JAVA ASSIGNMENTS

1) **Write a program to cover all Java OOPS concepts. Topics need to cover:**

   a) **Class and Object**
   b) **Class constructor**
   c) **Polymorphism**
   d) **Method overloading**
   e) **Method overriding**
   f) **Inheritance**
   g) **Interface**
   h) **Abstract class**
   i) **Abstraction and Encapsulation**
   j) **Composition and Aggregation**
   k) **Generalization and Specialization**

Code:

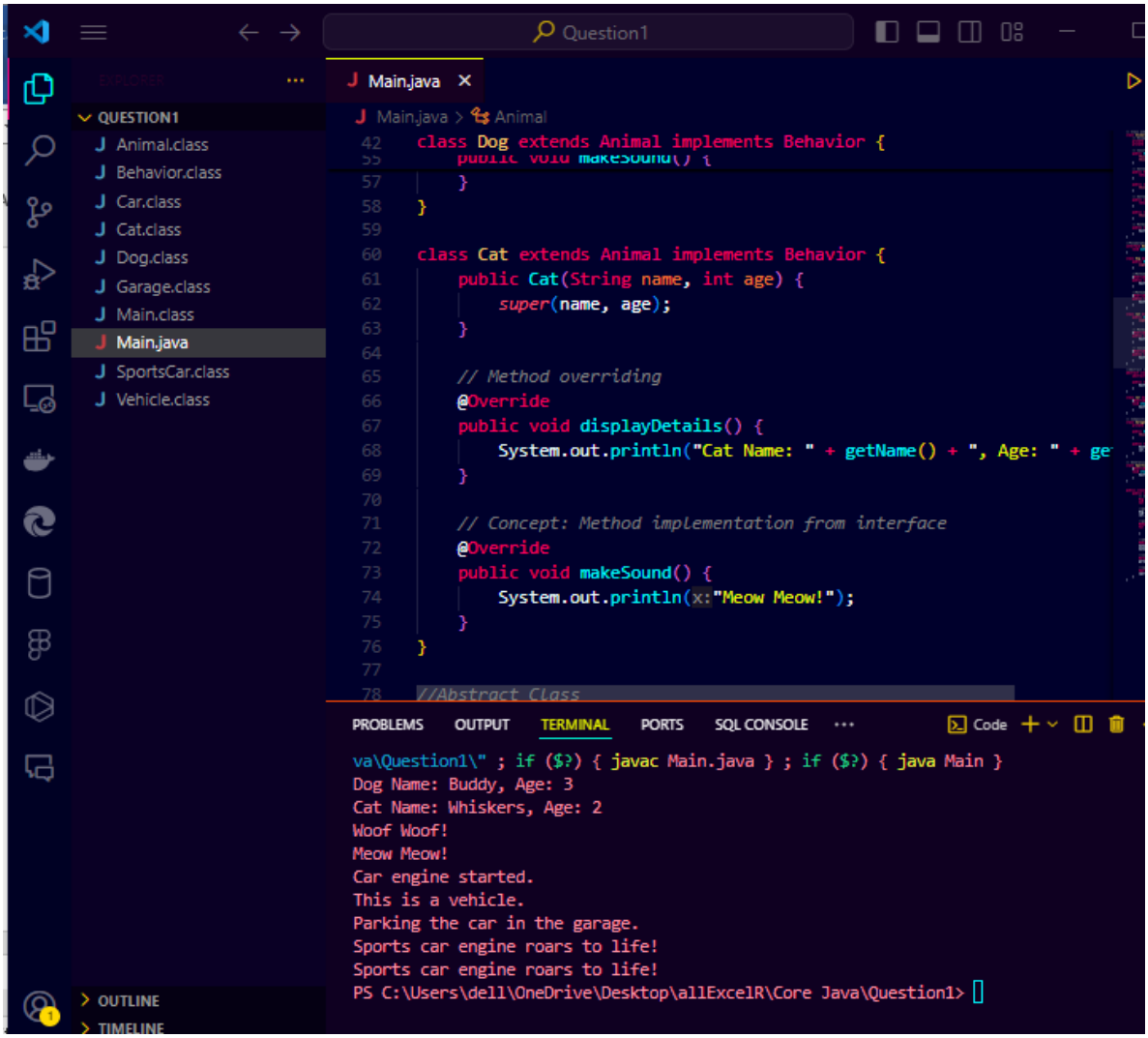
```
// Class and Object
class Animal {
    // Encapsulation: Private fields and public access getter/setter methods
    private String name;
    private int age;

    // Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter and Setter method
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Getter and Setter for age
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
```

```
    }

    // Method to display details ----Abstraction
    public void displayDetails() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

//Polymorphism
interface Behavior {
    void makeSound(); // Abstract method
}

class Dog extends Animal implements Behavior {
    public Dog(String name, int age) {
        super(name, age);
    }

    //Method overriding
    @Override
    public void displayDetails() {
        System.out.println("Dog Name: " + getName() + ", Age: " + getAge());
    }

    //Method implementation from interface
    @Override
    public void makeSound() {
        System.out.println("Woof Woof!");
    }
}

class Cat extends Animal implements Behavior {
    public Cat(String name, int age) {
        super(name, age);
    }

    // Method overriding
    @Override
    public void displayDetails() {
        System.out.println("Cat Name: " + getName() + ", Age: " + getAge());
    }

    // Concept: Method implementation from interface
    @Override
    public void makeSound() {
        System.out.println("Meow Meow!");
    }
}

//Abstract Class
abstract class Vehicle {
```

```
    abstract void startEngine(); //Abstract method

    public void displayType() {
        System.out.println("This is a vehicle.");
    }
}

class Car extends Vehicle {
    @Override
    void startEngine() {
        System.out.println("Car engine started.");
    }
}

// Composition and Aggregation
class Garage {
    private Car car; // Aggregation

    public Garage(Car car) {
        this.car = car;
    }

    public void parkCar() {
        System.out.println("Parking the car in the garage.");
        car.startEngine();
    }
}

// Generalization and Specialization
class SportsCar extends Car { // Specialization
    @Override
    void startEngine() {
        System.out.println("Sports car engine roars to life!");
    }
}


public class Main {
    public static void main(String[] args) {
        // Creating objects of Animal class
        Dog dog = new Dog("Buddy", 3);
        Cat cat = new Cat("Whiskers", 2);

        //polymorphism and method overriding
        dog.displayDetails();
        cat.displayDetails();

        //polymorphism through interface
        Behavior[] animals = {dog, cat};
        for (Behavior animal : animals) {
            animal.makeSound();
```

```
        }

        //abstract class
        Vehicle myCar = new Car();
        myCar.startEngine();
        myCar.displayType();

        //composition and aggregation
        Garage garage = new Garage(new SportsCar());
        garage.parkCar();

        //specialization
        SportsCar sportsCar = new SportsCar();
        sportsCar.startEngine();
    }
}
```

