Task 1:

What do you understand by exceptions?

Exceptions are things that go wrong when a computer program is running. They are problems or errors that happen that the program didn't expect.

For example:

The user types in something the program doesn't understand

The program tries to open a file that doesn't exist

The program tries to do math that doesn't make sense, like dividing by zero

When these unexpected problems happen, the normal flow of the program gets interrupted. Instead of continuing to run like normal, the program has to deal with the exception that came up.

Exceptions are the program's way of handling these unexpected problems. They allow the program to stop what it was doing, figure out what went wrong, and then either fix the problem or tell the user what happened.

The main thing to understand about exceptions is that they give the program a structured way to deal with things going wrong, instead of the program just crashing or shutting down completely.

Task 2:

What are the categories of Exceptions do we have in Java? What are they?

Solution : The main difference is that checked exceptions are things the program has to be prepared for, while unchecked exceptions and errors are more unexpected problems the program has to deal with as best it can.

1. Checked Exceptions:

These are problems that the computer knows could happen, like trying to open a file that doesn't exist. The program has to have a plan for dealing with these kinds of exceptions.

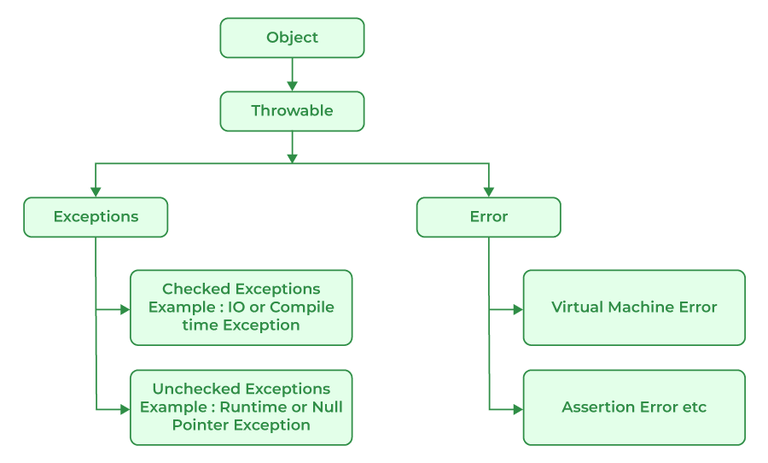
2. Unchecked Exceptions (Runtime Exceptions):

These are unexpected problems that can happen while the program is running, like trying to use a number that's too big. The program doesn't always have to have a specific plan for these.

3. Errors:

These are really serious problems that the program just can't fix, like if the computer runs out of memory. There's not much the program can do when these big errors happen.

Plz refer the below image .. for more details.



Task 3:

Can you try the below code snippet and let me know which kind of exception is this

// Java program to demonstrates handling

// the exception using try-catch block

import java.io.\*;

class Geeks {

    public static void main(String[] args)

    {

        int n = 10;

        int m = 0;

        try {

            // Code that may throw an exception

            int ans = n / m;

            System.out.println("Answer: " + ans);

        }

        catch (ArithmeticException e) {

            // Handling the exception

            System.out.println(

                "Error: Division by zero is not allowed!");

        }

        finally {

            System.out.println(

                "Program continues after handling the exception.");

        }

    }

}

Solution :

The exception in your code is an ArithmeticException.

Explanation (with reference to the image):

When you try to divide n by m (where m is 0), Java throws an ArithmeticException.

According to the image, ArithmeticException is an Unchecked Exception (also called a Runtime Exception).

Unchecked exceptions are a type of Exception that are not checked at compile time, but occur at runtime.

Summary:

This is an Unchecked Exception (Runtime Exception), specifically an ArithmeticException, which occurs when dividing by zero.

The program ArithmeticExceptionDemo.java demonstrates an ArithmeticException (an unchecked exception) in Java:

When dividing by zero, Java throws an ArithmeticException at runtime.

The catch block handles the error and prints:

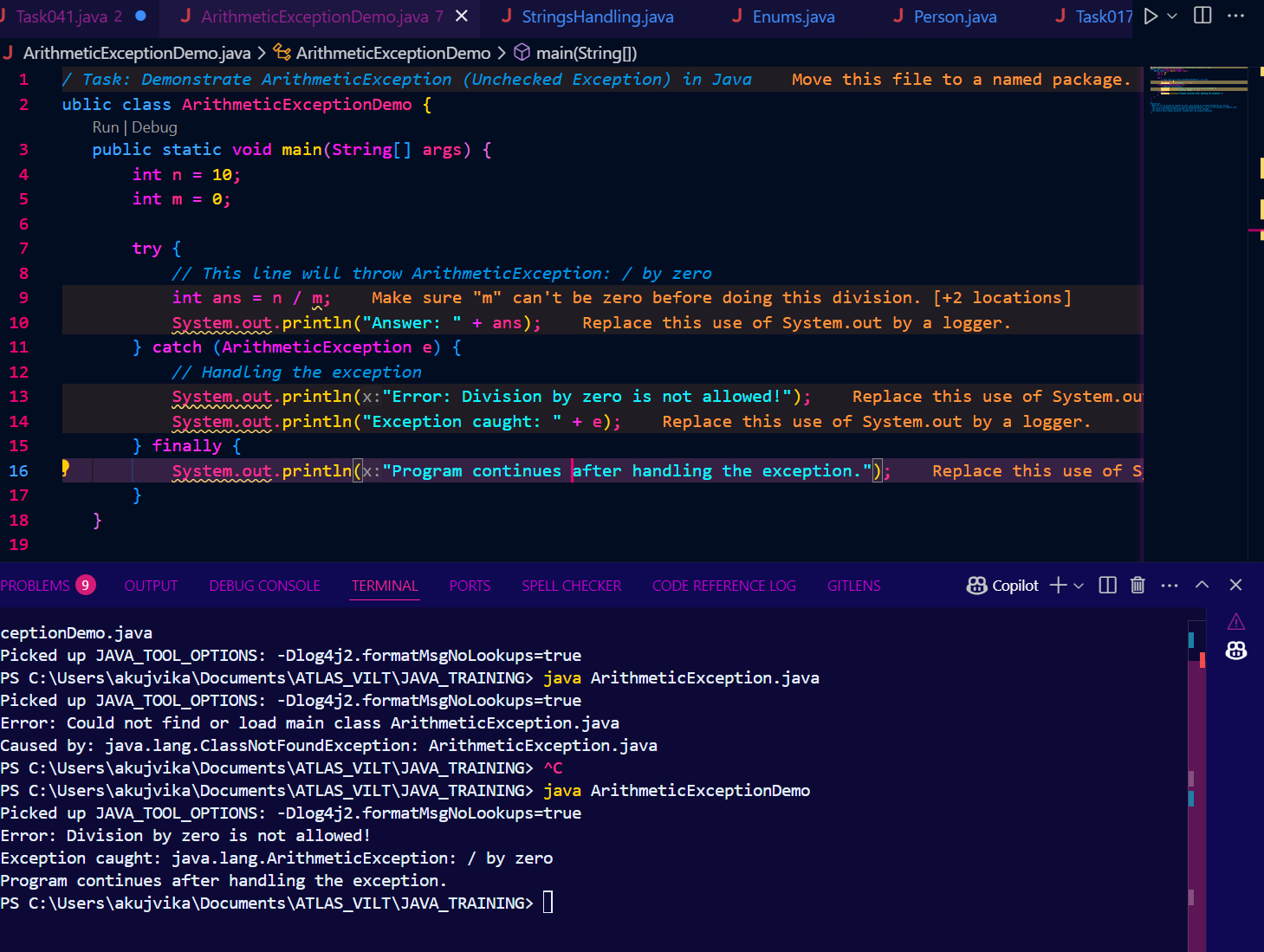
{Error: Division by zero is not allowed!

Exception caught: java.lang.ArithmeticException: / by zero}

The finally block shows the program continues:

{Program continues after handling the exception.}

This shows how Java handles runtime (unchecked) exceptions and how you can catch and manage them in your code.



Task 4: List of checked and unchecked exceptions.

Solution :

Checked Exceptions:

- FileNotFoundException - Occurs when a file cannot be found

- IOException - Occurs when there is an error reading or writing a file

- SQLException - Occurs when there is a database error

- ClassNotFoundException - Occurs when a class cannot be found

Unchecked Exceptions (Runtime Exceptions):

- NullPointerException - Occurs when trying to access a member on a null object

- ArrayIndexOutOfBoundsException - Occurs when accessing an array index that is out of bounds

- NumberFormatException - Occurs when a string cannot be parsed as a number

- ArithmeticException - Occurs when there is an error in a mathematical operation, like division by zero

Errors:

- OutOfMemoryError - Occurs when the Java Virtual Machine has run out of memory

- StackOverflowError - Occurs when the system runs out of stack space

- UnknownError - Represents unknown errors

The key difference is that checked exceptions have to be explicitly handled or declared, while unchecked exceptions and errors do not have to be explicitly handled by the programmer.

Task 5:

Try with Multiple catch blocks  …. Execute the below code snippet n display the out .. along with reason..

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         int b = 0;

         int c = 1/b;

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArrayIndexOutOfBoundsException e) {

         System.out.println("ArrayIndexOutOfBoundsException thrown  :" + e);

      }catch (Exception e) {

          System.out.println("Exception thrown  :" + e);

      }

      System.out.println("Out of the block");

   }

}

Solution :

Output given :

Exception thrown :java.lang.ArithmeticException: / by zero

Out of the block

Reason:

The code tries to divide by zero (int c = 1/b;), which throws an ArithmeticException.

