Task 19: Wap to display the content of the below enum.. (main  needs to be added)

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);

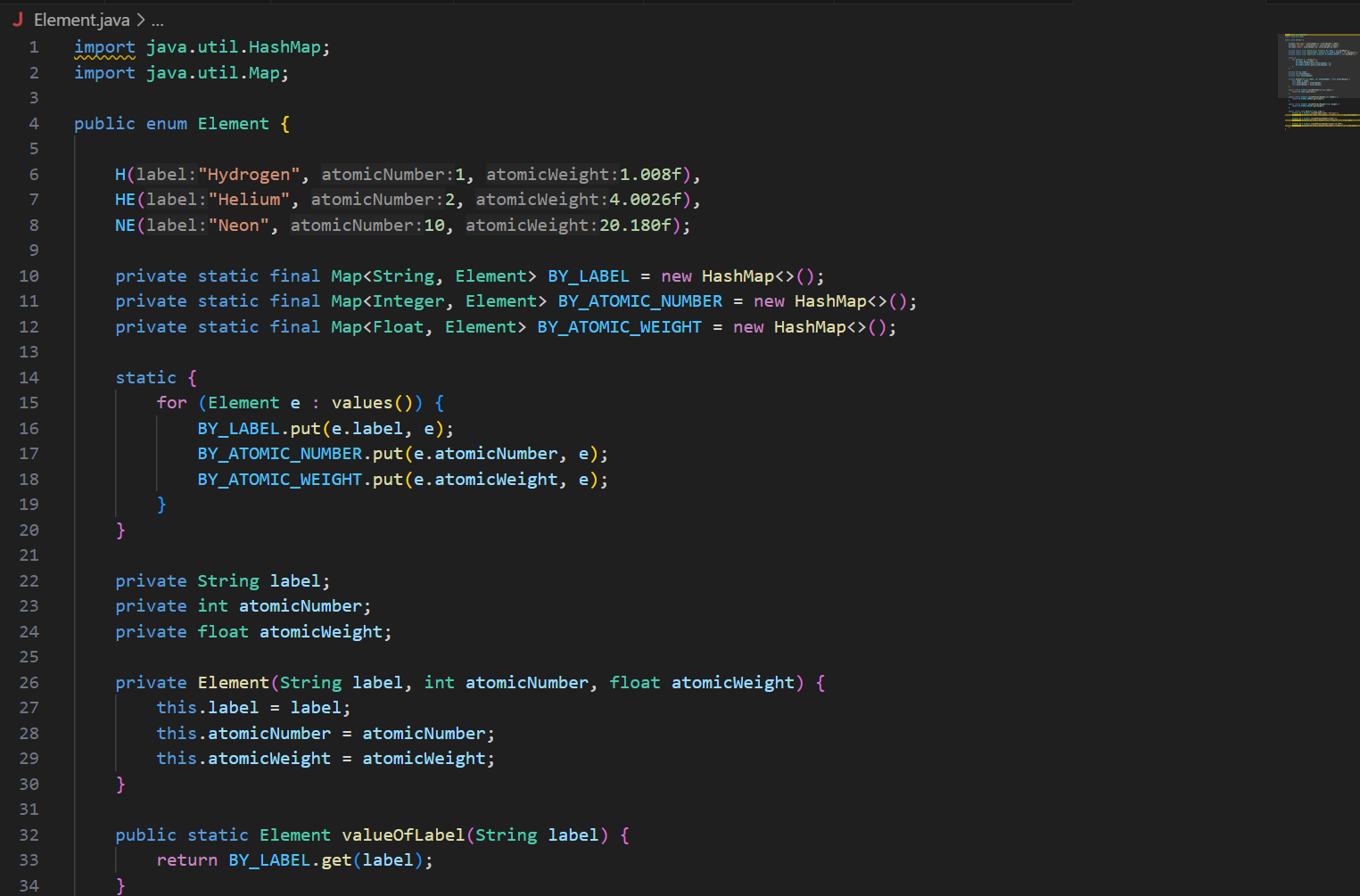
    }

    public static Element valueOfAtomicWeight(float weight) {

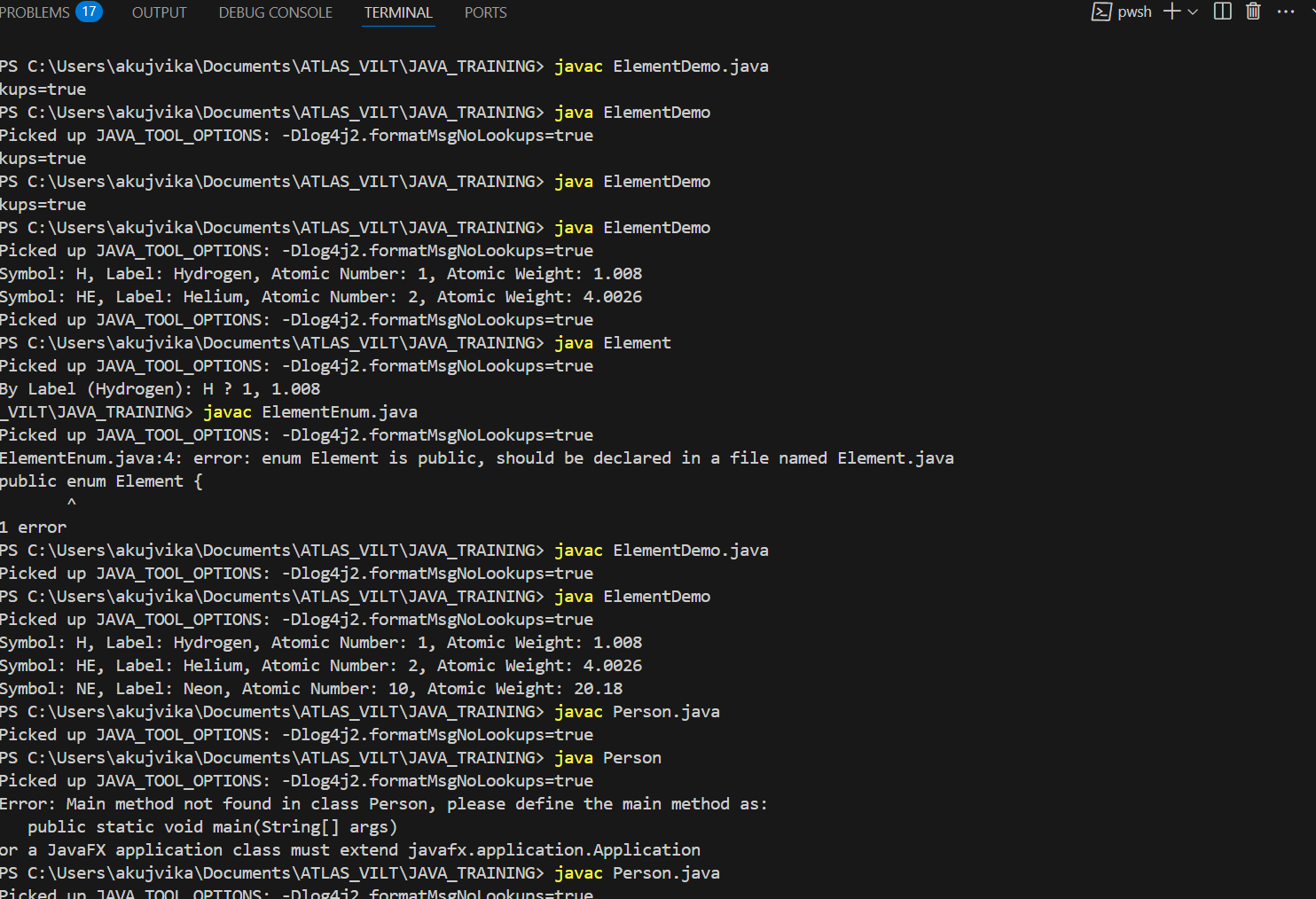
        return BY\_ATOMIC\_WEIGHT.get(weight);

    }

}







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();

