

Experiment No:01

Experiment Name: Write a Java program that reads data from a file named " data.txt ". Implement error handling using try-catch blocks to handle FileNotFoundException . If the file is not found, print an error message indicating the issue.

Code:

```
package javaassignment;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        String filePath = "data.txt";
        String line;
        StringBuilder fileContent = new StringBuilder();

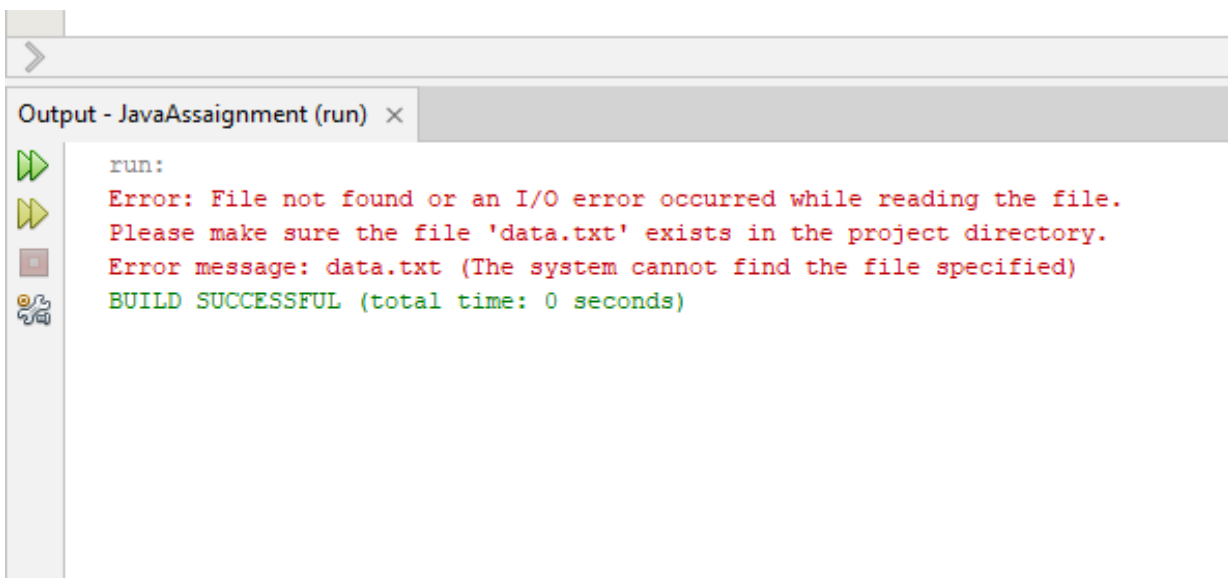
        try {
            FileReader fileReader = new FileReader(filePath);
            BufferedReader bufferedReader = new BufferedReader(fileReader);

            while ((line = bufferedReader.readLine()) != null) {
                fileContent.append(line).append("\n");
            }

            bufferedReader.close();
            System.out.println("File content:");
            System.out.println(fileContent);

        } catch (IOException e) {
            System.err.println("Error: File not found or an I/O error occurred while reading the file.");
            System.err.println("Please make sure the file 'data.txt' exists in the project directory.");
            System.err.println("Error message: " + e.getMessage());
        }
    }
}
```

Output:



Experiment No:02

Experiment Name: Write a Java program that initializes an array of integers and attempts to access an element at an index beyond the array's length. Implement try-catch blocks to handle the Array Index Out Of BoundsException that may occur. If the exception occurs, print a message indicating the invalid index.

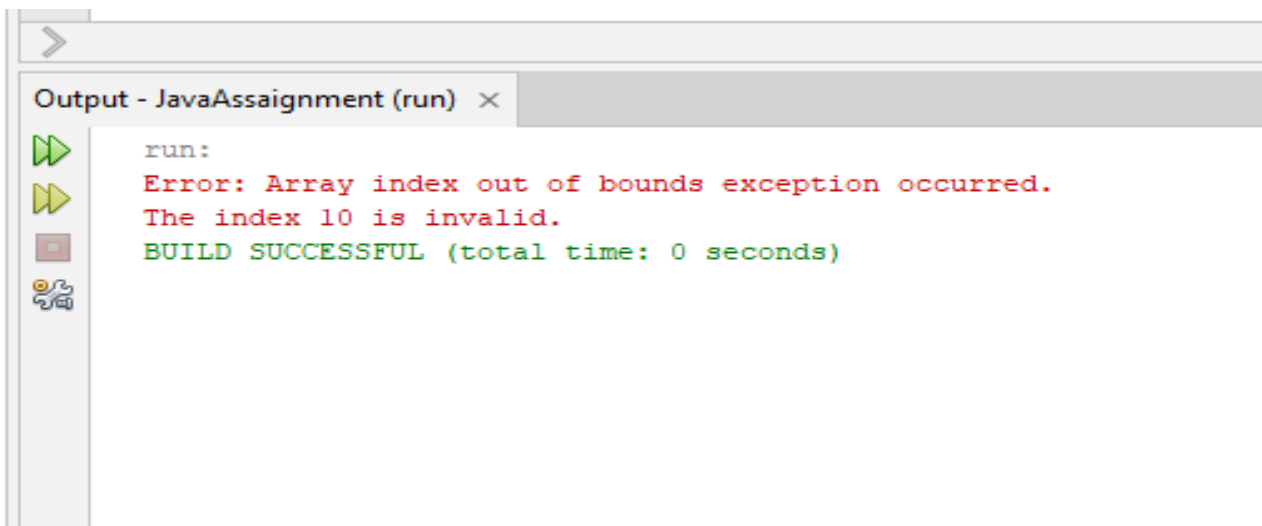
Code:

```
package javaassignment;

public class Main {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        int index = 10;

        try {
            System.out.println("Accessing element at index " + index + ": " + array[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Error: Array index out of bounds exception occurred.");
            System.err.println("The index " + index + " is invalid.");
        }
    }
}
```

Output:



```
Output - JavaAssaignment (run) x
run:
Error: Array index out of bounds exception occurred.
The index 10 is invalid.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment No:03

Experiment Name: Write a Java program to simulate bank account transactions. Implement try-catch blocks to handle exceptions that may occur during withdrawal or deposit operations, such as `Insufficient FundsException` for insufficient balance and `NegativeAmountException` for negative amounts. Use a finally block to ensure that resources are properly released after each transaction.

Code:

```
package javaassignment;

public class Main {
    private double balance;

    public Main(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) throws NegativeAmountException {
        if (amount < 0) {
            throw new NegativeAmountException("Cannot deposit a negative amount.");
        }
        balance += amount;
    }

    public void withdraw(double amount) throws InsufficientFundsException,
        NegativeAmountException {
        if (amount < 0) {
            throw new NegativeAmountException("Cannot withdraw a negative amount.");
        }
        if (balance < amount) {
            throw new InsufficientFundsException("Insufficient balance to complete the withdrawal.");
        }
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }

    public static void main(String[] args) {
        Main account = new Main(100);

        System.out.println("Initial balance: " + account.getBalance());

        try {
            account.deposit(50);
            System.out.println("Deposited 50, new balance: " + account.getBalance());

            account.withdraw(120);
            System.out.println("Withdrew 120, new balance: " + account.getBalance());

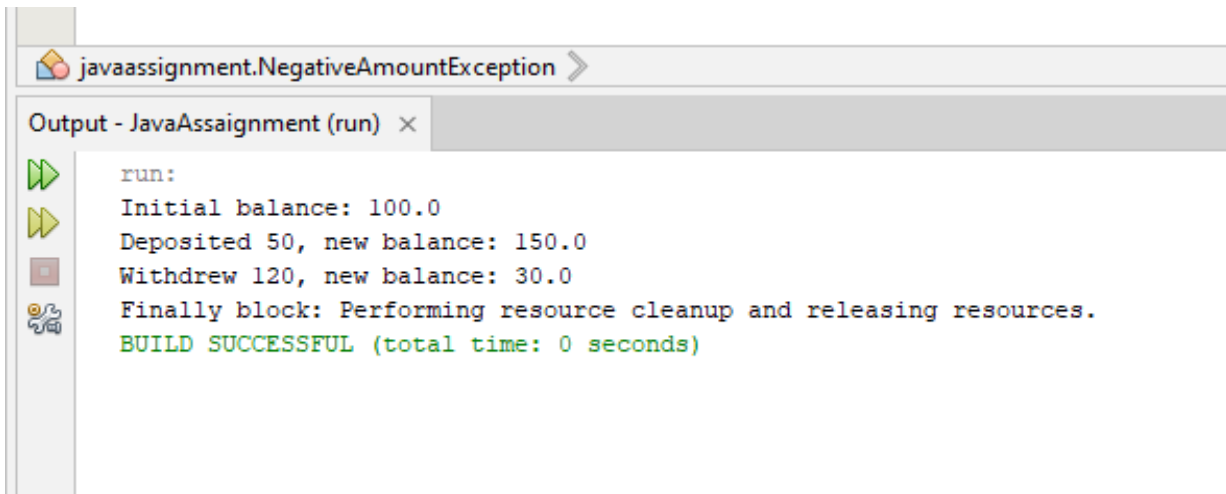
        } catch (NegativeAmountException e) {
            System.err.println("Error: " + e.getMessage());
        } catch (InsufficientFundsException e) {
            System.err.println("Error: " + e.getMessage());
        } finally {
            // Resource release logic would go here
        }
    }
}
```

```
        System.out.println("Finally block: Performing resource cleanup and releasing resources.");
    }
}

class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

class NegativeAmountException extends Exception {
    public NegativeAmountException(String message) {
        super(message);
    }
}
```

Output:



```
run:
Initial balance: 100.0
Deposited 50, new balance: 150.0
Withdrew 120, new balance: 30.0
Finally block: Performing resource cleanup and releasing resources.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment No:04

Experiment Name: Imagine you have a bank account. You can deposit and withdraw money from your account. You should keep in mind that the total amount of money withdrawn from your account must not exceed the total balance present in your account. If such a scenario happens, you need to safely execute from the banking system. Implement the above case in Java with the proper utilization of user-defined exception mechanism. Ensure that resources are properly released after each transaction.