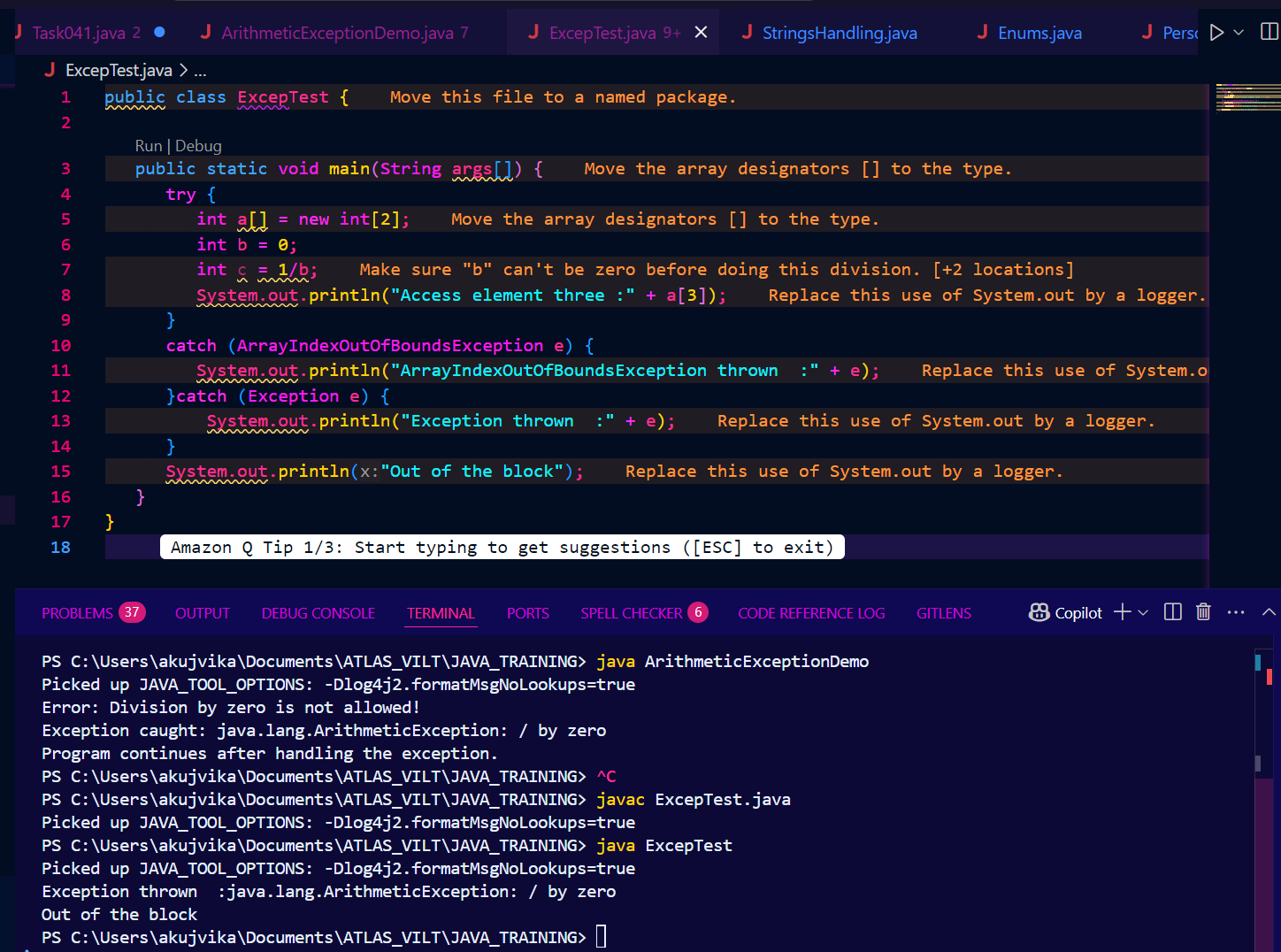
The first catch block is for ArrayIndexOutOfBoundsException, which does not match.

The second catch block is for the general Exception, which catches ArithmeticException (since it is a subclass of Exception).

The program prints the exception message and then "Out of the block".

Summary:

The code demonstrates how multiple catch blocks work. The most specific catch block is checked first, and if it doesn't match, the more general one is used. Here, the general Exception catch block handles the division by zero error.



Task 5.2 :

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         int b = 6;

         int c = 1/b;

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArrayIndexOutOfBoundsException e) {

         System.out.println("ArrayIndexOutOfBoundsException thrown  :" + e);

      }catch (Exception e) {

          System.out.println("Exception thrown  :" + e);

      }

      System.out.println("Out of the block");

   }

}

Output :

ArrayIndexOutOfBoundsException thrown :java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 2

Out of the block

Explanation:

With b = 6, int c = 1/b; does not cause an error (result is 0).

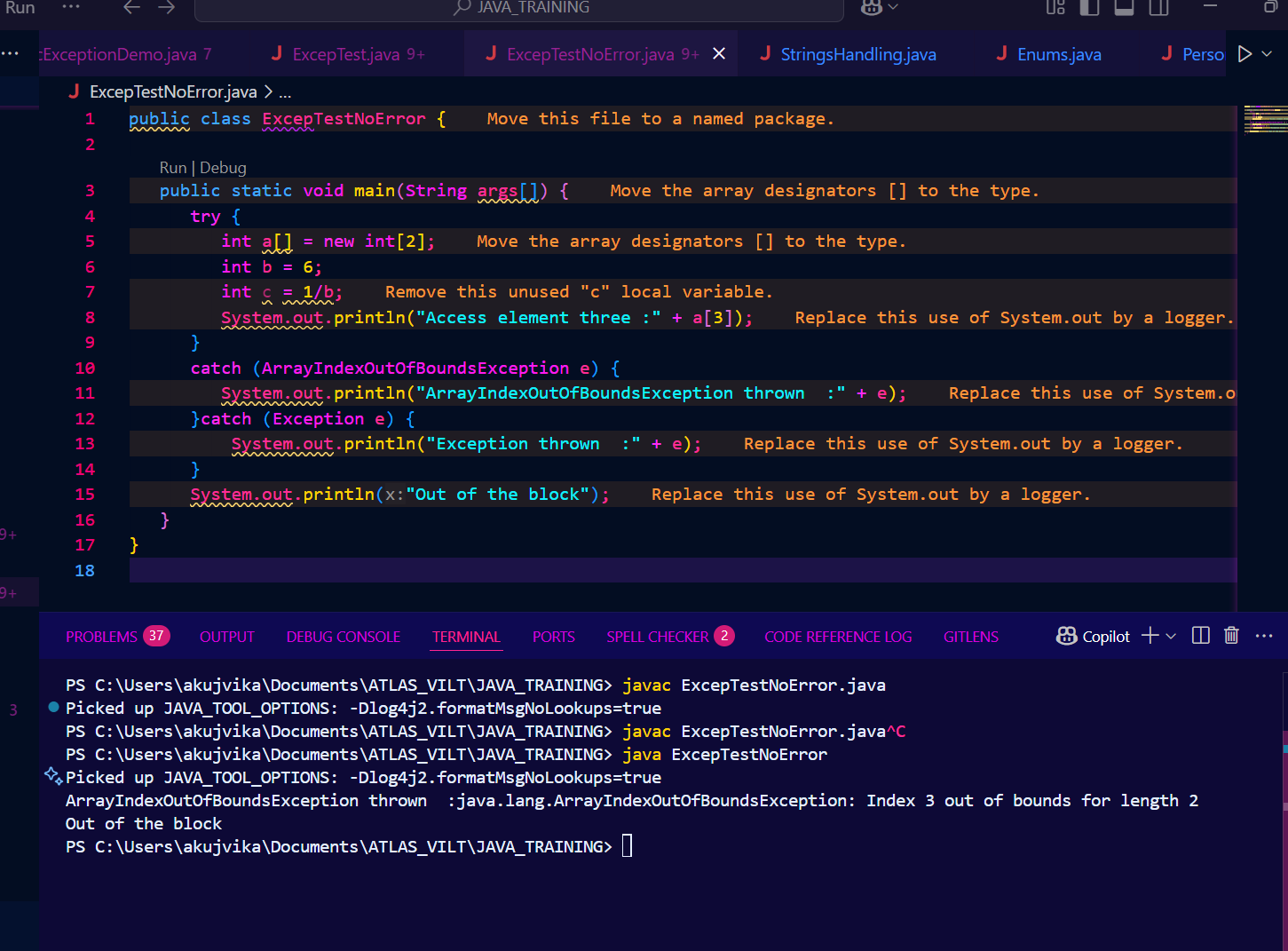
The next line tries to access a[3], but the array a only has indices 0 and 1.

This throws an ArrayIndexOutOfBoundsException, which is caught by the first catch block.

The program prints the exception message and then "Out of the block".

Summary:

When b = 6, the division is fine, but accessing an invalid array index causes an ArrayIndexOutOfBoundsException, which is handled by the corresponding catch block.



Task 6:

What is the output of the below code… give your reason for the output

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         int b = 0;

         int c = 1/b;

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArithmeticException e) {

         System.out.println("ArithmeticException thrown  :" + e);

      }

      catch (ArrayIndexOutOfBoundsException e) {

         System.out.println("ArrayIndexOutOfBoundsException thrown  :" + e);

      }catch (Exception e) {

          System.out.println("Exception thrown  :" + e);

      }

      System.out.println("Out of the block");

   }

}

Solution :

Reason:

The code tries to divide by zero (int c = 1/b;), which throws an ArithmeticException.

