***Task 1:***

**Debugging Exercise 1: Array Manipulation**

* ***Error****:*

**i < numbers.length**

* ***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++ ){  
 System.*out*.println(numbers[i]);  
 }  
 }  
}**

* ***Error Explanation:***

When the code is being run it shows “**ArrayIndexOutOfBoundsException” .** The erroris shown because the array index starts from 0. In the given program the total length is 5 but according to the index it is 4. In the previous program the for loop iterates till numbers[5] which doesn’t exist. Using **i < numbers.length** will correct the issue and will not show any error

**Debugging Exercise 2:Object- Oriented Programming**

* ***Error:***

cannot find symbol method 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 void* stop() {  
 System.*out*.println("Stopping the car.");  
 }  
}  
  
*public class* Main {  
 *public static void* main(String[] args) {  
 Car car = *new* Car("Toyota","Camry");  
 car.start();  
 car.stop();  
 }  
}**

* ***Error Explanation:***

Here it is missing a method called stop(). This is why it is showing an error called **cannot find symbol method stop()**. Adding a stop() method to the previous code will make it run without any error.

**Debugging Exercise 3: Exception Handling**

* ***Error:***

int result = divide(10, 0);  
 System.out.println("Result:"+result);

* ***Corrected Code:***

**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.");  
 }  
  
 try {  
 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) {  
 return a / b;  
 }  
}**

* ***Error Explanation:***

In the previous code we are attempting to divide by zero in the method divide() which causes an **“ArithmeticException”** error. To make the code run smoothly we are adding another try- catch block.

**try {  
 int result = divide(10, 0);  
 System.out.println("Result: " + result);  
} catch (ArithmeticException e) {  
 System.out.println("Cannot divide by zero.");  
}**

This will make the code run smoothly.

**Debugging Exercise 4:**

* ***Error:***

The previous code correctly calculates the Fibonacci number. But the code can be inefficient for larger n values.

* ***Corrected Code:***

**public class Fibonacci {  
 public static int fibonacci(int n) {  
 if (n <= 1) {  
 return n;  
 }  
  
 int a = 0, b = 1;  
 int c = 0;  
 for (int i = 2; i <= n; i++) {  
 c = a + b;  
 a = b;  
 b = c;  
 }  
 return c;  
 }  
  
 public static void main(String[] args) {  
 int n = 6;  
 int result = fibonacci(n);  
 System.out.println("The fibonacci number at position " + n + " is: " + result);  
 }  
}**

* ***Error Explanation:***

The recursive approach recalculates the same Fibonacci numbers multiple times, resulting in an exponential time complexity (O(2^n)). To improve performance, we use memorization to store and reuse results, or switch to an iterative approach that computes each value once, reducing the time complexity to O(n).

**Debugging Exercise 5:**

* ***Error:***

The previous code correctly finds the prime number, but the code can be inefficient for larger n values.

Because it checks divisibility up to i-1 for each number i.

* ***Corrected Code:***

**import java.util.\*;  
  
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 \* 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);  
 }  
}**

* ***Error Explanation:***

In the original code, the inner loop **for (int j = 2; j < i; j++)** checks every number **j** from **2** to **i-1** to see if it divides **i** without a remainder. For larger values of **i**, this results in many unnecessary checks, especially as **i** gets larger.