Name: Aravind kasanagottu

Id: mvsnarav

**Tasks from day7:**

Task16

What is the output of the below code snippet

package Enumerations;

enum color{

red, blue, green, yellow

}

public class Demo01 {

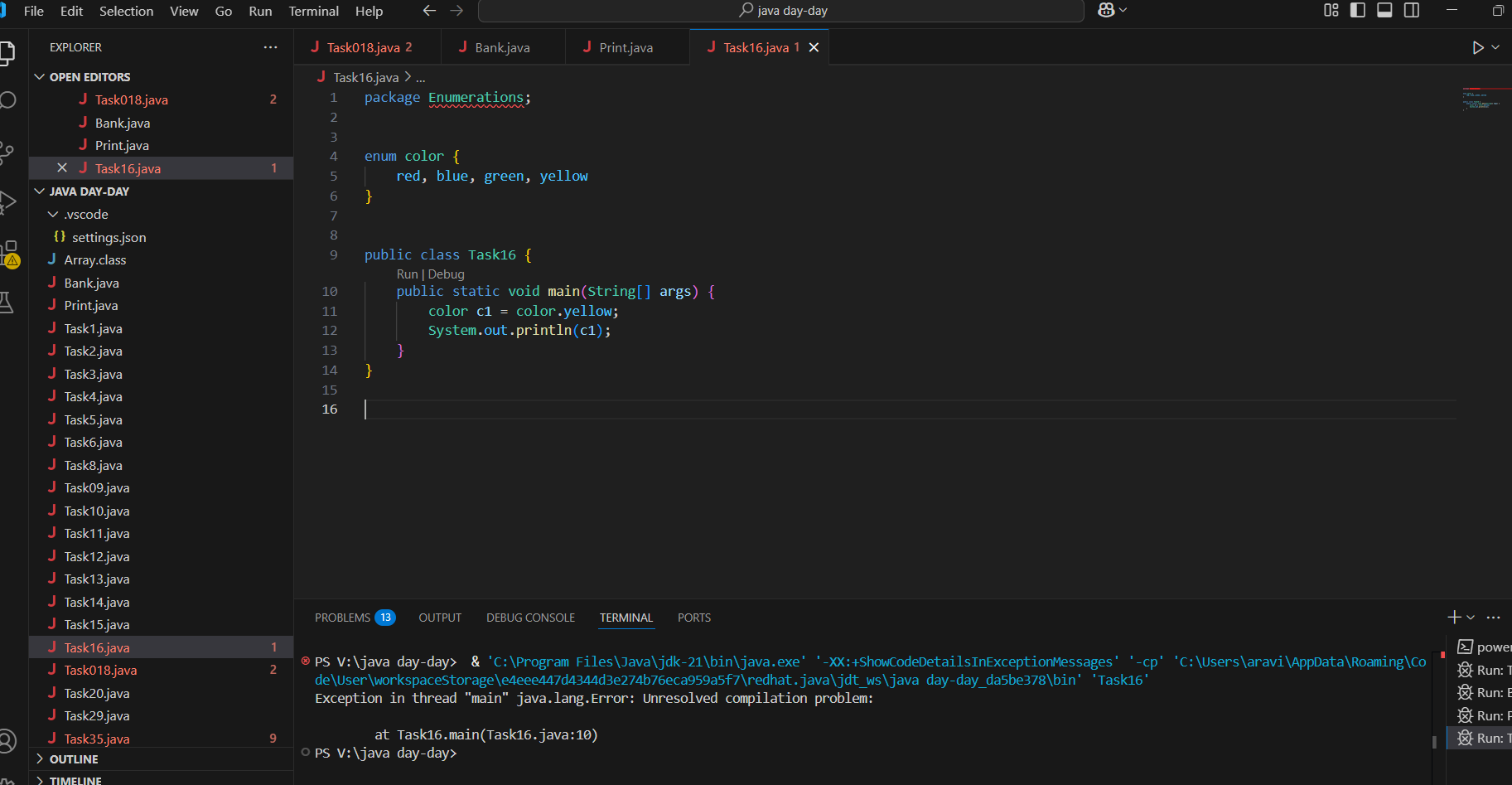
public static void main(String[] args) {

color c1 = color.yellow;

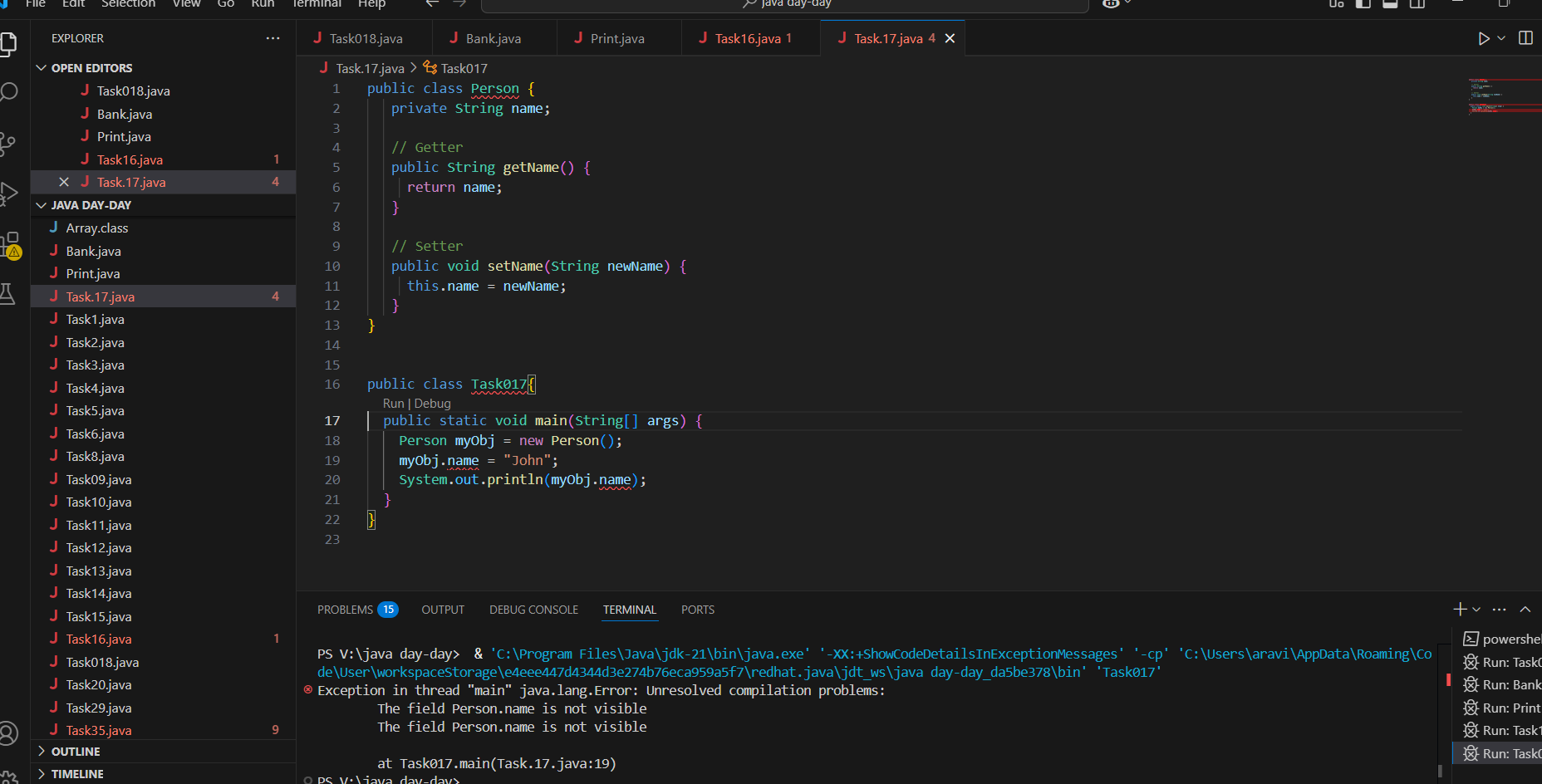
System.out.println(c1);

}

}



Task 17:



Reason for error we are declared name as private variable so we can acces this variable only inside the class only   
with use of setter and getters we can fix this error.

Task 18:

Now create one more program named Task018.java

public class Main {

  public static void main(String[] args) {

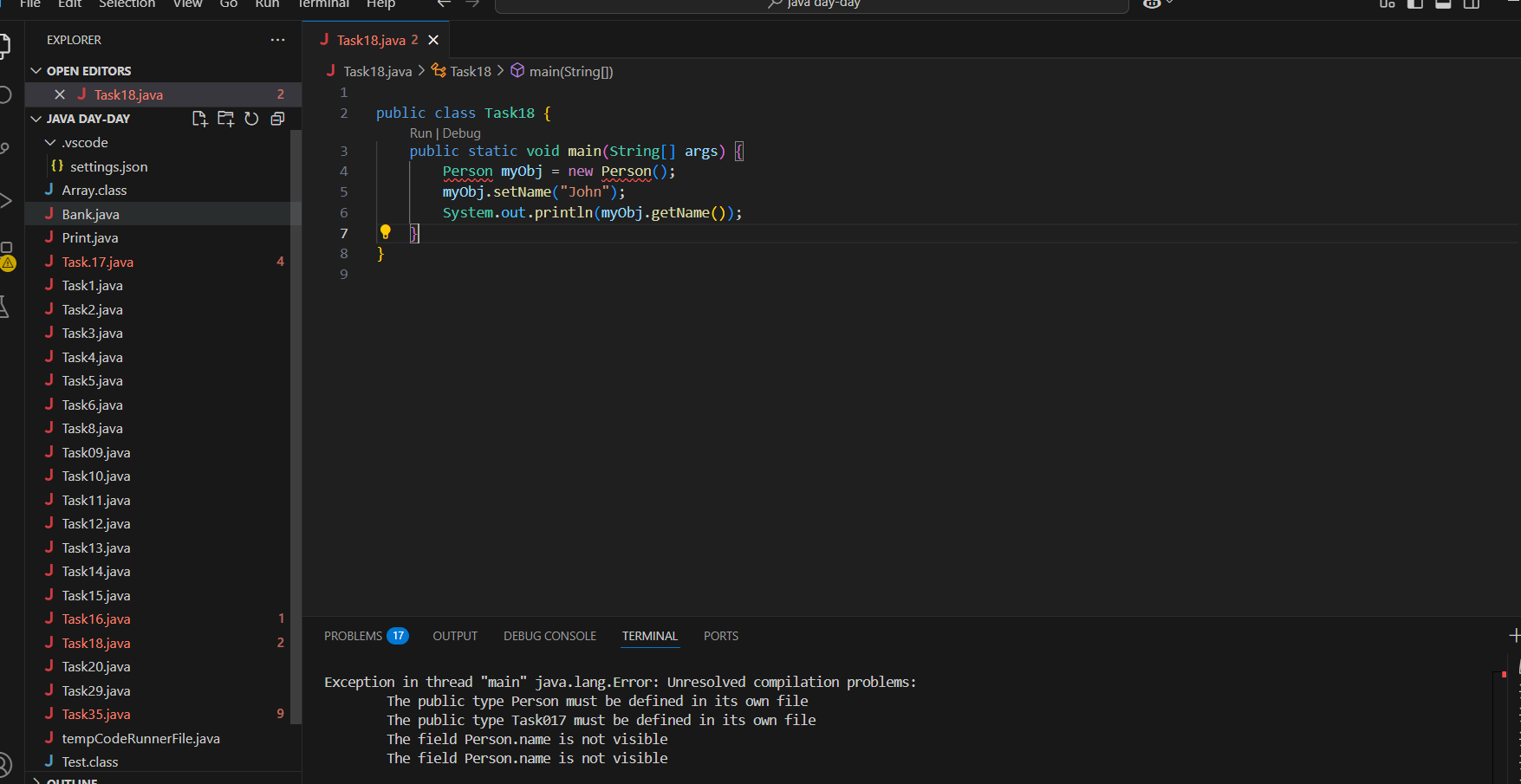
    Person myObj = new Person();

    myObj.setName("John");