Code:

```
package javaassignment;

public class Main {
    private double balance;

    public Main(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException("Insufficient balance to complete the withdrawal.");
        }
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }

    public static void main(String[] args) {
        Main account = new Main(100);

        System.out.println("Initial balance: " + account.getBalance());

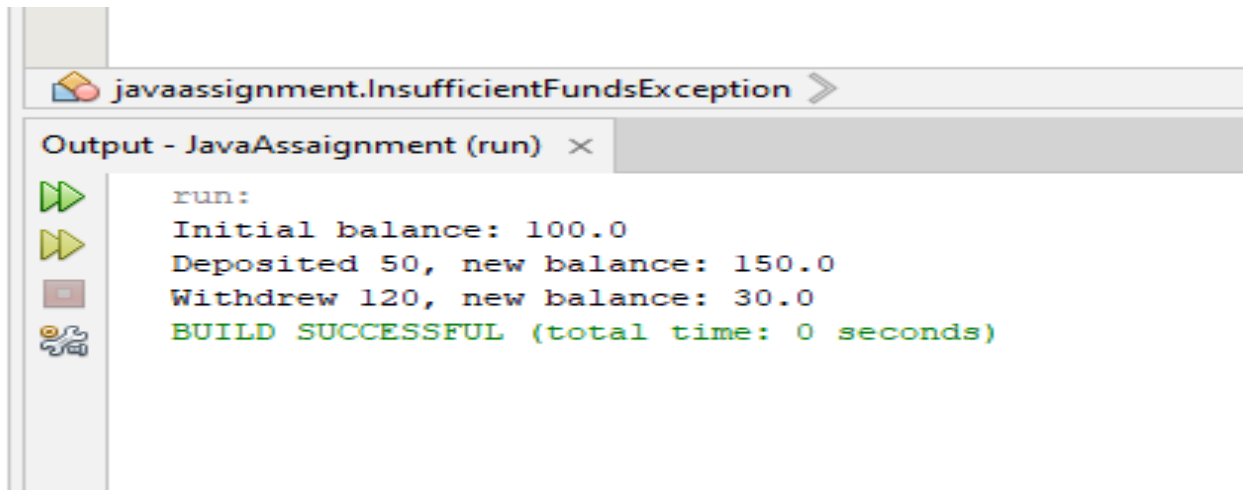
        try {
            account.deposit(50);
            System.out.println("Deposited 50, new balance: " + account.getBalance());

            account.withdraw(120);
            System.out.println("Withdrew 120, new balance: " + account.getBalance());

        } catch (InsufficientFundsException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}

class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}
```

Output:



Experiment No:05

Experiment Name: Write a Java program to validate an email address entered by the user. Implement multiple catch blocks to handle different types of exceptions that may occur during validation, such as `IllegalArgumentException` for invalid format and `NullPointerException` for null input. Use a finally block to close any resources opened during validation.

Code:

```
package javaassignment;

import java.util.Scanner;
import java.util.regex.Pattern;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an email address: ");
        String email = scanner.nextLine();
        scanner.close();

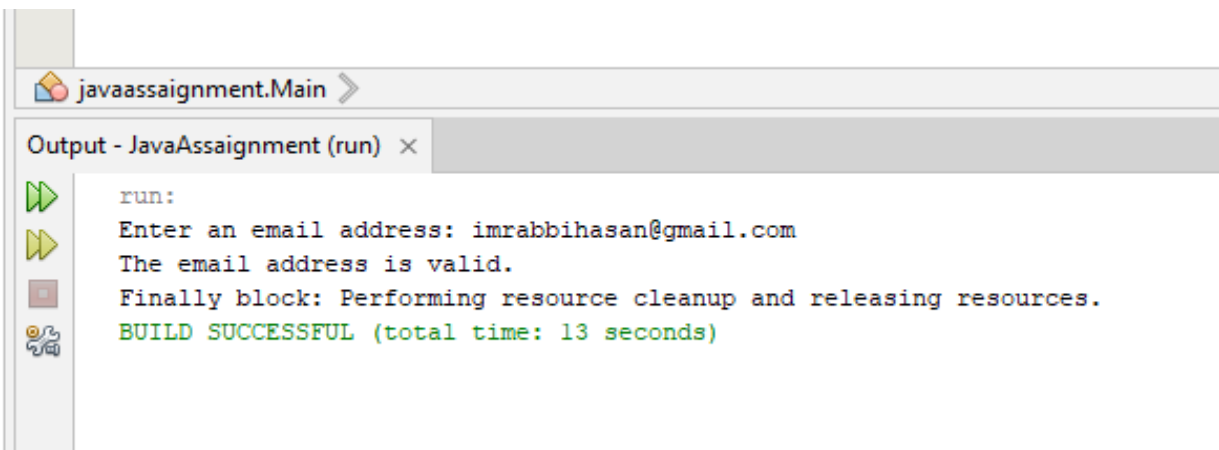
        try {
            validateEmail(email);
            System.out.println("The email address is valid.");
        } catch (IllegalArgumentException e) {
            System.err.println("Error: " + e.getMessage());
        } catch (NullPointerException e) {
            System.err.println("Error: " + e.getMessage());
        } finally {
            System.out.println("Finally block: Performing resource cleanup and releasing resources.");
        }
    }

    private static void validateEmail(String email) throws IllegalArgumentException,
    NullPointerException {
        if (email == null) {
            throw new NullPointerException("Email address cannot be null.");
        }
    }
}
```

```
String regex = "^([\\w!#$%&'*/+=?`{|}~^-.]+(?:\\.[\\w!#$%&'*/+=?`{|}~^-.]+)*@(?:[a-zA-Z0-9-]+\\\\.)+[a-zA-Z]{2,6})$";
Pattern pattern = Pattern.compile(regex);

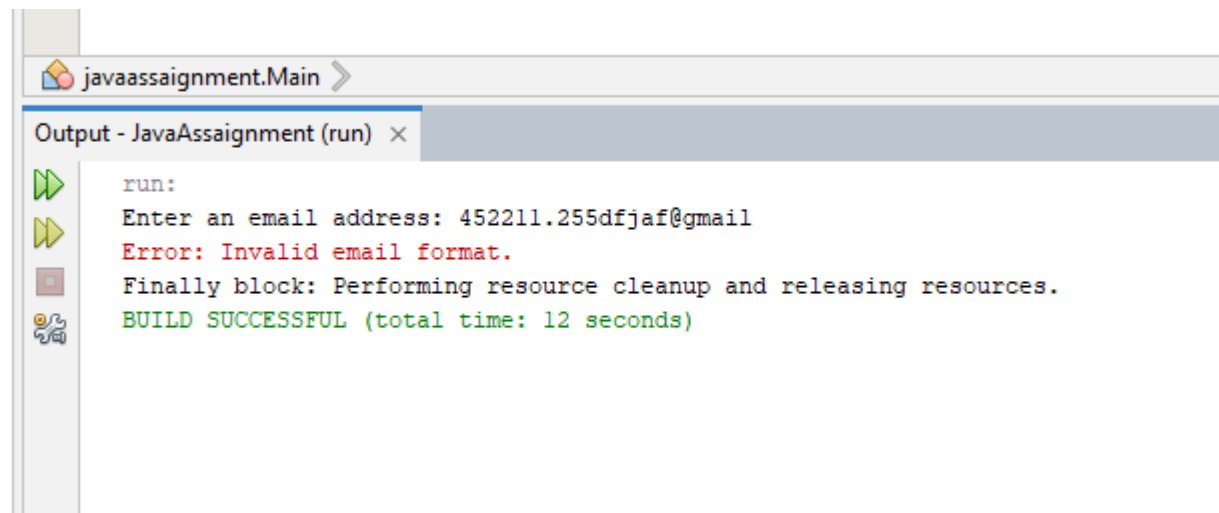
if (!pattern.matcher(email).matches()) {
    throw new IllegalArgumentException("Invalid email format.");
}
}
```

Output:



The screenshot shows an IDE window titled 'javaassignment.Main'. Below it is an 'Output - JavaAssaignment (run)' window. The output text is as follows:

```
run:
Enter an email address: imrabbihasan@gmail.com
The email address is valid.
Finally block: Performing resource cleanup and releasing resources.
BUILD SUCCESSFUL (total time: 13 seconds)
```



The screenshot shows an IDE window titled 'javaassignment.Main'. Below it is an 'Output - JavaAssaignment (run)' window. The output text is as follows:

```
run:
Enter an email address: 452211.255dfjaf@gmail
Error: Invalid email format.
Finally block: Performing resource cleanup and releasing resources.
BUILD SUCCESSFUL (total time: 12 seconds)
```

Experiment No:06

Experiment Name: Write a program to create four threads. Inside the first thread print your Dept. 10 times but wait for 2 second before printing each time. Inside the second thread print your Name 20 times. Inside the third thread print your ID 30 times. Make sure second thread gets more OS access than the first thread and the third thread starts after finishing the second thread.