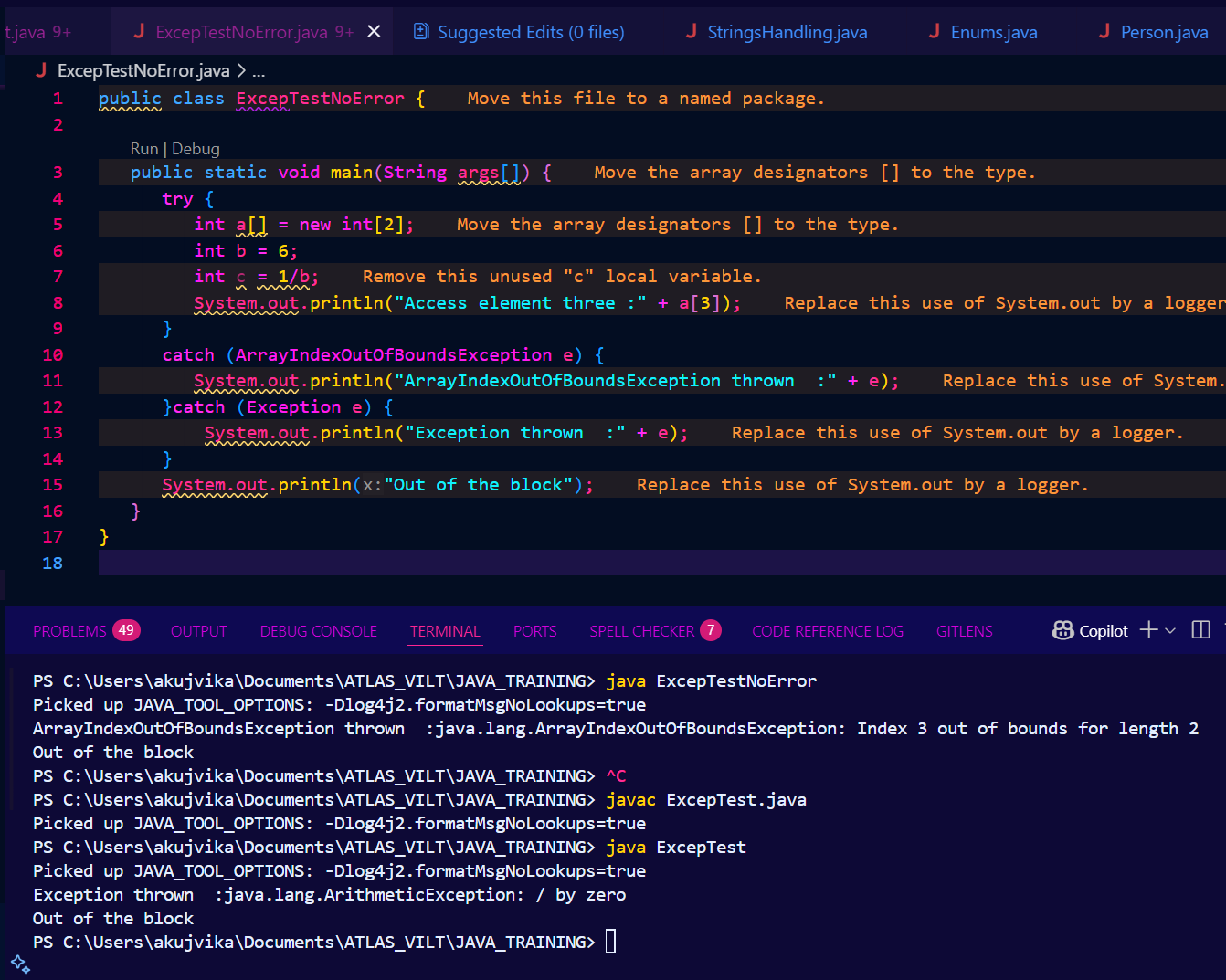
The first catch block for ArithmeticException matches and handles the exception, printing the message.

The line System.out.println("Access element three :" + a[3]); is never executed because the exception occurs before it.

The program then prints "Out of the block" after the try-catch block.

Summary:

The output is due to the division by zero, which is caught by the ArithmeticException catch block. The array access line is never reached.



Task 7:

In the below code we are having use multiple catch in a single statement: find the output and try to understand the code..

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

int b = 0;

int c = 1/b;

System.out.println("Access element three :" + a[3]);

}

catch (ArrayIndexOutOfBoundsException | ArithmeticException e) {

System.out.println("Exception thrown :" + e);

}

System.out.println("Out of the block");

}

}

Solution :

Output : Exception thrown :java.lang.ArithmeticException: / by zero

Out of the block

Explanation:

The code tries to divide by zero (int c = 1/b;), which throws an ArithmeticException.

The catch block uses multi-catch: catch (ArrayIndexOutOfBoundsException | ArithmeticException e), so it can catch either exception type.

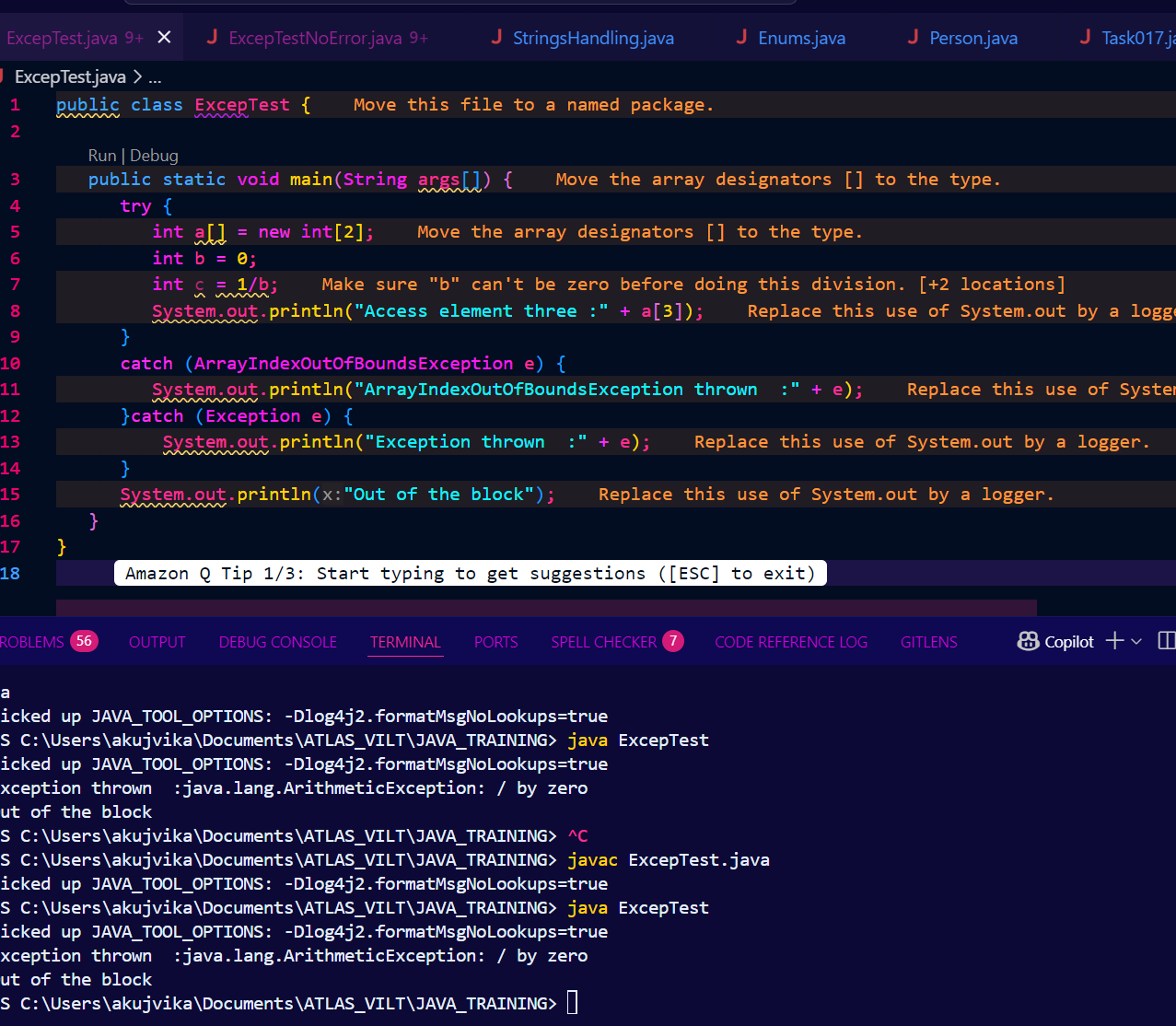
Since an ArithmeticException occurs, it is caught and the message is printed.

The line accessing a[3] is never executed because the exception occurs before it.

The program then prints "Out of the block".

Summary:

This code demonstrates Java's multi-catch feature, allowing you to catch multiple exception types in a single catch block. The first exception (ArithmeticException) is caught and handled, and the program continues.



