**Debugging Exercise 1**: Array Manipulation

Objective: To identify and fix errors in a Java program that manipulates arrays.

public class ArrayManipulation {

public static void main(String[] args) {

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

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

System.out.println(numbers[i]);

}

}

Corrected code

} public class ArrayManipulation {

public static void main(String[] args) {

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

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

Explanation

The error in the provided code is in the loop condition of the **for** loop. The loop should run while **i** is less than the length of the array (**numbers.length**), not less than or equal to the length. The valid indices of an array are from 0 to length-1. Therefore, the correct loop condition should be **i < numbers.length**. Here's the corrected code:

java

Debugging Exercise 2:

class Car {

private String make;

private String model;

public Car(String make, String model) {

this.make = make;

this.model = model;

}

public void start() {

System.out.println("Starting the car.");

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car("Toyota", "Camry");

car.start();

car.stop();

}

}

Corrected code

class Car {

private String make;

private String model;

public Car(String make, String model) {

this.make = make;

this.model = model;

}

public void start() {

System.out.println("Starting the car.");

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car("Toyota", "Camry");

car.start();

// Remove the following line if you don't have a stop method

// car.stop();

}

}

**Explanation**

In the provided code, the **Car** class does not have a **stop()** method defined, but the **main** method in the **Main** class is attempting to call **car.stop()**

To fix this issue, you should either add a **stop()** method to the **Car** class or remove the **car.stop()** line from the **main** method if you don't intend to have a stop functionality.

Debugging Exercise 3

public class ExceptionHandling {

public static void main(String[] args) {

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

try {

System.out.println(numbers[10]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index out of bounds.");

}

int result = divide(10, 0);

System.out.println("Result: " + result);

}

public static int divide(int a, int b) {

return a / b;

}

}

Correction

public class ExceptionHandling {

public static void main(String[] args) {

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

try {

// Accessing an index that is out of bounds

System.out.println(numbers[10]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index out of bounds.");

}

try {

// Handling division by zero

int result = divide(10, 0);

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Cannot divide by zero.");

}

}

public static int divide(int a, int b) {

if (b == 0) {

throw new ArithmeticException("Division by zero");

}

return a / b;

}

}

Explanation

The error in the provided code is that you are accessing an index that is out of bounds in the **numbers** array (**numbers[10]**). Additionally, there's a potential division by zero error in the **divide** method. You should handle the division by zero exception to prevent a runtime error. This corrected code includes a try-catch block for both the array index out of bounds exception and the division by zero exception. The **divide** method now checks if the divisor is zero and throws an **ArithmeticException** to handle the division by zero case

Debugging Exercise 4

public class Fibonacci {

public static int fibonacci(int n) {

if (n <= 1)

return n;

else

return fibonacci(n-1) + fibonacci(n-2);

}

public static void main(String[] args) {

int n = 6;

int result = fibonacci(n);

System.out.println("The Fibonacci number at position " + n + " is: " + result);

}

}

Corrected code

public class Fibonacci {

public static int fibonacci(int n) {

if (n <= 1)

return n;

else

return fibonacci(n - 1) + fibonacci(n - 2);

}

public static void main(String[] args) {

int n = 6;

int result = fibonacci(n);

System.out.println("The Fibonacci number at position " + n + " is: " + result);

}

}

Explanation

The provided code is correct for calculating the nth Fibonacci number using recursion. However, keep in mind that the recursive approach has exponential time complexity, and it may not be efficient for large values of n. If performance is a concern, you might want to consider using an iterative or memoization approach.

Debugging exercise 5

public class PrimeNumbers {

public static List<Integer> findPrimes(int n) {

List<Integer> primes = new ArrayList<>();

for (int i = 2; i <= n; i++) {

boolean isPrime = true;

for (int j = 2; j < i; j++) {

if (i % j == 0) {

isPrime = false;

break;

}

}

if (isPrime) {

primes.add(i);

}

}

return primes;

}

public static void main(String[] args) {

int n = 20;

List<Integer> primeNumbers = findPrimes(n);

System.out.println("Prime numbers up to " + n + ": " + primeNumbers);

}

}

Corrected code

import java.util.ArrayList;

import java.util.List;

public class PrimeNumbers {

public static List<Integer> findPrimes(int n) {

List<Integer> primes = new ArrayList<>();

if (n >= 2) {

primes.add(2);

}

for (int i = 3; i <= n; i += 2) {

boolean isPrime = true;

int sqrt = (int) Math.sqrt(i);

for (int j = 3; j <= sqrt; j += 2) {

if (i % j == 0) {

isPrime = false;

break;

}

}

if (isPrime) {

primes.add(i);

}

}

return primes;

}

public static void main(String[] args) {

int n = 20;

List<Integer> primeNumbers = findPrimes(n);

System.out.println("Prime numbers up to " + n + ": " + primeNumbers);

}

}

Explanation

The code you provided to find prime numbers has a logical error in the inner loop. The inner loop should check for divisibility up to the square root of **i**, not up to **i-1**. Additionally, the code could be optimized further by skipping even numbers (except for 2) since all even numbers greater than 2 are not prime.