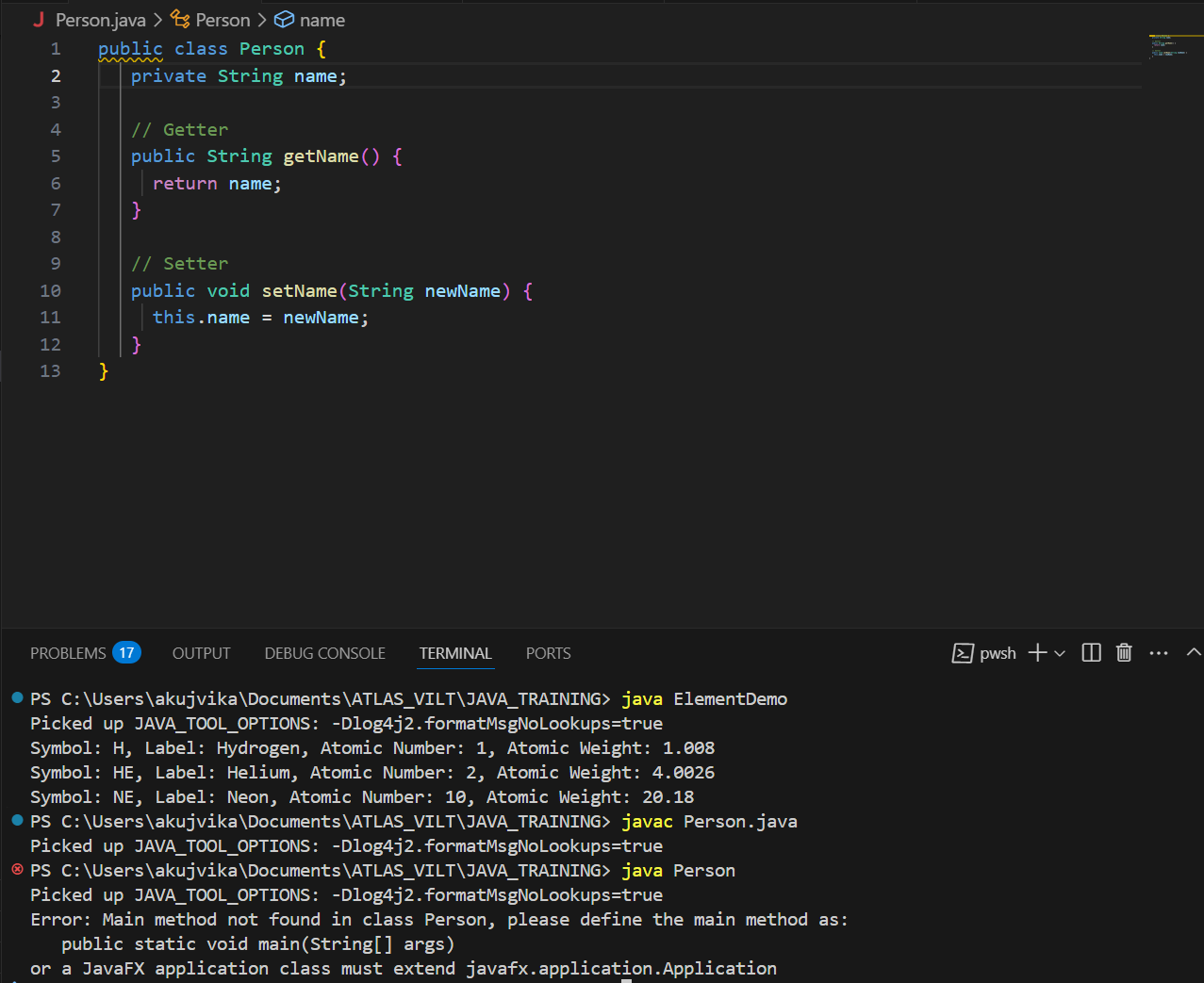
    myObj.name = "John";

    System.out.println(myObj.name);

  }

}

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



The error in the original Task017.java code is because we are trying to access the `name` variable directly, But in the Person class, the name variable is declared as private. The private keyword means the name variable can only be accessed within the Person class itself.We cannot access or modify it directly from outside the class (like in Task017.java).To fix this, should use the setter and getter methods provided in the Person class

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());

  }

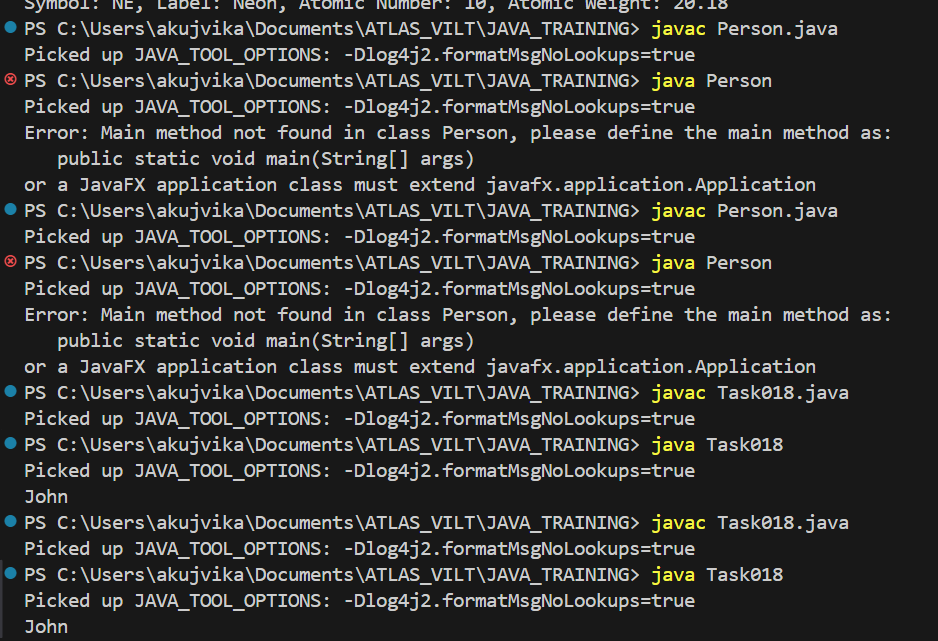
}

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

myObj.setName("John"); sets the name to "John".

System.out.println(myObj.getName()); prints the name, which is "John".

As long as you have the Person class with the correct getter and setter in your project, this will be the output.



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”);

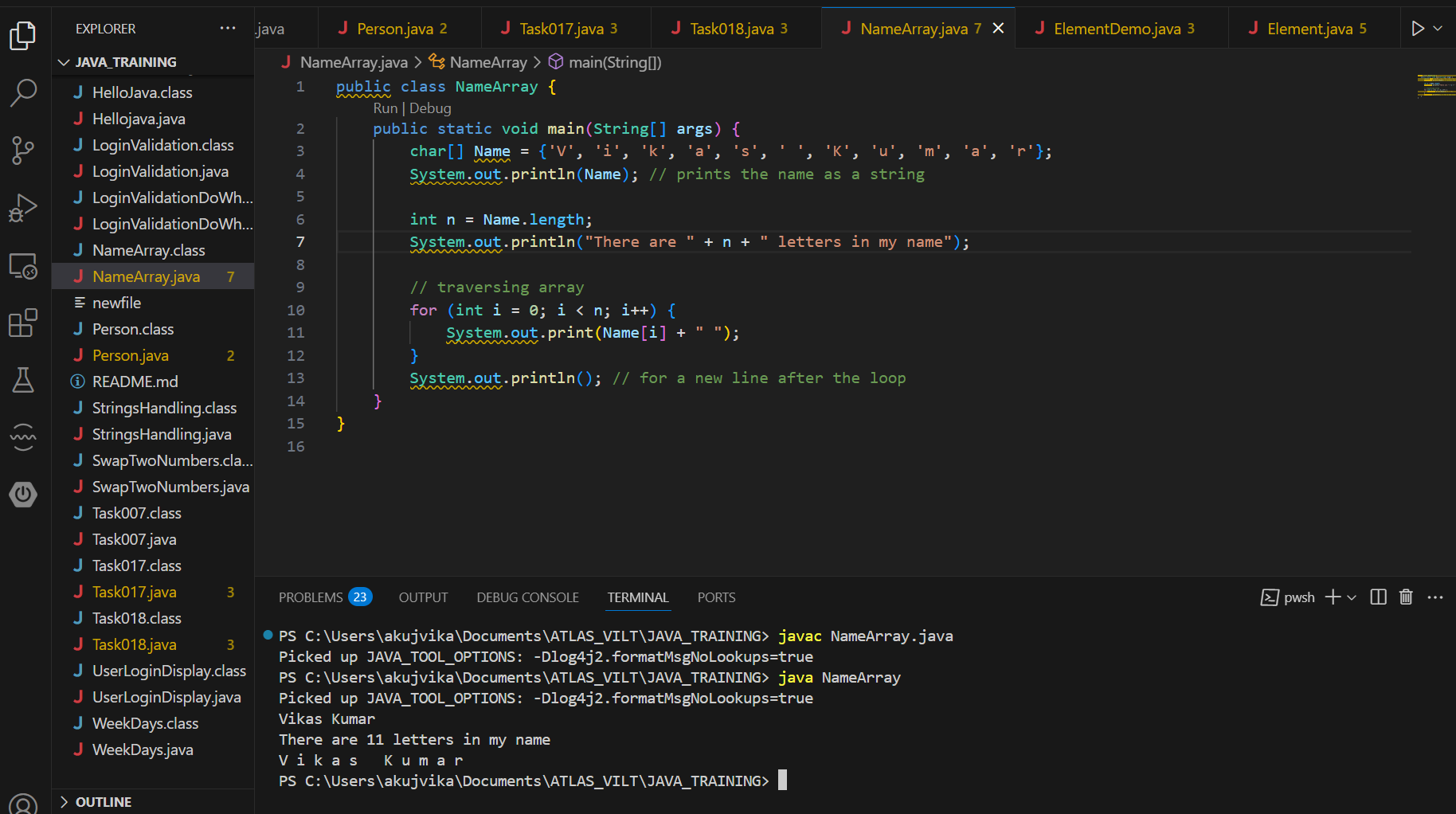
Use for loop to display each letter..

HInt: use ghe below code snippet…

// traversing array

        for (int i = 0; i < n; i++)

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



*Task 022 - home task*

#### Implementation:

// Java program to illustrate creating an array

// of integers,  puts some values in the array,

// and prints each value to standard output.