Nested try blocks :

Task 008:

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

try {

int b = 0;

int c = 1/b;

}catch(Exception e) {

System.out.println("Exception thrown: " + e);

}

System.out.println("Access element three :" + a[3]);

}

catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Exception thrown: " + e);

}

System.out.println("Out of the block");

}

}

Solution :

Output : Exception thrown: java.lang.ArithmeticException: / by zero

Exception thrown: java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 2

Out of the block

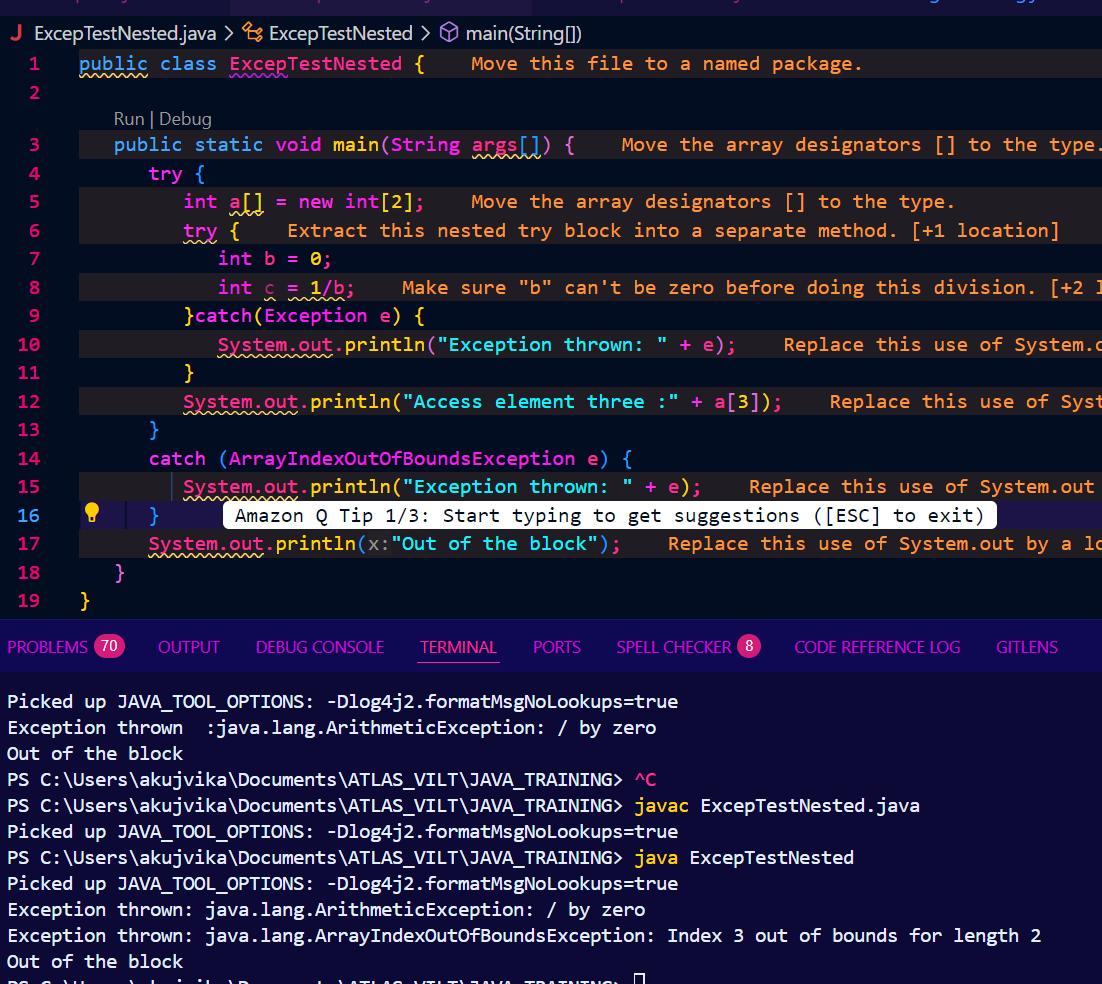
Explanation:

The inner try block throws an ArithmeticException (division by zero), which is caught and printed.

After the inner try-catch, the code tries to access a[3], which is out of bounds for the array (size 2), so an ArrayIndexOutOfBoundsException is thrown and caught by the outer catch block.

The program then prints "Out of the block".

This demonstrates how nested try-catch blocks work: each exception is caught by the nearest matching catch block.



Throw and Throws:

Task 009

// Demonstrating how to throw an exception

class MyClass {

static void fun() throws IllegalAccessException

{

System.out.println("Inside fun(). ");

throw new IllegalAccessException("demo");

}

public static void main(String args[])

{

try {

fun();

//method2(); → arrayindex…

//Method3() —> file not found….

}

catch (IllegalAccessException e) {

System.out.println("Caught in main.");

}

}

}

Solution :

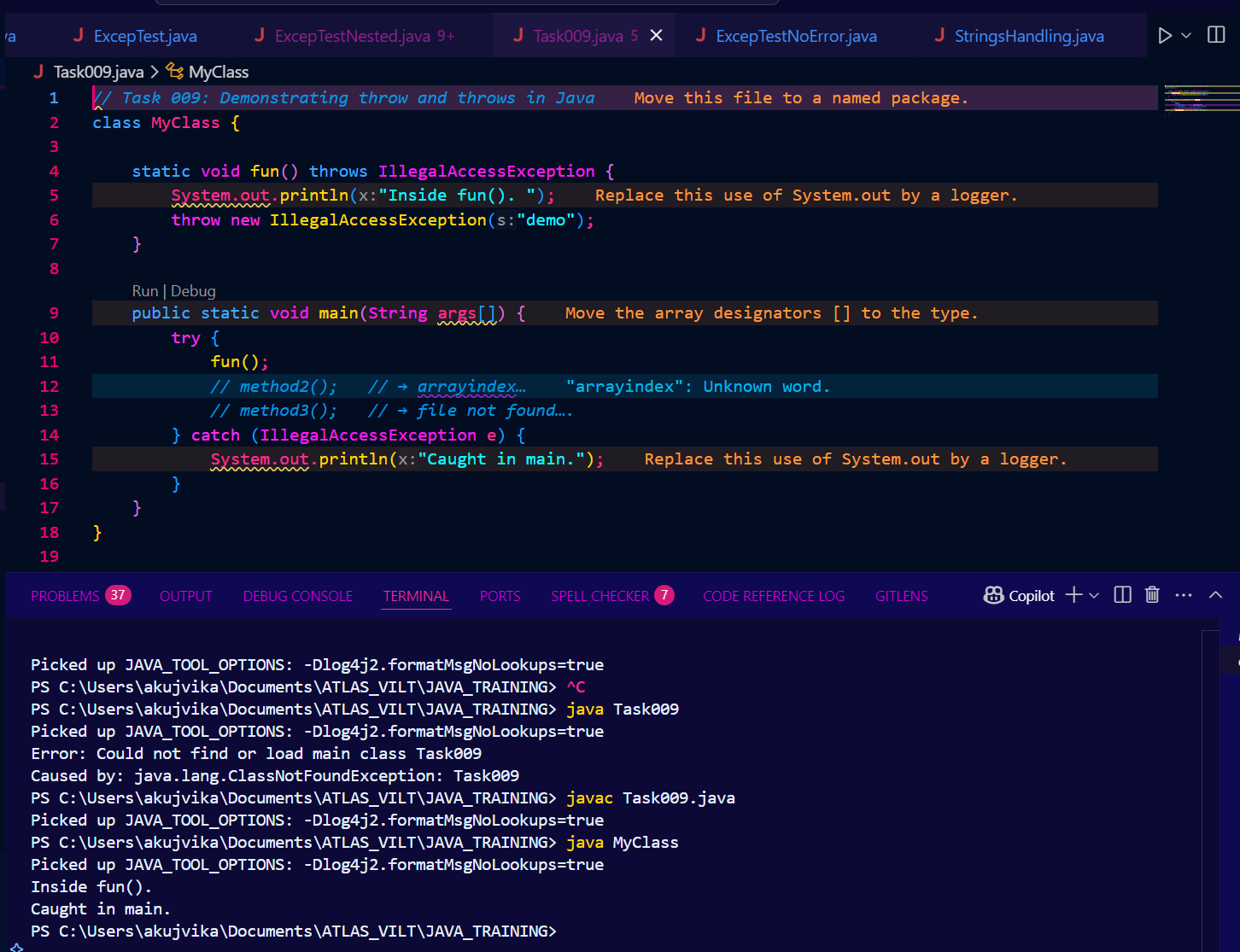
The fun() method is called, prints "Inside fun().", and then throws an IllegalAccessException.

The exception is not handled inside fun(), so it propagates to the main method.

The main method catches the exception and prints "Caught in main."

This demonstrates the use of throw (to throw an exception) and throws (to declare that a method might throw an exception).

If you want to see more about throw and throws or add more methods as in your comments, let me know!



Collection framework:

Custom exception — Abstraction with classes done … with interfaces - Interfaces not done..

Solution : The Java Collection Framework is a set of tools that help you work with groups of related objects in your Java programs. It has three main parts:

Interfaces - These are like blueprints that define the basic actions you can perform on a collection, like adding, removing, and searching for items.

Classes - These are the actual implementations of the interfaces. They provide ready-to-use collections you can use in your code, like lists, sets, and maps.

Algorithms - These are special methods that can do things like sort or search the items in a collection.

In your case:

Custom Exception - You've created your own special type of error that can happen when working with your collections.

Abstraction with classes - You've used abstract classes to provide a partially completed collection that other classes can build upon.

Interfaces not done - You haven't finished defining the blueprints for how your collections should behave.