Code:

```
package javaassignment;

public class Main {
    public static void main(String[] args) {
        Thread deptThread = new DeptThread();
        Thread nameThread = new NameThread();
        Thread idThread = new IDThread();

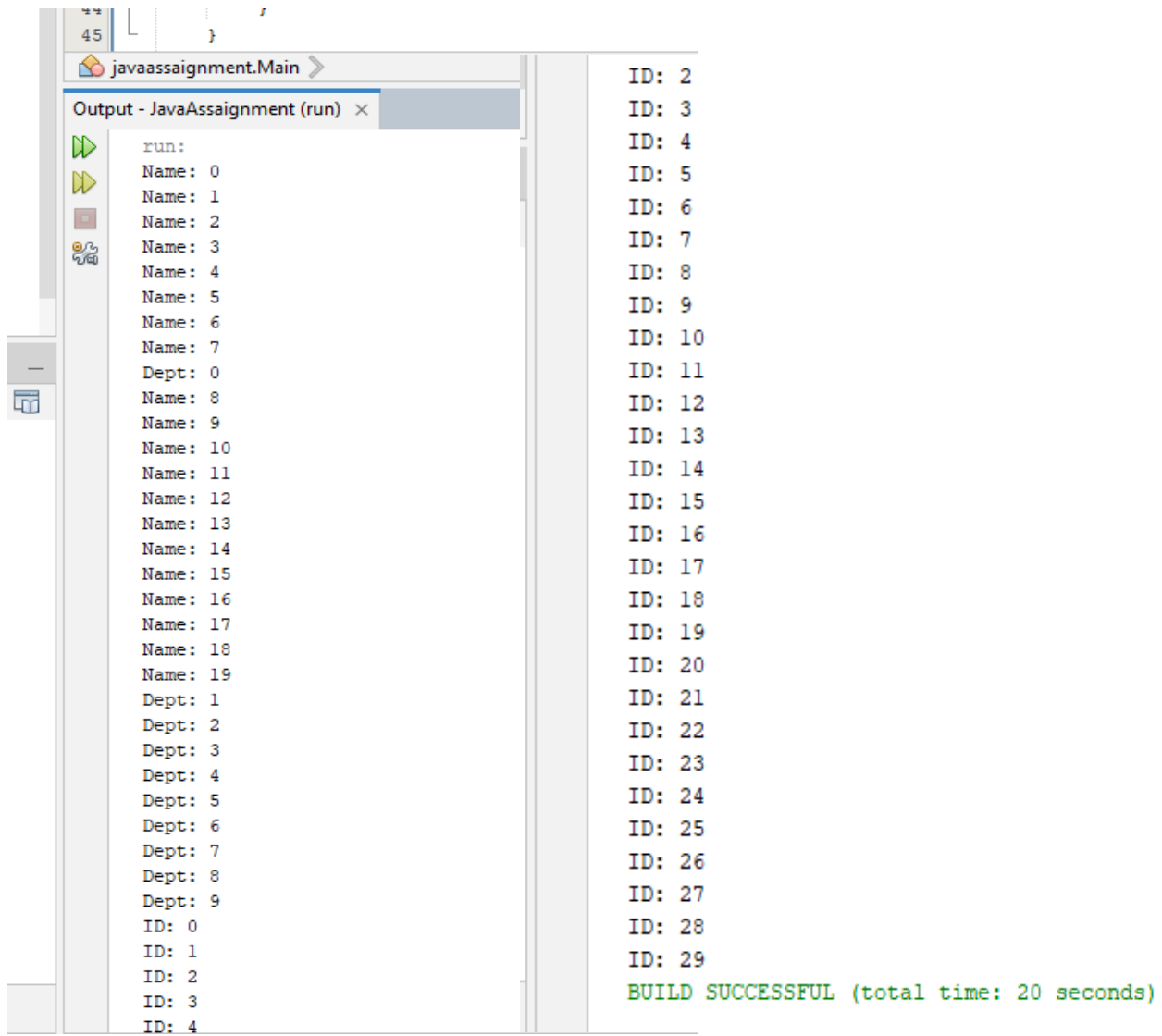
        deptThread.setPriority(Thread.MIN_PRIORITY);
        nameThread.setPriority(Thread.MAX_PRIORITY);

        deptThread.start();
        nameThread.start();

        try {
            deptThread.join();
            nameThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        idThread.start();
    }
    private static class DeptThread extends Thread {
        public void run() {
            for (int i = 0; i < 10; i++) {
                System.out.println("Dept: " + i);
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
    private static class NameThread extends Thread {
        public void run() {
            for (int i = 0; i < 20; i++) {
                System.out.println("Name: " + i);
            }
        }
    }
    private static class IDThread extends Thread {
        public void run() {
            for (int i = 0; i < 30; i++) {
                System.out.println("ID: " + i);
            }
        }
    }
}
```


Output:



The screenshot shows an IDE window titled 'javaassignment.Main'. The code editor contains a Java program with a loop that prints 'Name' and 'Dept' for IDs 0 through 19, and 'ID' for IDs 0 through 29. The output window, titled 'Output - JavaAssignment (run)', displays the results of the program execution. The output consists of two columns of text: the first column contains 'Name: 0' through 'Name: 19', 'Dept: 0' through 'Dept: 9', and 'ID: 0' through 'ID: 4'; the second column contains 'ID: 2' through 'ID: 29'. The output ends with the message 'BUILD SUCCESSFUL (total time: 20 seconds)'.

```
run:
Name: 0
Name: 1
Name: 2
Name: 3
Name: 4
Name: 5
Name: 6
Name: 7
Dept: 0
Name: 8
Name: 9
Name: 10
Name: 11
Name: 12
Name: 13
Name: 14
Name: 15
Name: 16
Name: 17
Name: 18
Name: 19
Dept: 1
Dept: 2
Dept: 3
Dept: 4
Dept: 5
Dept: 6
Dept: 7
Dept: 8
Dept: 9
ID: 0
ID: 1
ID: 2
ID: 3
ID: 4
ID: 2
ID: 3
ID: 4
ID: 5
ID: 6
ID: 7
ID: 8
ID: 9
ID: 10
ID: 11
ID: 12
ID: 13
ID: 14
ID: 15
ID: 16
ID: 17
ID: 18
ID: 19
ID: 20
ID: 21
ID: 22
ID: 23
ID: 24
ID: 25
ID: 26
ID: 27
ID: 28
ID: 29
BUILD SUCCESSFUL (total time: 20 seconds)
```

Experiment No:07

Experiment Name: Write a Java program to perform matrix multiplication using multithreading for parallel computation. Implement a method that takes two matrices as input and computes their product using multiple threads, each responsible for computing a portion of the result matrix. Ensure efficient utilization of resources and minimize thread synchronization overhead.

Code:

```
package javaassignment;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {

    private int[][] matrixA;
    private int[][] matrixB;
    private int[][] resultMatrix;
    private int numThreads;

    public Main(int[][] matrixA, int[][] matrixB, int numThreads) {
        this.matrixA = matrixA;
        this.matrixB = matrixB;
        this.numThreads = numThreads;
        this.resultMatrix = new int[matrixA.length][matrixB[0].length];
    }

    public void multiply() {
        int rowsPerThread = matrixA.length / numThreads;
        ExecutorService executor = Executors.newFixedThreadPool(numThreads);

        for (int i = 0; i < numThreads; i++) {
            int startRow = i * rowsPerThread;
            int endRow = (i == numThreads - 1) ? matrixA.length : (i + 1) * rowsPerThread;
            executor.execute(new MatrixMultiplierThread(startRow, endRow));
        }

        executor.shutdown();
        while (!executor.isTerminated()) {
            // Wait for all threads to finish
        }
    }

    private class MatrixMultiplierThread extends Thread {
        private int startRow;
        private int endRow;

        public MatrixMultiplierThread(int startRow, int endRow) {
            this.startRow = startRow;
            this.endRow = endRow;
        }

        @Override
        public void run() {
            for (int i = startRow; i < endRow; i++) {
                for (int j = 0; j < matrixB[0].length; j++) {
                    for (int k = 0; k < matrixA[0].length; k++) {
                        resultMatrix[i][j] += matrixA[i][k] * matrixB[k][j];
                    }
                }
            }
        }
    }
}
```

```

    }
    }
}

public int[][] getResultMatrix() {
    return resultMatrix;
}

public static void main(String[] args) {
    int[][] matrixA = {
        { 1, 2, 3},
        { 4, 5, 6},
        { 7, 8, 9}
    };

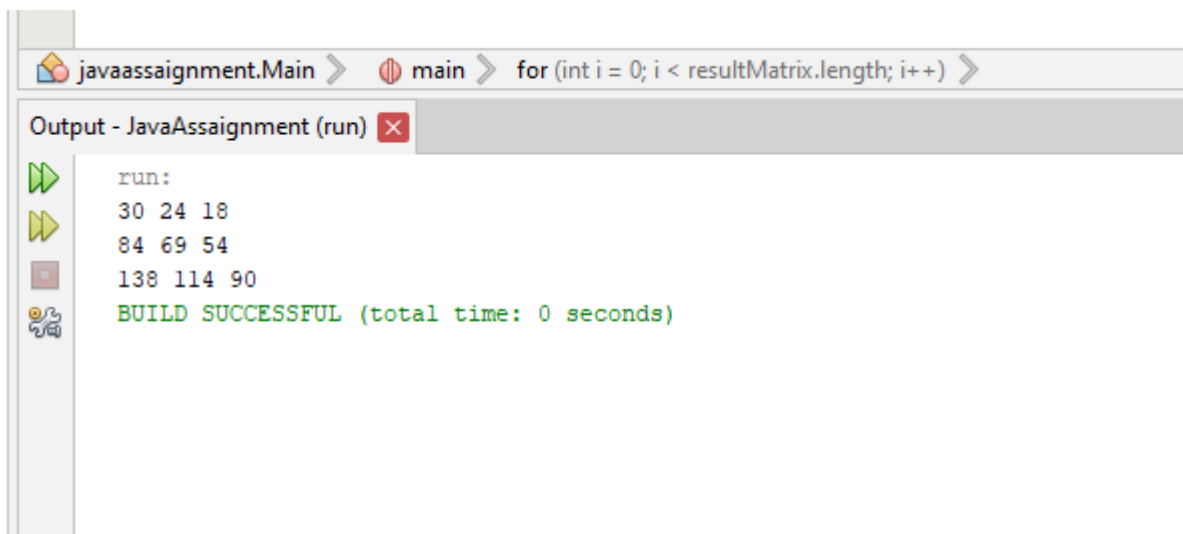
    int[][] matrixB = {
        { 9, 8, 7},
        { 6, 5, 4},
        { 3, 2, 1}
    };

    Main multiplier = new Main (matrixA, matrixB, 3);
    multiplier.multiply();

    int[][] resultMatrix = multiplier.getResultMatrix();
    for (int i = 0; i < resultMatrix.length; i++) {
        for (int j = 0; j < resultMatrix[0].length; j++) {
            System.out.print(resultMatrix[i][j] + " ");
        }
        System.out.println();
    }
}
}

```

Output:



```

javaassignment.Main > main > for (int i = 0; i < resultMatrix.length; i++) >
Output - JavaAssaignment (run)
run:
30 24 18
84 69 54
138 114 90
BUILD SUCCESSFUL (total time: 0 seconds)

```

Experiment No:08

Experiment Name: Write a Java program to compute the factorial of a given number using multi threading. Create two threads, one for computing the factorial of even numbers and the other for computing the factorial of odd numbers. Combine the results to get the final factorial.

