public getter and setter methods

b) Using static methods
c) Using final keyword
d) Using abstract classes

9. What is the main concept of inheritance in Java?
a) Reusability of code
b) Hiding implementation
c) Providing static methods
d) Overloading methods

10. In Java, multiple inheritance is:
a) Supported through classes
b) Supported through interfaces
c) Not supported at all
d) Supported through abstract classes

11. What keyword is used to inherit a class in Java?
a) inherit
b) extend
c) implement
d) super

12. Polymorphism in Java can be achieved through:
a) Method overloading
b) Method overriding
c) Both a and b
d) Constructors only

13. Which of the following is a correct statement about method overloading?
a) It allows multiple methods with the same name but different signatures
b) It only works with static methods
c) It can only be done with methods having the same return type
d) It allows methods with the same name and signature

14. Method overriding requires:
a) Methods to have the same name and signature
b) Methods to have different return types
c) Methods to be static
d) Constructors to be overloaded

15. What is the use of the super keyword in inheritance?
a) To call a parent class constructor or method
b) To call a static method
c) To create a new object
d) To override methods

16. Can static methods be overridden in Java?
a) Yes
b) No

17. Which of the following is NOT true about abstract classes?
a) They can have both abstract and non-abstract methods
b) They cannot be instantiated
c) They must implement all methods from an interface
d) They can have constructors

18. Which of the following is an example of dynamic polymorphism?
a) Method overloading
b) Method overriding
c) Constructor overloading
d) Interface implementation

19. What is the output when you try to access a non-static variable from a static method?
a) Compilation error
b) NullPointerException
c) Default value of the variable
d) Runtime exception

20. Which of the following keywords prevents a method from being overridden?
a) final
b) static
c) abstract
d this

1. What will be the output of the following code?
```
class Test {
    int x = 5;

    public static void main(String[] args) {
        Test obj1 = new Test();
        Test obj2 = new Test();
        obj1.x = 10;
        System.out.println(obj2.x);
    }
}
```

a) 5
b) 10
c) Compilation error
d) Runtime error

2. What is the output of the following code snippet?
```
class Test {
    static int count = 0;

    Test() {
        count++;
    }
    public static void main(String[] args) {
        Test t1 = new Test();
        Test t2 = new Test();
        System.out.println(Test.count);
    }
}
```
a) 0
b) 1
c) 2
d) Compilation error

3. Which of the following is true about the code snippet below?
```
class Parent {
    public void display() {
        System.out.println("Parent class");
    }
}

class Child extends Parent {
    public void display() {
        System.out.println("Child class");
    }
}

public class Test {
    public static void main(String[] args) {
        Parent obj = new Child();
        obj.display();
    }
}
```
a) It will print "Parent class"
b) It will print "Child class"
c) Compilation error
d) Runtime error

4. What will be the output of the following code?
```
class Test {
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        System.out.println(x + y + " is the sum.");
    }
}
```

a) 30 is the sum.
b) 1020 is the sum.
c) Compilation error
d) Runtime error

5. What will happen when you compile and run the following code?

```
class Test {
    private int value;

    public void setValue(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
    public static void main(String[] args) {
        Test obj = new Test();
        obj.setValue(100);
        System.out.println(obj.getValue());
    }
}
```

a) Prints 100
b) Prints 0
c) Compilation error
d) Runtime error

6. What is the result of the following program?

```
class Test {
    static void display() {
        System.out.println("Static Method");
    }
    public static void main(String[] args) {
        Test obj = null;
        obj.display();
    }
}
```
a) Prints "Static Method"
b) Compilation error
c) NullPointerException at runtime
d) Runtime error

7. What is the output of the following code snippet?
```
class Test {
    public static void main(String[] args) {
        int a = 5, b = 10;
        System.out.println(a > b ? "A is greater" : "B is greater");
    }
}
```
a) A is greater
b) B is greater
c) Compilation error
d) Runtime error

8. Which of the following is correct about this code?
```
class Parent {
    private void show() {
        System.out.println("Parent method");
    }
}
class Child extends Parent {
    public void show() {
        System.out.println("Child method");
    }
}
public class Test {
    public static void main(String[] args) {
        Parent obj = new Child();
        obj.show();
    }
}
```
a) It will print "Parent method"
b) It will print "Child method"
c) Compilation error
d) Runtime error

9. What will be the output of this code?
```
class Test {
    public static void main(String[] args) {
        int x = 5;
        increment(x);
        System.out.println(x);
    }
    public static void increment(int x) {
        x = x + 1;
    }
}
```
a) 5
b) 6
c) Compilation error
d) Runtime error

10. What will the following code print?
```
class A {
    A() {
        System.out.println("Class A Constructor");
    }
}
class B extends A {
    B() {
        System.out.println("Class B Constructor");
    }
}
public class Test {
    public static void main(String[] args) {
        B obj = new B();
    }
}
```

a) Class A Constructor
b) Class B Constructor
c) Both "Class A Constructor" and "Class B Constructor"
d) Compilation error

DAY8-

1. Inheritance types-
   1.1 is a relation
   1.2 has a relation

Example-

```
public class Employee
{
    int empno;
    String name;
    float sal;
    Address add;
public Employee(int empno, String name, float sal, Address add)
         {
                  this.empno = empno;
                  this.name = name;
                  this.sal = sal;
                  this.add = add;
         }
void dispEmp()
         {
         System.out.print(empno+name+sal+add.city+add.state+add.pin);
                  //add.dispAddress();
```

```
        }
public static void main(String[] args)
        {
                Address add = new Address("Bangalore","Karnataka",560038);
                Employee e = new Employee(1001,"shan",30000,add);
                e.dispEmp();
        }
}
public class Address
{
        String city;
    String state;
    int pin;
        public Address(String city, String state, int pin) {

                this.city = city;
                this.state = state;
                this.pin = pin;
        }
        void dispAddress()
        {
                System.out.println(city+state+pin);
        {
}
```

2. Abstraction-

abstraction a process of hiding the implementation details
showing only the functionality to the user.

2.1 abstract class

2.2 Interface

Format-

```
abstract class emp
{
   abstract void diplay();
   void print()
```

```
  {}
}
contract extends emp
{
  void display()
  {}
}
```

Example-

```
public class NetBankingPayment
{
        void payment(float amount)
        {
                System.out.println("Processing payment  "+ amount +" via net
Baning");
        }
}
class SBI extends NetBankingPayment
{
        void payment(float amount)
        {
                System.out.println("Processing  payment   "+  amount  +  "  via
SBI");
        }
}
class ICICI extends NetBankingPayment
{
        void payment(float amount)
        {
                System.out.println("Processing  payment   "+  amount  +  "  via
ICICI");
        }
}
class HDFC extends NetBankingPayment
{
        void payment(float amount)
        {
                System.out.println("Processing  payment    "+  amount  +  "  via 
HDFC");
        }
}
class testRuntimePolymorphism
{
        public static NetBankingPayment createbankobject(String bname)
        {
                if(bname.equals("SBI"))
                {
                   return   new SBI();
                }
        else if(bname.equals("ICICI"))
        {
                return new ICICI();
        }
        else if(bname.equals("HDFC"))
        {
                return  new HDFC();
        }
        else
                return null;
        }
        public static void main(String[] args)
        {
                NetBankingPayment n;
                Scanner s  = new Scanner(System.in);
                System.out.println("enter the bank name to be processed");
                String bname=s.next();
                 n =createbankobject(bname);
                 n.payment(1000);
        }
}
```

3.Three stakeholders are there -WHO DEFINE RULES(METHOD), WHO IMPLEMENTS ,USER

4. Abstract class-

    4.1)class has abstract keyword

    4.2) class contain abstract method

Format-

```
Class Classname
{
  properties
  constructor
  method
}
```

DAY9-

Exception Handling-

Exception handling in Java is a mechanism to handle runtime errors and maintain the normal flow of the application. Exceptions are unexpected events that disrupt the normal execution of a program.

Throwable Hierarchy-

- Error: Represents serious problems that applications should not try to handle (e.g., OutOfMemoryError).
- Exception: Represents conditions that applications might want to catch.
- Checked Exceptions: Exceptions that must be declared in a method's throws clause if not handled within the method (e.g., IOException, SQLException).
- Unchecked Exceptions: Subclasses of RuntimeException. These don't need to be declared or caught (e.g., NullPointerException, ArithmeticException).
- try: Defines a block of code that might throw exceptions.
- catch: Catches and handles exceptions thrown by the try block.
- finally: Defines a block of code that will always execute, regardless of exceptions.
- throw: Used to explicitly throw an exception.
- throws: Declares exceptions that a method might throw.

Examples

1. Handling Checked Exception-

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class CheckedExceptionExample {
    public static void main(String[] args) {
        try {
            File file = new File("nonexistent.txt");
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } finally {
            System.out.println("Execution complete.");
        }
    }
}
```

2. Handling Unchecked Exception-

```
public class UncheckedExceptionExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // Causes ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero: " + e.getMessage());
        } finally {
            System.out.println("Execution complete.");
        }
    }
}
```

3. Using throw

```
public class ThrowExample {
    public static void main(String[] args) {
        try {
            checkAge(15);
        } catch (IllegalArgumentException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }

    public static void checkAge(int age) {
        if (age < 18) {
            throw new IllegalArgumentException("Age must be 18 or above.");
        }
    }
}
```

4. Declaring Exceptions with throws

```
import java.io.IOException;
public class ThrowsExample {
    public static void main(String[] args) {
        try {
            readFile();
        } catch (IOException e) {
            System.out.println("Caught IOException: " + e.getMessage());
        }
    }
    public static void readFile() throws IOException {
        throw new IOException("File reading error");
    }
}
```

Custom Exceptions-

You can create your own exception classes by extending Exception or RuntimeException.

```
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
public class CustomExceptionExample {
    public static void main(String[] args) {
        try {
            validateInput(-5);
        } catch (CustomException e) {
            System.out.println("Caught CustomException: " + e.getMessage());
        }
    }
    public static void validateInput(int number) throws CustomException {
        if (number < 0) {
            throw new CustomException("Input cannot be negative.");
        }
    }
}
```

Advantages of Exception Handling-

- Separates error-handling logic from regular code.
- Ensures program stability by managing runtime errors gracefully. Provides meaningful error messages for debugging.

DAY10-

File Handling-

File handling in Java involves creating, reading, writing, and deleting files. Java provides robust and versatile APIs to perform these operations through the java.io and java.nio.file packages.

Classes-

- File: Represents file and directory pathnames.
- FileReader: For reading character files.
- FileWriter: For writing to character files.
- BufferedReader: For efficient reading of text files.
- BufferedWriter: For efficient writing to text files.
- FileInputStream: For reading binary data.
- FileOutputStream: For writing binary data.

    - Files: Utility class in java.nio.file for advanced file operations.

DAY11-

Socket programming-

Socket programming in Java enables communication between two systems over a network. It is used to establish a connection between a client and a server, allowing data to be exchanged in real-time. The java.net package provides classes and interfaces for socket-based communication.

- Socket-A socket is an endpoint for communication between two machines.It uses an IP address and port number to establish a connection.
- ServerSocket-Used by the server to listen for incoming client requests.Creates a socket for each client connection.

Common protocols-

- TCP (Transmission Control Protocol): Reliable, connection-oriented protocol.
- UDP (User Datagram Protocol): Unreliable, connectionless protocol.

Key Methods-

- ServerSocket:
  accept(): Waits for a client to connect.
  close(): Closes the server socket.
- Socket:
  getInputStream(): Retrieves the input stream for reading data.
  getOutputStream(): Retrieves the output stream for writing data.

DAY12-

JDBC Connection-

JDBC (Java Database Connectivity) is an API provided by Java to interact with relational databases. It enables Java applications to execute SQL queries and manage database operations such as retrieving, inserting, updating, and deleting data.

Components of JDBC

- Driver Manager-Manages a list of database drivers.

- Connection-Represents a connection session with the database.
- Statement-Used to execute SQL queries.
- Statement- For executing simple SQL queries.
- PreparedStatement-For precompiled parameterized SQL queries.
- CallableStatement- For executing stored procedures.
- ResultSet-Represents the result of a SQL query.

Steps to Establish a JDBC Connection-

1. Import the JDBC Package:
2. Import java.sql.* package.
3. Register the JDBC Driver:
4. Load the database driver class.
5. Establish a Connection:
6. Use DriverManager.getConnection() to connect to the database.
7. Create a Statement:
8. Use Connection.createStatement() or Connection.prepareStatement().
9. Execute SQL Queries:
10. Use methods like executeQuery() or executeUpdate().
11. Process the Results:
12. Iterate through the ResultSet.
13. Close the Resources:
14. Close ResultSet, Statement, and Connection objects.

JDBC Driver Types-

Type 1: JDBC-ODBC Bridge Driver (deprecated).
Type 2: Native-API Driver.
Type 3: Network Protocol Driver.
Type 4: Thin Driver (pure Java, commonly used).

DAY13-

Multithreading-

- Multithreading in Java allows a program to execute multiple threads concurrently, enabling efficient utilization of CPU resources
- Thread-A lightweight process that runs within a program.
- Multiple threads can execute independently.

- Main Thread-The main thread is the default thread that starts executing a Java program.
- Additional threads can be created to perform tasks concurrently.
- Multitasking-Process-based multitasking: Multiple processes run simultaneously.
- Thread-based multitasking: Multiple threads within the same process run concurrently.

Creating Threads in Java-

Java provides two ways to create threads-

1. By extending the Thread class.
2. By implementing the Runnable interface.

Thread Pooling-

Thread pooling involves creating a pool of threads to reuse for tasks, which improves performance by avoiding the overhead of creating and destroying threads.

DAY14-

Garbage collection-

Garbage Collection (GC) in Java is the process by which Java programs automatically reclaim memory that is no longer in use. The Java Virtual Machine (JVM) performs garbage collection to manage memory efficiently by deleting objects that are no longer reachable in a program.

How Garbage Collection Works in Java-

Mark-and-Sweep Algorithm-

Mark phase: The garbage collector identifies which objects are still reachable.

Sweep phase: The garbage collector reclaims the memory occupied by unreachable objects.

DAY15-

Inner Class and Wrapper Class-

Inner Classes in Java-

An inner class is a class that is defined within another class. Inner classes can access members (including private members) of the outer class, providing more powerful and flexible ways of working with classes.

Wrapper Classes in Java-

Wrapper classes are part of the Java standard library and are used to convert primitive data types into objects. Java provides wrapper classes for each of the primitive types to allow them to be treated as objects. These classes are part of the java.lang package.

List of Wrapper Classes-

- Byte for byte
- Short for short
- Integer for int
- Long for long
- Float for float
- Double for double
- Character for char
- Boolean for boolean

Why Wrapper Classes are Used-

- Java's collections framework (e.g., ArrayList, HashMap) works with objects, but primitive types are not objects. Wrapper classes allow primitive types to be used as objects.
- Wrapper classes provide useful methods to convert strings to primitive types, or vice versa. For example, Integer.parseInt() converts a string to an int.
- Autoboxing: Java automatically converts a primitive type to its corresponding wrapper class (e.g., int to Integer).
- Unboxing: Java automatically converts a wrapper class object back to its primitive type (e.g., Integer to int).

DAY16-

Collection-

Java Collections Framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. The framework includes various classes for storing and manipulating groups of data.

Common Implementations-

- Queue: A collection designed for holding elements prior to processing. It follows the FIFO (First-In-First-Out) principle.

- Deque (Double-Ended Queue): A linear collection that supports adding and removing elements from both ends.
- Methods: addFirst(), addLast(), removeFirst(), removeLast().
- Map: An object that maps keys to values. A map cannot contain duplicate keys, and each key can map to at most one value.
- ArrayList: A dynamically resizable array. It provides fast access to elements (constant time complexity for get()), but it can be slower for insertions and deletions in the middle of the list (linear time complexity).
- LinkedList: A doubly linked list. It allows faster insertions and deletions at both ends, but accessing elements by index is slower (linear time complexity).
- Vector: Similar to ArrayList but synchronized. It is less commonly used because of better alternatives like ArrayList.
- Stack: A subclass of Vector that implements a stack data structure (LIFO).
- HashSet: A collection that does not allow duplicates and does not guarantee any order of elements.
- LinkedHashSet: A HashSet that maintains the insertion order of elements.
- TreeSet: A Set that stores elements in a sorted (ascending) order according to their natural ordering or a custom comparator.
- HashMap: A key-value pair map that allows null values and keys. It does not guarantee the order of keys.
- LinkedHashMap: A HashMap that maintains the order in which keys are inserted.
- TreeMap: A Map that stores keys in sorted order based on their natural ordering or a comparator.

DAY17-

Synchronization-

Synchronization is a mechanism that ensures that two or more threads do not access shared resources simultaneously. This is essential in multi-threading environments, where multiple threads can interfere with each other while accessing shared data, leading to inconsistent results.

Why Synchronization is Important- When multiple threads access shared data concurrently, and at least one thread modifies the data, it can lead to data corruption, inconsistency, and unpredictable behavior. For example, if two threads try to update the same variable at the same time, it may lead to lost updates or race conditions.

Ways to Achieve Synchronization in Java-

1. Synchronized Methods
2. Synchronized Blocks
3. Locks (Explicit Locking)
4. Volatile Keyword

Thread Communication (wait(), notify(), notifyAll())

1. Synchronized Methods-

A synchronized method ensures that only one thread can execute the method at a time on an object. If one thread is executing a synchronized method, other threads attempting to access the same method (on the same object) will be blocked until the first thread finishes.

2. Synchronized Blocks-

A synchronized block allows for more granular control over synchronization. Instead of synchronizing an entire method, a block of code within a method can be synchronized. This can improve performance, as it reduces the scope of synchronization to only the critical section.

3. Locks (Explicit Locking)-

Java provides explicit locking mechanisms through the java.util.concurrent.locks package, which offers greater flexibility and control over synchronization. The Lock interface (e.g., ReentrantLock) allows threads to acquire and release locks explicitly.

DAY18-

Lambada Expression-

Lambda expressions, introduced in Java 8, provide a clear and concise way to represent functional interfaces (interfaces with a single abstract method) using an expression. Lambda expressions allow you to write functional-style programming in Java, making your code more readable, compact, and easier to work with, especially when dealing with collections and concurrency.

Basic Syntax of Lambda Expressions-

(parameters) -> expression

parameters: This is the list of input parameters.

arrow (->): Separates the parameters and the body of the lambda expression.

expression: The body of the lambda expression, where logic is written.

forEach()-it is used to iterate over each element of the list and print it.

filter() is used to filter the names starting with 'A' and print them.

DAY19-

java.io and java.nio Package-

In Java, the java.io package and the java.nio package both deal with input and output operations, but they differ in terms of architecture, flexibility, and efficiency. Let’s explore both in detail:

java.io Package-

The java.io package is the original API for input and output operations in Java. It provides classes and interfaces to read and write data from files, data streams, and the console. The primary I/O operations are generally blocking, meaning each operation waits for completion before the next one begins.

Classes in java.io-

- File: Represents files and directories in the filesystem.
- FileInputStream reads bytes from a file.
- FileOutputStream writes bytes to a file.
- BufferedReader wraps around other readers and improves performance by reading larger chunks of data.
- BufferedWriter wraps around other writers to write text data more efficiently.
- PrintWriter: Prints formatted text to a file or output stream. It also provides convenience methods like println().
- ObjectInputStream / ObjectOutputStream: Used for reading and writing Java objects.
- DataInputStream / DataOutputStream: Used for reading and writing primitive data types.
- FileReader / FileWriter: Used for reading and writing character data.

- RandomAccessFile: Provides read and write access to any part of a file, and allows the file pointer to be moved around.

java.nio Package-

The java.nio package offers more scalable, flexible, and efficient I/O operations. It provides features like non-blocking I/O, memory-mapped files, and buffers that allow you to handle large amounts of data with greater performance and concurrency. The NIO package focuses on handling I/O operations more efficiently, especially for network and file systems.

Components of java.nio-

1.Buffers: A buffer is an object that holds data of a specific type (e.g., ByteBuffer, CharBuffer, etc.). Buffers are the core building blocks for reading and writing data in NIO.

2.Channels: A channel represents a connection to an I/O device. It is similar to a stream in java.io, but channels can be non-blocking.

Key Classes in java.nio-

Path: Represents a file or directory path. It is part of the NIO.2 file system API.

....