​

class GFG {

   public static void main(String[] args)

   {

       // declares an Array of integers.

       int[] arr;

​

       // allocating memory for 5 integers.

       arr = new int[5];

​

       // initialize the elements of the array

       // first to last(fifth) element

      arr[0] = 10;

      arr[1] = 20;

       arr[2] = 30;

       arr[3] = 40;

       arr[4] = 50;

​

       // accessing the elements of the specified array

       for (int i = 0; i < arr.length; i++)

           System.out.println("Element at index "

                              + i + " : " + arr[i]);

   }

}

**Output**

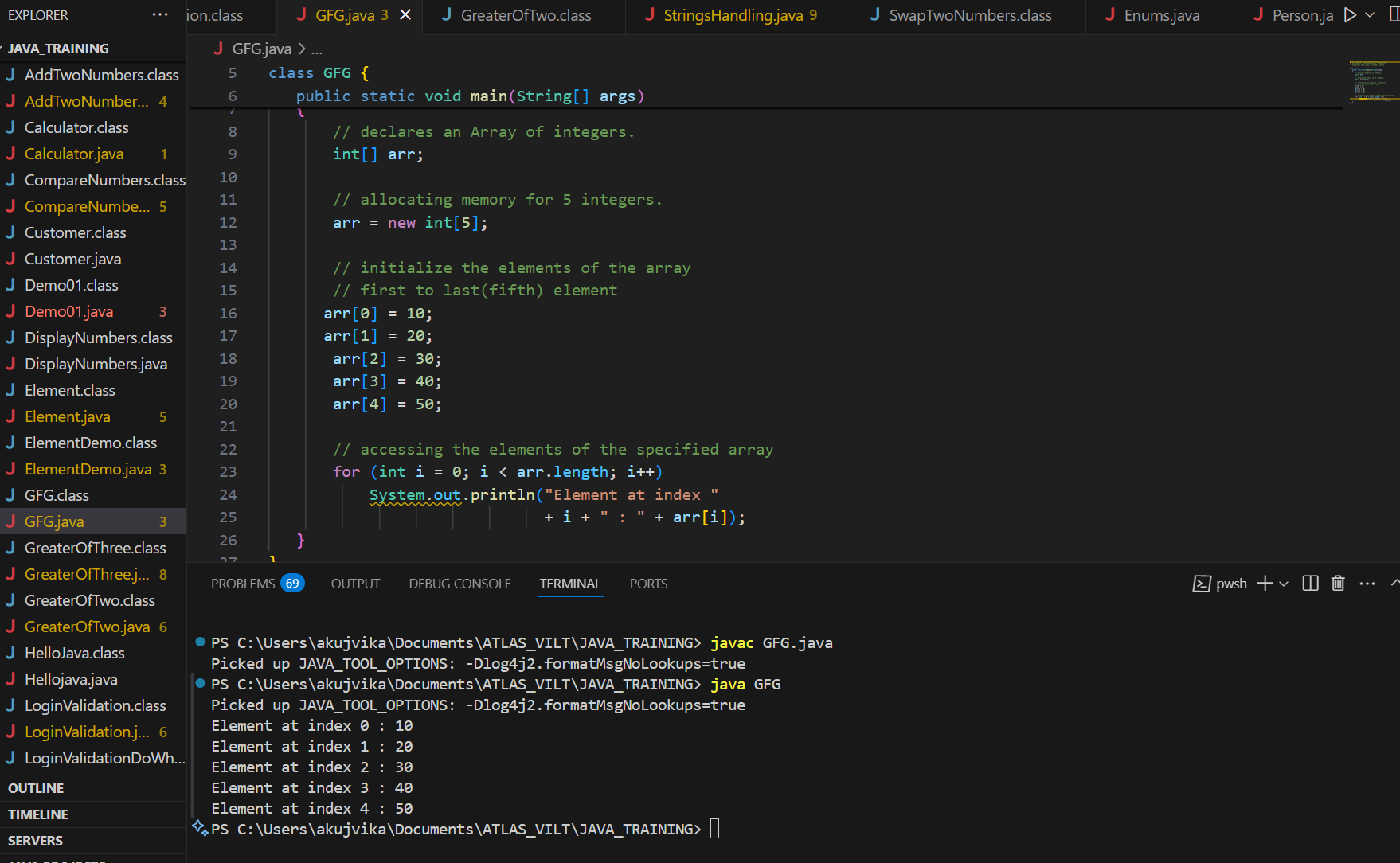
Element at index 0 : 10

Element at index 1 : 20

Element at index 2 : 30

Element at index 3 : 40

Element at index 4 : 50



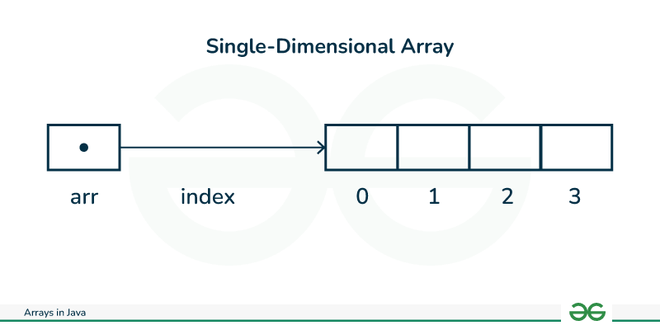
## Types of Arrays in Java

Java supports different types of arrays:

### 1. Single-Dimensional Arrays

These are the most common type of arrays, where elements are stored in a linear order.

***// A single-dimensional array****int[] singleDimArray = {1, 2, 3, 4, 5};*

**

### 2. Multi-Dimensional Arrays

Arrays with more than one dimension, such as two-dimensional arrays (matrices).

***// A 2D array (matrix)****int[][] multiDimArray = {  
      {1, 2, 3},  
     {4, 5, 6},  
     {7, 8, 9} };*

You can also access java arrays using [for each loops](https://www.geeksforgeeks.org/for-each-loop-in-java/).

## Arrays of Objects in Java

An array of objects is created like an array of primitive-type data items in the following way.

**Syntax:**

***Method 1:****ObjectType[] arrName;*

***Method 2:****ObjectType arrName[];*

### Example of Arrays of Objects

Task 023 - home task

**Example:** Here we are taking a student class and creating an array of Student with five Student objects stored in the array. The Student objects have to be instantiated using the constructor of the Student class, and their references should be assigned to the array elements.

// Java program to illustrate creating

//  an array of objects

​

class Student {

   public int roll\_no;

   public String name;

    Student(int roll\_no, String name){

       this.roll\_no = roll\_no;

       this.name = name;

   }

}

​

public class Main {

   public static void main(String[] args){

       // declares an Array of Student

       Student[] arr;

​

       // allocating memory for 5 objects of type Student.

       arr = new Student[5];

​

       // initialize the elements of the array

       arr[0] = new Student(1, "aman");

       arr[1] = new Student(2, "vaibhav");

       arr[2] = new Student(3, "shikar");

       arr[3] = new Student(4, "dharmesh");

       arr[4] = new Student(5, "mohit");

​

       // accessing the elements of the specified array

       for (int i = 0; i < arr.length; i++)

           System.out.println("Element at " + i + " : { "

                              + arr[i].roll\_no + " "

                              + arr[i].name+" }");

   }

}

**Output**

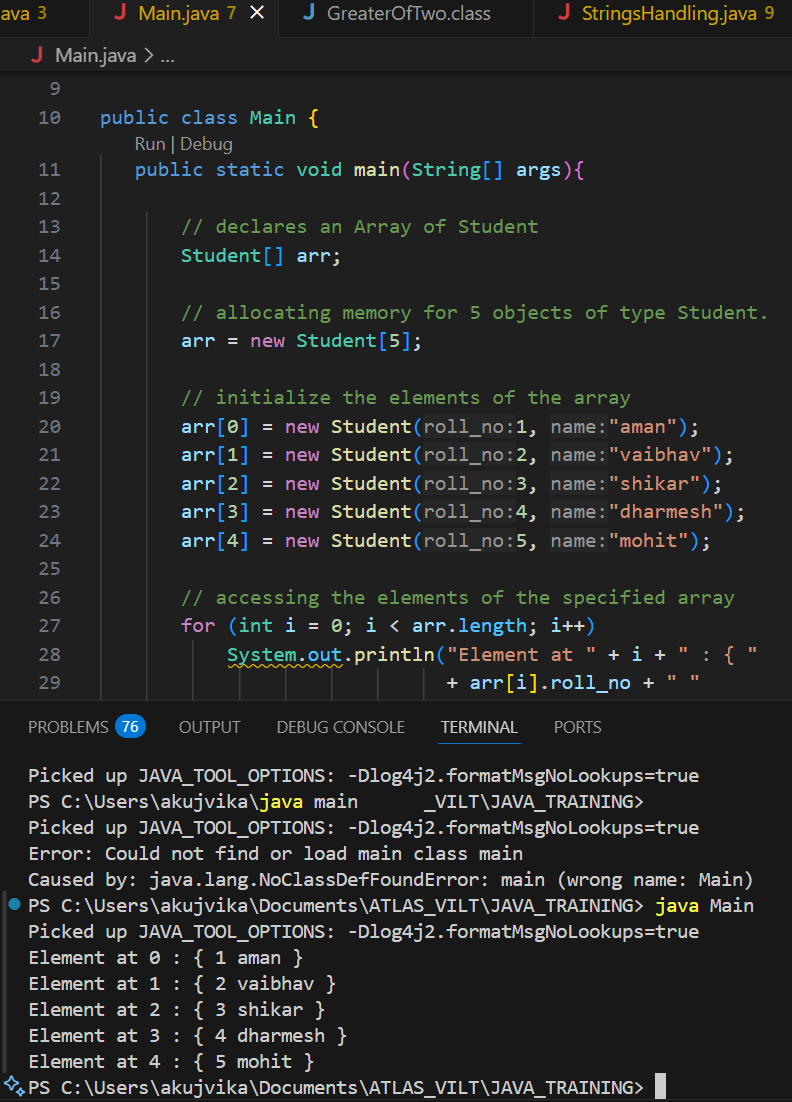
Element at 0 : { 1 aman }

Element at 1 : { 2 vaibhav }

Element at 2 : { 3 shikar }

Element at 3 : { 4 dharmesh }

Element at 4 : { 5 mohit }



Task 024 Home task

**Example:** An array of objects is also created like

// Java program to illustrate creating

//  an array of objects

 class Student{

   public String name;

    Student(String name){

       this.name = name;

   }

  @Override

   public String toString(){

       return name;

   }

}

 ​

public class Main{

   public static void main (String[] args){

       // declares an Array and initializing the

      // elements of the array

       Student[] myStudents = new Student[]{

         new Student("Dharma"),new Student("sanvi"),

         new Student("Rupa"),new Student("Ajay")