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

  }

}



**Tasks day 8:**

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) {

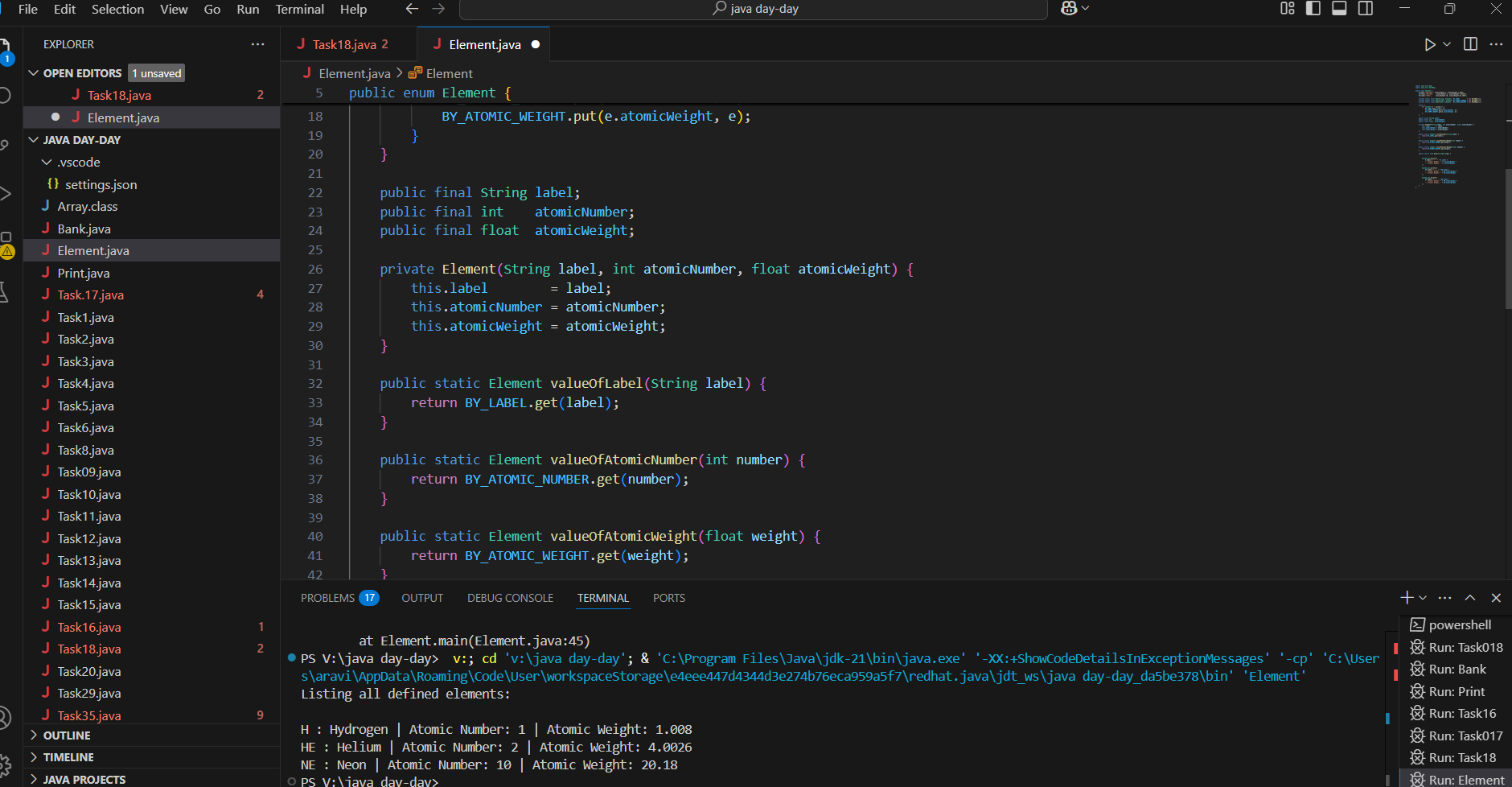
        return BY\_ATOMIC\_WEIGHT.get(weight);

    }

}

Task 19:

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



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”);

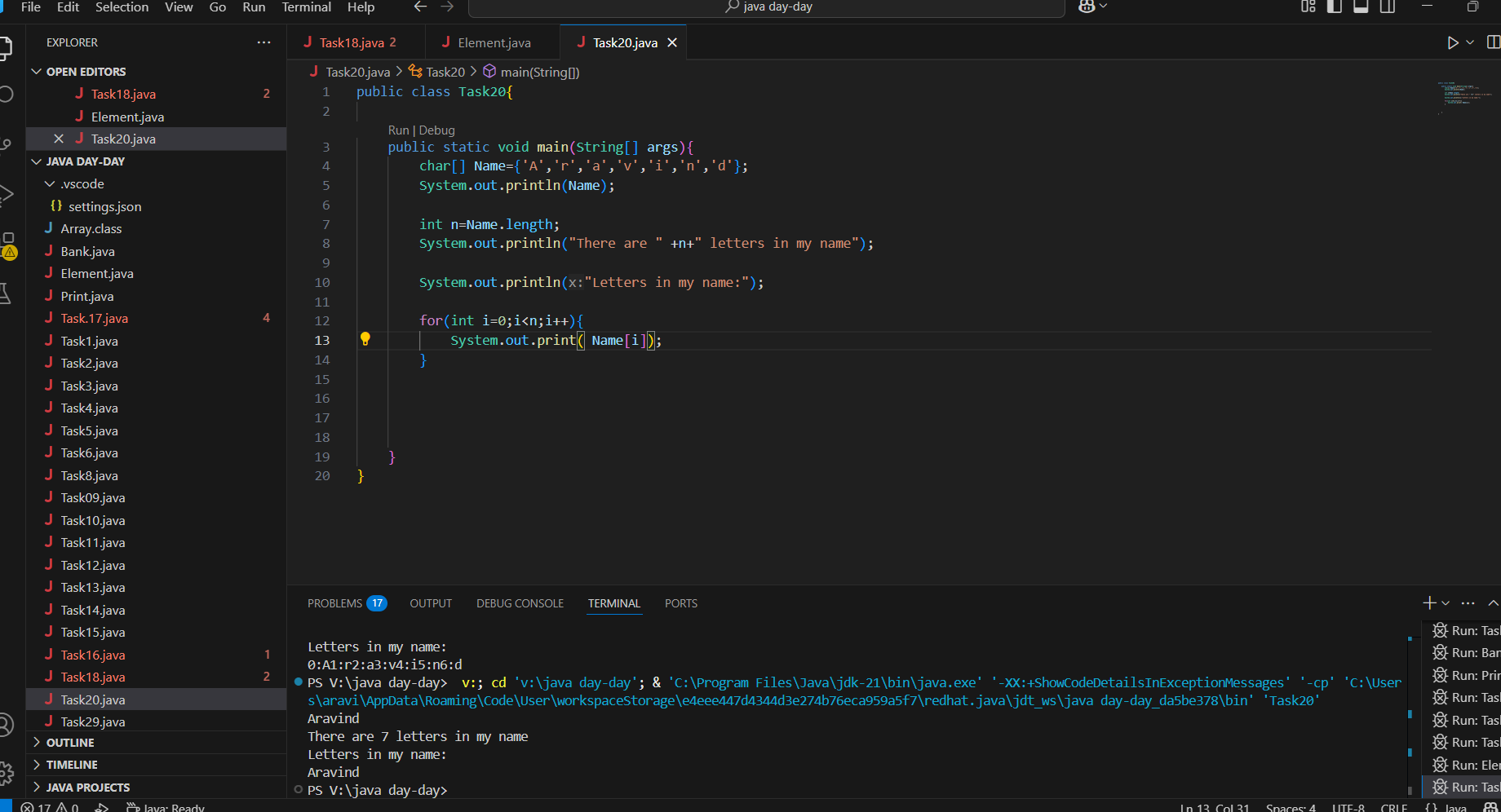
Use for loop to display each letter..

HInt: use ghe below code snippet…

// traversing array

        for (int i = 0; i < n; i++)

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



Task 21: Home Task

**Example:** This example demonstrates how to initialize an array and traverse it using a for loop to print each element.

public class Main {

   public static void main(String[] args)

   {

​

       // initializing array

       int[] arr = { 1, 2, 3, 4, 5 };

​

       // size of array

       int n = arr.length;

​

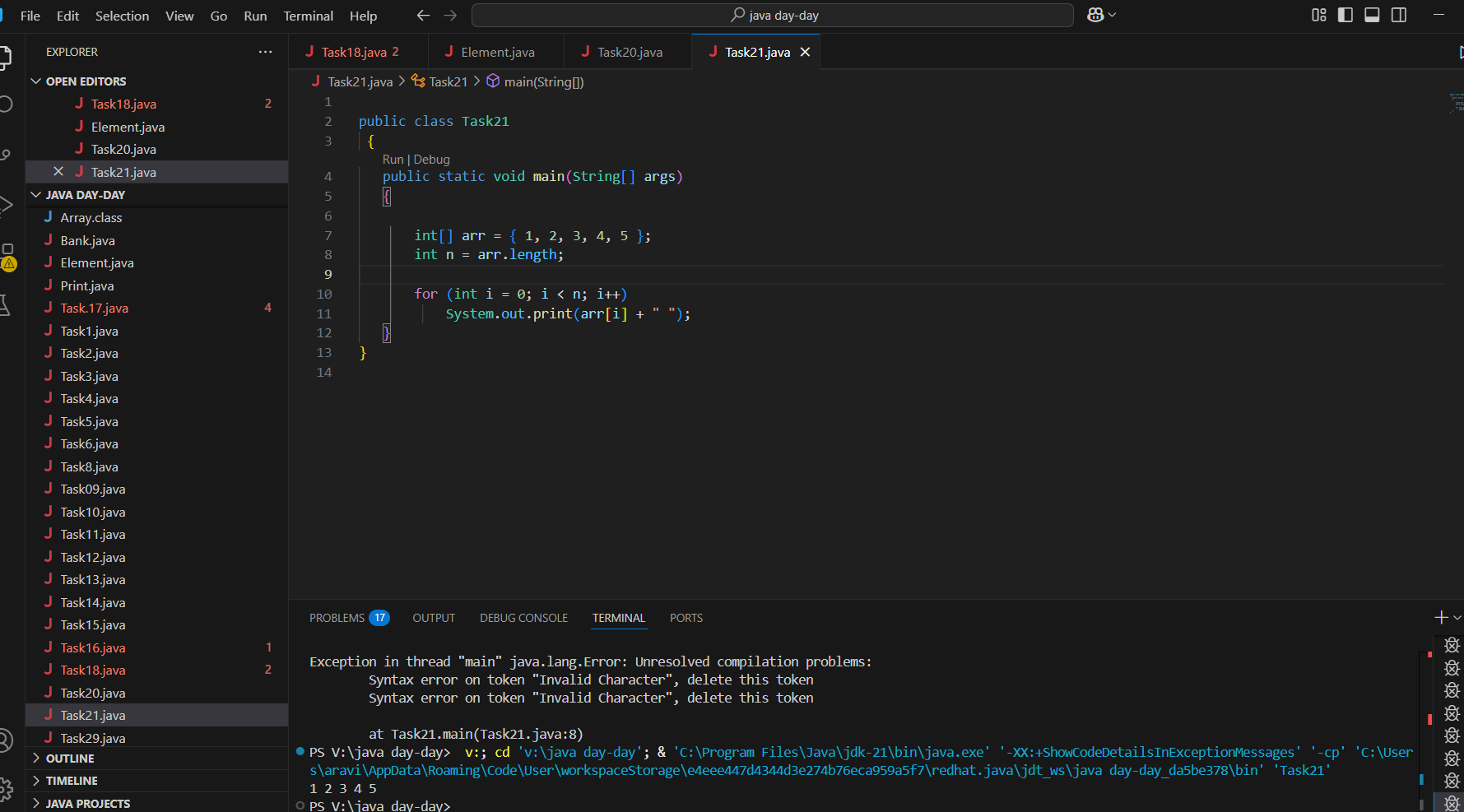
       // traversing array

       for (int i = 0; i < n; i++)