Output:



2) **Design a Java program that performs various string operations and uses control statements for user input validation. The program should allow the user to perform the following operations:**

   a) **Concatenate Strings: The user can enter two strings and the program should concatenate them.**
   b) **Find Length of a String: The user can enter a string, and the program should display its length.**
   c) **Convert to Uppercase and Lowercase: The user can enter a string, and the program should display it in both uppercase and lowercase.**
   d) **Extract Substring: The user can enter a string and specify the starting and ending index, and the program should extract and display the substring.**
   e) **Split a Sentence: The user can enter a sentence, and the program should split it into words and display them.**
   f) **Reverse a String: The user can enter a string, and the program should reverse and display it.**
   g) **Requirements:**
      i) **Use control statements (if-else, switch, loops) for input validation and handling possible errors.**
      ii) **Implement a user-friendly console interface for the user to interact with the program.**
      iii) **Cover all string concepts, such as concatenation, length, uppercase and lowercase conversion, substring extraction, splitting, and reversal.**

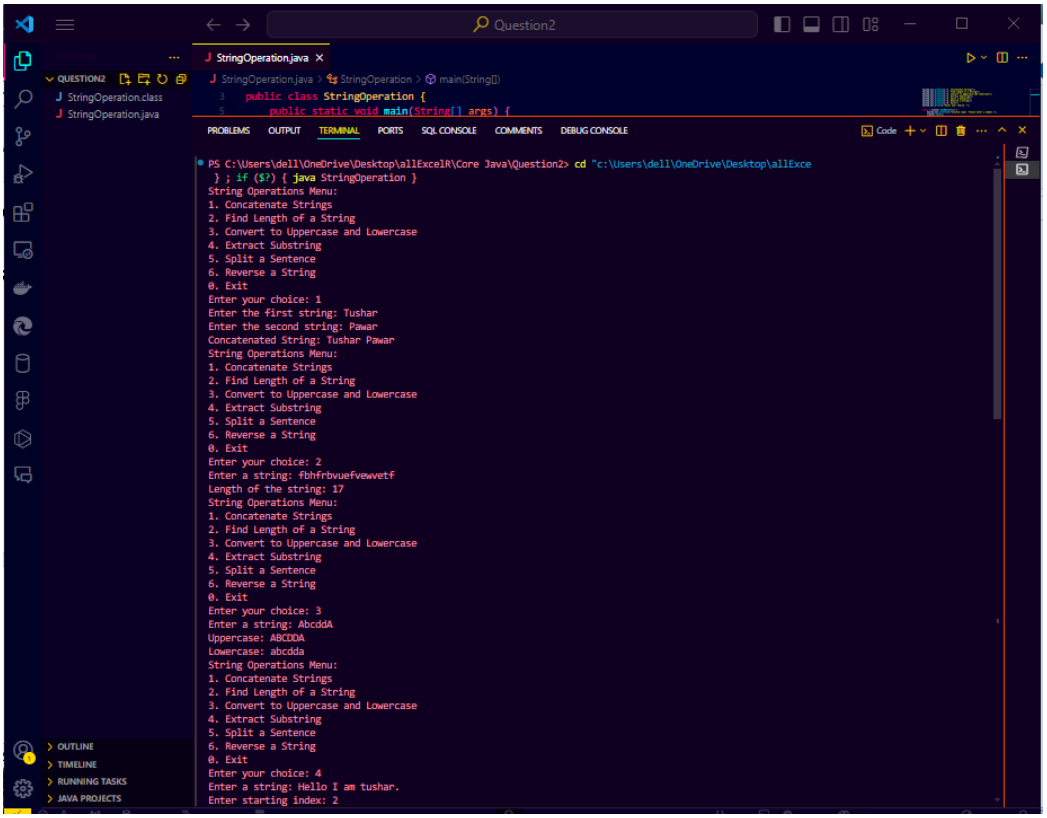
Code:

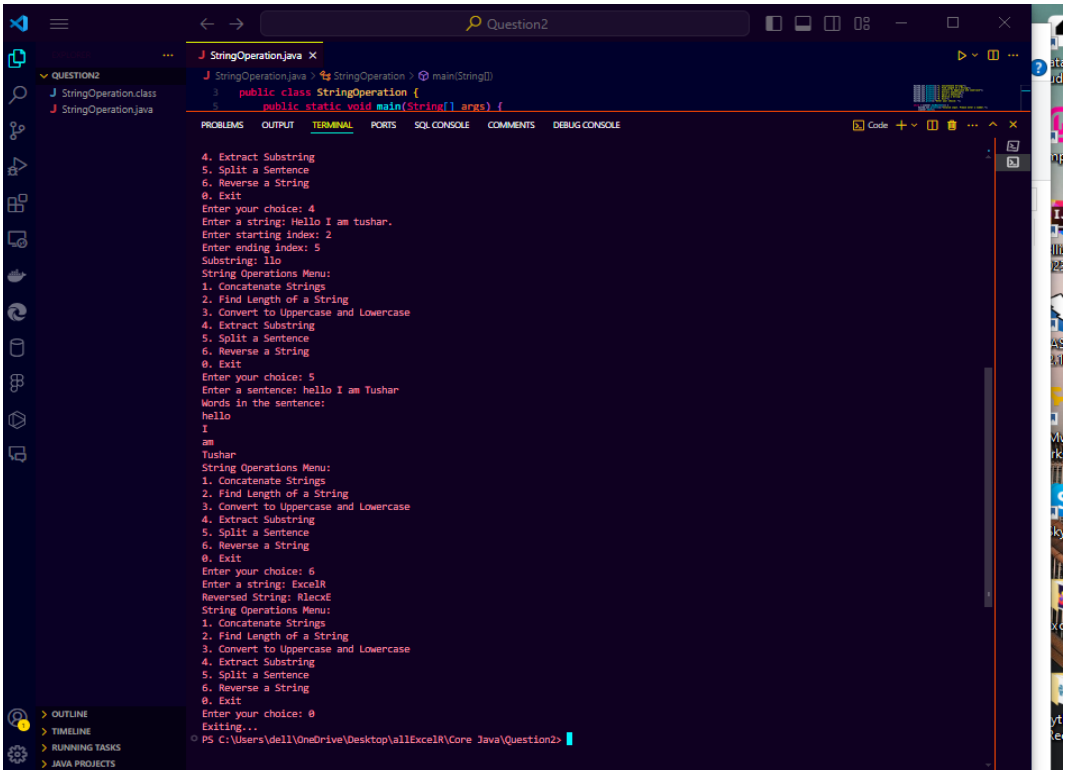
```
import java.util.Scanner;

public class StringOperation {

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

        do {
            System.out.println("String Operations Menu:");
            System.out.println("1. Concatenate Strings");
            System.out.println("2. Find Length of a String");
            System.out.println("3. Convert to Uppercase and Lowercase");
            System.out.println("4. Extract Substring");
            System.out.println("5. Split a Sentence");
            System.out.println("6. Reverse a String");
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");
```

```
while (!scanner.hasNextInt()) {
    System.out.println("Invalid input. Please enter a number.");
    scanner.next();
    System.out.print("Enter your choice: ");
}

choice = scanner.nextInt();
scanner.nextLine();

switch (choice) {
    case 1:
        System.out.print("Enter the first string: ");
        String str1 = scanner.nextLine();
        System.out.print("Enter the second string: ");
        String str2 = scanner.nextLine();
        System.out.println("Concatenated String: " + str1 + str2);
        break;

    case 2:
        System.out.print("Enter a string: ");
        String strLength = scanner.nextLine();
        System.out.println("Length of the string: " + strLength.length());
        break;

    case 3:
        System.out.print("Enter a string: ");
        String strCase = scanner.nextLine();
        System.out.println("Uppercase: " + strCase.toUpperCase());
        System.out.println("Lowercase: " + strCase.toLowerCase());
        break;

    case 4:
        System.out.print("Enter a string: ");
        String strSubstring = scanner.nextLine();
        System.out.print("Enter starting index: ");

        while (!scanner.hasNextInt()) {
            System.out.println("Invalid input. Please enter a number.");
            scanner.next();
            System.out.print("Enter starting index: ");
        }
        int startIndex = scanner.nextInt();

        System.out.print("Enter ending index: ");

        while (!scanner.hasNextInt()) {
            System.out.println("Invalid input. Please enter a number.");
            scanner.next();
            System.out.print("Enter ending index: ");
        }
        int endIndex = scanner.nextInt();
```

```
                scanner.nextLine();

                if (startIndex >= 0 && endIndex <= strSubstring.length() && startIndex
< endIndex) {
                    System.out.println("Substring: " +
strSubstring.substring(startIndex, endIndex));
                } else {
                    System.out.println("Invalid indices.");
                }
                break;

            case 5:
                System.out.print("Enter a sentence: ");
                String sentence = scanner.nextLine();
                String[] words = sentence.split("\\s+");
                System.out.println("Words in the sentence:");
                for (String word : words) {
                    System.out.println(word);
                }
                break;

            case 6:
                System.out.print("Enter a string: ");
                String strReverse = scanner.nextLine();
                String reversedString = new
StringBuilder(strReverse).reverse().toString();
                System.out.println("Reversed String: " + reversedString);
                break;

            case 0:
                System.out.println("Exiting...");
                break;

            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }

    } while (choice != 0);

    scanner.close();
  }
}
```

Output:


Question2
StringOperation.java
QUESTION2
StringOperation.class
StringOperation.java
StringOperation.java > StringOperation > main(String[])
public class StringOperation {
public static void main(String[] args) {
PROBLEMS OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEBUG CONSOLE
PS C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question2> cd "c:\Users\dell\OneDrive\Desktop\allExce
} ; if ($?) { java StringOperation }
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 1
Enter the first string: Tushar
Enter the second string: Pawar
Concatenated String: Tushar Pawar
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 2
Enter a string: fbhfrbvuefvewwetf
Length of the string: 17
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 3
Enter a string: AbcddA
Uppercase: ABCDDA
Lowercase: abcdda
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 4
Enter a string: Hello I am tushar.
Enter starting index: 2
OUTLINE
TIMELINE
RUNNING TASKS
JAVA PROJECTS