Code:

```
package javaassignment;

class FactorialEven implements Runnable {
    private int num;
    private long result;

    public FactorialEven(int num) {
        this.num = num;
    }

    @Override
    public void run() {
        result = computeFactorialEven();
    }

    private long computeFactorialEven() {
        long fact = 1;
        for (int i = 2; i <= num; i += 2) {
            fact *= i;
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return fact;
    }

    public long getResult() {
        return result;
    }
}

class FactorialOdd implements Runnable {
    private int num;
    private long result;

    public FactorialOdd(int num) {
        this.num = num;
    }

    @Override
    public void run() {
        result = computeFactorialOdd();
    }

    private long computeFactorialOdd() {
        long fact = 1;
        for (int i = 1; i <= num; i += 2) {
            fact *= i;
            try {
                Thread.sleep(100);
            }
        }
    }
}
```

```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    return fact;
}

public long getResult() {
    return result;
}
}

public class Main {
    public static void main(String[] args) {
        int num = 5;
        FactorialEven fe = new FactorialEven(num);
        FactorialOdd fo = new FactorialOdd(num);

        Thread t1 = new Thread(fe);
        Thread t2 = new Thread(fo);

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

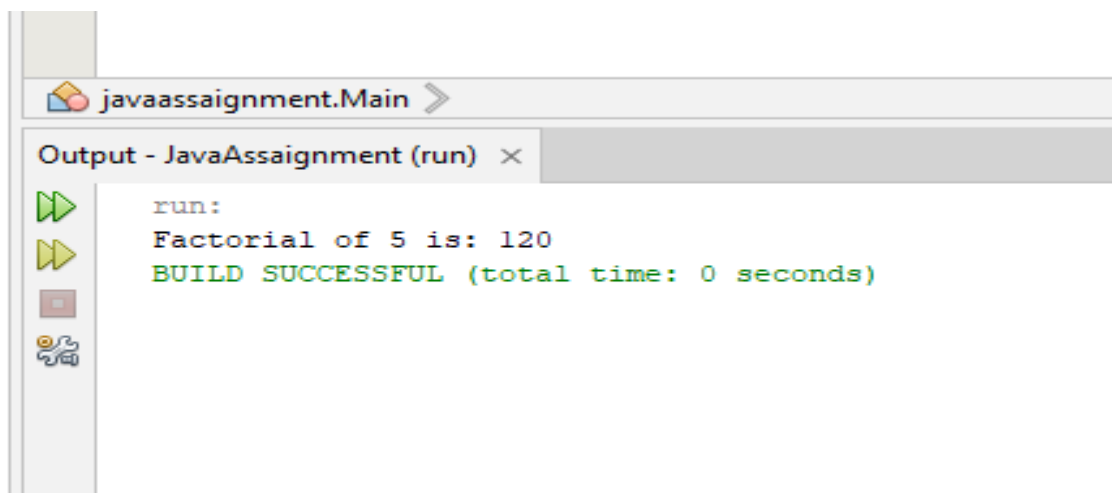
        long evenFact = fe.getResult();
        long oddFact = fo.getResult();

        long finalFact = evenFact * oddFact;

        System.out.println("Factorial of " + num + " is: " + finalFact);
    }
}

```

Output:



Experiment No:09

Experiment Name: Write a Java program that creates two threads, one for printing uppercase letters from A to Z and the other for printing lowercase letters from a to z. Ensure that the letters are printed in sequence, with uppercase letters followed by lowercase letters.

Code:

```
package javaassignment;

class UpperCaseThread extends Thread {
    private final Object lock;

    public UpperCaseThread(Object lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        synchronized (lock) {
            for (char ch = 'A'; ch <= 'Z'; ch++) {
                System.out.print(ch);
                lock.notify();
                try {
                    lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

class LowerCaseThread extends Thread {
    private final Object lock;

    public LowerCaseThread(Object lock) {
        this.lock = lock;
    }

    @Override
    public void run() {
        synchronized (lock) {
            for (char ch = 'a'; ch <= 'z'; ch++) {
                lock.notify();
                try {
                    lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.print(ch);
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Object lock = new Object();
        UpperCaseThread upperCaseThread = new UpperCaseThread(lock);
```

```
LowerCaseThread lowerCaseThread = new LowerCaseThread(lock);

upperCaseThread.start();
lowerCaseThread.start();

try {
    upperCaseThread.join();
    lowerCaseThread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
```

Output:



Experiment No: 10

Experiment Name: Write a Java program that calculates the sum of all numbers from 1 to 100 using multiple threads. Divide the range of numbers into equal segments and assign each thread to compute the sum of a segment. Then, combine the results from all threads to get the final sum.

Code:

```
package javaassignment;

class SumThread extends Thread {
    private int start;
    private int end;
    private long total;

    public SumThread(int start, int end) {
        this.start = start;
        this.end = end;
        this.total = 0;
    }

    public void run() {
        for (int i = start; i <= end; i++) {
            total += i;
        }
    }

    public long getTotal() {
        return total;
    }
}

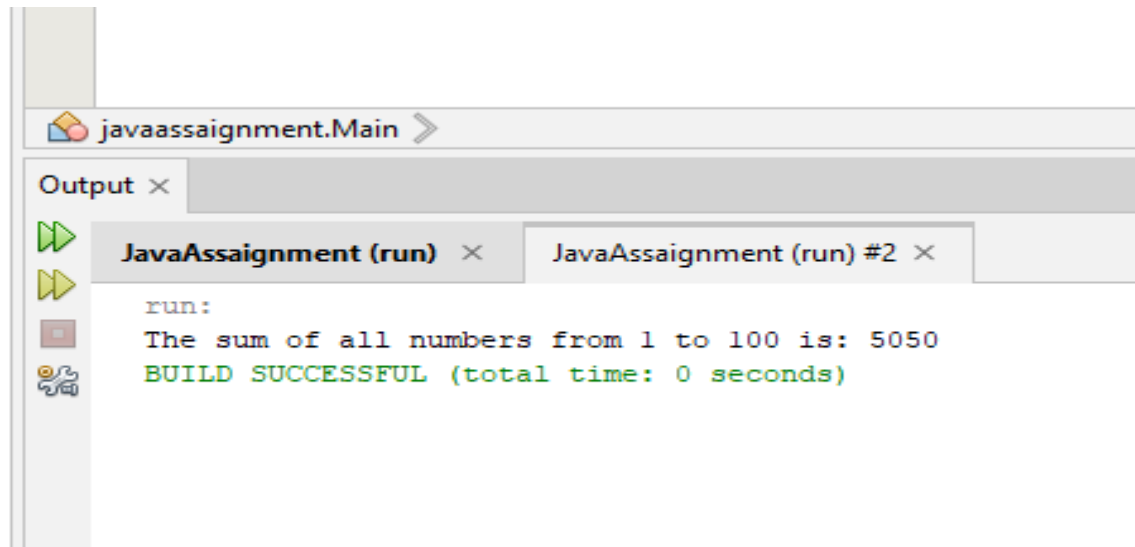
public class Main{
    public static void main(String[] args) {
        int numThreads = 4;
        int range = 100;
        int segmentSize = range / numThreads;

        SumThread[] threads = new SumThread[numThreads];
        for (int i = 0; i < numThreads; i++) {
            int start = i * segmentSize + 1;
            int end = (i == numThreads - 1) ? range : start + segmentSize - 1;
            threads[i] = new SumThread(start, end);
            threads[i].start();
        }

        long finalTotal = 0;
        for (SumThread thread : threads) {
            try {
                thread.join();
                finalTotal += thread.getTotal();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("The sum of all numbers from 1 to 100 is: " + finalTotal);
    }
}
```


Output:



Experiment No:11

Experiment Name : Write a program that takes a paragraph of text as input and counts the occurrences of each word. Additionally, identify the five most common words and display them along with their frequencies.

Code:

```
package javaassignment;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a paragraph of text:");
        String paragraph = scanner.nextLine();

        Map<String, Integer> wordFrequencies = countWordFrequencies(paragraph);
        printWordFrequencies(wordFrequencies);
        printTop5MostCommonWords(wordFrequencies);
    }

    private static Map<String, Integer> countWordFrequencies(String paragraph) {
        Map<String, Integer> wordFrequencies = new HashMap<>();
        String[] words = paragraph.split("\\s+");

        for (String word : words) {
            word = word.toLowerCase();
            if (wordFrequencies.containsKey(word)) {
                int count = wordFrequencies.get(word) + 1;
                wordFrequencies.put(word, count);
            } else {
                wordFrequencies.put(word, 1);
            }
        }
    }
}
```

```

    }
}

return wordFrequencies;
}

private static void printWordFrequencies(Map<String, Integer> wordFrequencies) {
    System.out.println("Word Frequencies:");
    for (Map.Entry<String, Integer> entry : wordFrequencies.entrySet()) {
        System.out.println(entry.getKey() + ": " + entry.getValue());
    }
}

private static void printTop5MostCommonWords(Map<String, Integer> wordFrequencies) {
    int maxCount = 0;
    String[] top5Words = new String[5];

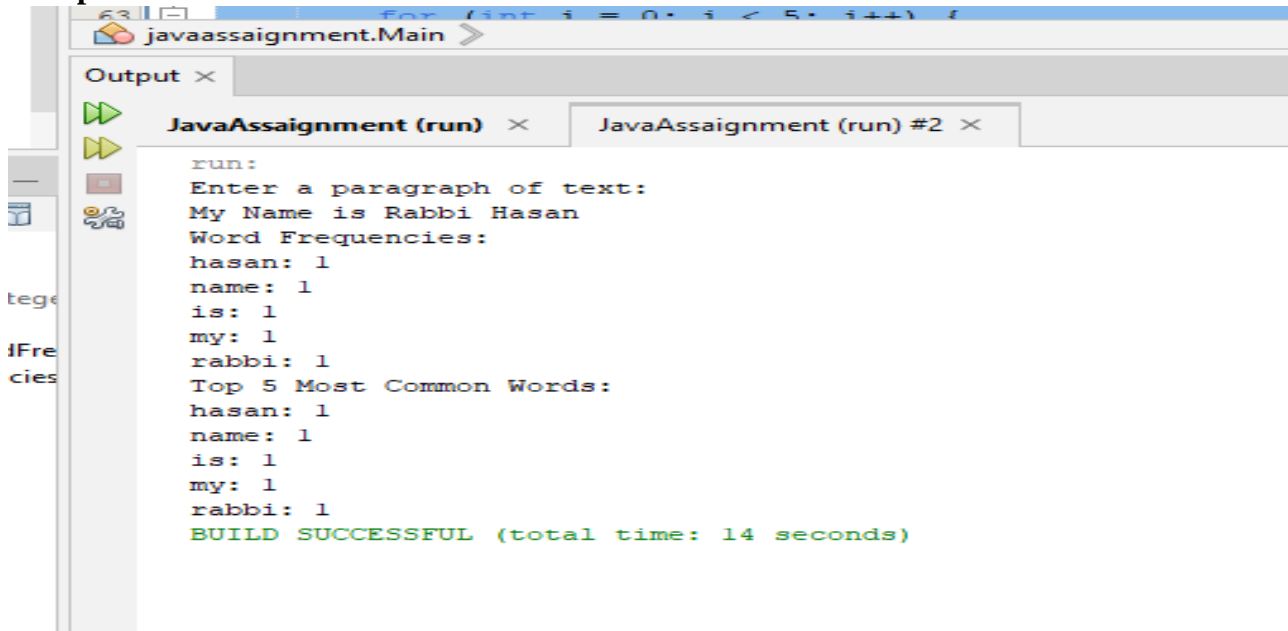
    for (Map.Entry<String, Integer> entry : wordFrequencies.entrySet()) {
        if (entry.getValue() > maxCount) {
            maxCount = entry.getValue();
        }
    }

    int count = 0;
    for (Map.Entry<String, Integer> entry : wordFrequencies.entrySet()) {
        if (entry.getValue() == maxCount) {
            top5Words[count++] = entry.getKey();
            if (count == 5) {
                break;
            }
        }
    }

    System.out.println("Top 5 Most Common Words:");
    for (int i = 0; i < 5; i++) {
        System.out.println(top5Words[i] + ": " + maxCount);
    }
}
}

