**Java Assignment 1**

1. **Explain the difference between primitive and reference data types with examples.**

Ans. In Java, data types are divided into two main categories: primitive data types and reference data types.

Primitive Data Types

Primitive data types are basic data types that store values directly in memory. They are not objects and do not have methods. There are eight primitive data types in Java:

1. byte: 8-bit signed integer

2. short: 16-bit signed integer

3. int: 32-bit signed integer

4. long: 64-bit signed integer

5. float: 32-bit floating-point number

6. double: 64-bit floating-point number

7. boolean: true or false value

8. char: 16-bit unsigned character

Example:

int x = 10; // primitive data type

System.out.println(x); // prints 10

Reference Data Types

Reference data types, on the other hand, store references to objects in memory. They are objects and have methods. Reference data types include:

1. Arrays: collections of primitive or reference data types

2. Classes: custom-defined data types

3. Interfaces: abstract data types

4. Strings: sequences of characters

Example:

String name = "John"; // reference data type

System.out.println(name); // prints "John"

// changing the reference

name = "Jane";

System.out.println(name); // prints "Jane"

Key differences:

- Memory allocation: Primitive data types store values directly in memory, while reference data types store references to objects in memory.

- Null value: Primitive data types cannot be null, while reference data types can be null.

- Methods: Primitive data types do not have methods, while reference data types have methods.

- Assignment: When you assign a primitive data type to another variable, it creates a copy of the value. When you assign a reference data type to another variable, it creates a copy of the reference.

1. **Explain the concept of encapsulation with a suitable example.**

Ans. Encapsulation is a fundamental concept in object-oriented programming (OOP) that binds together the data and the methods that manipulate that data, and keeps both safe from outside interference and misuse.

Let's consider a "Student" class as an example of encapsulation.

Example: Student Class

The "Student" class has attributes like name, roll number, and marks, which are private and cannot be accessed directly. The class also has methods like setMarks(), getMarks(), and calculateGrade() that manipulate and provide access to these attributes.



Explanation

In this example:

- The name, rollNumber, and marks variables are private, meaning they can only be accessed within the Student class.

- The setMarks, getMarks, and calculateGrade methods are public, meaning they can be accessed from outside the class. These methods manipulate the marks attribute and ensure it remains consistent and valid.

- The Student class encapsulates the name, rollNumber, and marks attributes and the methods that manipulate them, making it a self-contained unit.

Benefits

Encapsulation provides several benefits:

- Data Hiding: Protects the data from outside interference and misuse.

- Code Organization: Keeps related data and methods together, making the code more organized and easier to maintain.

- Improved Security: Reduces the risk of data corruption or unauthorized access.

By encapsulating the data and methods, we've created a robust and maintainable Student class that ensures the integrity of student data.

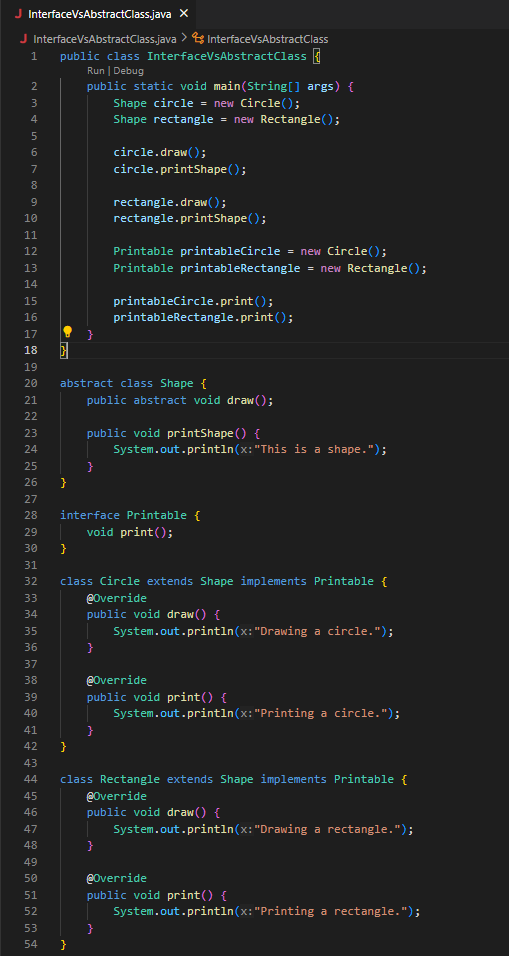
1. **Explain the concept of interfaces and abstract classes with examples.**

Ans. Interfaces: An interface is a abstract concept that defines a set of methods that must be implemented by any class that implements it. Interfaces are used to achieve abstraction and multiple inheritance.