Question2
StringOperation.java
QUESTION2
StringOperation.class
StringOperation.java
StringOperation.java > StringOperation > main(String[])
public class StringOperation {
public static void main(String[] args) {
PROBLEMS OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEBUG CONSOLE
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 4
Enter a string: Hello I am tushar.
Enter starting index: 2
Enter ending index: 5
Substring: llo
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 5
Enter a sentence: hello I am Tushar
Words in the sentence:
hello
I
am
Tushar
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 6
Enter a string: ExcelR
Reversed String: RlecxE
String Operations Menu:
1. Concatenate Strings
2. Find Length of a String
3. Convert to Uppercase and Lowercase
4. Extract Substring
5. Split a Sentence
6. Reverse a String
0. Exit
Enter your choice: 0
Exiting...
PS C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question2>
OUTLINE
TIMELINE
RUNNING TASKS
JAVA PROJECTS

3) **Design a Java program to cover all File related topics, demonstrating various File operations in Java. The program should allow users to perform the following tasks:**

   a) **Create a new directory.**
   b) **Create a new text file and write content to it.**
   c) **Read the content from an existing text file.**
   d) **Append new content to an existing text file.**
   e) **Copy the content from one text file to another.**
   f) **Delete a text file.**
   g) **List all files and directories in a given directory.**
   h) **Search for a specific file in a directory and its subdirectories.**
   i) **Rename a file.**
   j) **Get information about a file (e.g., file size, last modified time).**
   k) **Requirements:**
      i) **Use File Input and Output streams for reading and writing text files.**
      ii) **Implement exception handling to handle possible errors during file operations.**
      iii) **Provide a user-friendly console interface for the user to interact with the program.**

Code :

```
import java.io.*;
import java.nio.file.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;

public class FileOperations {

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

        do {
            System.out.println("File Operations Menu:");
            System.out.println("1. Create a new directory");
            System.out.println("2. Create a new text file and write content to it");
            System.out.println("3. Read content from an existing text file");
            System.out.println("4. Append new content to an existing text file");
            System.out.println("5. Copy content from one text file to another");
            System.out.println("6. Delete a text file");
            System.out.println("7. List all files and directories in a given directory");
            System.out.println("8. Search for a specific file in a directory and its
subdirectories");
            System.out.println("9. Rename a file");
```