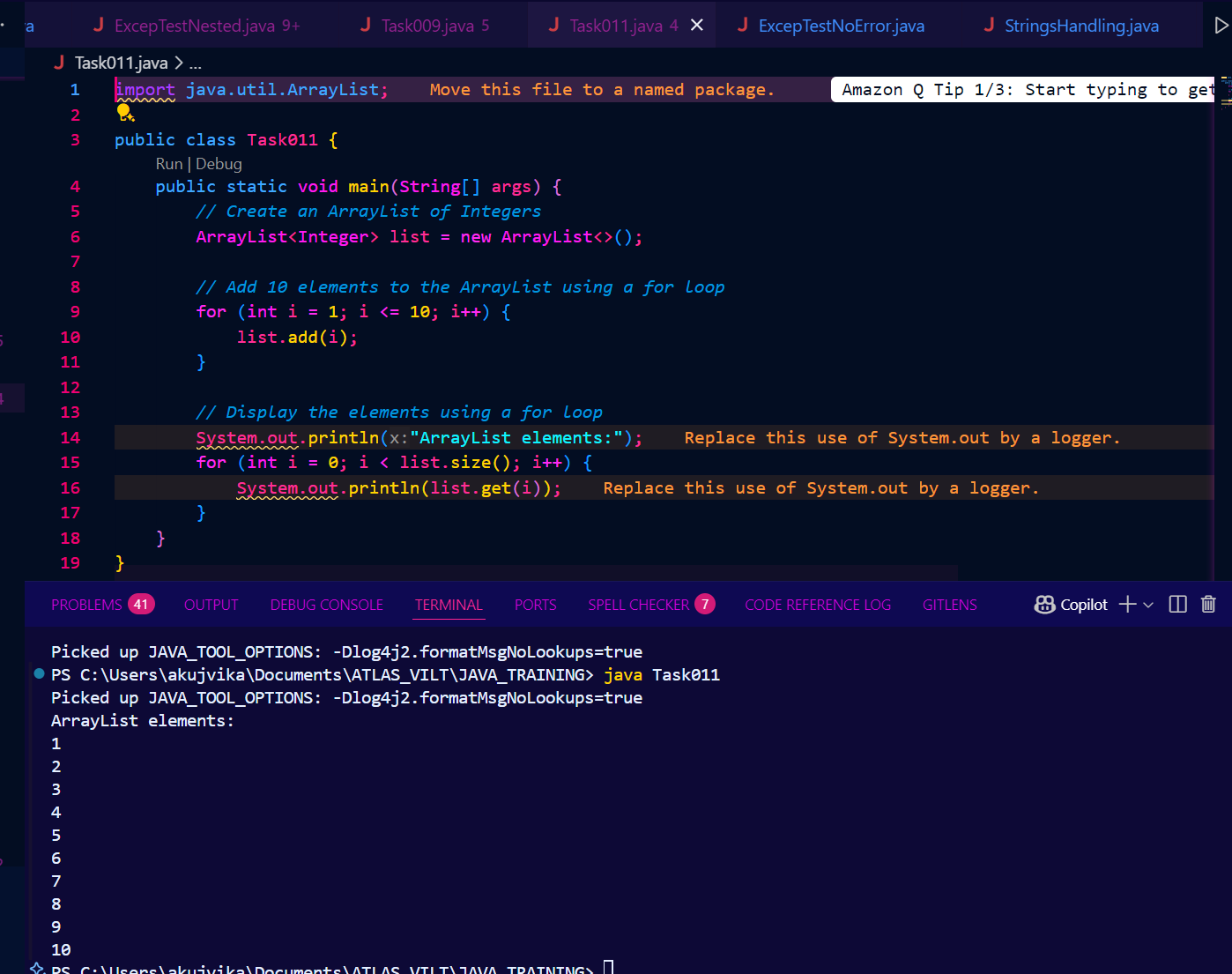
So in summary, the Java Collection Framework gives you a standard way to work with groups of related objects in your Java programs, using interfaces, classes, and algorithms. But it seems you still have some work to do to fully implement the interfaces for your collections.

Task 011 : Wap to create an array list to display 10 elements using for loop.

Solution :

The code creates an ArrayList, adds numbers 1 to 10 using a for loop, and prints each element.

This demonstrates how to use an ArrayList and a for loop to display 10 elements in Java.



Task 012

// Addition, Deletion and Updation of Element

import java.util.\*;

class Main {

public static void main(String args[]){

ArrayList<String> al = new ArrayList<>();

al.add("Prasunamba");

al.add("Meher");

System.out.println("Orignal List : "+al);

al.add(1, "Hello");

System.out.println("After Adding element at index 1 : "+ al);

al.remove(0);

System.out.println("Element removed from index 0 : "+ al);

al.remove("Prasunamba");

System.out.println("Element Prasunamba removed : "+ al);

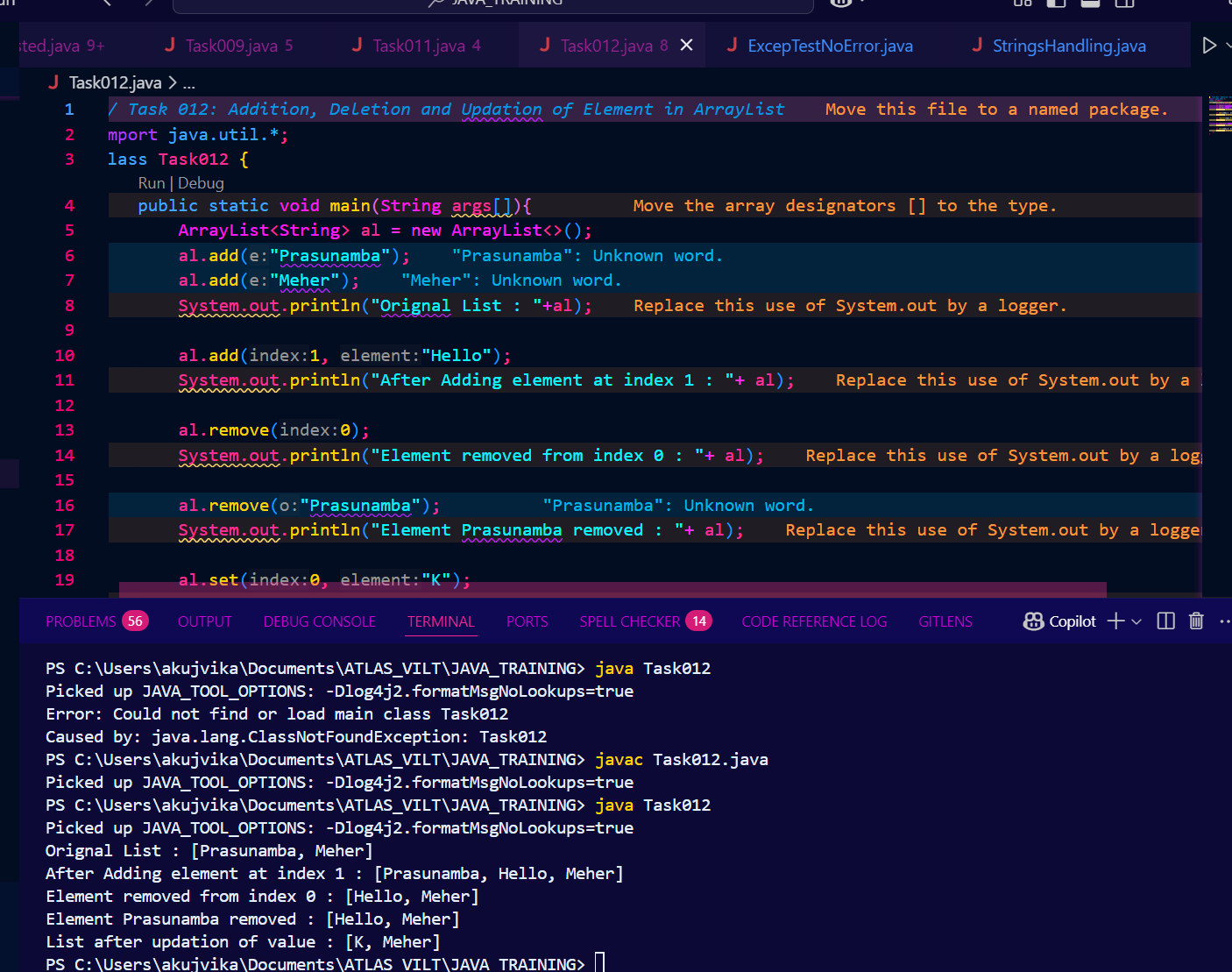
al.set(0, "K");

System.out.println("List after updation of value : "+al);

}

}

Solution : Elements are added, inserted, removed, and updated in the ArrayList, and the list is printed after each operation to show the changes. This demonstrates basic ArrayList operations in Java



Task 013 : Run the code and see hope the user defined exception works..

User defined Exception:

// A Class that represents user-defined exception

class Customer extends Exception {// predefined class Exception

public Customer(String m) { // constructor with parameters

super(m); // parent class constructor

}

}

// A Class that uses the above Customer

public class setText {

public static void main(String args[]) {

try {

// Throw an object of user-defined exception

throw new Customer("This is a custom exception");