       };

        // accessing the elements of the specified array

       for(Student m:myStudents){

           System.out.println(m);

       }

   }

}

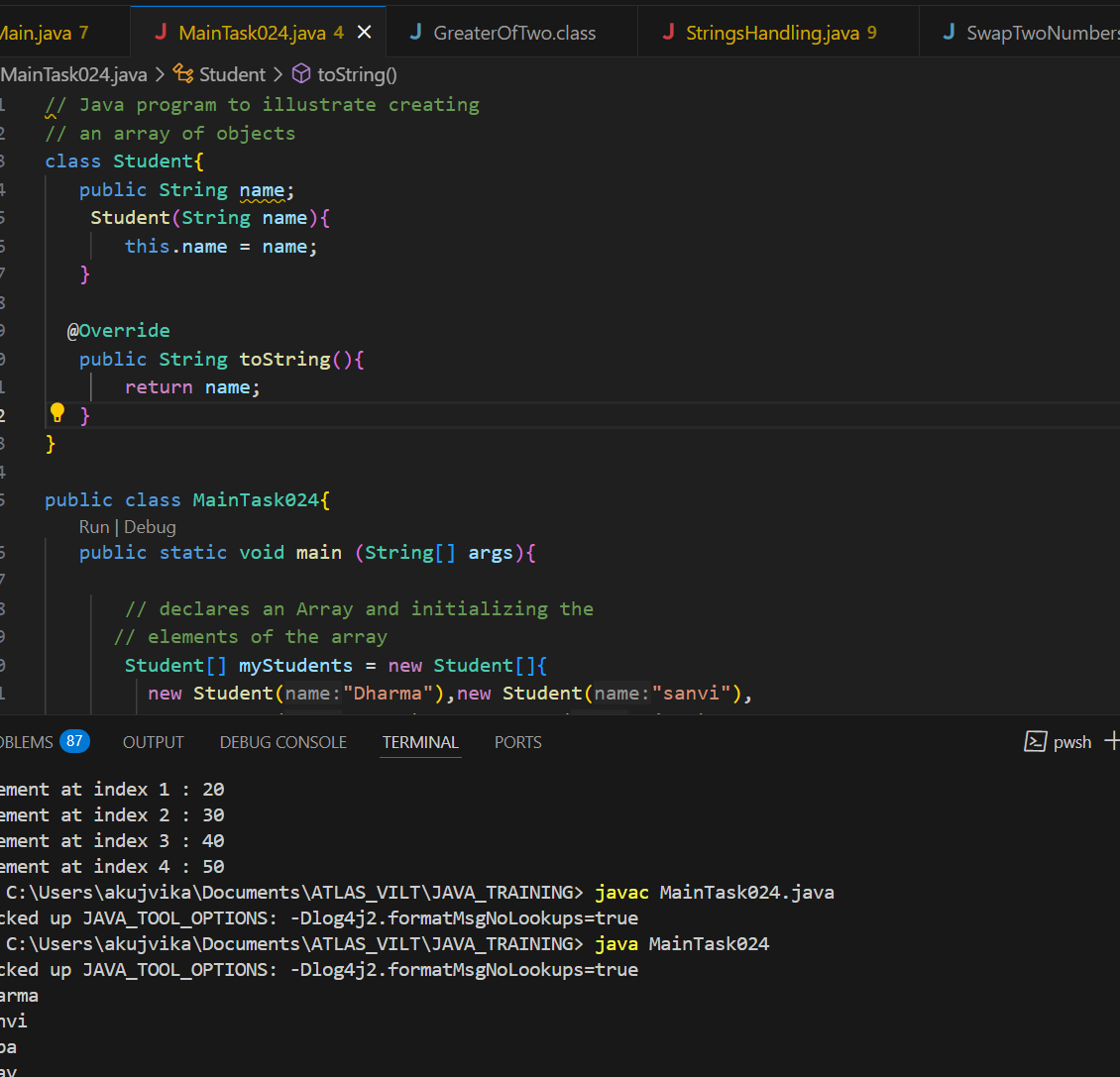
**Output**

Dharma

sanvi

Rupa

Ajay



## What happens if we try to access elements outside the array size?

JVM throws **ArrayIndexOutOfBoundsException** to indicate that the array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of an array.

**Below code shows what happens if we try to access elements outside the array size:**

**Task 025 - home Task**

// Code for showing error "ArrayIndexOutOfBoundsException"

​

public class GFG {

   public static void main(String[] args)

   {

       int[] arr = new int[4];

       arr[0] = 10;

       arr[1] = 20;

       arr[2] = 30;

       arr[3] = 40;

​

       System.out.println(

           "Trying to access element outside the size of array");

       System.out.println(arr[5]);

   }

}

**Output**

Trying to access element outside the size of array

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 4

at GFG.main(GFG.java:13)

## Multidimensional Arrays in Java

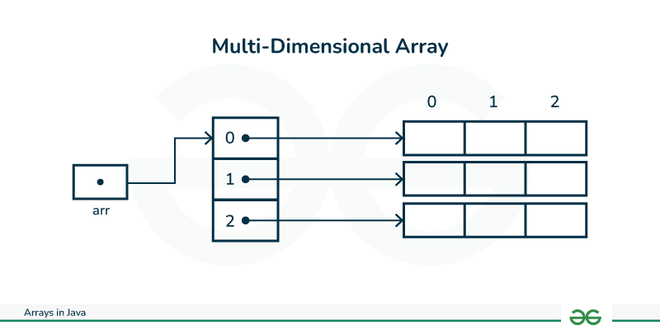
Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

**Syntax:**

There are **2 methods** to declare Java Multidimensional Arrays as mentioned below:

***// Method 1****datatype [][] arrayrefvariable;*

***// Method 2****datatype arrayrefvariable[][];*

**

### Declaration:

***// 2D array or matrix****int[][] intArray = new int[10][20];*

***// 3D array****int[][][] intArray = new int[10][20][10];*

### Java Multidimensional Arrays Examples

Task 025  - home task

**Example:** Let us start with basic two dimensional Array declared and initialized.

// Java Program to demonstrate

// Multidimensional Array

import java.io.\*;

​

class GFG {

   public static void main(String[] args){

       // Two Dimensional Array

      // Declared and Initialized

      int[][] arr = new int[3][3];

​

       // Number of Rows

       System.out.println("Rows : " + arr.length);

       // Number of Columns

       System.out.println("Columns : " + arr[0].length);

   }

}

**Output**

Rows:3

Columns:3

Task 026 - Home Task

**Example:** Now, after declaring and initializing the array we will check how to Traverse the Multidimensional Array using for loop.

// Java Program to Multidimensional Array

​

// Driver Class

public class multiDimensional {

     // main function

   public static void main(String args[])

   {

       // declaring and initializing 2D array

       int arr[][] = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

​

       // printing 2D array

       for (int i = 0; i < 3; i++) {

           for (int j = 0; j < 3; j++)

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

​

           System.out.println();

       }

   }

}

**Output**

2 7 9

3 6 1

7 4 2

## Passing Arrays to Methods

Like variables, we can also pass arrays to methods. For example, the below program passes the array to method *sum* to calculate the sum of the array's values.

Task 27 - Home task

// Java program to demonstrate

// passing of array to method

​

public class Test {

   // Driver method

   public static void main(String args[])

   {

       int arr[] = { 3, 1, 2, 5, 4 };

​

       // passing array to method m1

       sum(arr);

   }

​

   public static void sum(int[] arr)

   {

       // getting sum of array values

       int sum = 0;

​

       for (int i = 0; i < arr.length; i++)

           sum += arr[i];

​

       System.out.println("sum of array values : " + sum);

   }

}

**Output**

sum of array values : 15

## Returning Arrays from Methods

As usual, a method can also return an array. For example, the below program returns an array from method *m1*.

Task 28 - Home Task

// Java program to demonstrate

// return of array from method

​

class Test {

   // Driver method

   public static void main(String args[])

   {

       int arr[] = m1();

​

       for (int i = 0; i < arr.length; i++)

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

   }

​

   public static int[] m1()

   {

       // returning  array

       return new int[] { 1, 2, 3 };

   }

}

**Output**

1 2 3

## Java Array Members

Now, as you know that arrays are objects of a class, and a direct superclass of arrays is a class Object.

The members of an array type are all of the following:

* The public final field *length* contains the number of components of the array. Length may be positive or zero.
* All the members are inherited from class Object; the only method of Object that is not inherited is its [clone](https://www.geeksforgeeks.org/clone-method-in-java-2/) method.
* The public method *clone()* overrides the clone method in class Object and throws no [checked exceptions](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/).

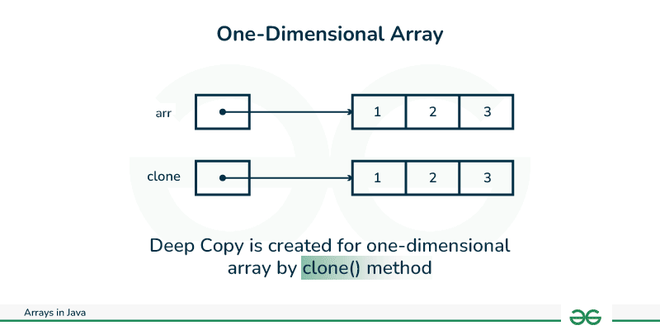
### Arrays Types and Their Allowed Element Types

|  |  |
| --- | --- |
| **Array Types** | **Allowed Element Types** |
| Primitive Type Arrays | Any type which can be implicitly promoted to declared type. |
| Object Type Arrays | Either declared type objects or it's child class objects. |
| Abstract Class Type Arrays | Its child-class objects are allowed. |
| Interface Type Arrays | Its implementation class objects are allowed. |

## Cloning Arrays in Java

### 1. Cloning of Single-Dimensional Array

When you clone a single-dimensional array, such as Object[], a **shallow copy** is performed. This means that the new array contains references to the original array's elements rather than copies of the objects themselves. A deep copy occurs only with arrays containing primitive data types, where the actual values are copied.