```

Output:



```

run:
Enter a paragraph of text:
My Name is Rabbi Hasan
Word Frequencies:
hasan: 1
name: 1
is: 1
my: 1
rabbi: 1
Top 5 Most Common Words:
hasan: 1
name: 1
is: 1
my: 1
rabbi: 1
BUILD SUCCESSFUL (total time: 14 seconds)

```

Experiment No:12

Experiment Name: Write a program that takes a sentence and a word as input and finds whether the word is present as a substring in the sentence.

Code:

```
package javaassignment;

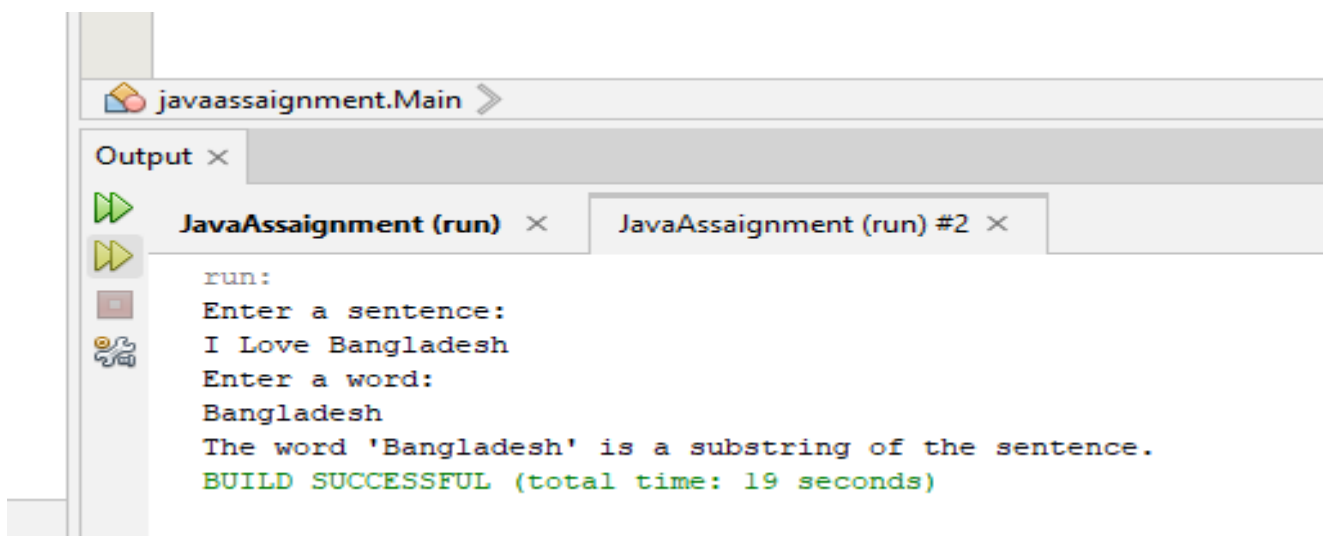
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a sentence:");
        String sentence = scanner.nextLine();
        System.out.println("Enter a word:");
        String word = scanner.nextLine();

        boolean isSubstring = isSubstring(sentence, word);
        if (isSubstring) {
            System.out.println("The word " + word + " is a substring of the sentence.");
        } else {
            System.out.println("The word " + word + " is not a substring of the sentence.");
        }
    }

    private static boolean isSubstring(String sentence, String word) {
        sentence = sentence.toLowerCase();
        word = word.toLowerCase();
        return sentence.contains(word);
    }
}
```

Output:



```
javaassignment.Main >
Output x
JavaAssaignment (run) x JavaAssaignment (run) #2 x
run:
Enter a sentence:
I Love Bangladesh
Enter a word:
Bangladesh
The word 'Bangladesh' is a substring of the sentence.
BUILD SUCCESSFUL (total time: 19 seconds)
```

Experiment No:13

Experiment Name: Write a program that takes a sentence as input and capitalizes the first letter of each word. For example, "hello world" should become "Hello World".

Code:

```
package javaassignment;

import java.util.Scanner;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a sentence:");
        String sentence = scanner.nextLine();

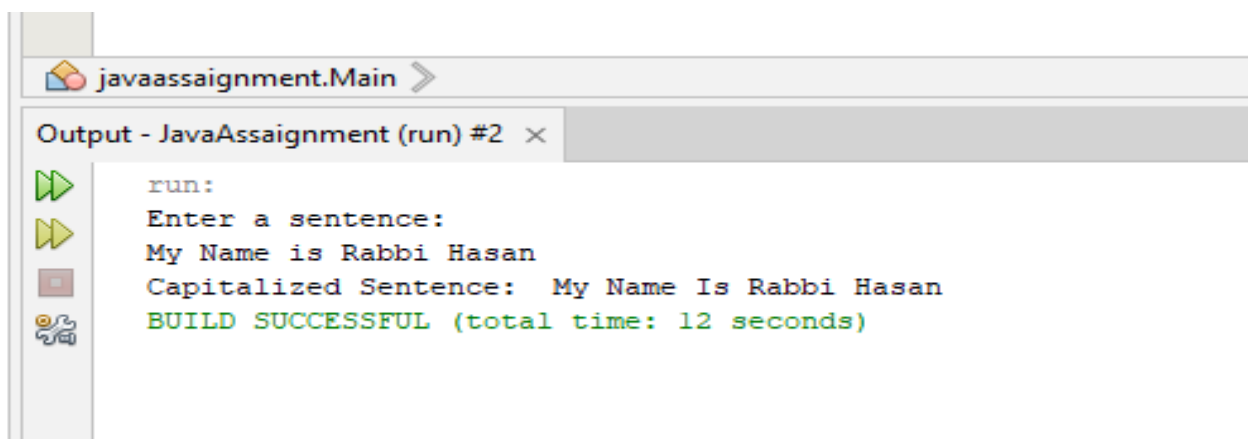
        String capitalizedSentence = capitalizeFirstLetter(sentence);
        System.out.println("Capitalized Sentence: " + capitalizedSentence);
    }

    private static String capitalizeFirstLetter(String sentence) {
        StringBuilder sb = new StringBuilder();
        boolean capitalizeNext = true;

        for (char c : sentence.toCharArray()) {
            if (Character.isWhitespace(c)) {
                capitalizeNext = true;
            } else if (capitalizeNext) {
                sb.append(" ");
                sb.append(Character.toUpperCase(c));
                capitalizeNext = false;
            } else {
                sb.append(c);
            }
        }

        return sb.toString();
    }
}
```

Output:



Experiment No:14

Experiment Name: Create a function that takes a sentence as input and reverses the order of words in it. For example, "Hello world" should become "world Hello".

Code:

```
package javaassignment;

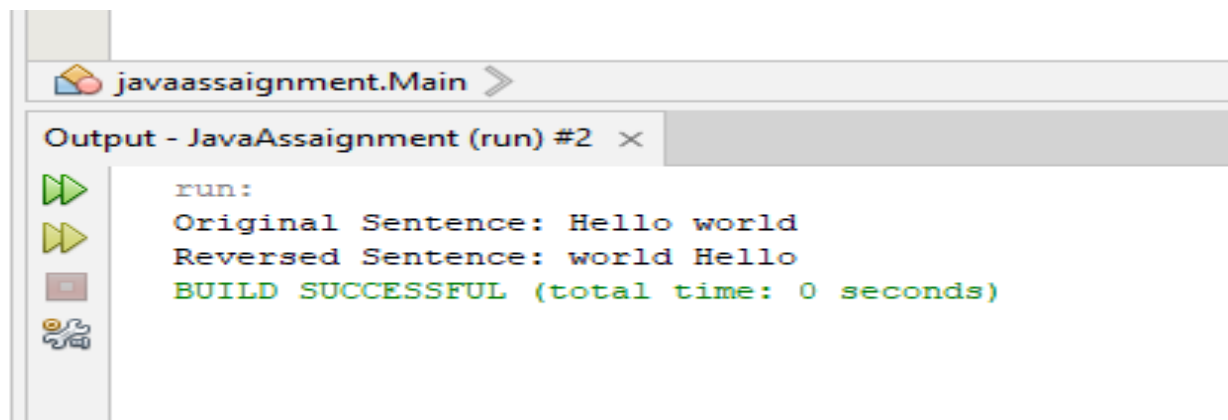
public class Main {
    public static void main(String[] args) {
        String sentence = "Hello world";
        String reversedSentence = reverseWords(sentence);
        System.out.println("Original Sentence: " + sentence);
        System.out.println("Reversed Sentence: " + reversedSentence);
    }

    private static String reverseWords(String sentence) {
        String[] words = sentence.split(" ");
        StringBuilder sb = new StringBuilder();

        for (int i = words.length - 1; i >= 0; i--) {
            if (i != words.length - 1) {
                sb.append(" ");
            }
            sb.append(words[i]);
        }

        return sb.toString();
    }
}
```

Output:



Experiment No:15

Experiment Name: Write a program that counts the occurrences of each character in a given string and displays the count for each character.