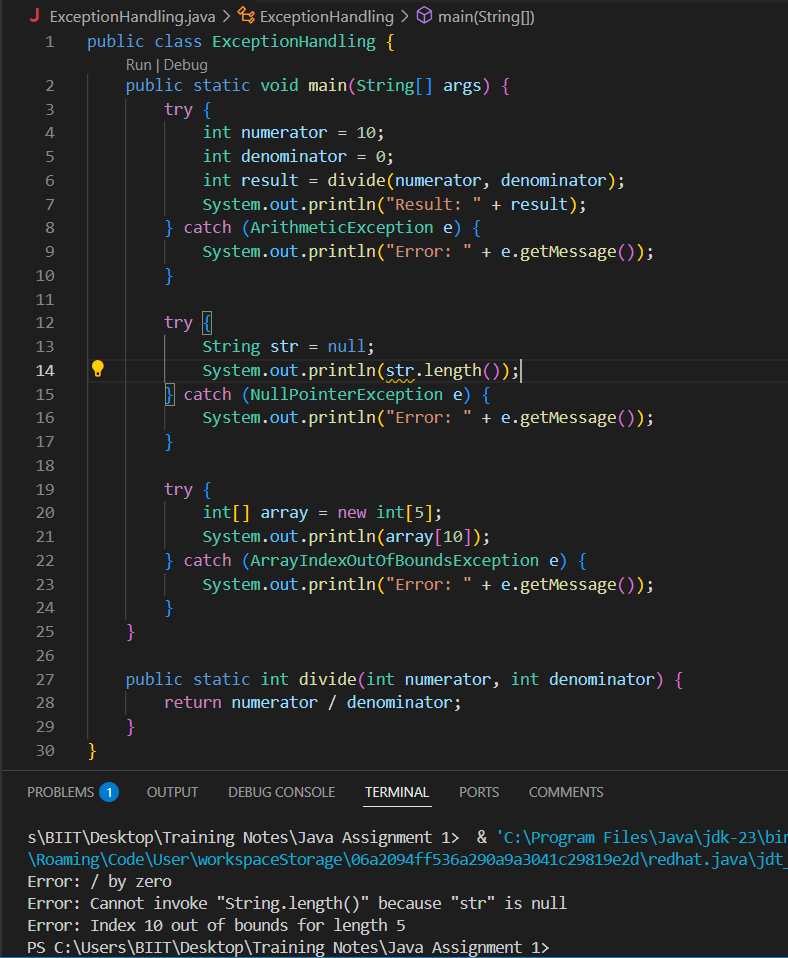
Abstract Classes: An abstract class is a class that cannot be instantiated and is designed to be inherited by other classes. Abstract classes can have both abstract and concrete methods.

- Abstract Class Example: The Shape class is an abstract class that defines an abstract method draw() and a concrete method printShape(). The Circle and Rectangle classes extend the Shape abstract class.

- Interface Example: The Printable interface defines a method print(). The Circle and Rectangle classes implement the Printable interface.



1. **Create a program to handle exceptions using try-catch blocks.**



1. We define three try-catch blocks to handle different types of exceptions.

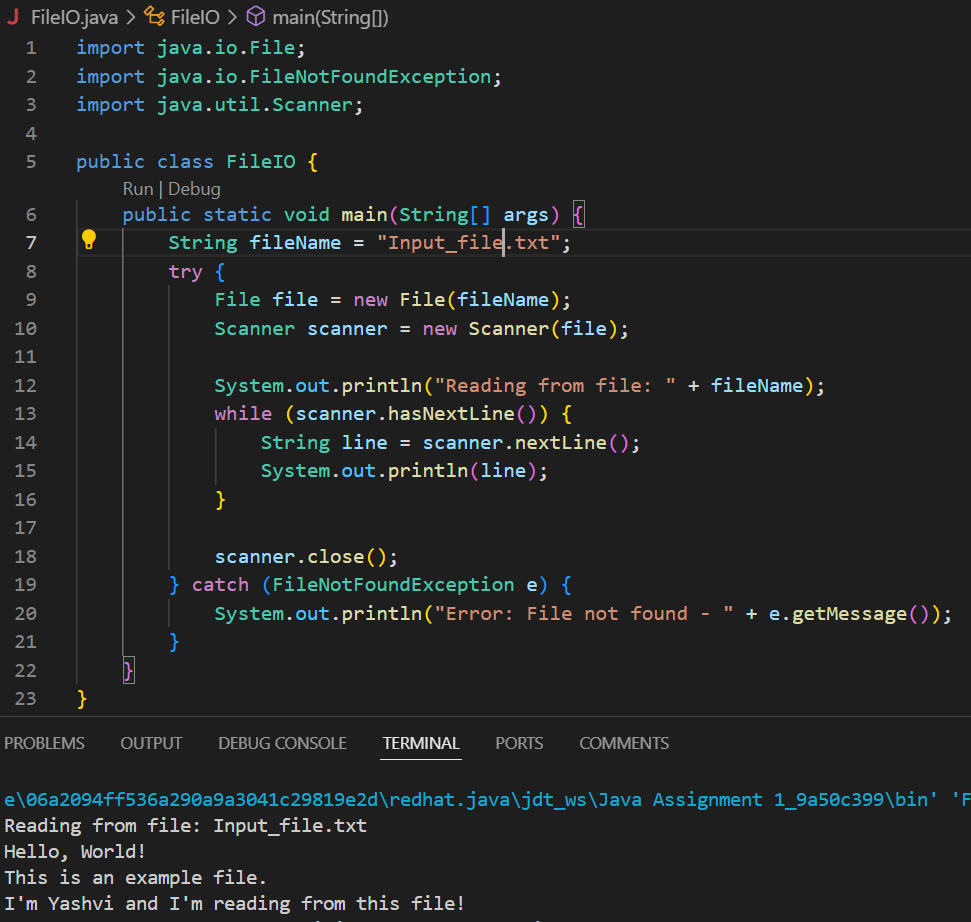
2. In the first try-catch block, we attempt to divide an integer by zero, which throws an ArithmeticException.

3. In the second try-catch block, we attempt to access the length of a null string, which throws a NullPointerException.

4. In the third try-catch block, we attempt to access an array index that is out of bounds, which throws an ArrayIndexOutOfBoundsException.

5. In each catch block, we print an error message along with the exception message.

**5. Implement a simple file I/O operation to read data from a text file.**



1. We import the necessary classes: File and Scanner.

2. We specify the name of the file we want to read (example.txt).

3. We create a File object using the file name.

4. We create a Scanner object to read the file.

5. We use a while loop to read the file line by line.

6. We print each line to the console.

7. We close the Scanner object to prevent resource leaks.

8. We catch the FileNotFoundException exception in case the file does not exist.