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

   }

}

  
Task 22:

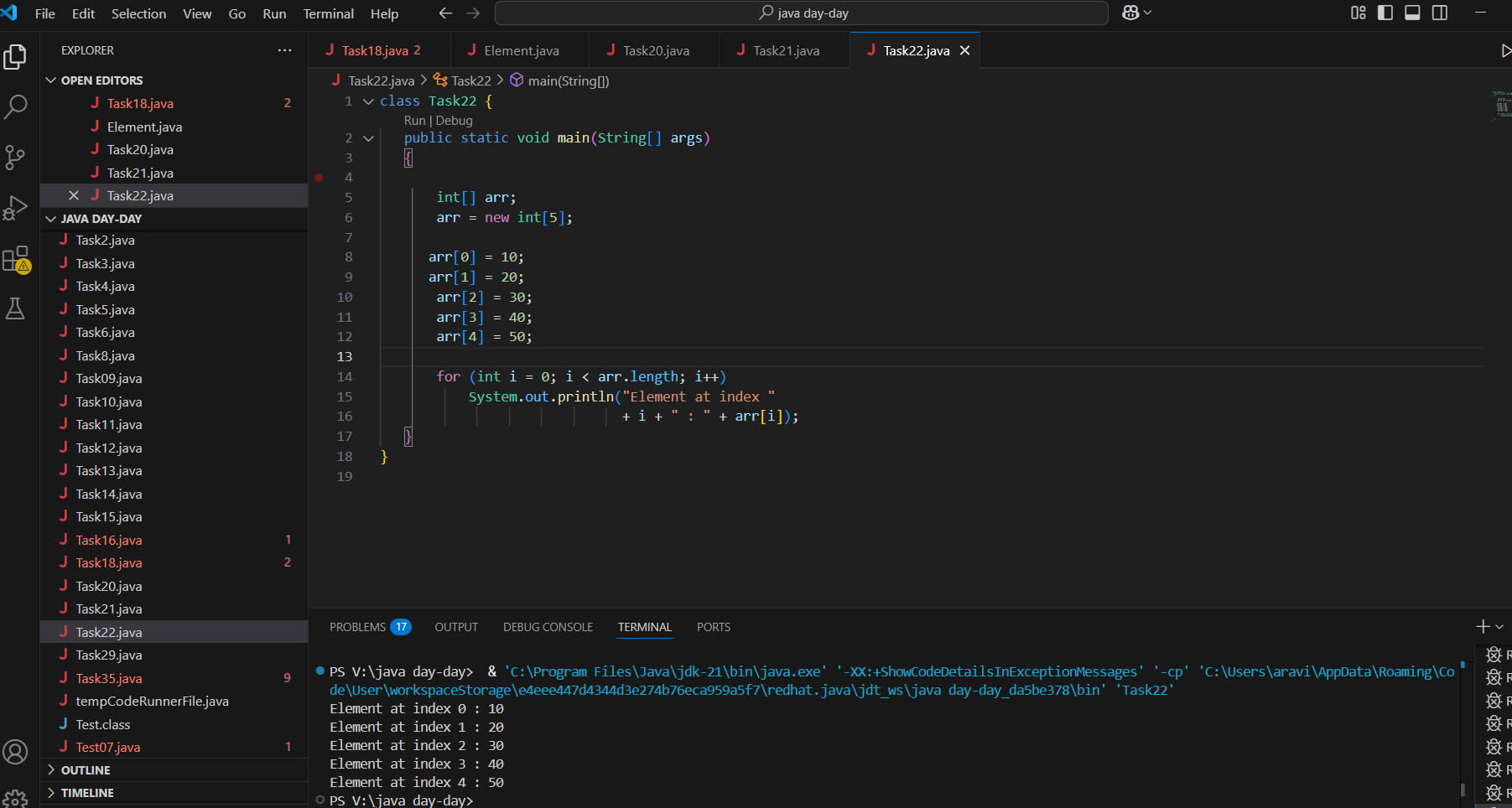
*Task 022 - home task*

**Implementation:**

// Java program to illustrate creating an array

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

// and prints each value to standard output.



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;

   }

// destructors in c++... which is **not** in Java… in java we gave GC (garbage collectors)

~Student(){

roll \_no =0;

name = “”;

}

}

​

public class Main {

   public static void main(String[] args){

Student sobj1 = new Student(); // **default** constructor is called automatically when object is initialised.

Student sobj2 = new Student();

Student sobj3 = new Student();

       // 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++)

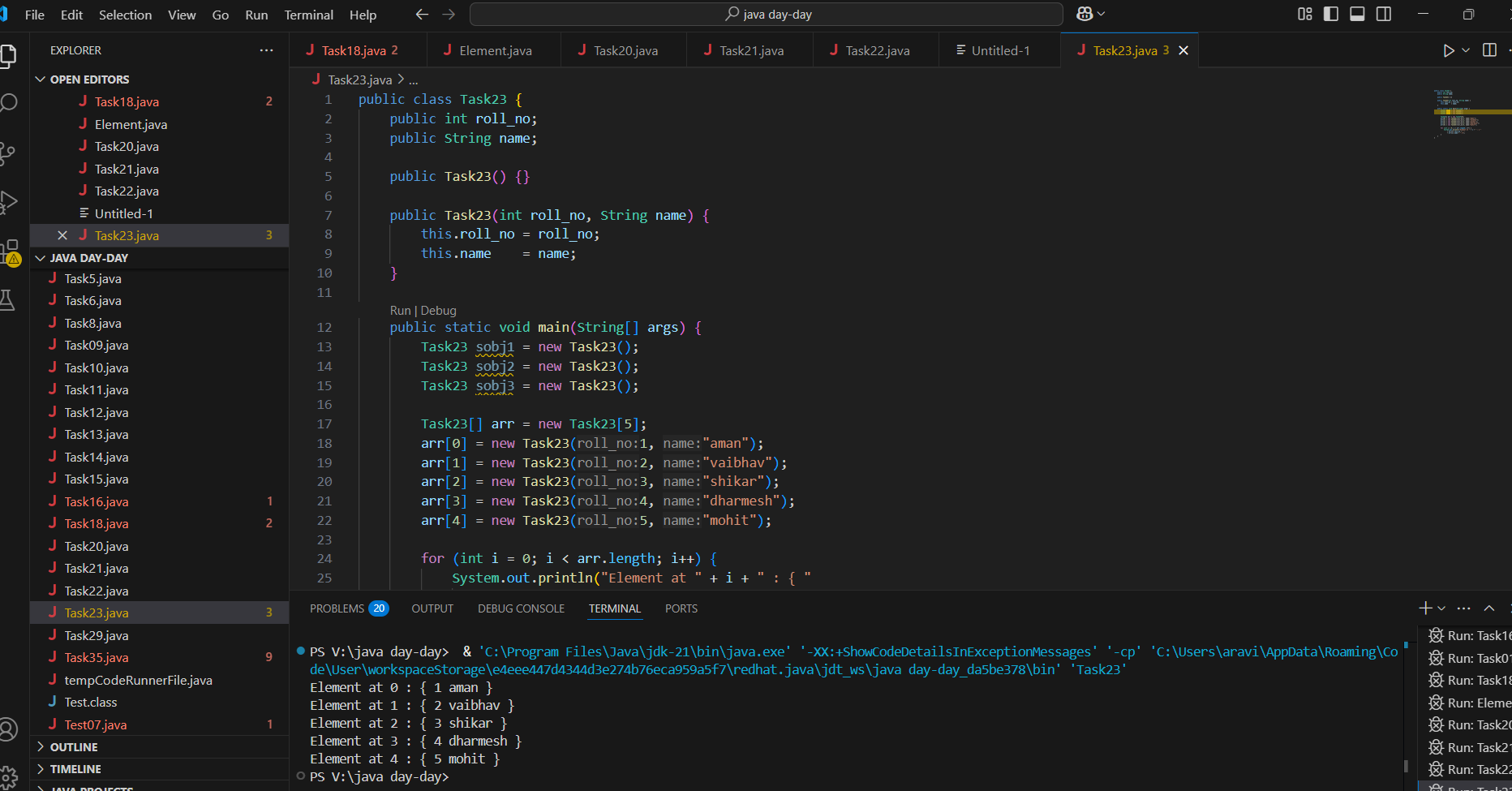
           System.out.println("Element at " + i + " : { "

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

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

   }

}



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

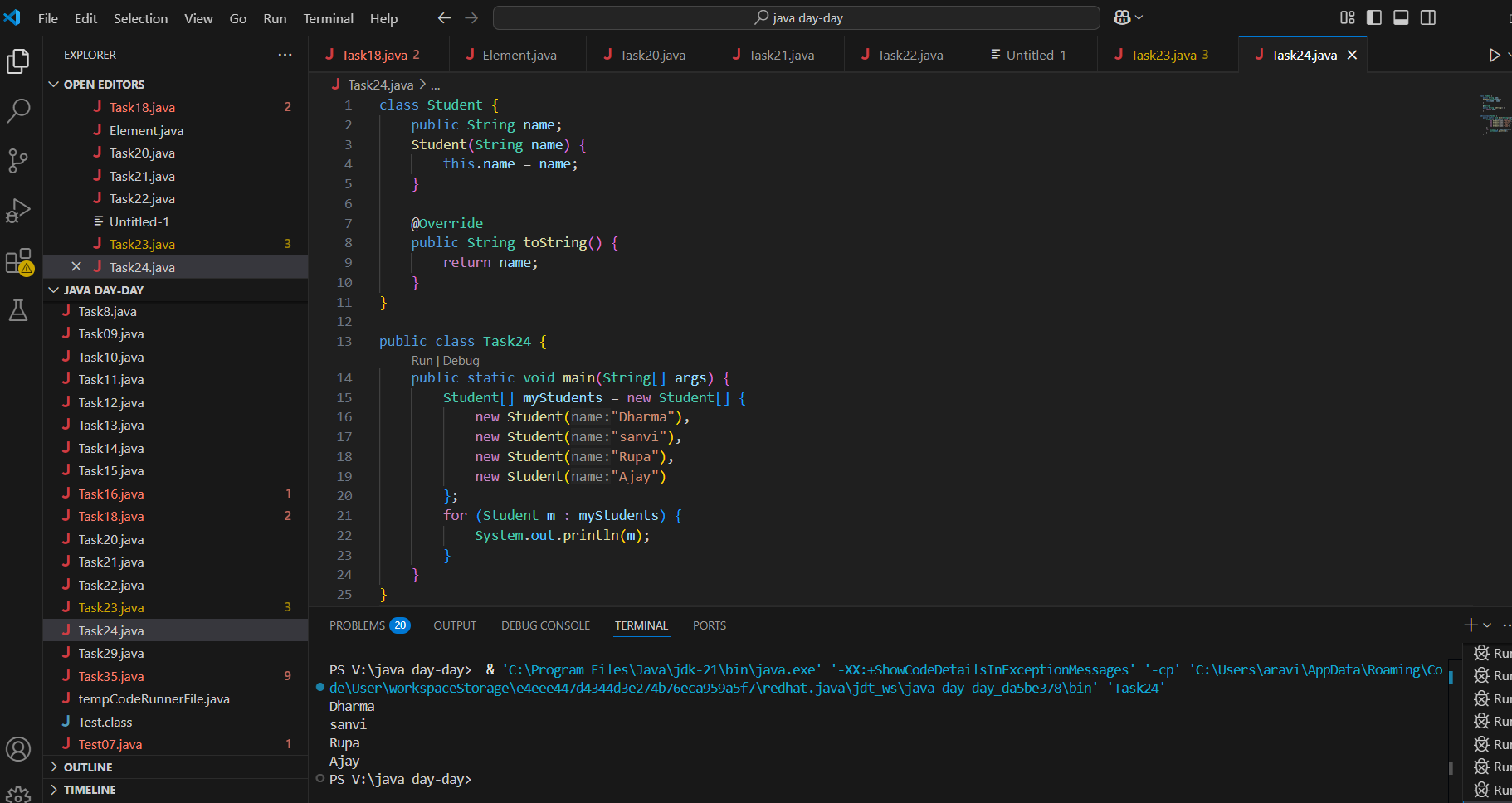
       for(Student m:myStudents){

           System.out.println(m);