Code:

```
package javaassignment;

import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        String input = "Hello, World!";
        countCharacterFrequency(input);
    }

    private static void countCharacterFrequency(String input) {
        Map<Character, Integer> frequencyMap = new HashMap<>();

        for (char c : input.toCharArray()) {
            if (frequencyMap.containsKey(c)) {
                frequencyMap.put(c, frequencyMap.get(c) + 1);
            } else {
                frequencyMap.put(c, 1);
            }
        }

        System.out.println("Character Frequency:");
        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

Output:

```
run:
Character Frequency:
: 1
!: 1
r: 1
d: 1
e: 1
W: 1
H: 1
l: 3
,: 1
o: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment No:16

Experiment Name: Write a program that takes the first name and last name of a person as input and concatenates them to form a full name.

Code:

```
package javaassignment;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

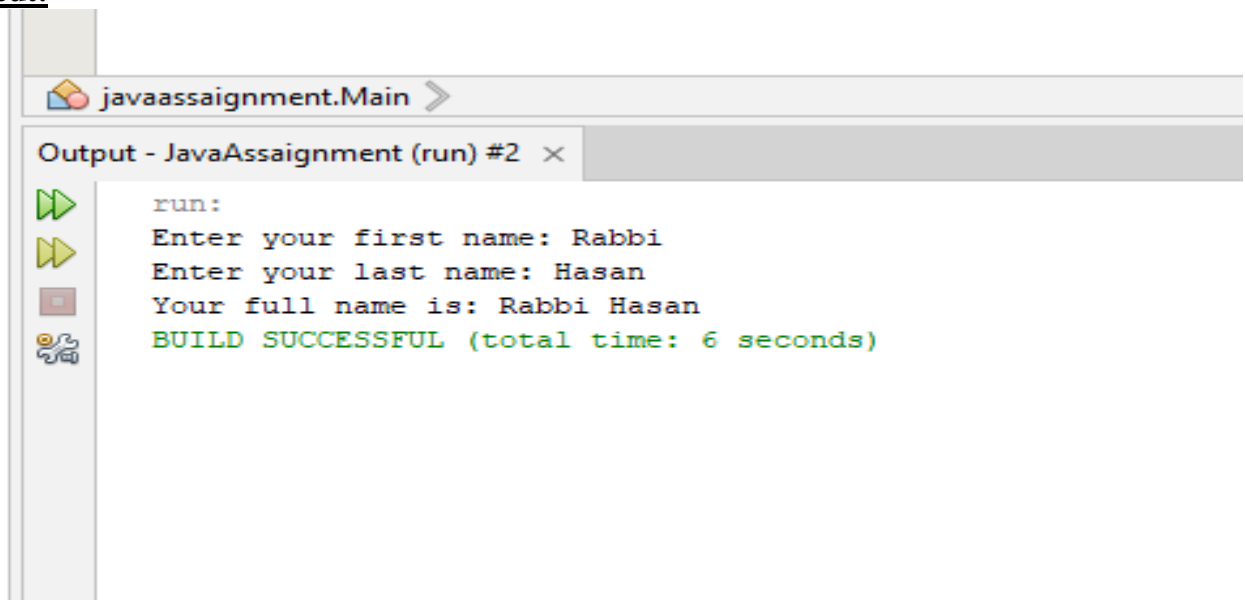
        System.out.print("Enter your first name: ");
        String firstName = scanner.nextLine();

        System.out.print("Enter your last name: ");
        String lastName = scanner.nextLine();

        String fullName = concatenateFullName(firstName, lastName);
        System.out.println("Your full name is: " + fullName);
    }

    private static String concatenateFullName(String firstName, String lastName) {
        return firstName + " " + lastName;
    }
}
```

Output:



Experiment No:17

Experiment Name: Given the following strings : A = " The early bird catches the worm " B = "Patience is a virtue" Your task is to extract the word "early" from A and "virtue" from B. Then, concatenate these two words to form a sentence. After that, capitalize the sentence and find the last occurrence of the letter 'V' from the capitalized sentence. Perform all of these tasks using proper String class function.

Code:

```
package javaassignment;

import java.util.Scanner;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String A = " The early bird catches the worm ";
        String B = "Patience is a virtue";

        String wordA = extractWord(A, "early");
        String wordB = extractWord(B, "virtue");

        String sentence = concatenateWords(wordA, wordB);
        String capitalizedSentence = capitalize(sentence);

        int lastVIndex = findLastV(capitalizedSentence);

        System.out.println("Sentence: " + sentence);
        System.out.println("Capitalized Sentence: " + capitalizedSentence);
        System.out.println("Last occurrence of 'V' in the capitalized sentence: " + lastVIndex);
    }

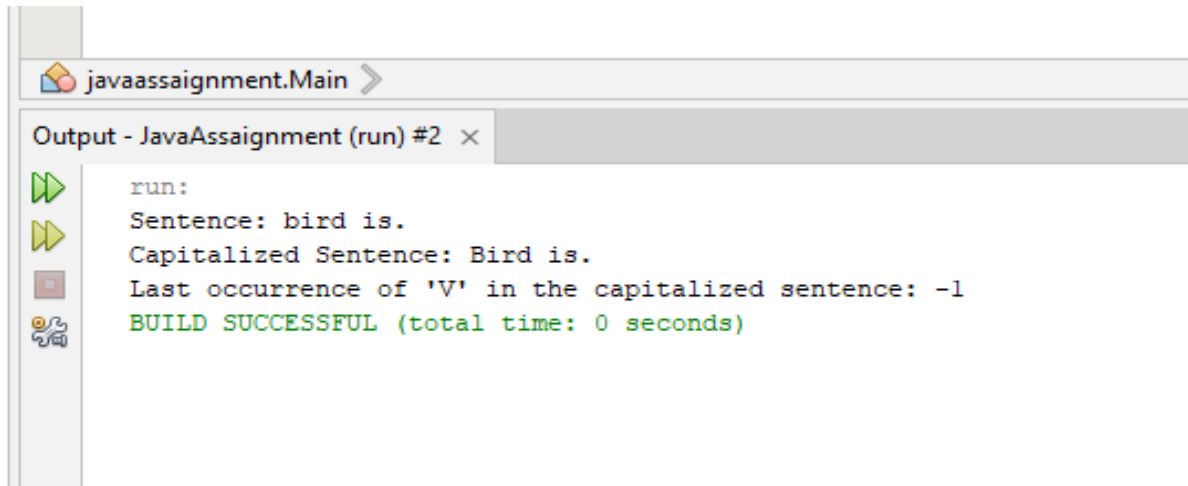
    private static String extractWord(String input, String word) {
        return input.split(" ")[input.split(" ").length / 2 + input.split(word).length / 2 - 1];
    }

    private static String concatenateWords(String wordA, String wordB) {
        return wordA + " " + wordB + ".";
    }

    private static String capitalize(String input) {
        return input.substring(0, 1).toUpperCase() + input.substring(1);
    }

    private static int findLastV(String input) {
        int index = input.lastIndexOf('V');
        while (index > 0 && Character.isUpperCase(input.charAt(index - 1))) {
            index = input.lastIndexOf('V', index - 1);
        }
        return index;
    }
}
```

Output:



Experiment No:18

Experiment Name: You are developing a ticket booking system for a movie theater. Design a Java program that uses a Queue to manage ticket requests, where each request represents a customer wanting to book a ticket. Implement methods to add new booking requests, process bookings in the order they were received, and display the status of ticket bookings.

Code:

```
package javaassignment;

import java.util.LinkedList;
import java.util.Queue;

public class Main {
    private Queue<TicketRequest> requests;

    public Main () {
        requests = new LinkedList<>();
    }

    public void addRequest(TicketRequest request) {
        requests.add(request);
    }

    public void processRequests() {
        while (!requests.isEmpty()) {
            TicketRequest request = requests.poll();
            processRequest(request);
        }
    }

    public void processRequest(TicketRequest request) {
        System.out.println("Processing request for seat " + request.getSeat() + " for showtime " +
request.getShowtime());
        // Add code here to process the ticket request, such as checking availability, charging the customer, etc.
    }

    public void displayStatus() {
        System.out.println("Ticket booking status:");
        for (TicketRequest request : requests) {
```

```

        System.out.println("- Request for seat " + request.getSeat() + " for showtime " +
request.getShowtime());
    }
}

public static void main(String[] args) {
    Main system = new Main ();

    system.addRequest(new TicketRequest("A1", "10:00 AM"));
    system.addRequest(new TicketRequest("B2", "1:00 PM"));
    system.addRequest(new TicketRequest("C3", "4:00 PM"));

    system.displayStatus();

    system.processRequests();

    system.displayStatus();
}
}

class TicketRequest {
    private String seat;
    private String showtime;

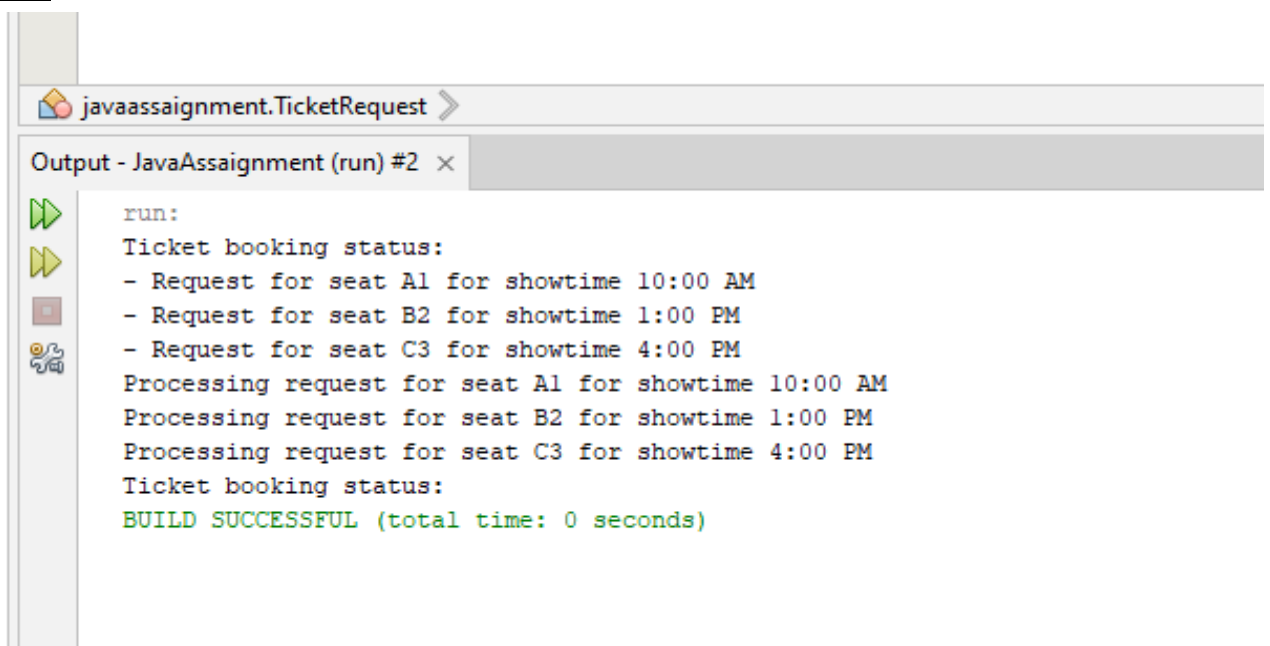
    public TicketRequest(String seat, String showtime) {
        this.seat = seat;
        this.showtime = showtime;
    }

    public String getSeat() {
        return seat;
    }

    public String getShowtime() {
        return showtime;
    }
}