}

catch (Customer ex) {

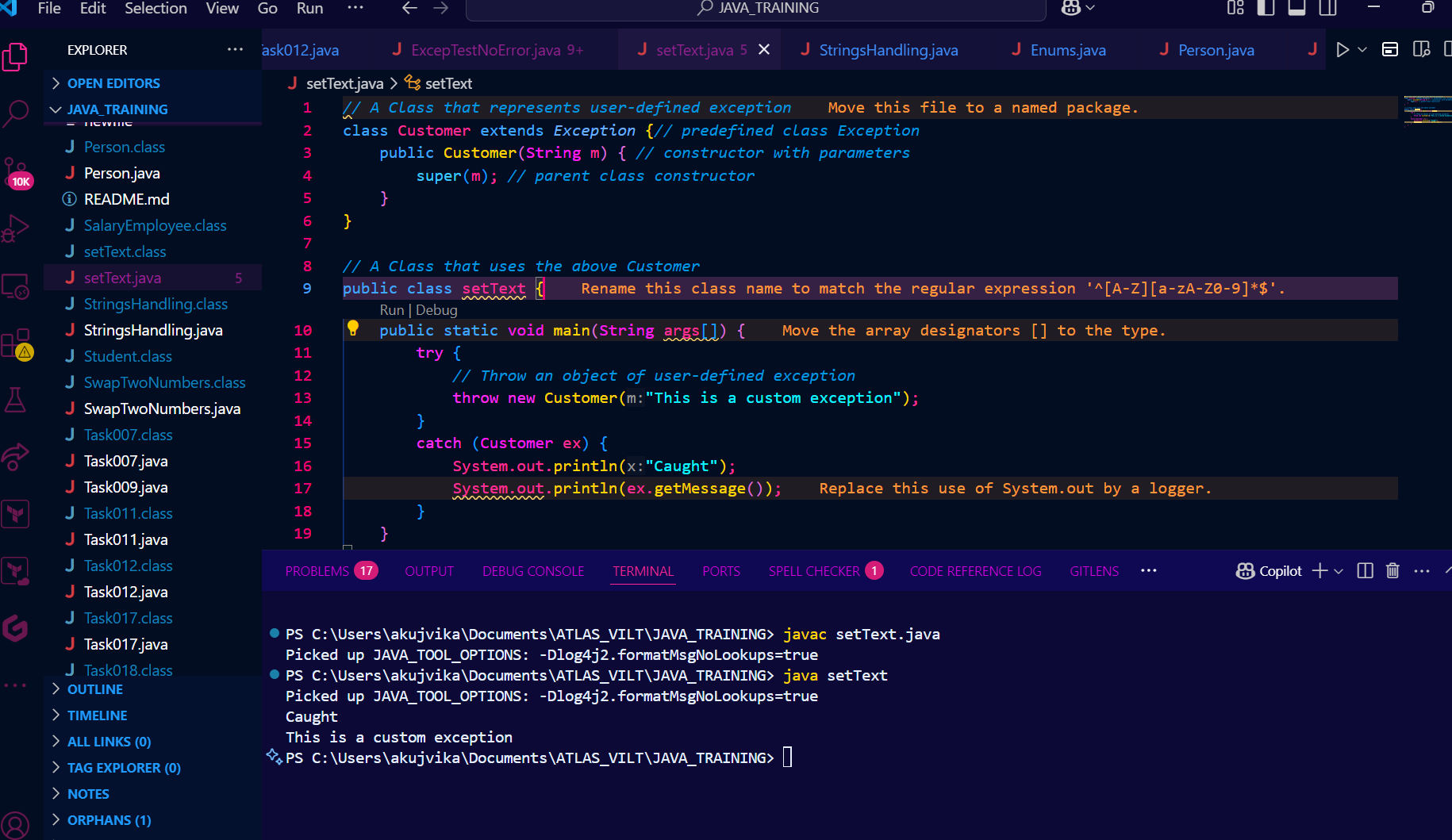
System.out.println("Caught");

System.out.println(ex.getMessage());

}

}

}

Solution : 