       }

   }

}



**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;

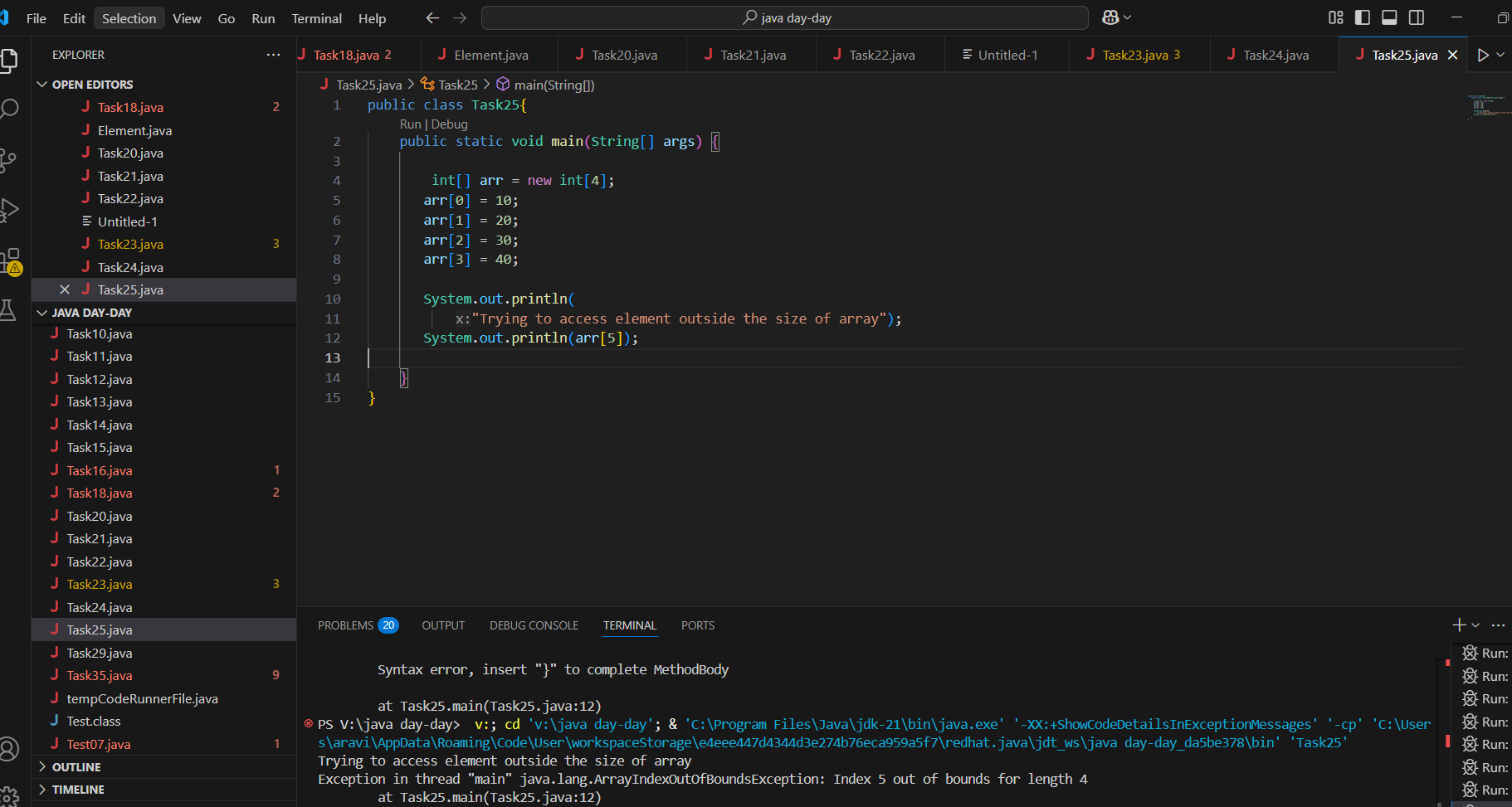
       arr[3] = 40;

​

       System.out.println(

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

       System.out.println(arr[5]);



Task 026  - 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);

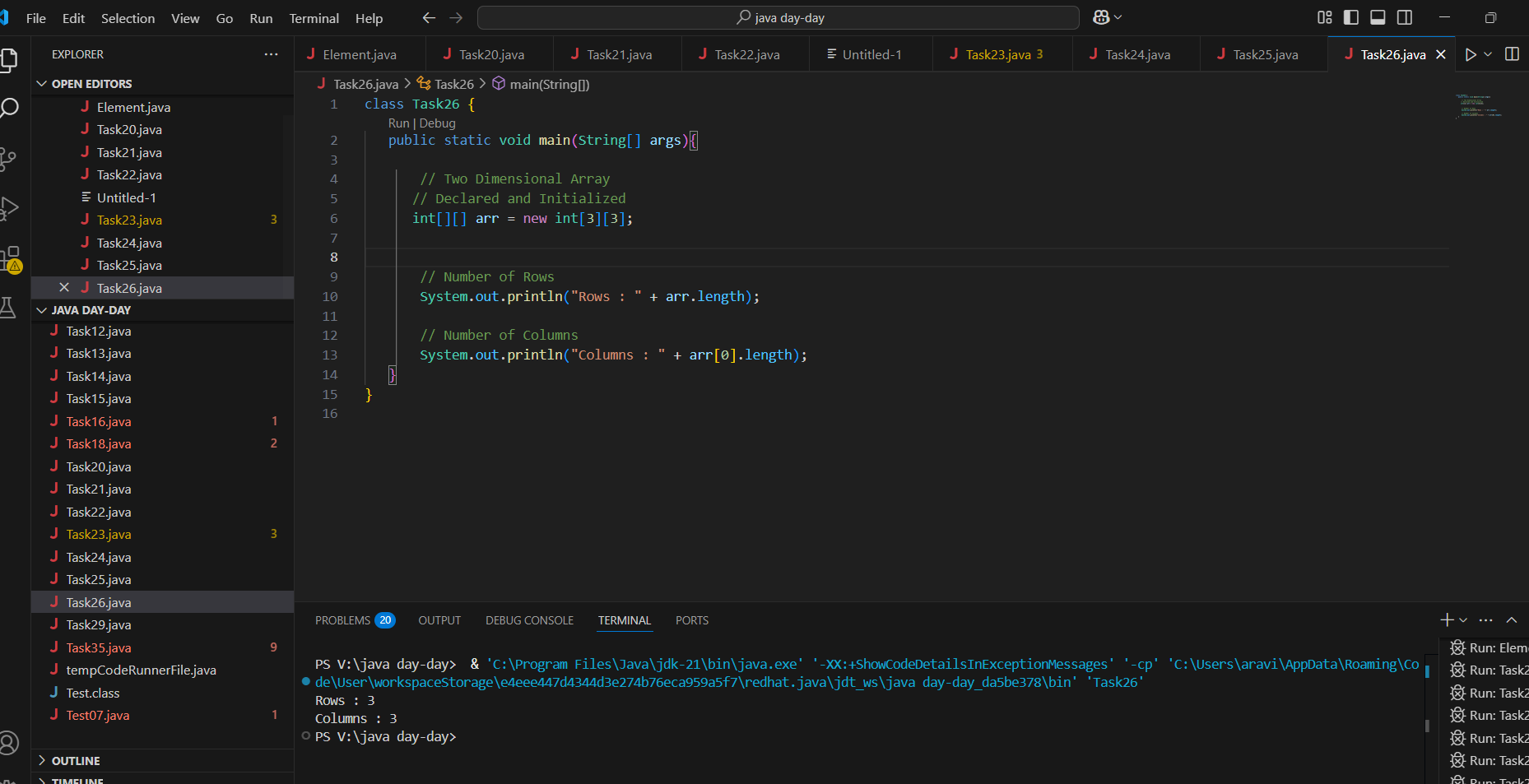
   }

}

**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++) { // rows

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

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

​

           System.out.println();