```

Output:



```

run:
Ticket booking status:
- Request for seat A1 for showtime 10:00 AM
- Request for seat B2 for showtime 1:00 PM
- Request for seat C3 for showtime 4:00 PM
Processing request for seat A1 for showtime 10:00 AM
Processing request for seat B2 for showtime 1:00 PM
Processing request for seat C3 for showtime 4:00 PM
Ticket booking status:
BUILD SUCCESSFUL (total time: 0 seconds)

```

Experiment No:19

Experiment Name: Create a class named Car with properties such as price (double), brand (String), and speed (double). These properties will be initialized when an object of the class is created. Create five objects of the Car class and add them to an ArrayList. Display the cars whose price is over 2000000 takas. Complete the program.

Code:

```
package javaassignment;
import java.util.ArrayList;

public class Car {
    private double price;
    private String brand;
    private double speed;

    public Car(double price, String brand, double speed) {
        this.price = price;
        this.brand = brand;
        this.speed = speed;
    }

    public double getPrice() {
        return price;
    }

    public String getBrand() {
        return brand;
    }

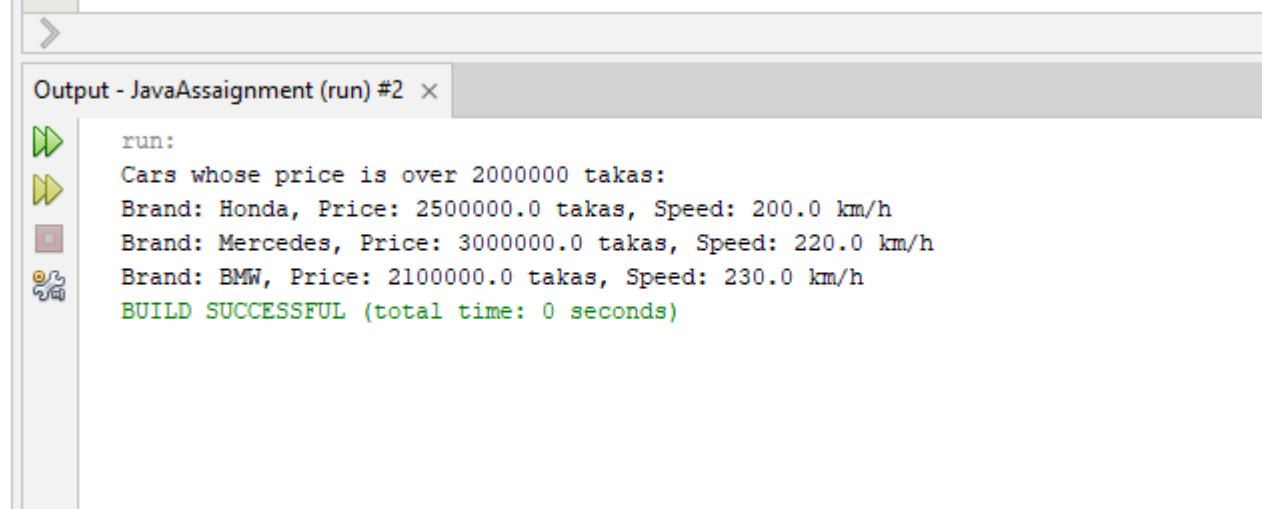
    public double getSpeed() {
        return speed;
    }

    public static void main(String[] args) {
        ArrayList<Car> cars = new ArrayList<>();

        cars.add(new Car(1500000, "Toyota", 180));
        cars.add(new Car(2500000, "Honda", 200));
        cars.add(new Car(3000000, "Mercedes", 220));
        cars.add(new Car(1800000, "Ford", 190));
        cars.add(new Car(2100000, "BMW", 230));

        System.out.println("Cars whose price is over 2000000 takas:");
        for (Car car : cars) {
            if (car.getPrice() > 2000000) {
                System.out.println("Brand: " + car.getBrand() + ", Price: " + car.getPrice() + " takas, Speed: " +
car.getSpeed() + " km/h");
            }
        }
    }
}
```

Output:



The screenshot shows an IDE's output window titled "Output - JavaAssaignment (run) #2". On the left side of the window, there is a vertical toolbar with four icons: a green double arrow (run), a yellow double arrow (debug), a red square (stop), and a magnifying glass (search). The output text is as follows:

```
run:
Cars whose price is over 2000000 takas:
Brand: Honda, Price: 2500000.0 takas, Speed: 200.0 km/h
Brand: Mercedes, Price: 3000000.0 takas, Speed: 220.0 km/h
Brand: BMW, Price: 2100000.0 takas, Speed: 230.0 km/h
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment No:20

Experiment Name: Write a basic Java program for managing student IDs and their grades in a gradebook system. Implement methods to add new student IDs, remove existing student IDs, display the list of student IDs, and store/display grades for each student. Utilize simple data structures like arrays for storing student IDs and grades.

Code:

```
package javaassignment;
import java.util.ArrayList;

import java.util.Scanner;

public class Main {
    private String[] studentIds;
    private double[][] grades;
    private int numStudents;

    public Main(int capacity) {
        studentIds = new String[capacity];
        grades = new double[capacity][];
        numStudents = 0;
    }

    public void addStudentId(String id) {
        if (numStudents < studentIds.length) {
            studentIds[numStudents] = id;
            grades[numStudents] = new double[3]; // assume 3 grades per student
            numStudents++;
            System.out.println("Student ID added: " + id);
        } else {
            System.out.println("Gradebook is full. Cannot add more students.");
        }
    }

    public void removeStudentId(String id) {
        for (int i = 0; i < numStudents; i++) {
            if (studentIds[i].equals(id)) {
                // shift elements to fill the gap
                for (int j = i; j < numStudents - 1; j++) {
                    studentIds[j] = studentIds[j + 1];
                    grades[j] = grades[j + 1];
                }
                numStudents--;
                System.out.println("Student ID removed: " + id);
                return;
            }
        }
        System.out.println("Student ID not found: " + id);
    }

    public void displayStudentIds() {
        System.out.println("Student IDs:");
        for (int i = 0; i < numStudents; i++) {
            System.out.println(studentIds[i]);
        }
    }

    public void storeGrade(String id, double grade, int index) {
```

```

        for (int i = 0; i < numStudents; i++) {
            if (studentIds[i].equals(id)) {
                grades[i][index] = grade;
                System.out.println("Grade stored for " + id + ": " + grade);
                return;
            }
        }
        System.out.println("Student ID not found: " + id);
    }

    public void displayGrades(String id) {
        for (int i = 0; i < numStudents; i++) {
            if (studentIds[i].equals(id)) {
                System.out.println("Grades for " + id + ":");
                for (int j = 0; j < grades[i].length; j++) {
                    System.out.println("Grade " + (j + 1) + ": " + grades[i][j]);
                }
                return;
            }
        }
        System.out.println("Student ID not found: " + id);
    }

    public static void main(String[] args) {
        Main gradebook = new Main(5);

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("1. Add student ID");
            System.out.println("2. Remove student ID");
            System.out.println("3. Display student IDs");
            System.out.println("4. Store grade");
            System.out.println("5. Display grades");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");
            int option = scanner.nextInt();

            switch (option) {
                case 1:
                    System.out.print("Enter student ID: ");
                    String id = scanner.next();
                    gradebook.addStudentId(id);
                    break;
                case 2:
                    System.out.print("Enter student ID: ");
                    id = scanner.next();
                    gradebook.removeStudentId(id);
                    break;
                case 3:
                    gradebook.displayStudentIds();
                    break;
                case 4:
                    System.out.print("Enter student ID: ");
                    id = scanner.next();
                    System.out.print("Enter grade: ");
                    double grade = scanner.nextDouble();
                    System.out.print("Enter grade index (0-2): ");
                    int index = scanner.nextInt();
                    gradebook.storeGrade(id, grade, index);

