**ASSIGNMENT-1**

**Exercise 1:** Array Manipulation  
  
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]);

        }

    }

}

**Explanation:**

The error in the given code is in the ‘for’ loop condition. The loop should run while ‘i’ is less than the array(‘number.length’), but in the given code has ‘<=’ instead of ‘<’.

**Here’s the 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]);

}

}

}

**Exercise 2:** Object-Oriented Programming

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();

    }

}

**Explanation:**

The error in the given code is the ‘stop()’ method is not defined in the ‘Car’ class. In order to fix this, either we need to add it in the class or remove it from the main class.

**Here’s the 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();

}

}

**Exercise 3:** Exception Handling

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;

    }

}

**Explanation:**

In the ‘try’ block, the element that access an element at the index 10 in the numbers array, which has only upto to index 4. That leads to ‘ArrayIndexOutOfBoundsException’.

In the ‘divide’ method, it divides by zero(‘a/b’), which leads to ‘ArithemeticException’.

**Here’s the corrected code:**

public class ExceptionHandling {

public static void main(String[] args) {

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

try {

System.out.println(numbers[2]);

} 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) {

if (b == 0) {

throw new ArithmeticException("Cannot divide by zero.");

}

return a / b;

}

}

**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);

    }

}

**Explanation:**

The error the given code is inefficient to the implementation of larger values of ‘n’. It is only used to calculate the simple recursion to calculate the Fibonacci numbers. For more efficiency, we can use memorization to store intermediate results and avoid redundant recursive calls.

**Here’s the corrected code:**

import java.util.HashMap;

import java.util.Map;

public class Fibonacci {

private static Map<Integer, Integer> memo = new HashMap<>();

public static int fibonacci(int n) {

if (n <= 1)

return n;

if (memo.containsKey(n)) {

return memo.get(n);

} else {

int result = fibonacci(n - 1) + fibonacci(n - 2);

memo.put(n, result);

return result;

}

}

public static void main(String[] args) {

int n = 6;

int result = fibonacci(n);

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

}

}

**Exercise 5:**  
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 < 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);

    }

}

**Explanation:**

The error in the given code is in the inner loop of the ‘findPrimes’ method. The inner loop should iterate up to the square root of ‘i’ instead of ‘i-1’ to improve efficiency. This is because if a number ‘i’ is not prime, it may have a factor smaller than or equal to its square root.

**Here’s the 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 <= 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<Integer> primeNumbers = findPrimes(n);

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

}

}