VaultofCodes

JAVA INTERNSHIP

EXERCISE 1:

```
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]);
		}
	}
}
```

Debugged Code (1) :

```
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]);
		}
	}
}
```

Debugged Code (2) :

```
public class ArrayManipulation

{
	public static void main(String[] args)
	{
		int[] numbers = {1, 2, 3, 4, 5};
		for (int i = 0; i <=(numbers.length-1); i++)
		{
			System.out.println(numbers[i]);
		}
	}
}
```

Explanation:

There exists an Runtime Exception rather than an error in Excercise1. Since the length of the given array of numbers starts counting from one(1) and ends with the respective length where as the indexing starts with Zero(0). As the printing statement follows indexing of the given array **ArrayIndexOutOfBoundsException** arises. To overcome this the limit of the loop must be set by 1 decrement or lessthan(<) to the length of the array. As a result the Indexing and loop limit matches.

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();
        }
}
```

Debugged Code (1) :

```
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();
        }
}
```

Debugged Code (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 void stop()
        {
                System.out.println("Stoping the car.");
        }

}
public class Main
{
        public static void main(String[] args)
        {
                Car car = new Car("Toyota", "Camry");
                car.start();
                car.stop();
        }
}
```

Explanation:

The error arises due to the undeclared and undefined method **car.stop()**. This Method isn’t declared and defined in the given code. So there arises an error **cannot find symbol**. To overcome this error either the stop method must be removed from code or the method must be defined.

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;
	}
}
```

Debugged Code (1) :

```
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.");
		}
	}
}
```

Debugged Code (2) :

```
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("Arithmetic error occurred since an integer cannot divided
			by zero.");
		}
	}
	public static int divide(int a, int b)
	{
		return a / b;
	}
}
```

Explanation:

In the code given, even though the exceptional case which is due to the array bounds is cleared with including exceptional condition which resembles and another exceptional case comes into play which arises due to the division with Zero (0) namely represented as **ArithmeticException: / by zero**. To overcome the exception raised the code must be modified by including the exceptional case which tries the division operation and catches the exceptional case and performs the task with in catch section. The other way to clear this this exception is to remove the task which includes the exception **ArithmeticException: / by zero**.

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);
	}
}
```

Debugged 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-1);
                System.out.println("The Fibonacci number at position " + n + " is: " + result);
        }
}
```

Explanation:

The actual **6th** term of Fibonacci series is **5** but by the given code we receive the output as **8** for the term **6**. The reason for this incorrect output is the term accessing to the function and the recursions within the code. While considering to the code and function within the task performer the we pass the **nth** term to the function and this arises the case at which the accessing ends at 1. While the accessing ends at 1 the counter need to get one more term as per the given input. To clear this count weinitialize the method just by **n-1** terms. This must be done as the Fibonacci series starts with Zero (0).
Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …………………

EXERCISE 5:

```
import java.util.*;
public class PrimeNumbers
{
        public static List findPrimes(int n)
        {
                List 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 primeNumbers = findPrimes(n);
        System.out.println("Prime numbers up to " + n + ": " + primeNumbers);
        }
}
```

Debugged Code:

```
import java.util.*;
public class PrimeNumbers
{
        public static List findPrimes(int n)
        {
                List primes = new ArrayList<>();
                for (int i = 2; i <= n; i++)
                {
                        boolean isPrime = true;
                        for (int j = 2; j < Math.sqrt(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 primeNumbers = findPrimes(n);
        System.out.println("Prime numbers up to " + n + ": " + primeNumbers);
        }
}
```

Explanation:

As per the syntax and other classifications of the function within the code the inefficiency of the code is existing even we get the correct output with **no CompileTime** and **RunTime Errors** . This is due to the **Inner for loop** of code. To overcome this issue the loop can be modified to **for (int j = 2; j < Math.sqrt(i); j++)** from **for (int j = 2; j < i; j++)**. By this the efficiency of the code exists and the same correct ouput can be achieved. This can also be correct due to the logic i.e.; if a number has a factor greater than its square root, it must also have a factor smaller than its square root.

The End