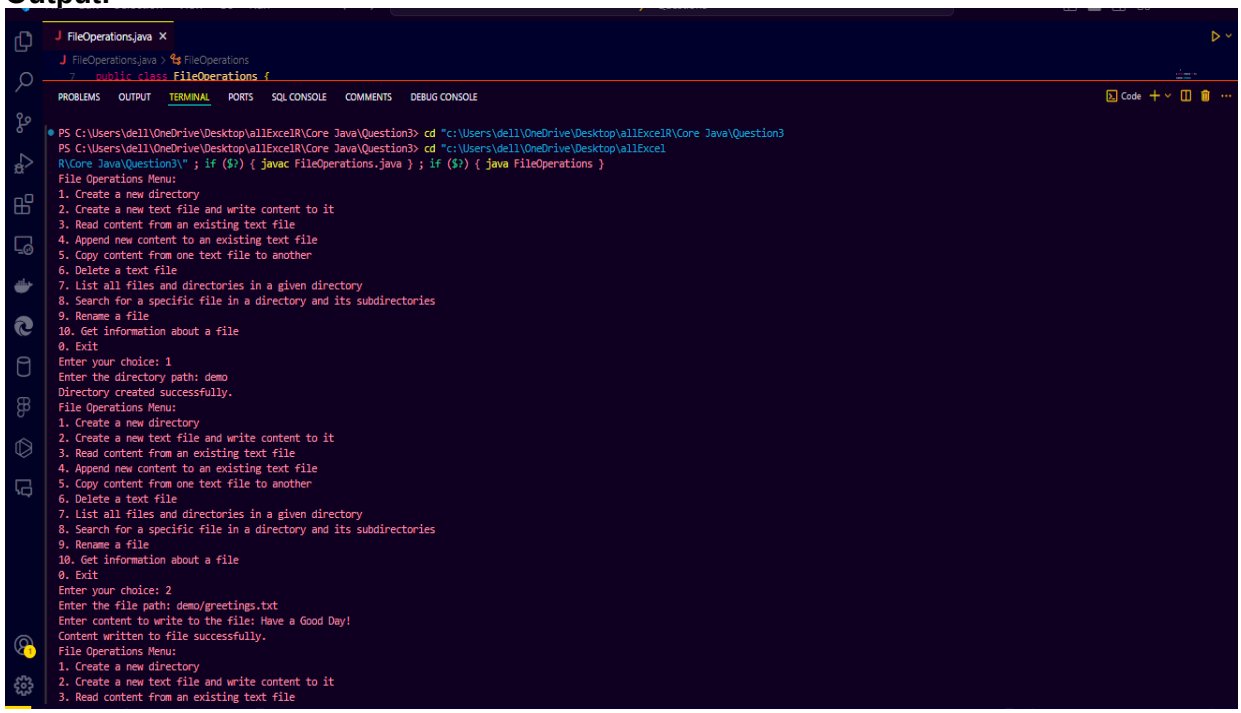
```
        System.out.println("10. Get information about a file");
        System.out.println("0. Exit");
        System.out.print("Enter your choice: ");

        while (!scanner.hasNextInt()) {
            System.out.println("Invalid input. Please enter a number.");
            scanner.next();
            System.out.print("Enter your choice: ");
        }

        choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                System.out.print("Enter the directory path: ");
                String dirPath = scanner.nextLine();
                File dir = new File(dirPath);
                if (dir.mkdirs()) {
                    System.out.println("Directory created successfully.");
                } else {
                    System.out.println("Failed to create directory or directory already
exists.");
                }
                break;

            case 2:
                System.out.print("Enter the file path: ");
                String filePath = scanner.nextLine();
                System.out.print("Enter content to write to the file: ");
                String content = scanner.nextLine();
                try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath))) {
                    writer.write(content);
                    System.out.println("Content written to file successfully.");
                } catch (IOException e) {
                    System.out.println("An error occurred while writing to the file: " +
e.getMessage());
                }
                break;

            case 3:
                System.out.print("Enter the file path: ");
                String readFilePath = scanner.nextLine();
                try (BufferedReader reader = new BufferedReader(new
FileReader(readFilePath))) {
                    String line;
                    while ((line = reader.readLine()) != null) {
                        System.out.println(line);
                    }
                } catch (IOException e) {
```

```
                System.out.println("An error occurred while reading the file: " +
e.getMessage());
            }
            break;

        case 4:
            System.out.print("Enter the file path: ");
            String appendFilePath = scanner.nextLine();
            System.out.print("Enter content to append: ");
            String appendContent = scanner.nextLine();
            try (BufferedWriter writer = new BufferedWriter(new
FileWriter(appendFilePath, true))) {
                writer.write(appendContent);
                System.out.println("Content appended to file successfully.");
            } catch (IOException e) {
                System.out.println("An error occurred while appending to the file: "
+ e.getMessage());
            }
            break;

        case 5:
            System.out.print("Enter the source file path: ");
            String sourceFilePath = scanner.nextLine();
            System.out.print("Enter the destination file path: ");
            String destFilePath = scanner.nextLine();
            try {
                Files.copy(Paths.get(sourceFilePath), Paths.get(destFilePath),
StandardCopyOption.REPLACE_EXISTING);
                System.out.println("File copied successfully.");
            } catch (IOException e) {
                System.out.println("An error occurred while copying the file: " +
e.getMessage());
            }