       }

   }

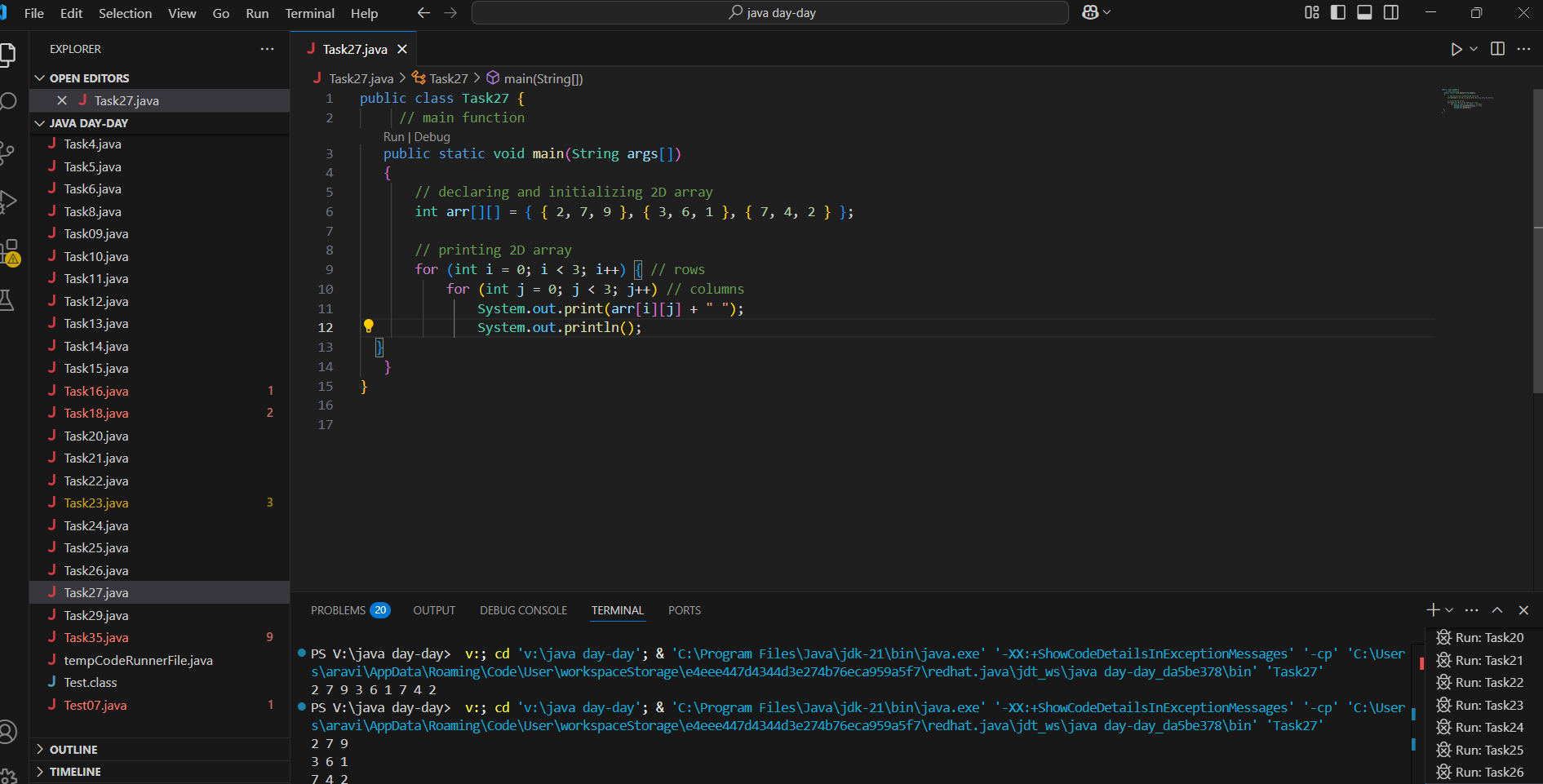
}

**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];

​

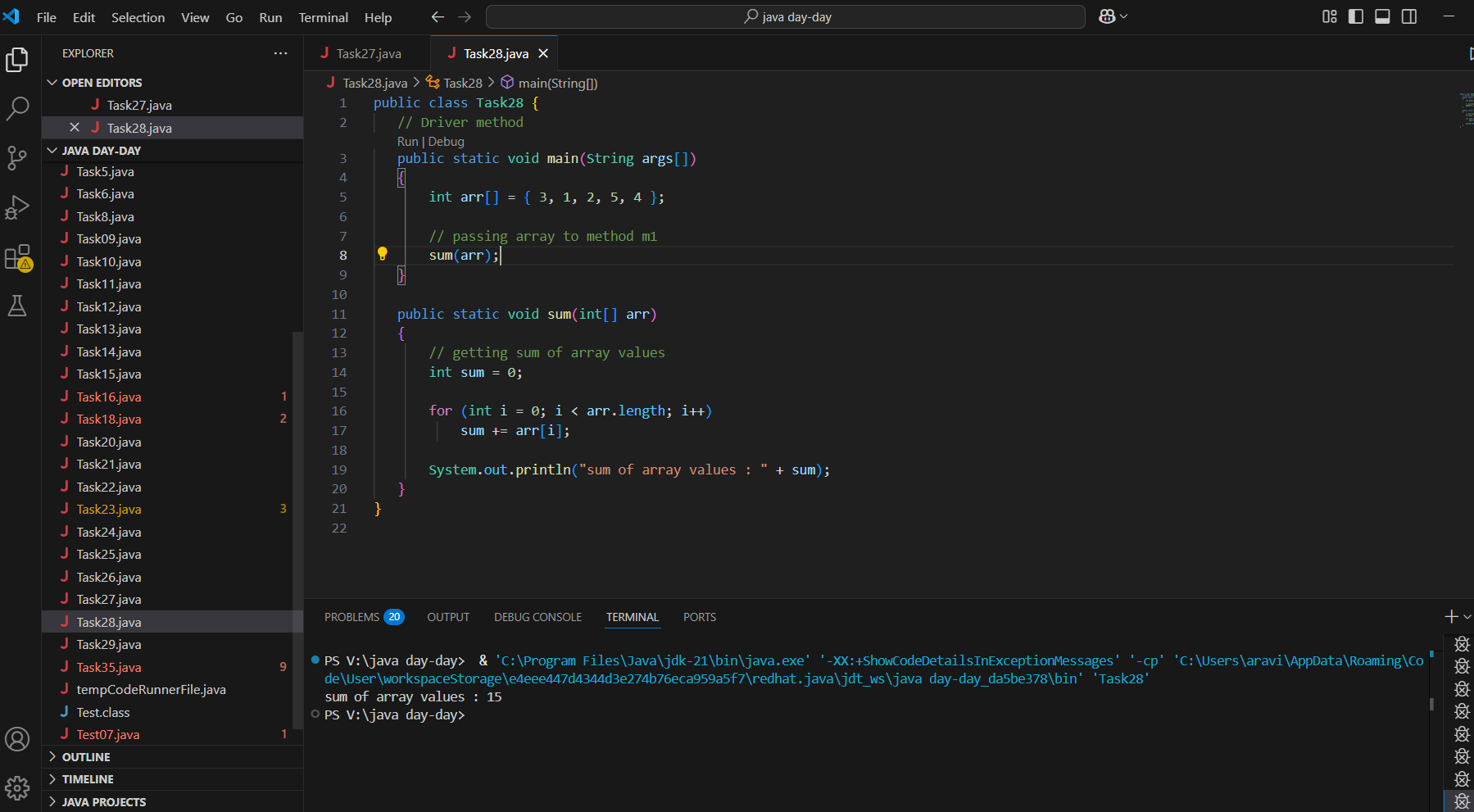
       System.out.println("sum of array values : " + sum);

   }

}

**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

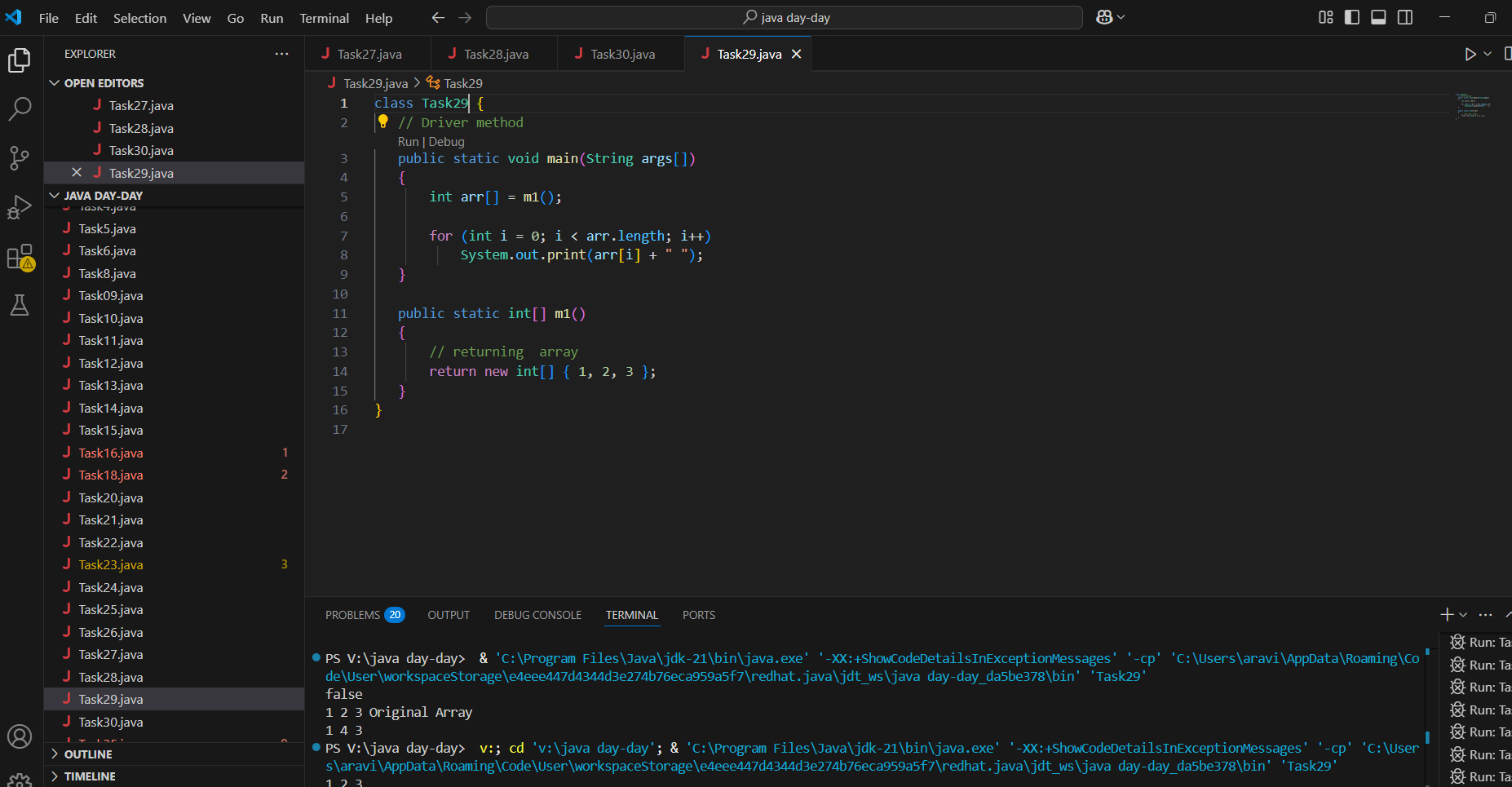
       return new int[] { 1, 2, 3 };

   }

}

**Output**

1 2 3



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);

​

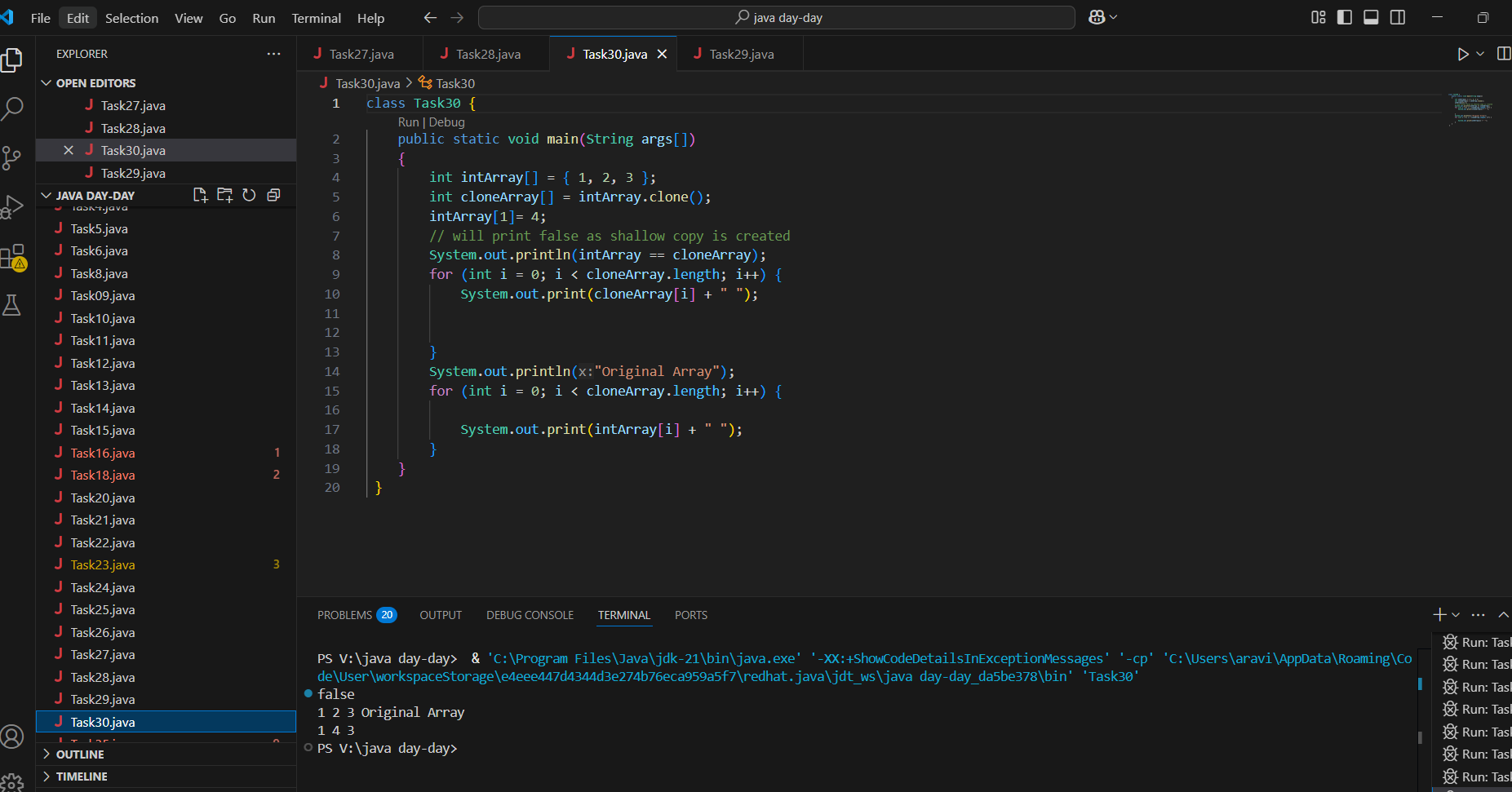
       for (int i = 0; i < cloneArray.length; i++) {

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

       }

   }

}



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]);