#### Below is the implementation of the above method:

Task 029 home Task

// Java program to demonstrate

// cloning of one-dimensional arrays

​

class Test {

   public static void main(String args[])

   {

       int intArray[] = { 1, 2, 3 };

​

       int cloneArray[] = intArray.clone();

​

       // will print false as shallow copy is created

       System.out.println(intArray == cloneArray);

​

       for (int i = 0; i < cloneArray.length; i++) {

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

       }

   }

}

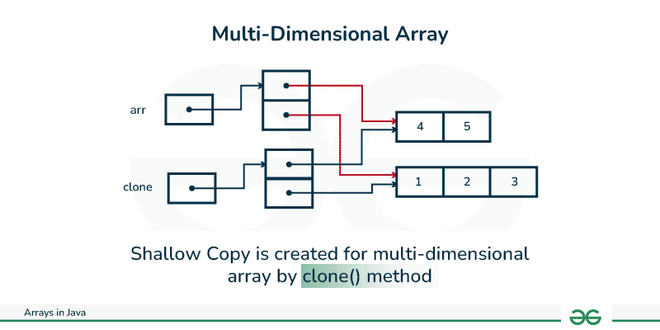
**Output**

false

1 2 3

### 2. Cloning Multidimensional Array

A clone of a multi-dimensional array (like Object[][]) is a "shallow copy," however, which is to say that it creates only a single new array with each element array a reference to an original element array, but subarrays are shared.



**Below is the implementation of the above method:**

Task 030 Home Task

// Java program to demonstrate

// cloning of multi-dimensional arrays

​

class Test {

   public static void main(String args[])

   {

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

​

       int cloneArray[][] = intArray.clone();

​

       // will print false

       System.out.println(intArray == cloneArray);

​

       // will print true as shallow copy is created

       // i.e. sub-arrays are shared

       System.out.println(intArray[0] == cloneArray[0]);

       System.out.println(intArray[1] == cloneArray[1]);

   }

}

**Output**

false

true

true

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

**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

Types of inheritance

Single inheritance

Multi level inheritance

hYBRID IN

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();

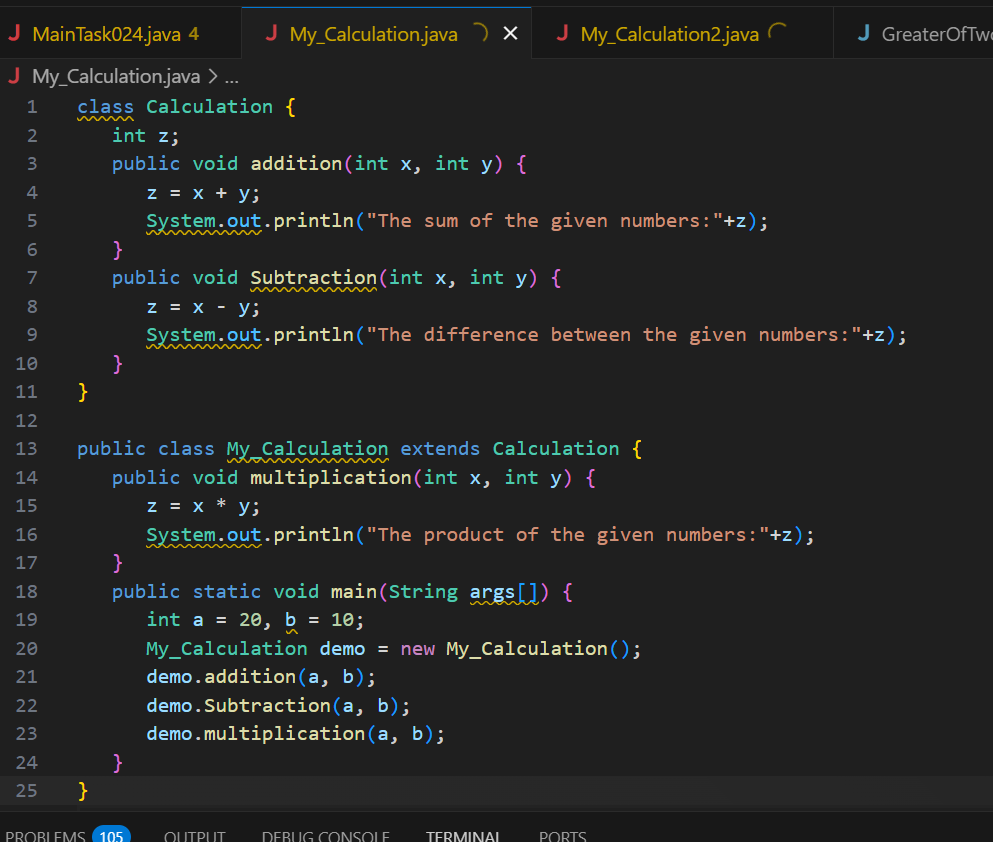
      demo.addition(a, b);

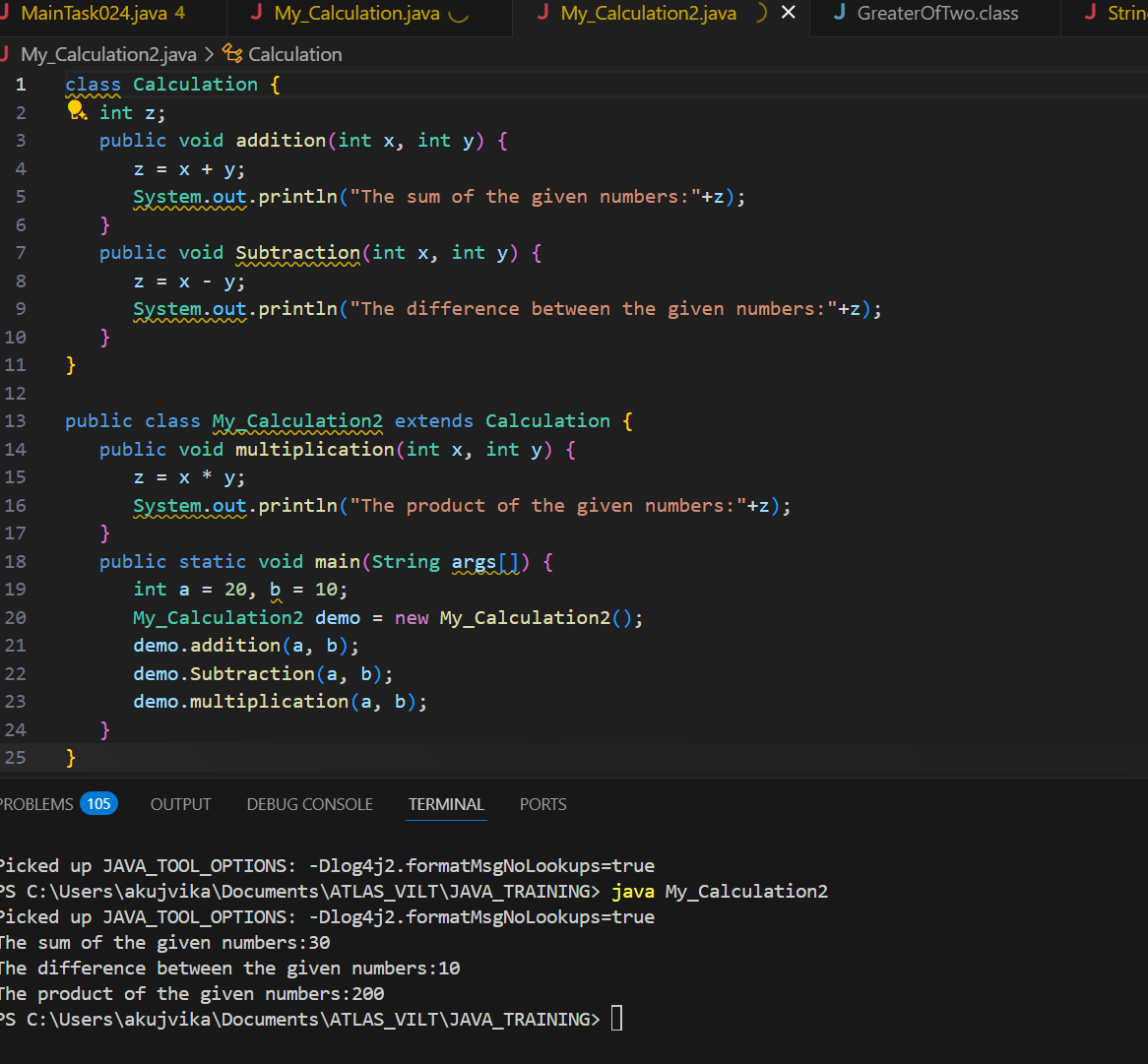
      demo.Subtraction(a, b);