            break;

        case 6:
            System.out.print("Enter the file path: ");
            String deleteFilePath = scanner.nextLine();
            File fileToDelete = new File(deleteFilePath);
            if (fileToDelete.delete()) {
                System.out.println("File deleted successfully.");
            } else {
                System.out.println("Failed to delete the file or file does not exist.");
            }
            break;

        case 7:
            System.out.print("Enter the directory path: ");
            String listDirPath = scanner.nextLine();
            File listDir = new File(listDirPath);
            if (listDir.isDirectory()) {
```

```
                File[] files = listDir.listFiles();
                if (files != null) {
                    for (File file : files) {
                        System.out.println((file.isDirectory() ? "Directory: " : "File: ") +
file.getName());
                    }
                } else {
                    System.out.println("No files found in the directory.");
                }
            } else {
                System.out.println("The specified path is not a directory.");
            }
            break;

        case 8:
            System.out.print("Enter the directory path: ");
            String searchDirPath = scanner.nextLine();
            System.out.print("Enter the file name to search for: ");
            String fileName = scanner.nextLine();
            searchFile(new File(searchDirPath), fileName);
            break;

        case 9:
            System.out.print("Enter the current file path: ");
            String oldFilePath = scanner.nextLine();
            System.out.print("Enter the new file path: ");
            String newFilePath = scanner.nextLine();
            File oldFile = new File(oldFilePath);
            File newFile = new File(newFilePath);
            if (oldFile.renameTo(newFile)) {
                System.out.println("File renamed successfully.");
            } else {
                System.out.println("Failed to rename the file or file does not exist.");
            }
            break;

        case 10:
            System.out.print("Enter the file path: ");
            String infoFilePath = scanner.nextLine();
            File infoFile = new File(infoFilePath);
            if (infoFile.exists()) {
                System.out.println("File Size: " + infoFile.length() + " bytes");
                SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy
HH:mm:ss");
                System.out.println("Last Modified: " + sdf.format(new
Date(infoFile.lastModified())));
            } else {
                System.out.println("File does not exist.");
            }
            break;
```

```
            case 0:
                System.out.println("Exiting...");
                break;

            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }

    } while (choice != 0);

    scanner.close();
  }


  private static void searchFile(File directory, String fileName) {
    File[] files = directory.listFiles();
    if (files != null) {
        for (File file : files) {
            if (file.isDirectory()) {
                searchFile(file, fileName);
            } else if (file.getName().equalsIgnoreCase(fileName)) {
                System.out.println("File found: " + file.getAbsolutePath());
            }
        }
    }
  }
}
```