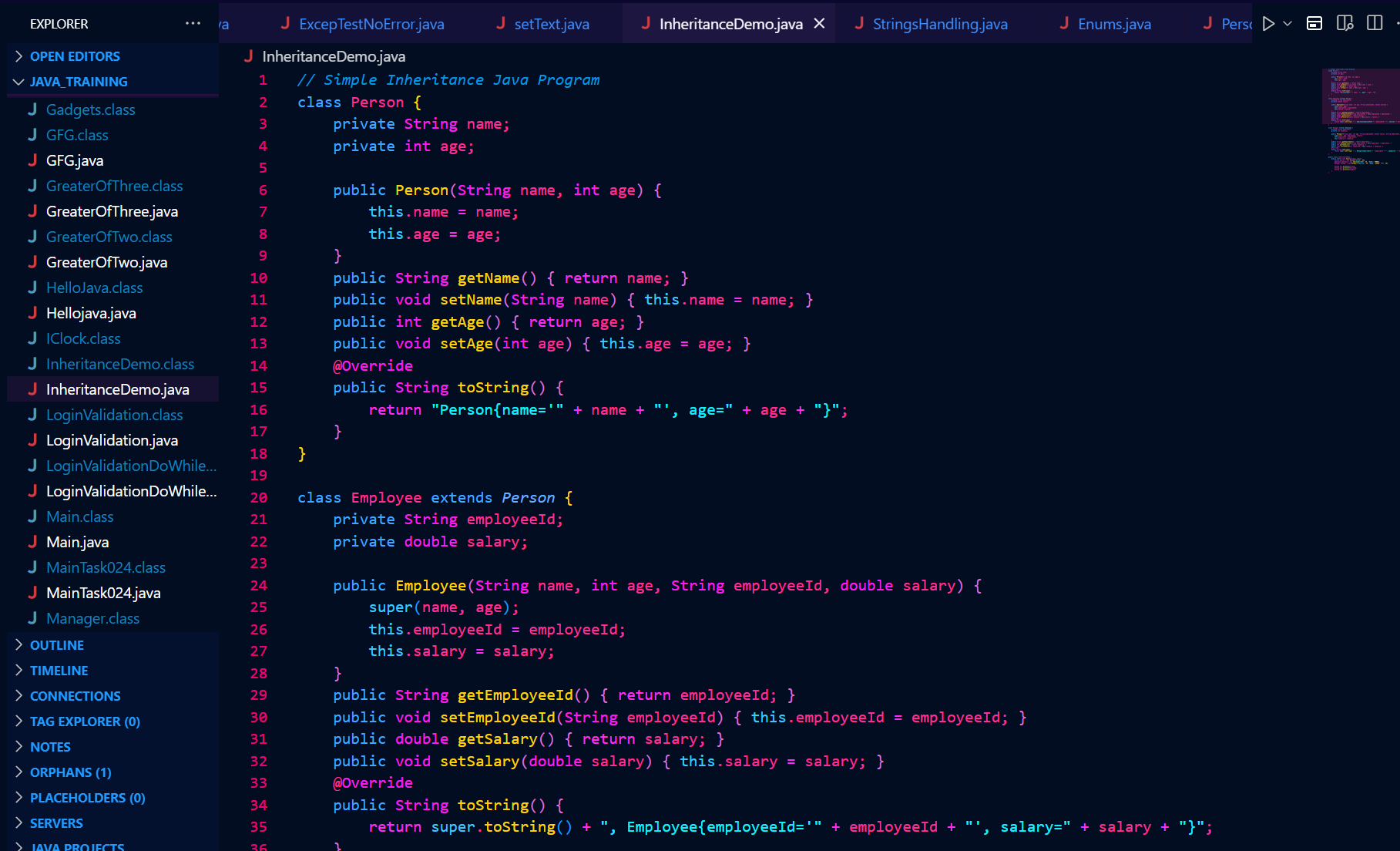
Task 014

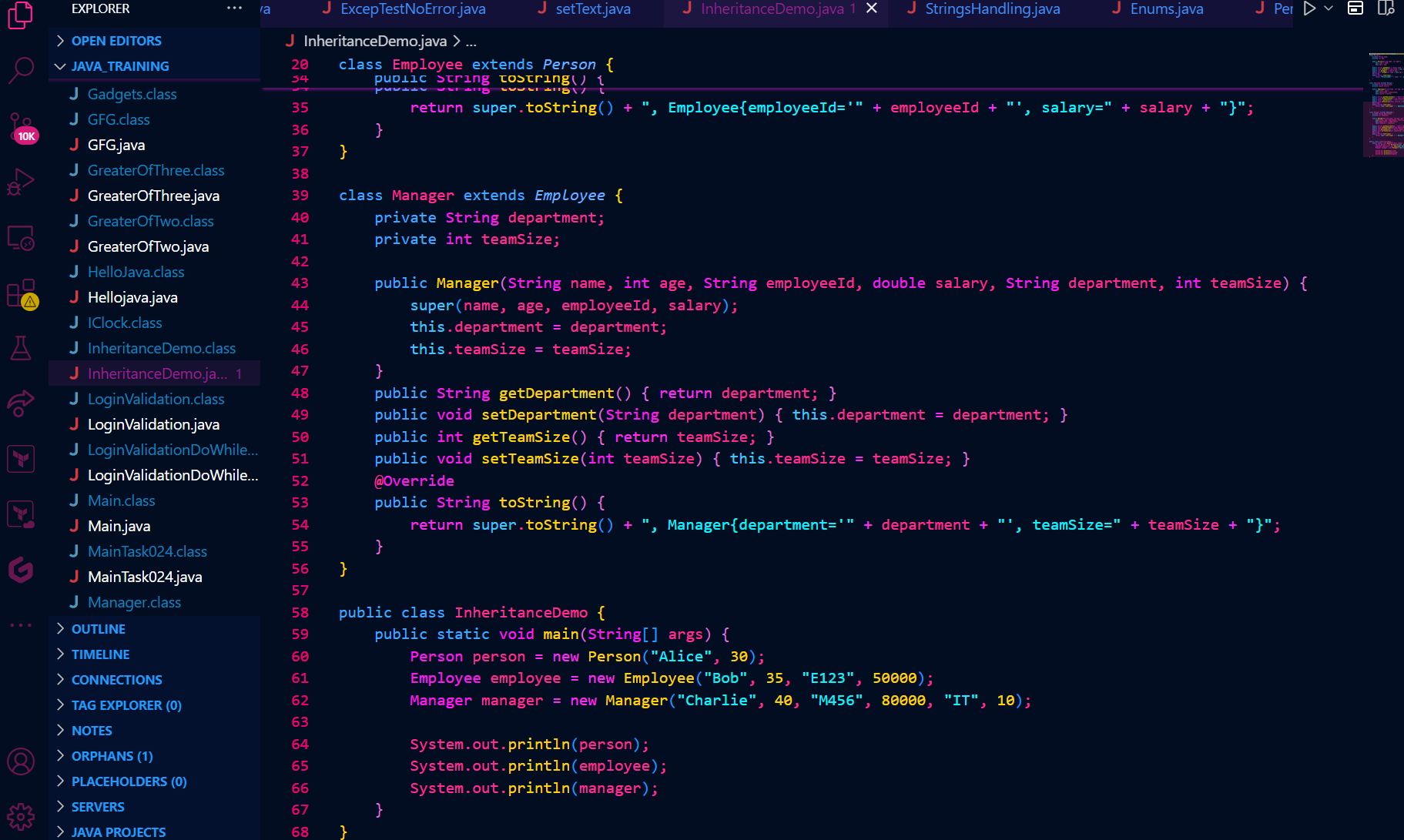
Inheritance

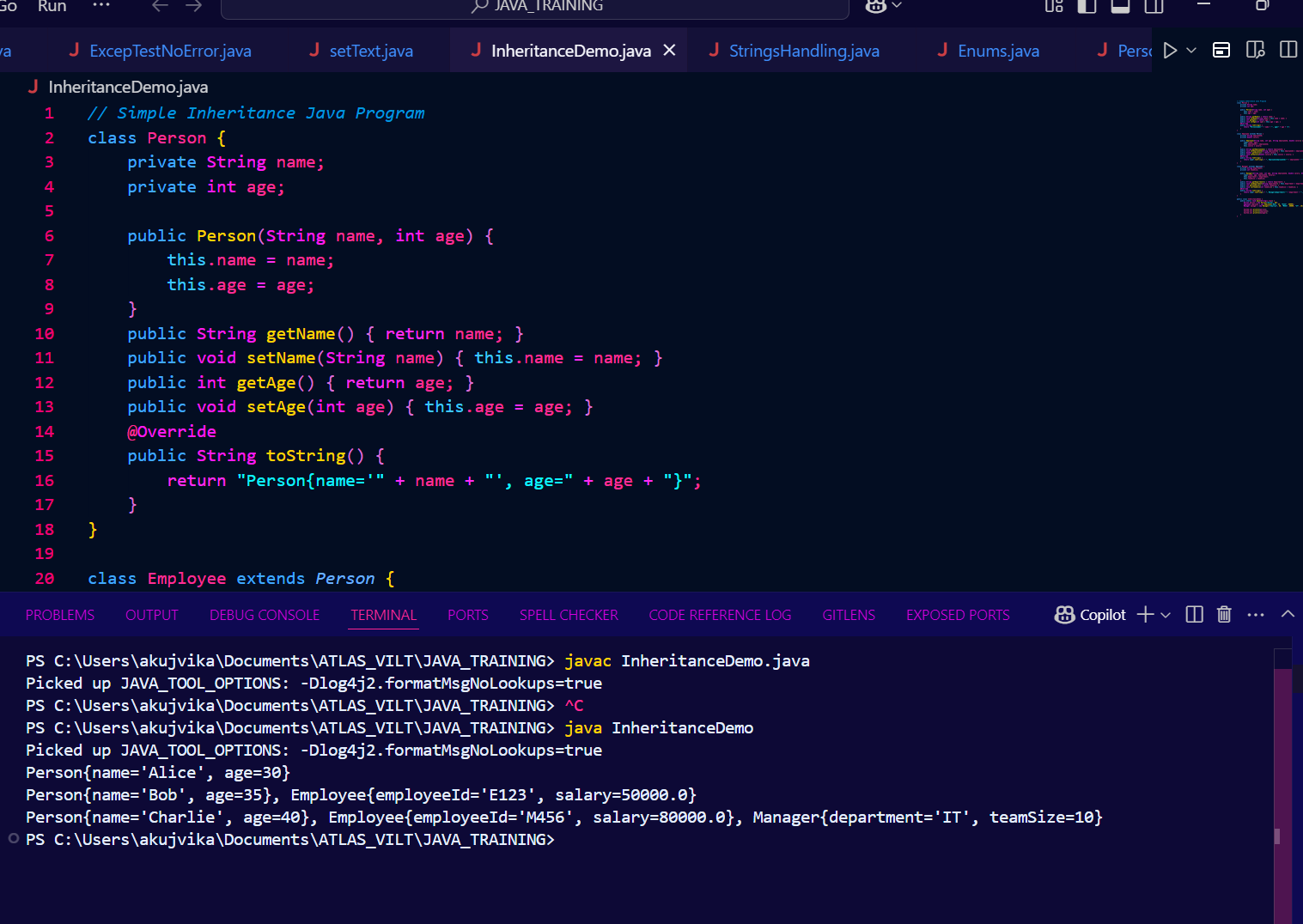
Classes customer/ person , employee, Manager … 2 variables in each class

Driver class – display all the variables… toString();

Hint : use getter and setters..







Inner classes

Task 015

What is the output of the below code snippet.. Explain ..

class OuterClass {

int x = 10;

class InnerClass {

int y = 5;

}

}

public class Main {

public static void main(String[] args) {

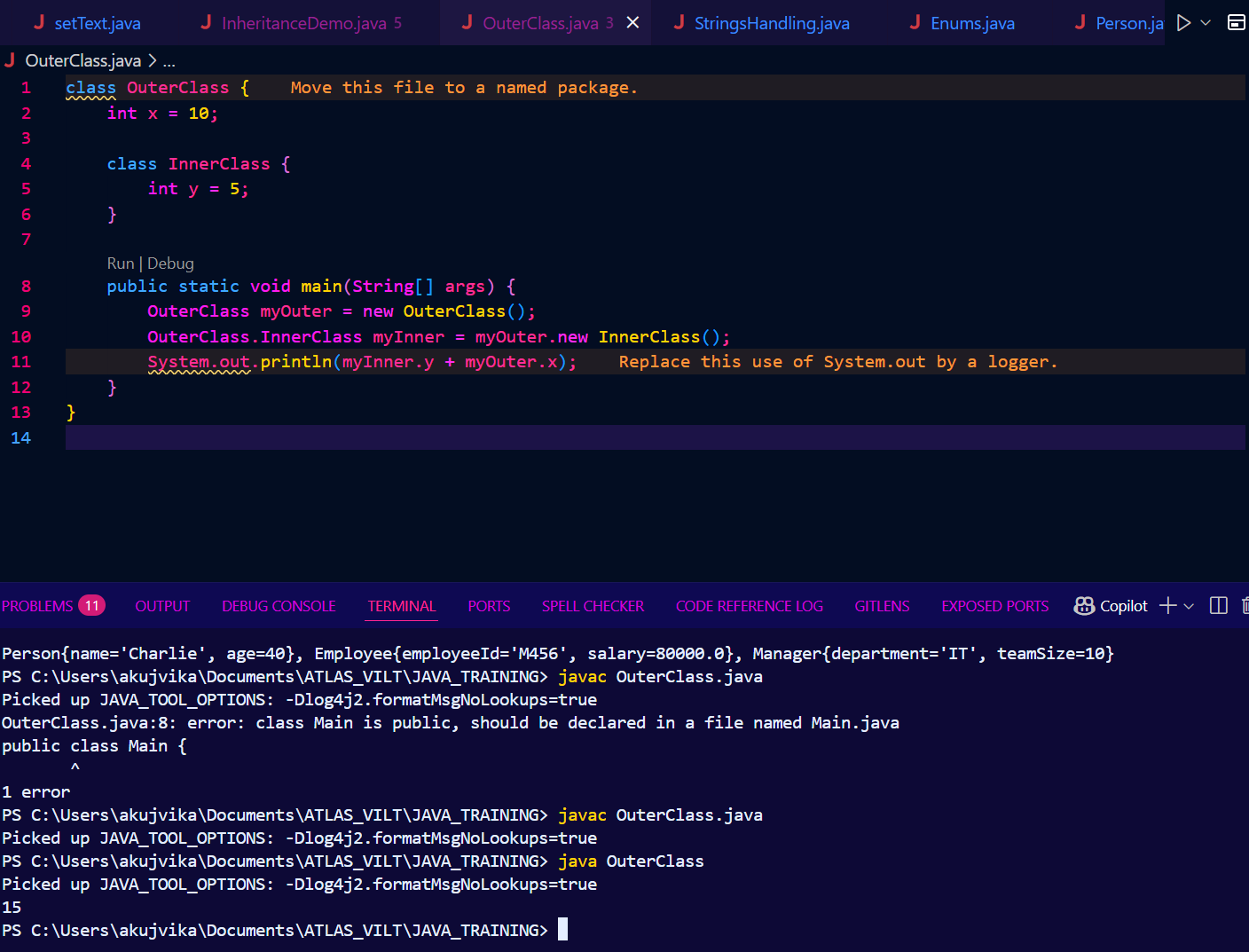
OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.y + myOuter.x);

}

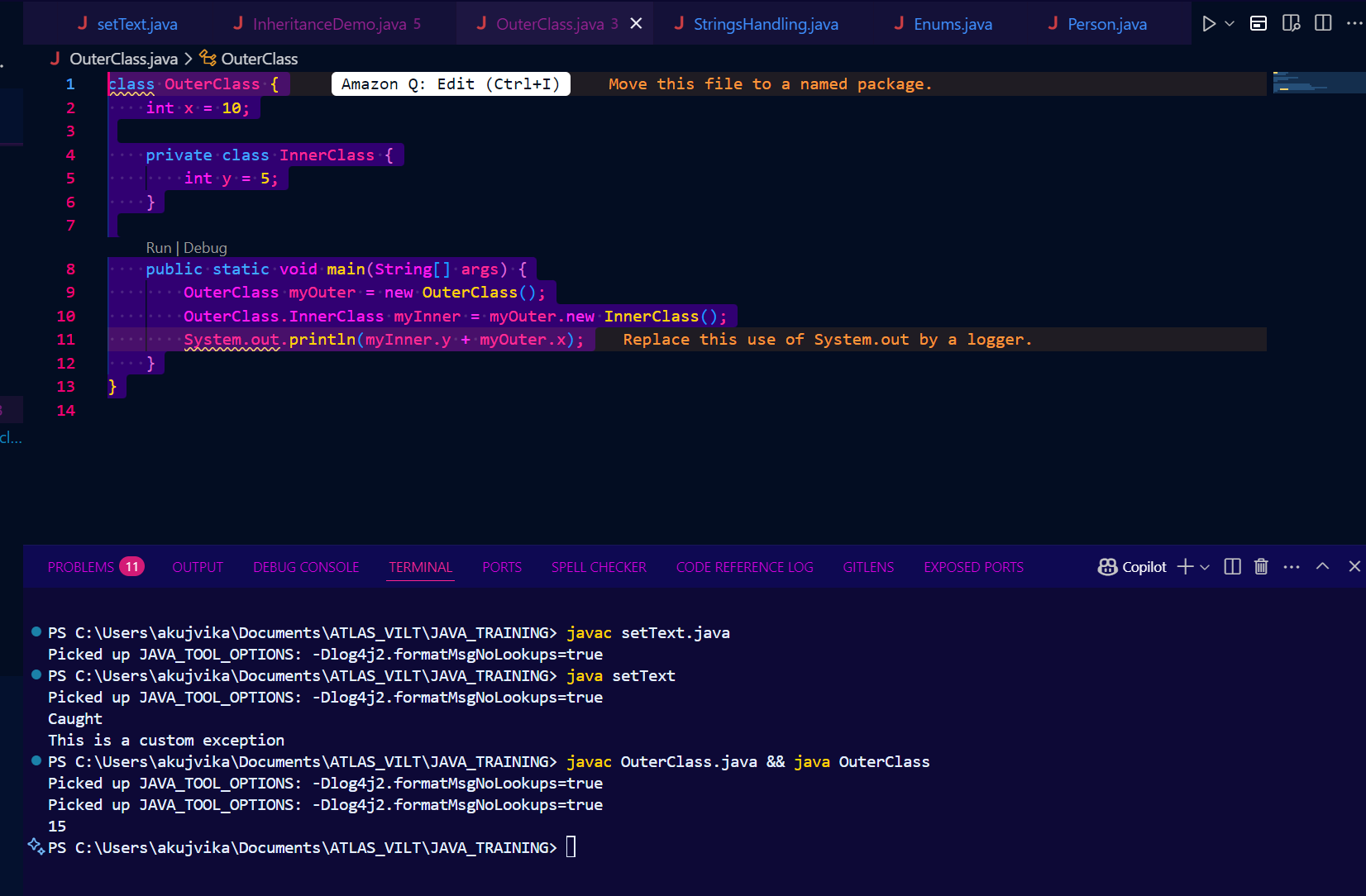
}



Task 016

Use the above code and make the inner class as private and see the output..

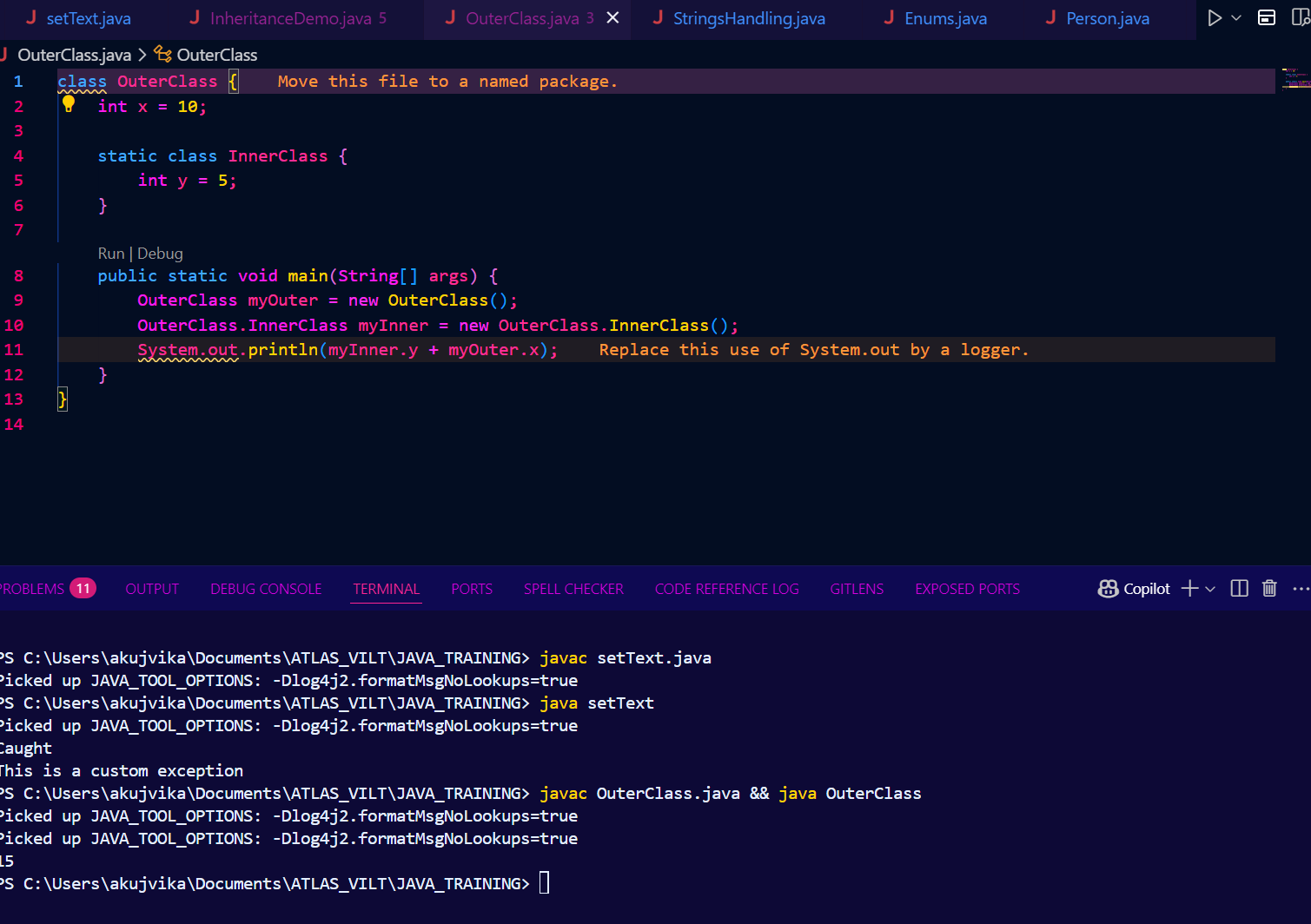
Ex: private class InnerClass {



Task 017

Use the above code Task 015 and make the inner class static … see the output and explain..

Ex: static class InnerClass {



Task 018

Use the above code Task 015 and create a method in innerclass and return the outer class variable

class OuterClass{

Int x = 50;

Class InnerClass {

Public int innerMethod() {

Return x;

}

}

}

Public class DriverClass {

psvm(){

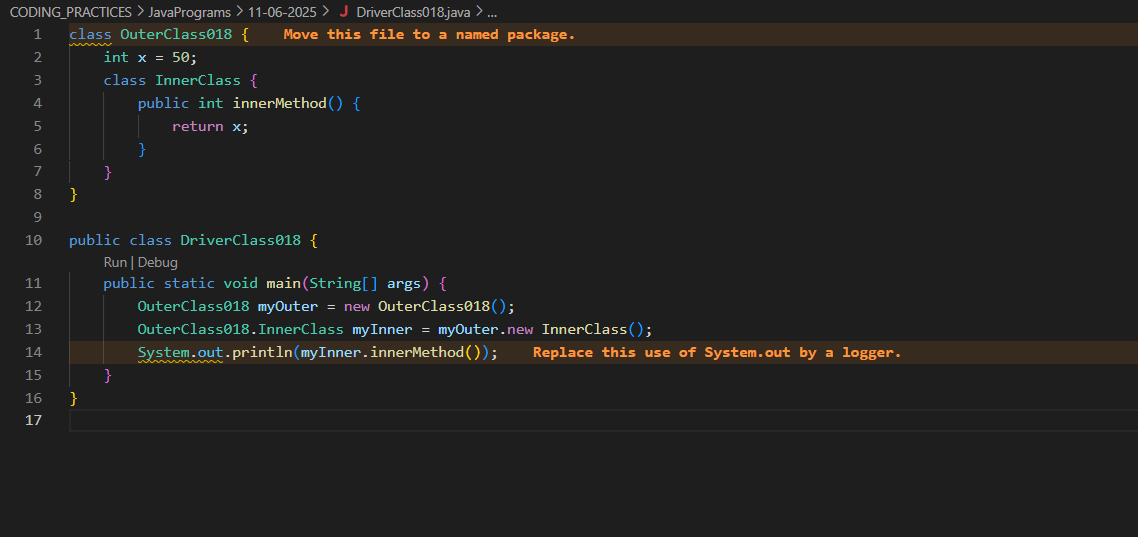
OuterClass myOuter = new OuterClass();

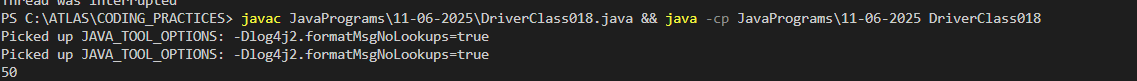
OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.innerMethod());

}

}





Task 019 — query by vivek

class OuterClass {

int x = 10;

static class InnerClass {

static int y = 5;

}

}

public class Main {

public static void main(String[] args) {

OuterClass.InnerClass myInner = new OuterClass.InnerClass();

System.out.println(myInner.y);

}

}