      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 {

—--

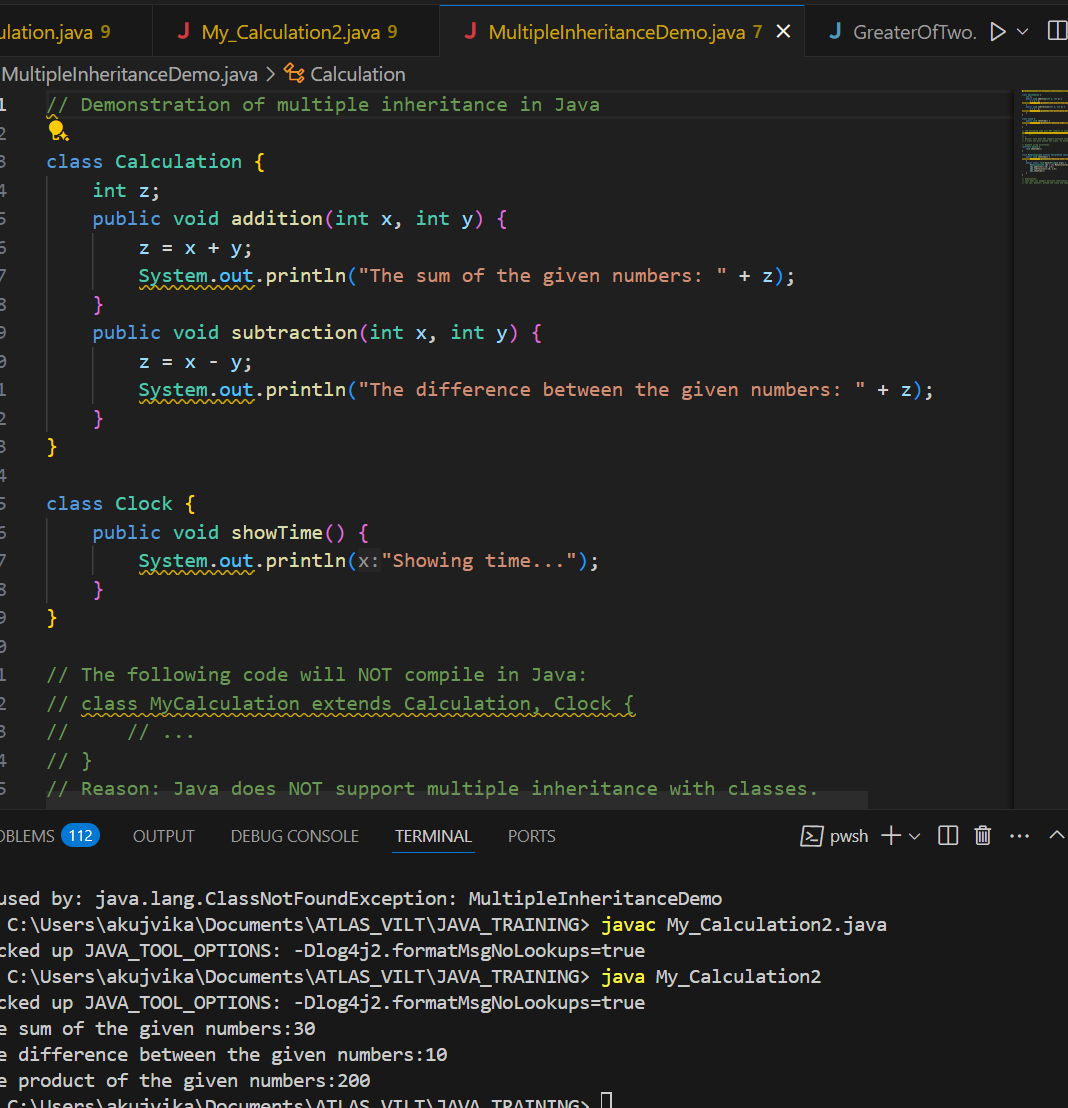
—--

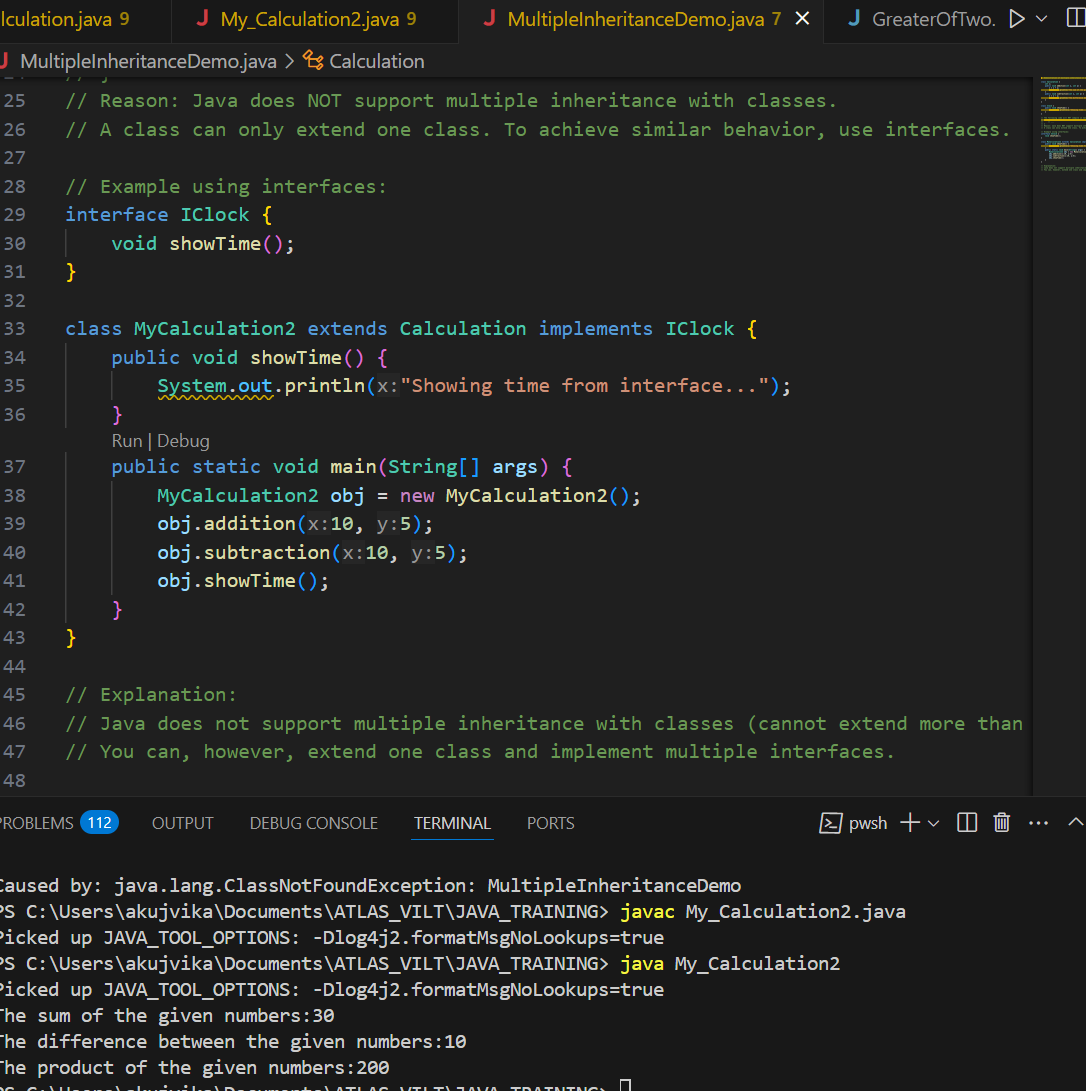
}

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

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

No, it is not possible in Java. Java does not support multiple inheritance with classes, meaning a class cannot extend more than one class. You can only extend one class, but you can implement multiple interfaces





}

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);

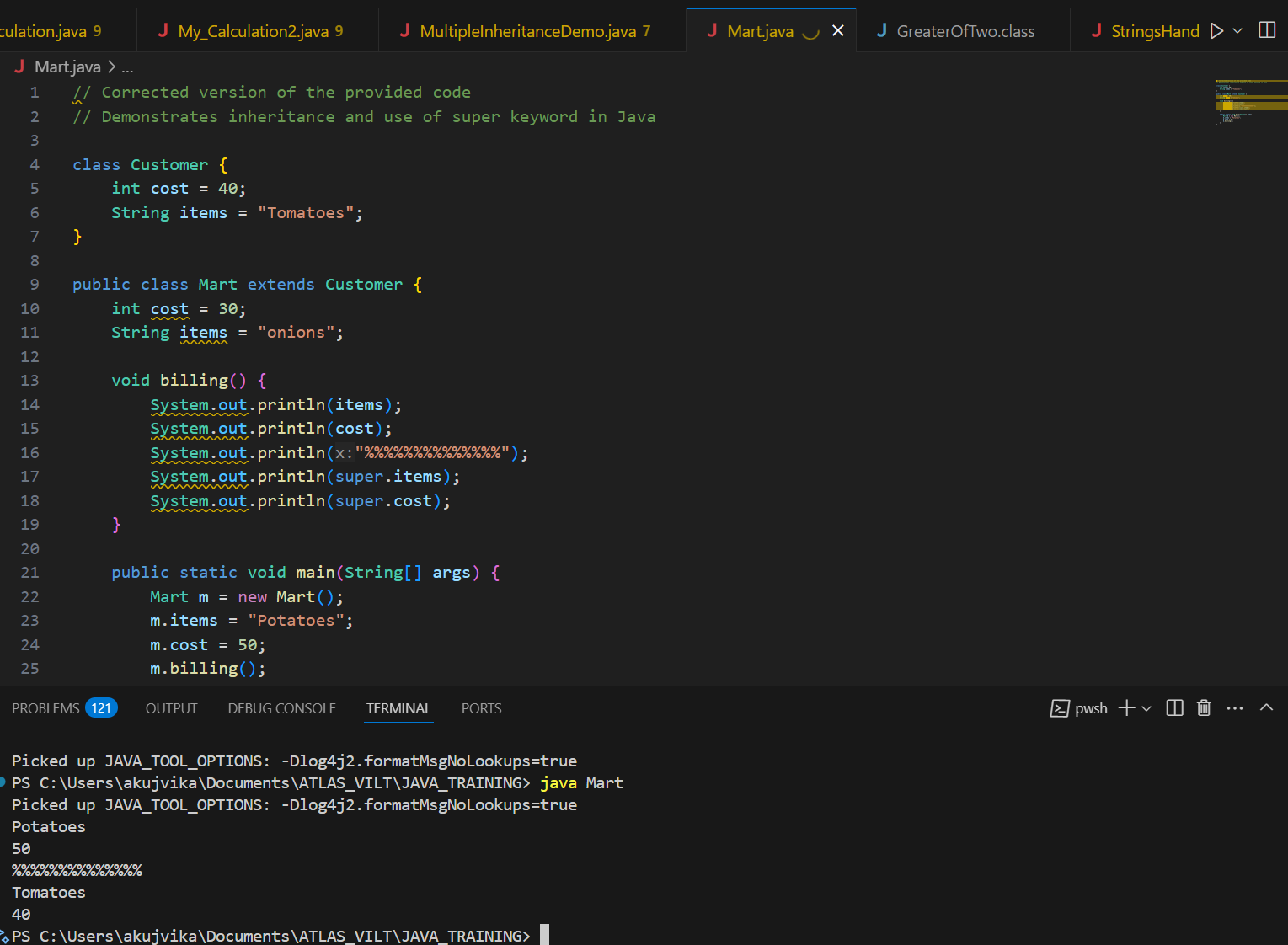
sout”(%%%%%%%%%%%%%%”);

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

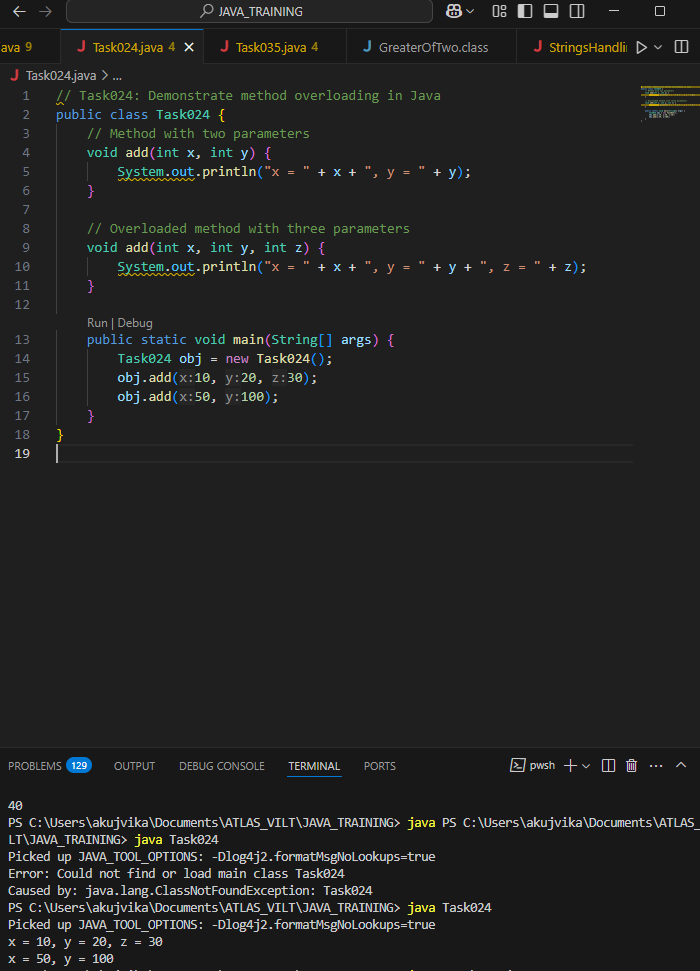
}

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

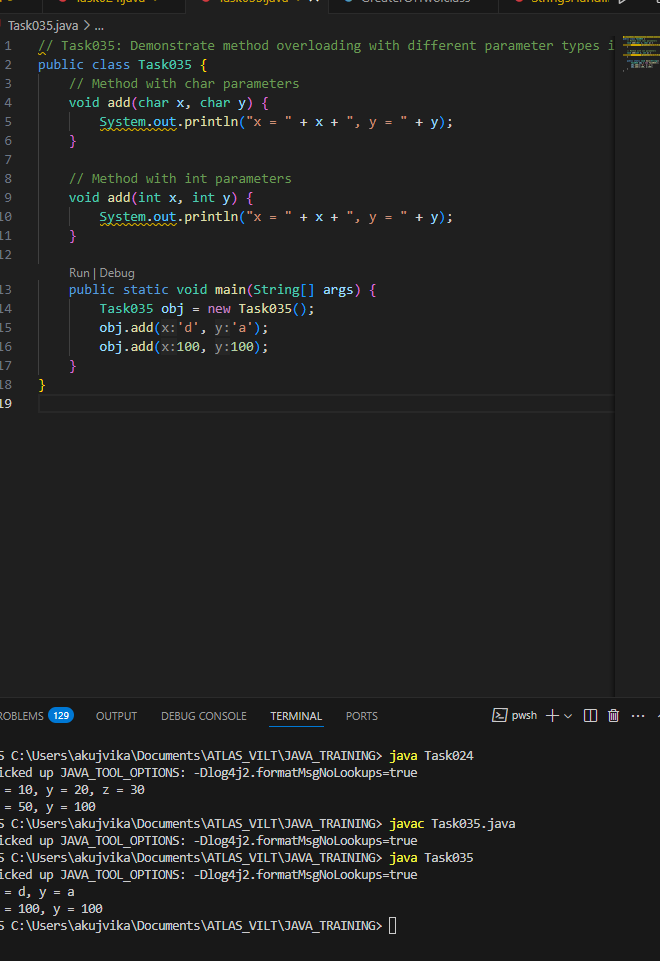
}

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

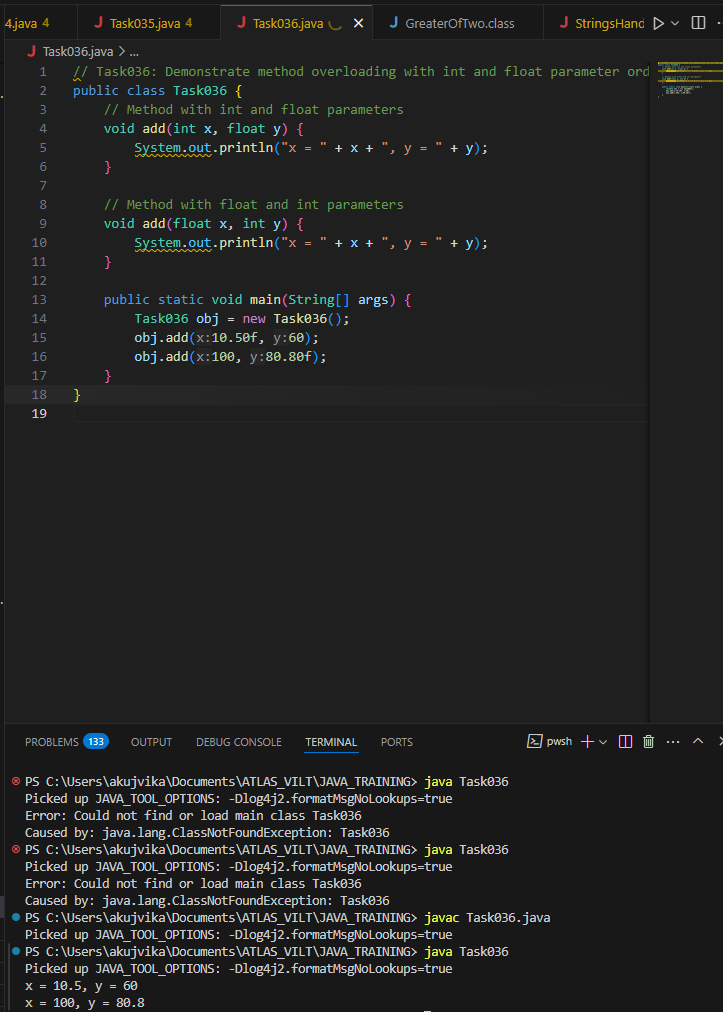
}

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 clas 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(){

}

}

// Task037: Demonstrate access modifiers and super keyword in Java inheritance

// Note: Java does not support destructors like C++. Also, 'super' cannot be used to access private members.

class Employee {

    private int pwd;

    protected int Salary;

    public int empid;

    Employee() {

        // Constructor

    }

    // No destructor in Java

}

class Hr extends Employee {

    Hr() {

        // super.pwd = 1254; // Error: Cannot access private member of superclass

        super.Salary = 50000; // OK: Salary is protected

        super.empid = 10001;  // OK: empid is public

    }

    public static void main(String[] args) {

        Hr hr = new Hr();

        // System.out.println(hr.pwd); // Not accessible

        System.out.println("Salary: " + hr.Salary);

        System.out.println("EmpID: " + hr.empid);

    }

}

// Explanation:

// - Private members (pwd) are not accessible in subclasses, even with 'super'.

// - Protected and public members can be accessed using 'super' or directly.

// - Java does not have destructors; cleanup is handled by the garbage collector.

Key points:

You cannot access or assign the private member pwd from the subclass using super.

You can access and assign the protected member Salary and the public member empid using super.

Java does not support destructors like C++.