Output:



PROBLEMS 6 OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEBUG CONSOLE

```
Enter your choice: 3
Enter the file path: demo/greetings.txt
Have a Good Day!
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 4
Enter the file path: !!
Enter content to append: t
Content appended to file successfully.
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 4
Enter the file path: demo/greetings.txt
Enter content to append: !!
Content appended to file successfully.
File Operations Menu:
1. Create a new directory
```

PROBLEMS 6 OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEBUG CONSOLE

```
Enter your choice: 3
Enter the file path: demo/greetings.txt
Have a Good Day!!!
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 5
Enter the source file path: demo/greetings.txt
Enter the destination file path: copy.txt
File copied successfully.
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 7
Enter the directory path:
The specified path is not a directory.
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
```

```
PROBLEMS 6   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE

9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 7
Enter the directory path:
The specified path is not a directory.
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 7
Enter the directory path: demo
File: greetings.txt
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 8
Enter the directory path: demo
Enter the file name to search for: greetings.txt
File found: C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question3\demo\greetings.txt
```

```
PROBLEMS 6   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE   Coc

3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
6. Delete a text file
7. List all files and directories in a given directory
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 10
Enter the file path: greetings.txt
File Size: 13 bytes
Last Modified: 07/24/2024 08:10:51
File Operations Menu:
1. Create a new directory
2. Create a new text file and write content to it
3. Read content from an existing text file
4. Append new content to an existing text file
5. Copy content from one text file to another
6. Delete a text file
7. List all files and directories in a given directory
8. Search for a specific file in a directory and its subdirectories
9. Rename a file
10. Get information about a file
0. Exit
Enter your choice: 0
Exiting...
```