   }

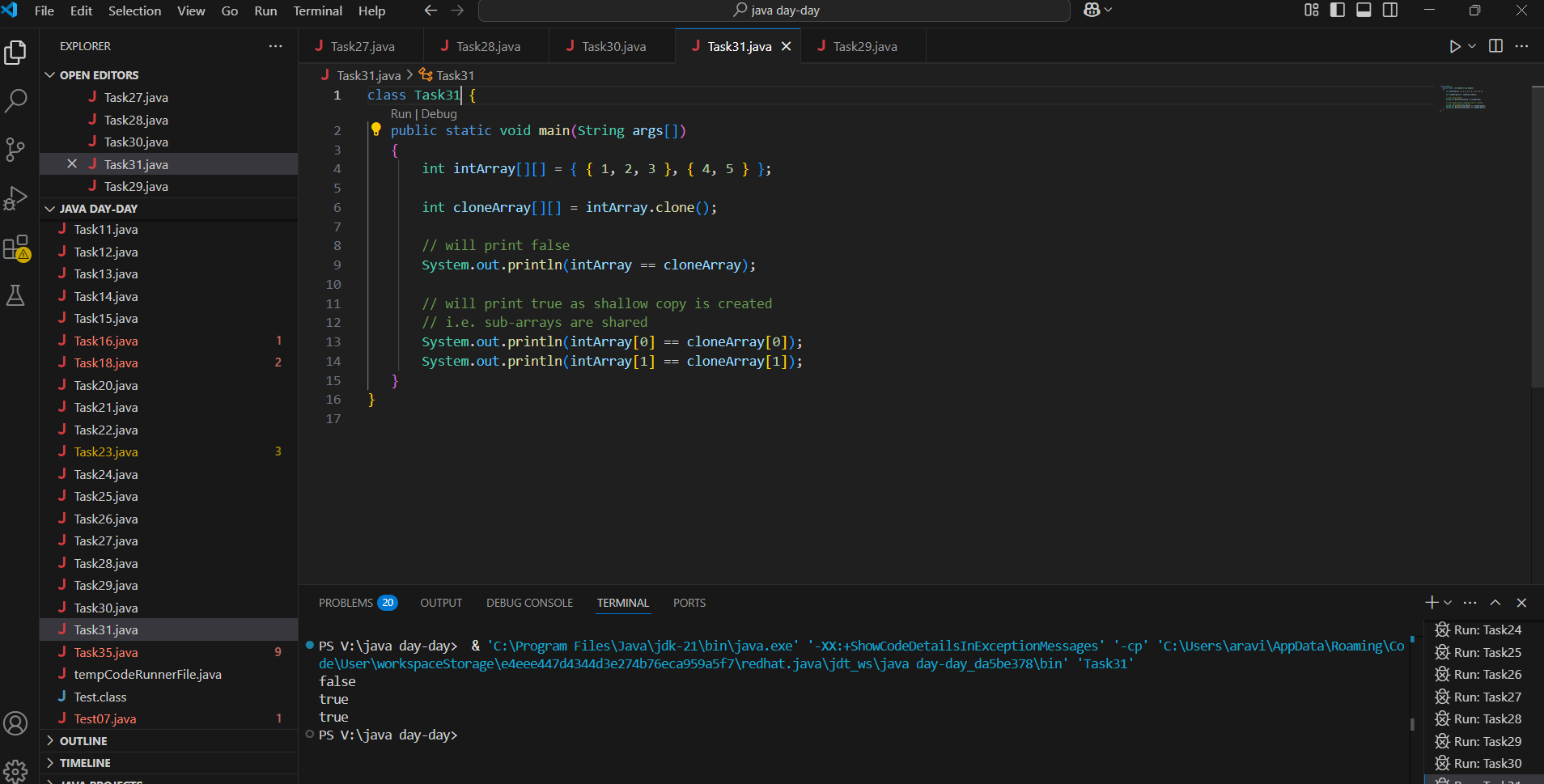
}

**Output**

false

true

tru



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

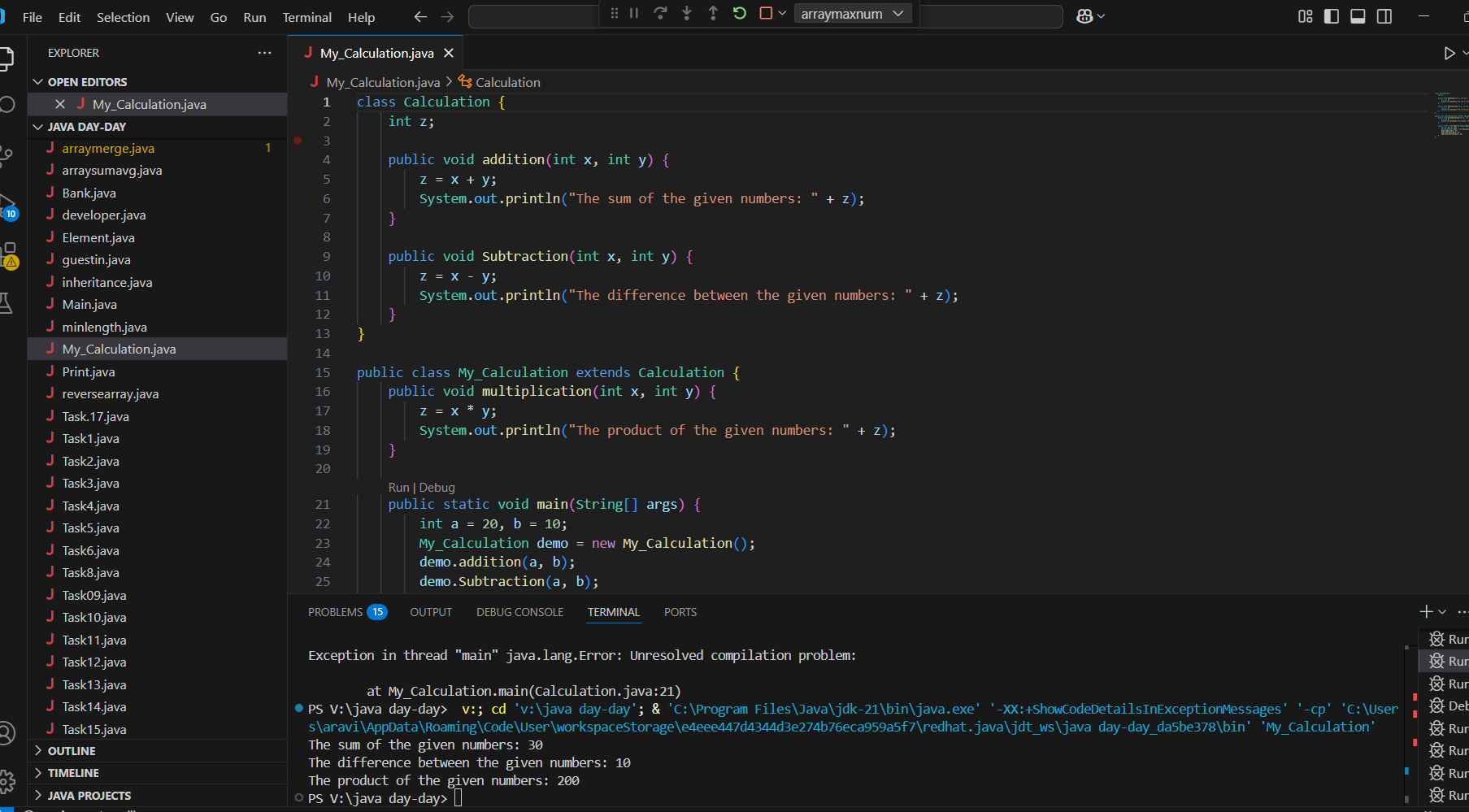
It avoids - duplication

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

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

Ans:NO ,That’s because **Java only allows single inheritance**—a class can extends exactly one other class

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);

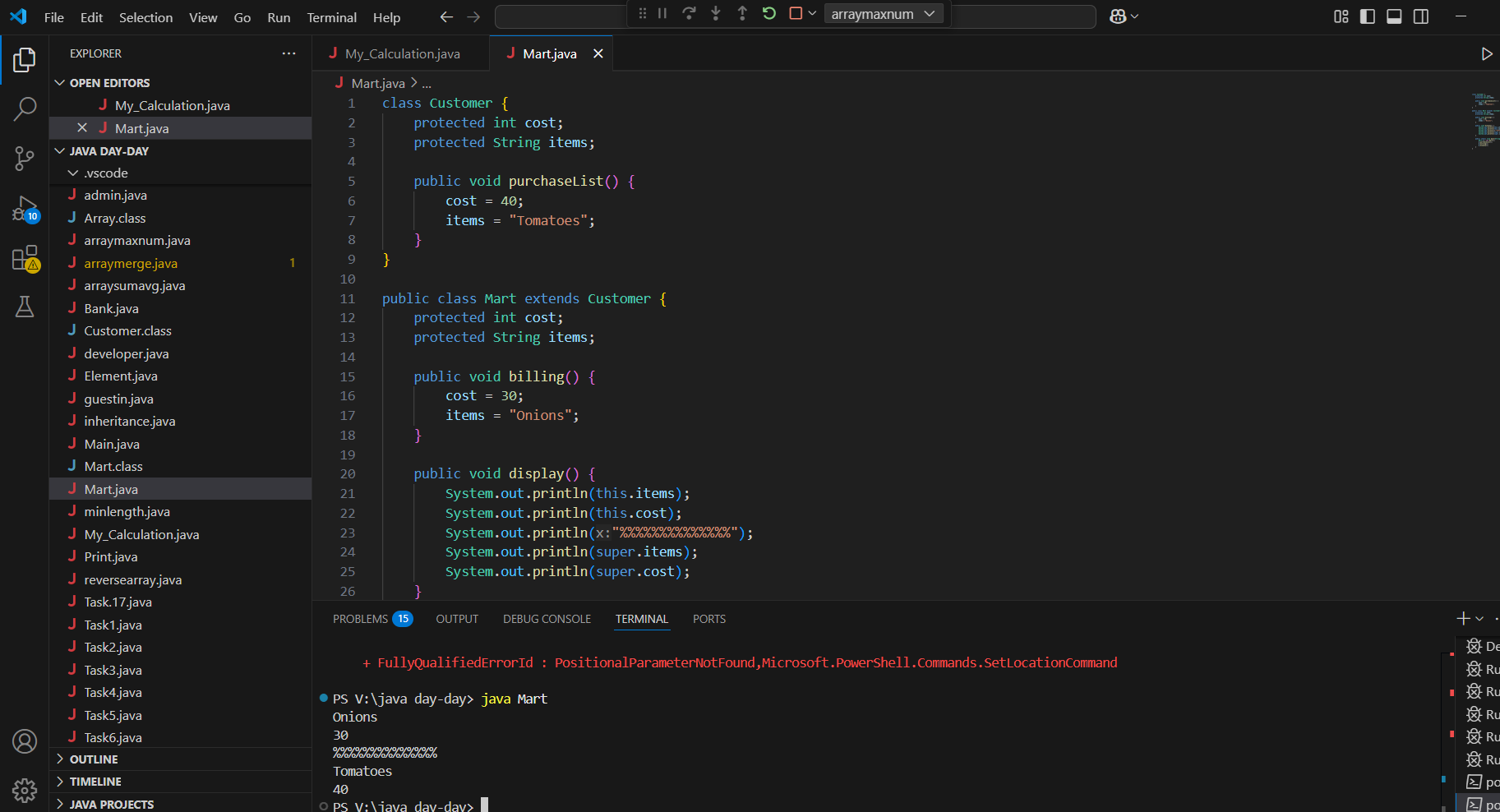
sout”(%%%%%%%%%%%%%%”);

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

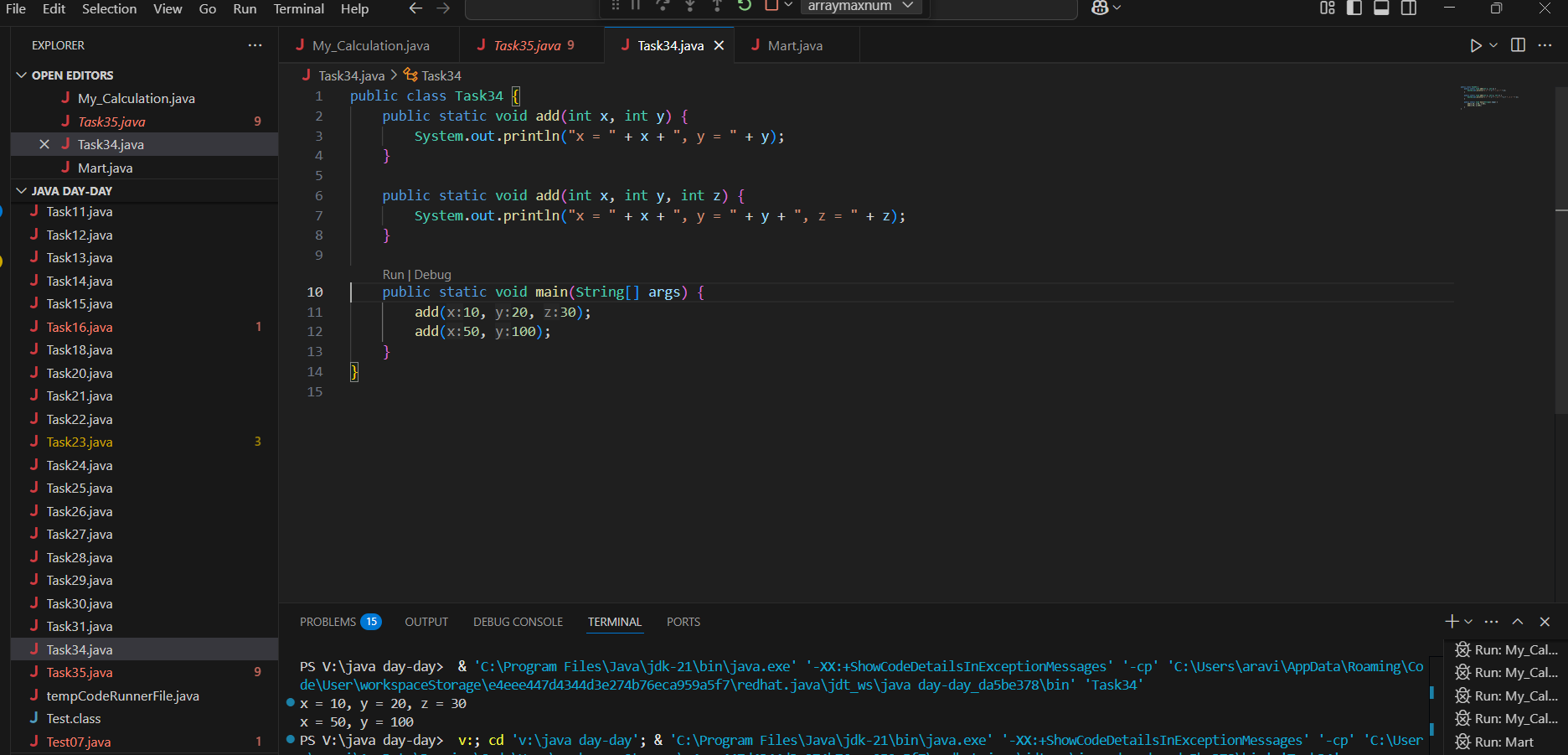
}

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

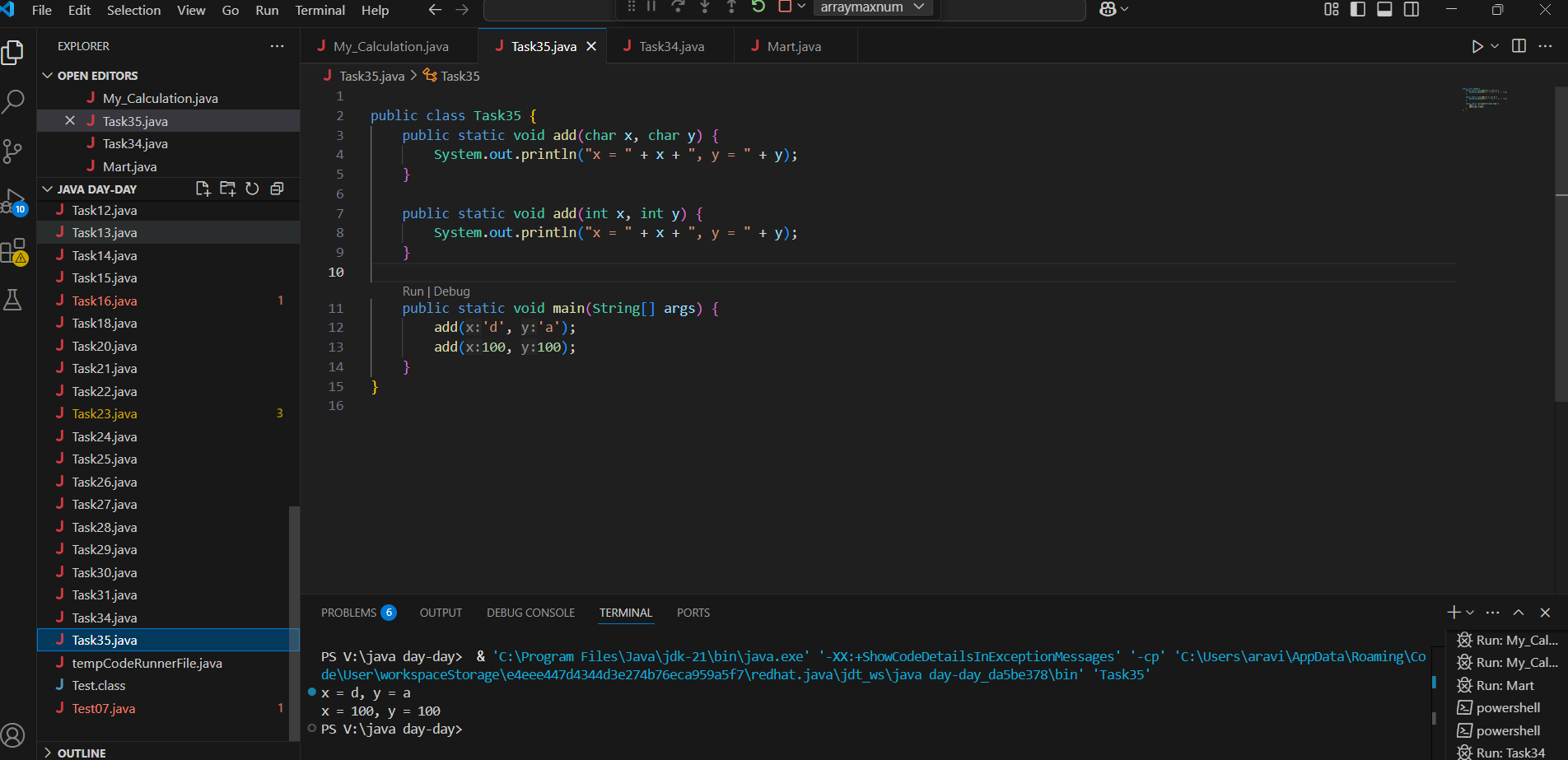
}

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

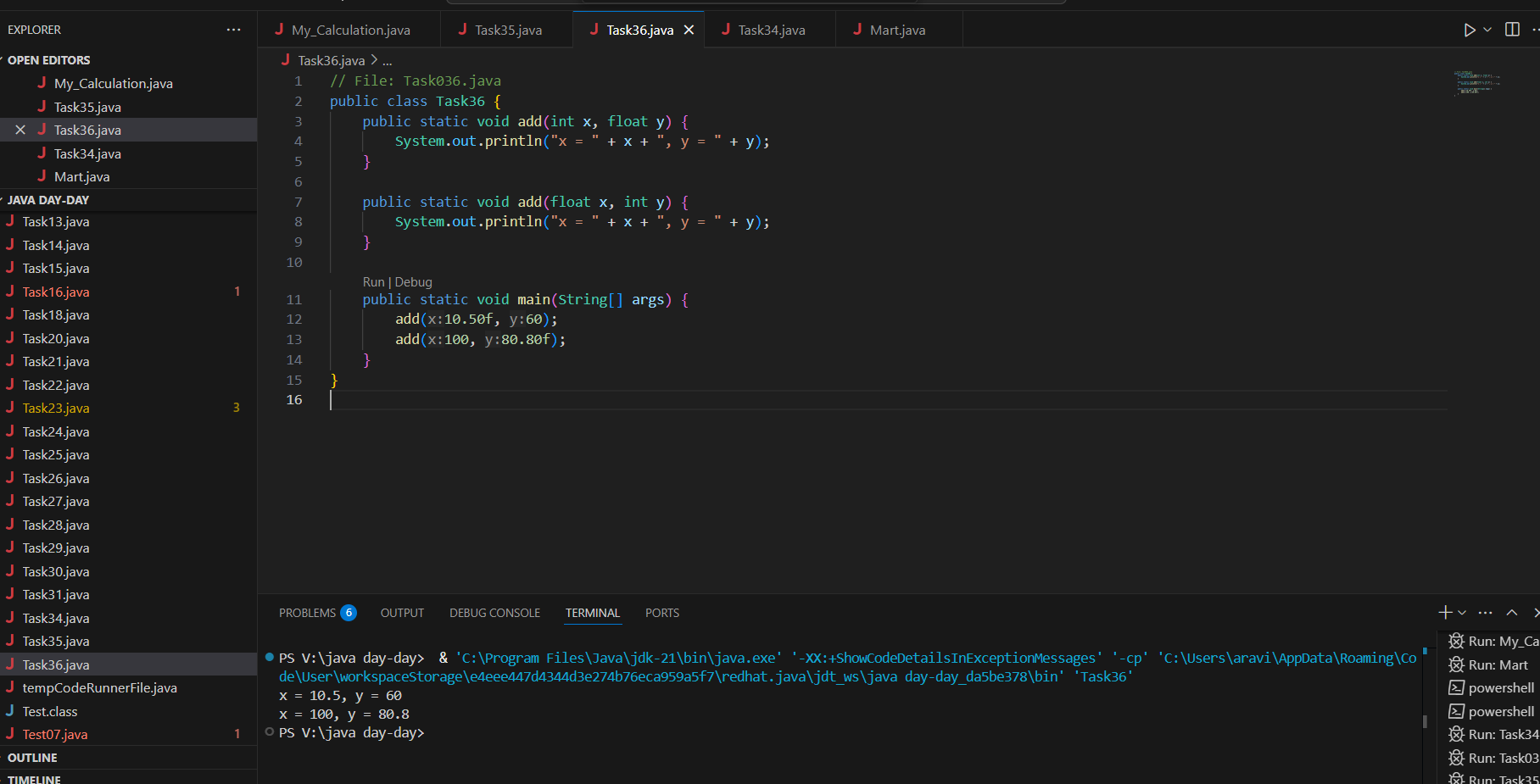
}

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; //===============>  ??????

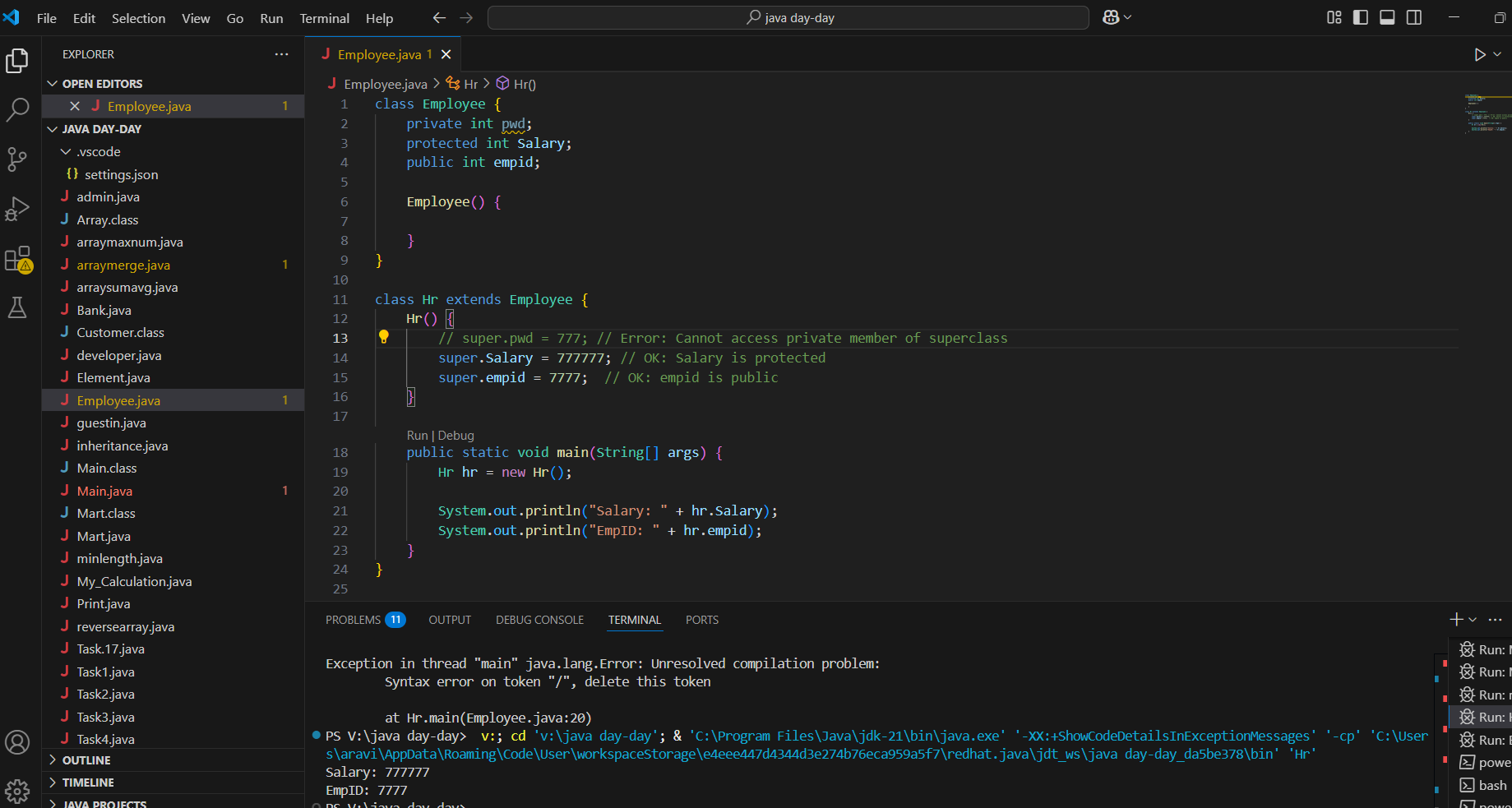
super.Salary = 50000; //==================>  ?

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() {

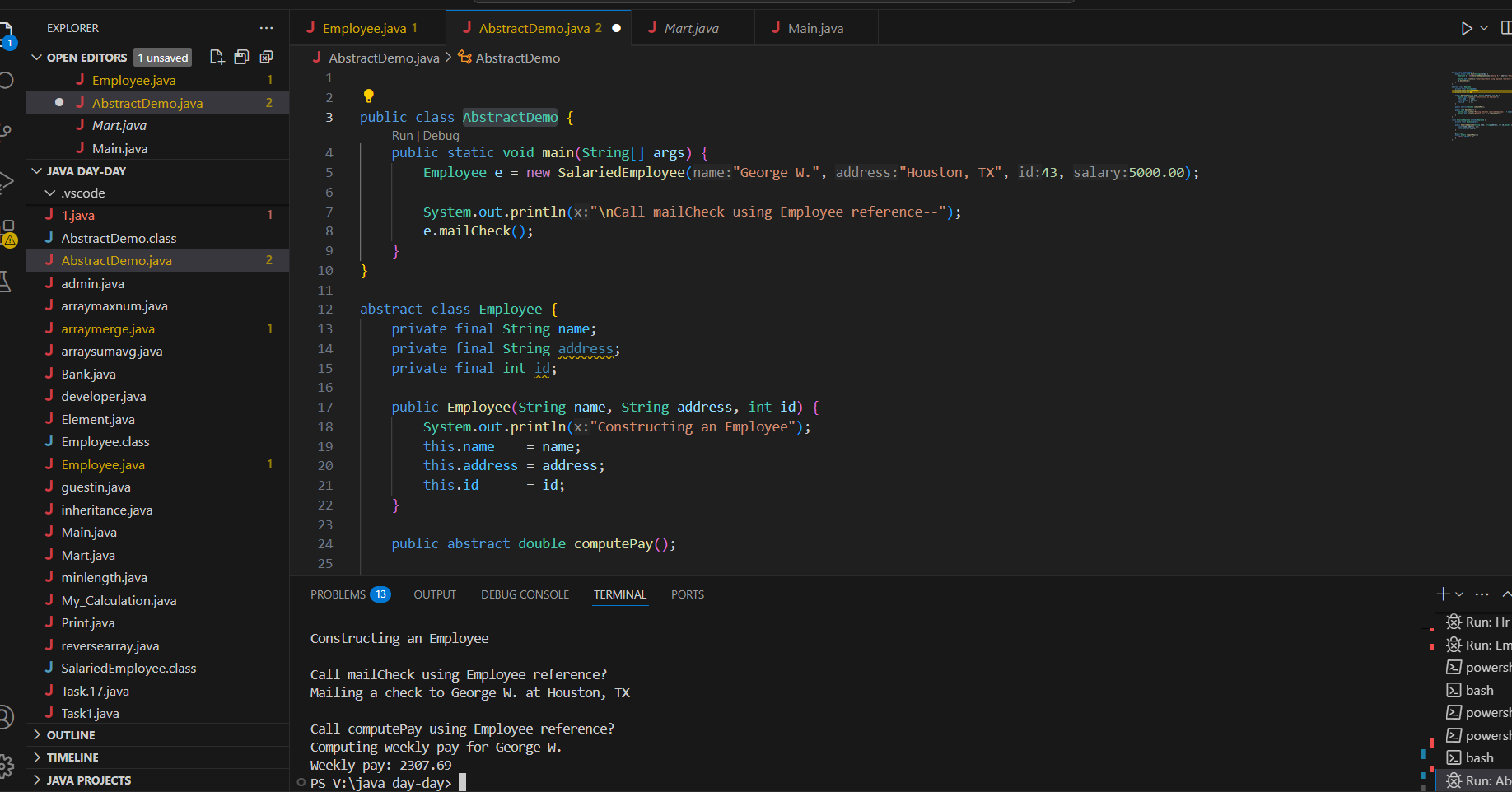
      return number;

   }

}

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();

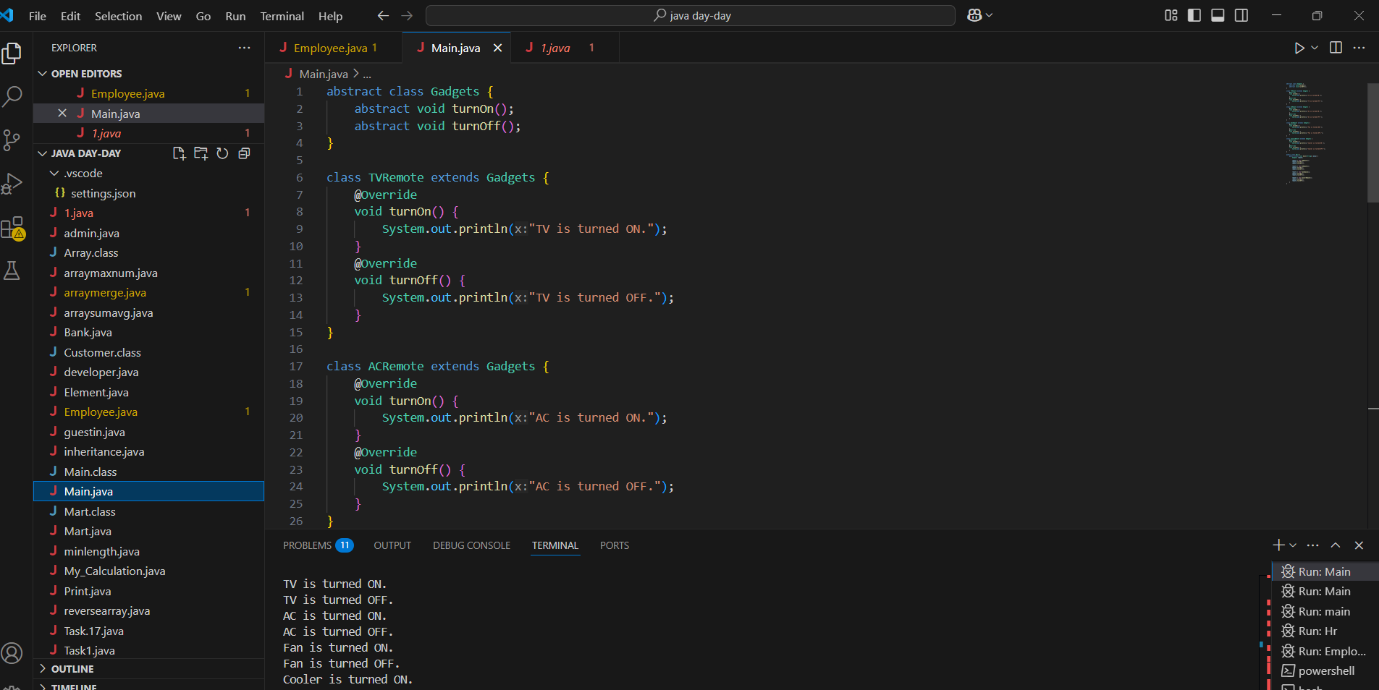
        Gadgets remote = new CoolerRemote();

        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](https://www.geeksforgeeks.org/abstraction-in-java-2).
* By default, **variables in an interface are public, static, and final.**
* It is used to achieve abstraction and [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) in Java.
* It **supports loose coupling** (classes depend on behavior, not implementation).
* In other words, i**nterfaces 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);

    }

}

Task 041:

Difference between Final and constant

In Java, the concept of a "constant" is implemented using the final keyword. While final is the tool, the term "constant" refers to the idea of an unchangeable value. Here's a breakdown:

Final Keyword

* **Purpose:**The final keyword in Java is a non-access modifier that indicates that a variable, method, or class cannot be modified after its initial definition.
* **Variables:**When applied to a variable, final makes it a constant. Its value must be assigned at the time of declaration or within the constructor for instance variables. Once assigned, the value cannot be changed.
* **Methods:**A final method cannot be overridden by subclasses, ensuring its behavior remains consistent.
* **Classes:**A final class cannot be subclassed, preventing inheritance.

Constants

* **Concept:**A constant is a variable whose value is fixed and cannot be changed after initialization. Constants are used to represent values that should not be modified during the program's execution.
* **Implementation:**In Java, constants are created using the final keyword. It's common practice to declare constants as public static final, making them accessible from anywhere and ensuring they have a single copy throughout the program.
* **Compile-Time vs. Run-Time:**
  + final variables can be initialized at compile-time or run-time.
  + When final variables are initialized with literal values or constant expressions at compile-time, they behave as true constants.
  + When final variables are initialized at run-time, they become run-time constants, meaning their value is fixed after their first assignment but not known until the program runs.

Key Differences

* **Terminology:**"Constant" is a conceptual idea, while final is the Java keyword to realize that idea.
* **Flexibility:**final variables can be initialized at run-time, allowing for values that are not known at compile time.
* **Scope:**final can be applied to variables, methods, and classes, while "constant" is primarily associated with variables.

In summary, final is the mechanism to create constants in Java. By declaring a variable as final, you ensure its value remains unchanged, promoting code clarity and preventing accidental modifications.

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