1. Given a string s consisting of words and spaces, return the length of the last word in the string. A word is a maximal substring consisting of non-space characters only. There will be at least one word, consists of only English letters and spaces ' '.

public class Main {

public static void main(String[] args) {

String s = "Hello World ";

int length = lengthOfLastWord(s);

System.out.println("The length of the last word is: " + length);

}

public static int lengthOfLastWord(String s) {

// Trim any leading or trailing spaces from the string

s = s.trim();

// Split the string into an array of words

String[] words = s.split("\\s+");

// Return the length of the last word

return words[words.length - 1].length();

}

}

Output:

The length of the last word is: 5

2. Using the concepts of thread with implementing Runnable interface in Java to find whether a given number is prime or not.

public class PrimeChecker implements Runnable {

private int number;

public PrimeChecker(int number) {

this.number = number;

}

@Override

public void run() {

isPrime(number);

}

public void isPrime(int number) {

boolean isPrime = true;

if (number <= 1) {

isPrime = false;

} else {

for (int i = 2; i <= number / 2; i++) {

if (number % i == 0) {

isPrime = false;

break;

}

}

}

if (isPrime) {

System.out.println(number + " is a prime number.");

} else {

System.out.println(number + " is not a prime number.");

}

}

public static void main(String[] args) {

int num = 17;

PrimeChecker primeChecker = new PrimeChecker(num);

Thread thread = new Thread(primeChecker);

thread.start();

}

}

Output:

17 is a prime number.

3. Generate a Java code to find the sum of N numbers using array and throw Array Index Out Of Bounds Exception when the loop variable beyond the size N.

import java.util.Scanner;

public class SumOfNNumbers {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the value of N: ");

int n = scanner.nextInt();

int[] numbers = new int[n];

int sum = 0;

for (int i = 0; i <= n; i++) {

try {

System.out.print("Enter number " + (i + 1) + ": ");

numbers[i] = scanner.nextInt();

sum += numbers[i];

} catch (ArrayIndexOutOfBoundsException e) {

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

System.exit(1);

}

}

System.out.println("The sum of the " + n + " numbers is: " + sum);

}

}

Output:

Enter the value of N: 5

Enter number 1: 1

Enter number 2: 2

Enter number 3: 3

Enter number 4: 4

Enter number 5: 5

Enter number 6: The sum of the 5 numbers is: 15

Enter the value of N: 6

Enter number 1: 1

Enter number 2: 2

Enter number 3: 3

Enter number 4: 4

Enter number 5: 5

Error: Array index out of bounds.

4. Using the concepts of thread with implementing Runnable interface in Java to generate Fibonacci series.

import java.util.Scanner;

public class FibonacciGenerator implements Runnable {

private int num;

public FibonacciGenerator(int num) {

this.num = num;

}

@Override

public void run() {

fibonacci(num);

}

public void fibonacci(int num) {

int n1 = 0, n2 = 1, n3, i;

System.out.println("Fibonacci Series upto " + num + " terms:");

for (i = 1; i <= num; i++) {

System.out.print(n1 + " ");

n3 = n1 + n2;

n1 = n2;

n2 = n3;

}

}

public static void main(String[] args) {

int num = 10;

FibonacciGenerator fibonacciGenerator = new FibonacciGenerator(num);

Thread thread = new Thread(fibonacciGenerator);

thread.start();

}

}

Output:

Fibonacci Series upto 10 terms:

0 1 1 2 3 5 8 13 21 34

5. Display Multiplication table for 5 and 10 using various stages of life cycle of the thread by generating a suitable code in Java.

class MultiplicationTable implements Runnable {

private int num;

public MultiplicationTable(int num) {

this.num = num;

}

@Override

public void run() {

System.out.println("Multiplication table for " + num + ":");

for (int i = 1; i <= 10; i++) {

System.out.println(num + " x " + i + " = " + (num \* i));

}

}

public static void main(String[] args) {

MultiplicationTable table5 = new MultiplicationTable(5);

MultiplicationTable table10 = new MultiplicationTable(10);

Thread thread5 = new Thread(table5);

Thread thread10 = new Thread(table10);

thread5.start();

thread10.start();

try {

thread5.join();

thread10.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Multiplication tables have been generated.");

}

}

Output:

Multiplication table for 5:

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

Multiplication table for 10:

10 x 1 = 10

10 x 2 = 20

10 x 3 = 30

10 x 4 = 40

10 x 5 = 50

10 x 6 = 60

10 x 7 = 70

10 x 8 = 80

10 x 9 = 90

10 x 10 = 100

Multiplication tables have been generated.