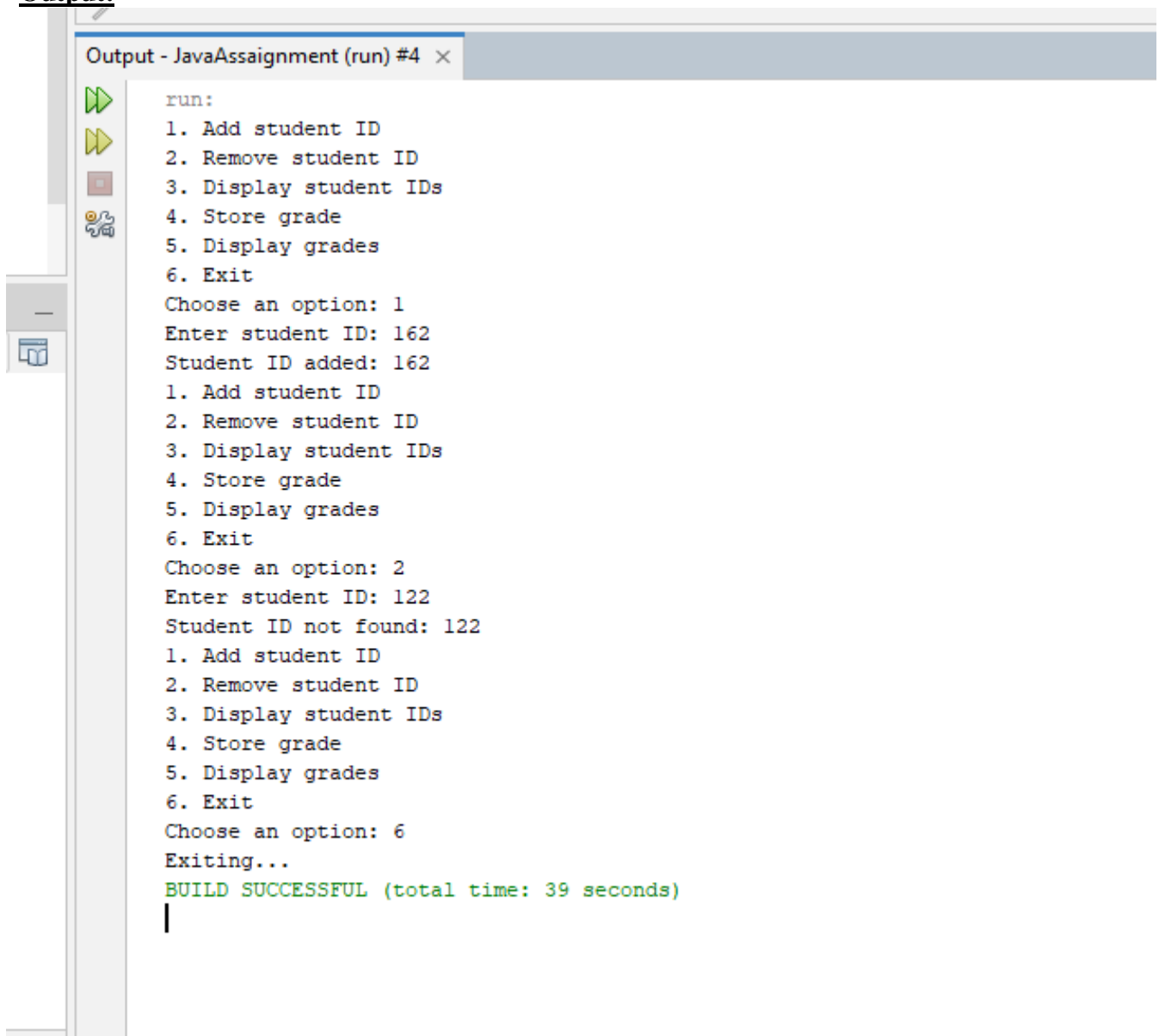
```

```

        break;
    case 5:
        System.out.print("Enter student ID: ");
        id = scanner.next();
        gradebook.displayGrades(id);
        break;
    case 6:
        System.out.println("Exiting...");
        return;
    default:
        System.out.println("Invalid option. Try again.");
    }
}
}
}

```

Output:



```

run:
1. Add student ID
2. Remove student ID
3. Display student IDs
4. Store grade
5. Display grades
6. Exit
Choose an option: 1
Enter student ID: 162
Student ID added: 162
1. Add student ID
2. Remove student ID
3. Display student IDs
4. Store grade
5. Display grades
6. Exit
Choose an option: 2
Enter student ID: 122
Student ID not found: 122
1. Add student ID
2. Remove student ID
3. Display student IDs
4. Store grade
5. Display grades
6. Exit
Choose an option: 6
Exiting...
BUILD SUCCESSFUL (total time: 39 seconds)
|

```

Experiment No:21

Experiment Name: Create a class named Student. Write a program to insert 10 Student objects in a Stack list. Now take user input for a variable named “menu”. If menu is 1 then insert another Student object. If menu is 2, delete the top Student object from the stack list. If menu is 3, just output the top Student object. Use proper stack list methods.

Code:

```
package javaassignment;
import java.util.ArrayList;
import java.util.Stack;
import java.util.Scanner;

class Student {
    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

public class Main {
    public static void main(String[] args) {
        Stack<Student> stack = new Stack<>();
        for (int i = 0; i < 10; i++) {
            stack.push(new Student("Student " + (i + 1), 20 + i));
        }

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("Menu:");
            System.out.println("1. Insert a new Student object");
            System.out.println("2. Delete the top Student object");
            System.out.println("3. Display the top Student object");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");
            int menu = scanner.nextInt();

            switch (menu) {
                case 1:
                    System.out.print("Enter student name: ");
                    String name = scanner.next();
                    System.out.print("Enter student age: ");
                    int age = scanner.nextInt();
                    stack.push(new Student(name, age));
                    System.out.println("Student object inserted successfully!");
                    break;
                case 2:
                    if (!stack.isEmpty()) {

```

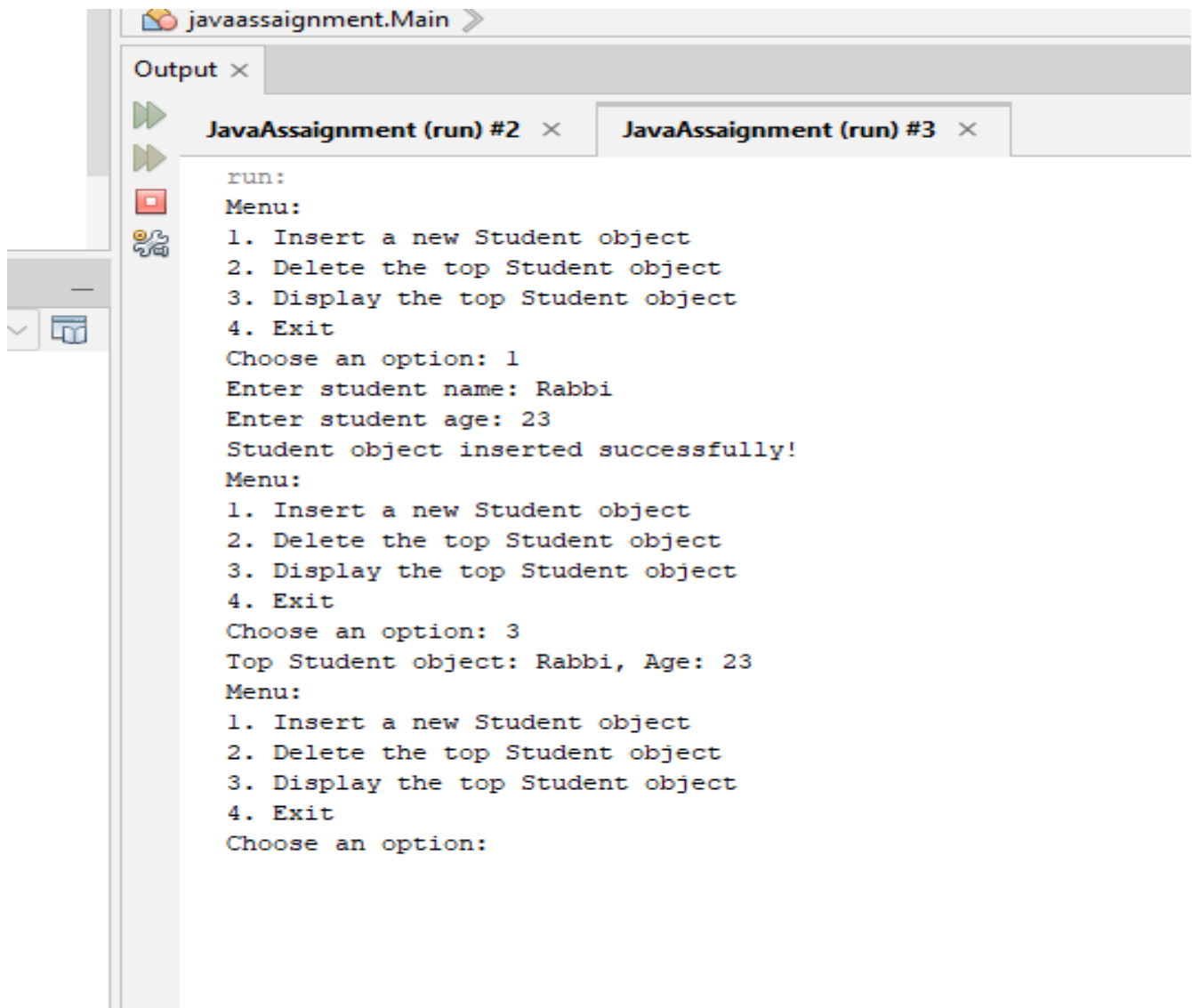


```

        stack.pop();
        System.out.println("Top Student object deleted successfully!");
    } else {
        System.out.println("Stack is empty!");
    }
    break;
case 3:
    if (!stack.isEmpty()) {
        Student topStudent = stack.peek();
        System.out.println("Top Student object: " + topStudent.getName() + ", Age: " +
topStudent.getAge());
    } else {
        System.out.println("Stack is empty!");
    }
    break;
case 4:
    System.out.println("Exiting...");
    return;
default:
    System.out.println("Invalid option. Try again.");
}
}
}
}
}

```

Output:



```

run:
Menu:
1. Insert a new Student object
2. Delete the top Student object
3. Display the top Student object
4. Exit
Choose an option: 1
Enter student name: Rabbi
Enter student age: 23
Student object inserted successfully!
Menu:
1. Insert a new Student object
2. Delete the top Student object
3. Display the top Student object
4. Exit
Choose an option: 3
Top Student object: Rabbi, Age: 23
Menu:
1. Insert a new Student object
2. Delete the top Student object
3. Display the top Student object
4. Exit
Choose an option:

```

Experiment No:22

Experiment Name: Write a Java program to remove duplicates from a list of strings. Implement a method remove duplicates that takes a List of strings as input and removes any duplicate elements, keeping only the first occurrence of each element.

Code:

```
package javaassignment;

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("apple");
        list.add("banana");
        list.add("apple");
        list.add("orange");
        list.add("banana");
        list.add("grape");
        System.out.println("Original list: " + list);
        List<String> uniqueList = removeDuplicates(list);
        System.out.println("List after removing duplicates: " + uniqueList);
    }
    public static List<String> removeDuplicates(List<String> list) {
        List<String> uniqueList = new ArrayList<>();

        for (String str : list) {
            if (!uniqueList.contains(str)) {
                uniqueList.add(str);
            }
        }
        return uniqueList;
    }
}
```

Output:

