**Day -08 Java**

**Mohammed Arsalain**

**Employee ID: 109890529**

**mobarsal@amazon.com**

Day 8 - 05th June 2025

Enum , gettters and setters (encapsulation), Arrays, (pending from yesterday)

OOP's concepts Inheritance, Polymorphism, Encapsulation, Abstraction, Interfaces, Exception Handling,

Enums

//Attaching Multiple values

public enum Element {

H("Hydrogen", 1, 1.008f),

HE("Helium", 2, 4.0026f),

// ...

NE("Neon", 10, 20.180f);

private static final Map<String, Element> BY\_LABEL = new HashMap<>();

private static final Map<Integer, Element> BY\_ATOMIC\_NUMBER = new HashMap<>();

private static final Map<Float, Element> BY\_ATOMIC\_WEIGHT = new HashMap<>();

static {

for (Element e : values()) { //for each loop

BY\_LABEL.put(e.label, e);

BY\_ATOMIC\_NUMBER.put(e.atomicNumber, e);

BY\_ATOMIC\_WEIGHT.put(e.atomicWeight, e);

}

}

public final String label;

public final int atomicNumber;

public final float atomicWeight;

private Element(String label, int atomicNumber, float atomicWeight) {

this.label = label;

this.atomicNumber = atomicNumber;

this.atomicWeight = atomicWeight;

}

public static Element valueOfLabel(String label) {

return BY\_LABEL.get(label);

}

public static Element valueOfAtomicNumber(int number) {

return BY\_ATOMIC\_NUMBER.get(number);

}

public static Element valueOfAtomicWeight(float weight) {

return BY\_ATOMIC\_WEIGHT.get(weight);

}

}

Task 19:

Wap to display the content of the above enum.. (main needs to be added)

10.06 to 10.10

==================================================================

Code from yesterday

================================================================

Task 017:

Getter and setter

Create a program name Person.java

public class Person {

private String name;

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String newName) {

this.name = newName;

}

}

Create another program named Task017.java

public class Task017{

public static void main(String[] args) {

Person myObj = new Person();

myObj.name = "John";

System.out.println(myObj.name);

}

}

—----------------------------------what is the reason for the error —---------------explain

Task 018

Now create one more program named Task018.java

public class Main {

public static void main(String[] args) {

Person myObj = new Person();

myObj.setName("John");

System.out.println(myObj.getName());

}

}

Now —--------------think what is the output of the above code—--------------

========================================================================

4444444444444444444444$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$444444444444444444444========================================================================

Arrays

Task 020:

Create an array of your name

Hint : use

Char[] Name = {‘P’, “r’, ….}; // initializing an array

sout(Name);

Int n = Name.length; // size of your name

sout(“there are “+ n +”letters in my name”);

Use for loop to display each letter..

HInt: use ghe below code snippet…

// traversing array

for (int i = 0; i < n; i++)

System.out.print(Name[i] + " ");

11.17 to 17.20

Coffee break : 11.20 to 11.35

23 done put of 29

As of 11.40

sHALLOW copy and Deep copy

Copy Type

Description

Example

Shallow Copy

Both arrays share references to the same objects

Shallow Copied Array : [[obj1, obj2] , [obj3], [obj4]]

Deep Copy

New instances of objects are created.

Deep copied array : [[newObj1, newObj2] , [newObj3], [newObj4]]

+++++++++++++++++++++++++++++++++++++++++=============+++++++++++++++++

Task 21 to Task 30 – home tasks — plz refer Doc 17 Arrays in java..

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

=========================================+++++++++++++++++++++++++++++++

OOPS (Object Oriented Programming) ++++++++++++++++++++++++++++++++++++++++++==============================

Inheritance, Polymorphism, Encapsulation, Abstraction, Interfaces, Exception Handling,

4 pillers of OOPS

Inheritance, Polymorphism, Encapsulation, Abstraction

Inheritance : getting parental characteristics

Inheritance —- > over riding

Types of inheritance

Single inheritance

Multi level inheritance

Hybrid inheritance

Hierarchical inhe

Multiple Inheritance — not supported in java

Due to diamond problem

Task 031

class Calculation {

int z;

public void addition(int x, int y) {

z = x + y;

System.out.println("The sum of the given numbers:"+z);

}

public void Subtraction(int x, int y) {

z = x - y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class My\_Calculation extends Calculation {

public void multiplication(int x, int y) {

z = x \* y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]) {

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

}

public class My\_Calculation2 extends Calculation {

public void multiplication(int x, int y) {

z = x \* y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]) {

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

}

Inheritance provides - reusability

It avoids - duplication

Multi level inheritance

Clac < ========= My\_calculation < ======= calculation

Class calc extends My\_calculation{

}

—--Task 032 ------------------------------------------------------------------------------

In the above code add a class clock — and try to extend calculation and clock in the my calculation class.. Is it possible ???? give reason.

class clock {

—--

—--

}

class my\_calculation extends calculation , clock{ // multiple inheritance

// —---------------------------------- ???????????????????????

}

Task 033

—-----------------------------------------------------------------

class Customer {

Void purchage\_list{

Int cos = 40t;

String items = “Tomatoes”;

}

}

public class Mart extends Customer {

Void billing(){

String items = “onions”;

Int cost = 30;

}

Psvm (String[] args) {

Super.items = “Potatoes”

Super.cost = 50;

Sout(items);

sout(cost);

sout”(%%%%%%%%%%%%%%”);

Sout(super.items);

sout(suer.cost);

}

}

—----------------------------------------------------------------------------------------------------------------------------

Polymorphism – Method overloading

-------------------------------------------------------------------------------------------------------------------------------

Two or more methods having the same name but differ -

No of parameters

Type of parameters

Sequence of parameters

No of parameters:

Task 034

Void add(int x, int y){

Sout —> x and y values

}

Void add(int x, int y, int z){

Sout —-> x, y, z values

}

psvm(){

add(10,20,30);

add(50,100);

}

Type of parameters

Task 035

Void add(char x, char y){

Sout —-> x, y values

}

Void add(int x, int y) {

Sout —> x, y values

}

psvm(){

add(‘d’, ‘a’);

add(100, 100);

}

Sequence of Parameters

Task 036

Void add(int x, float y){

Sout → x, y values

}

Void add(float x, int y){

Sout → x, y

}

psvm(){

add(10.50f, 60);

add(100, 80.80f)

}

========================================================================

Encapsulation

========================================================================

Data hiding – secured data

Access modifiers 👍

private

Protected – inheritance

Public – anyone can access

Task 037:

Class Employee{

Private int pwd;

Protected int Salary;

Public int empid:

employee(){ // constructors are methods having same name as class name (we have in c++)

}

~employee(){// destructors used in c++ but not in java

}

}

Class Hr extends Employee {

super.pwd = 1254; //===============> ??????

super.Salary = 50000; //==================> ?

Super.empid = 10001; // ======================>?

psvm(){

}

}

========================================================================

Abstraction ========================================================================

Java Abstract Classes

A Java class which contains the abstract keyword in its declaration is known as abstract class.

Java abstract classes may or may not contain abstract methods, i.e., methods without body ( public void get(); )

But, if a class has at least one abstract method, then the class must be declared abstract.

If a class is declared abstract, it cannot be instantiated.

To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.

If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Task 038

/\* File name : AbstractDemo.java \*/

Public class AbstractDemo {

public static void main(String [] args) {

/\* Following is not allowed and would raise error \*/

Employee e = new Employee("George W.", "Houston, TX", 43);

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

abstract class Employee {

private String name;

private String address;

private int number;

public Employee(String name, String address, int number) {

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public double computePay() {

System.out.println("Inside Employee computePay");

return 0.0;

}

public void mailCheck() {

System.out.println("Mailing a check to " + this.name + " " + this.address);

}

public String toString() {

return name + " " + address + " " + number;

}

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public void setAddress(String newAddress) {

address = newAddress;

}

public int getNumber() {

return number;

}

}

Task 039

Rewrite the above code to give the output without errors..

Task 040

// Working of Abstraction in Java

abstract class Gadgets {

abstract void turnOn();

abstract void turnOff();

}

// Concrete class implementing the abstract methods

class TVRemote extends Gadgets {

@Override

void turnOn() {

System.out.println("TV is turned ON.");

}

@Override

void turnOff() {

System.out.println("TV is turned OFF.");

}

}

class ACRemote extends Gadgets {

@Override

void turnOn() {

System.out.println("AC is turned ON.");

}

@Override

void turnOff() {

System.out.println("AC is turned OFF.");

}

}

// Main class to demonstrate abstraction

public class Main {

public static void main(String[] args) {

Gadgets remote = new TVRemote();

Gadgets remote = new ACRemote();

remote.turnOn();

remote.turnOff();

Gadgets remote = new FanRemote();

Gadgets remote = new CoolerRemote();

remote.turnOn();

remote.turnOff();

}

}

12 pax done –@ 6.03

14 pax @ 6.10

========================================================================

Interfaces in java

========================================================================

An Interface in Java programming language is defined as an abstract type used to specify the behaviour of a class. An interface in Java is a blueprint of a behaviour. A Java interface contains static constants and abstract methods.

Key Properties of Interface:

The interface in Java is a mechanism to achieve abstraction.

By default, variables in an interface are public, static, and final.

It is used to achieve abstraction and multiple inheritance in Java.

It supports loose coupling (classes depend on behavior, not implementation).

In other words, interfaces primarily define methods that other classes must implement.

An interface in Java defines a set of behaviours that a class can implement, usually representing an IS-A relationship, but not always in every scenario.

Task 041

import java.io.\*;

// Interface Declared

//Driver Code Ends

interface testInterface {

// public, static and final

final int a = 10;

// public and abstract

void display();

}

// Class implementing interface

class TestClass implements testInterface {

// Implementing the capabilities of

// Interface

public void display(){

System.out.println("Myclass");

}

}

class Myclass

//Driver Code Starts

{

public static void main(String[] args)

{

TestClass t = new TestClass();

t.display();

System.out.println(t.a);

}

}er..