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) {

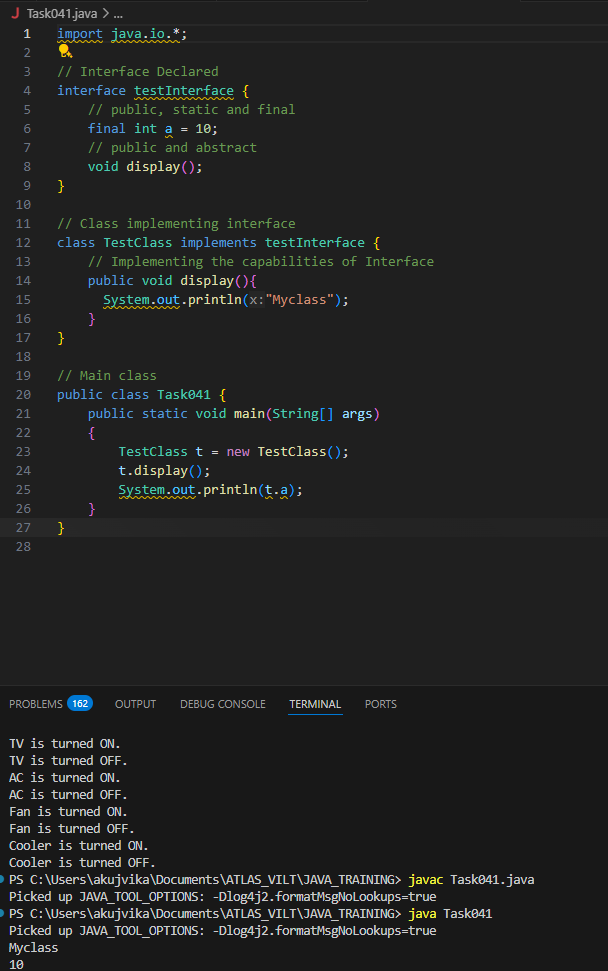
      address = newAddress;

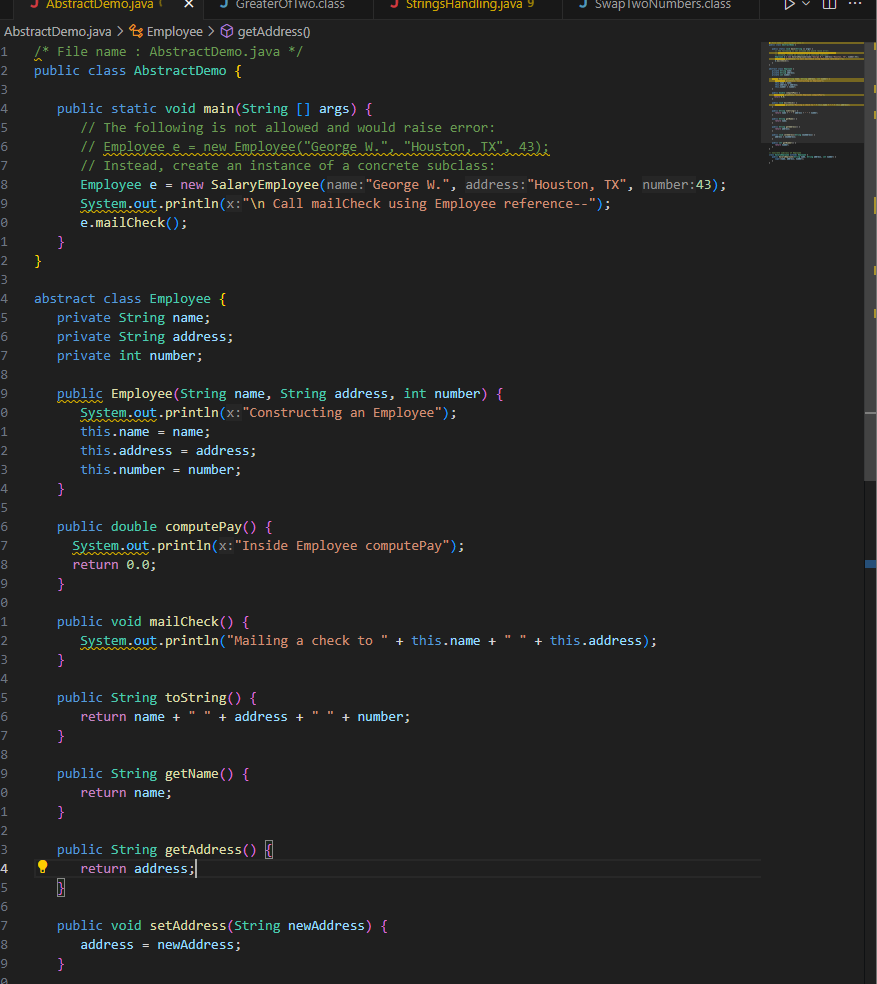
   }

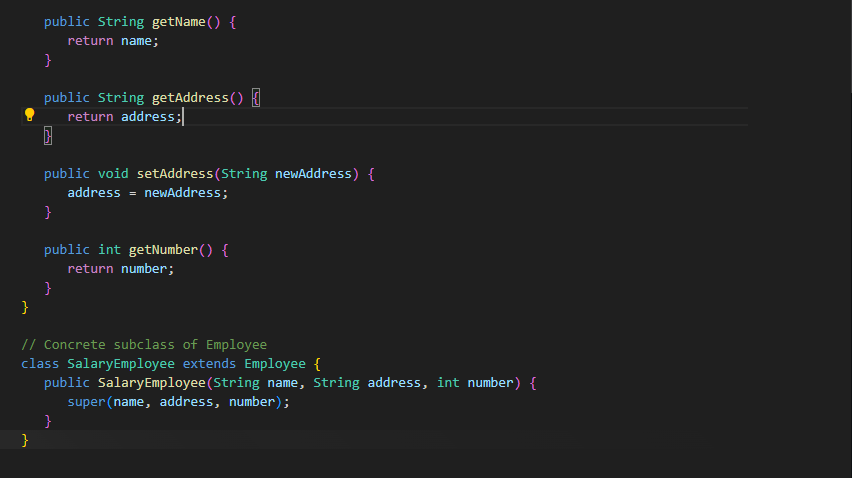
   public int getNumber() {

      return number;

   }

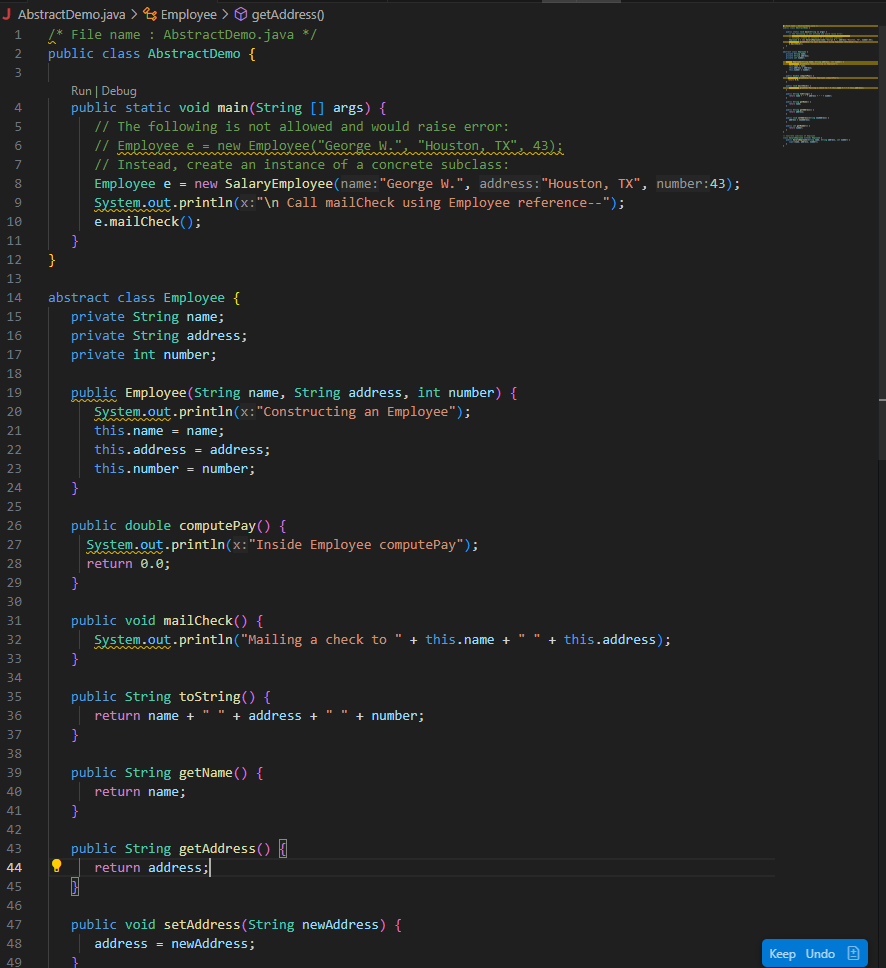


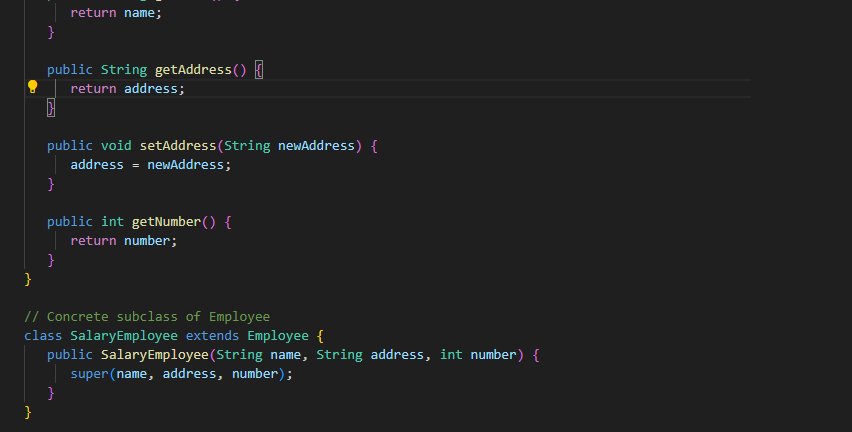




Task 039

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();

    }

}