4) **Design a Java program that covers all thread-related topics, demonstrating various multithreading concepts in Java. The program should allow users to perform the following tasks:**

   a) **Create and start multiple threads.**
   b) **Synchronize threads to avoid race conditions and ensure data consistency.**
   c) **Use wait() and notify() to implement thread communication.**
   d) **Use sleep() to pause threads for a specified duration.**
   e) **Demonstrate thread interruption and thread termination.**
   f) **Use thread pools to manage a group of threads efficiently.**
   g) **Implement thread synchronization using locks and conditions.**
   h) **Demonstrate deadlock and ways to avoid it.**
   i) **Use thread-local variables to handle thread-specific data.**
   j) **Implement producer-consumer problem using thread synchronization.**
   k) **Use Executors and Callable to perform parallel computation and get results.**
   l) **Requirements:**
      i) **Implement exception handling to handle possible errors during multithreaded operations.**
      ii) **Provide a user-friendly console interface for the user to interact with the program.**

Code :

```
import java.util.concurrent.*;


import java.util.concurrent.locks.*;
import java.util.*;

public class ThreadDemo {

    private static final int NUM_THREADS = 3;
    private static final int PRODUCER_COUNT = 1;
    private static final int CONSUMER_COUNT = 2;
    private static final BlockingQueue<Integer> queue = new LinkedBlockingQueue<>();
    private static final Lock lock = new ReentrantLock();
    private static final Condition condition = lock.newCondition();
    private static final ThreadLocal<Integer> threadLocalValue = ThreadLocal.withInitial(() -> 0);

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

        do {
            System.out.println("Thread Operations Menu:");
            System.out.println("1. Create and start multiple threads");
            System.out.println("2. Synchronize threads");
            System.out.println("3. Use wait() and notify()");
```

```
System.out.println("4. Use sleep() to pause threads");
System.out.println("5. Demonstrate thread interruption and termination");
System.out.println("6. Use thread pools");
System.out.println("7. Implement thread synchronization using locks and conditions");
System.out.println("8. Demonstrate deadlock and ways to avoid it");
System.out.println("9. Use thread-local variables");
System.out.println("10. Implement producer-consumer problem");
System.out.println("11. Use Executors and Callable");
System.out.println("0. Exit");
System.out.print("Enter your choice: ");

while (!scanner.hasNextInt()) {
    System.out.println("Invalid input. Please enter a number.");
    scanner.next();
    System.out.print("Enter your choice: ");
}

choice = scanner.nextInt();
scanner.nextLine();

switch (choice) {
    case 1:
        System.out.println("Creating and starting threads...");
        List<Thread> threads = new ArrayList<>();
        for (int i = 0; i < NUM_THREADS; i++) {
            Thread thread = new Thread(new RunnableTask("Thread-" + i));
            threads.add(thread);
            thread.start();
        }
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted: " + e.getMessage());
            }
        }
        break;

    case 2:
        System.out.println("Synchronizing threads...");
        Thread syncThread1 = new Thread(new SyncTask());
        Thread syncThread2 = new Thread(new SyncTask());
        syncThread1.start();
        syncThread2.start();
        try {
            syncThread1.join();
            syncThread2.join();
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        }
        break;
```

```
case 3:
    System.out.println("Demonstrating wait() and notify()...");
    ThreadWaitNotify twn = new ThreadWaitNotify();
    Thread consumerThread = new Thread(() -> twn.consumer());
    Thread producerThread = new Thread(() -> twn.producer());
    consumerThread.start();
    producerThread.start();
    try {
        consumerThread.join();
        producerThread.join();
    } catch (InterruptedException e) {
        System.out.println("Thread interrupted: " + e.getMessage());
    }
    break;

case 4:
    System.out.println("Demonstrating sleep()...");
    Thread sleepThread = new Thread(() -> {
        try {
            Thread.sleep(2000);
            System.out.println("Thread woke up after 2 seconds");
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        }
    });
    sleepThread.start();
    try {
        sleepThread.join();
    } catch (InterruptedException e) {
        System.out.println("Thread interrupted: " + e.getMessage());
    }
    break;

case 5:
    System.out.println("Demonstrating thread interruption...");
    ThreadInterrupt demo = new ThreadInterrupt();
    Thread interruptThread = new Thread(demo);
    interruptThread.start();
    try {
        Thread.sleep(1000);
        interruptThread.interrupt();
        interruptThread.join();
    } catch (InterruptedException e) {
        System.out.println("Main thread interrupted: " + e.getMessage());
    }
    break;

case 6:
    System.out.println("Using thread pools...");
    ExecutorService executor = Executors.newFixedThreadPool(NUM_THREADS);
```

```
    List<Future<?>> futures = new ArrayList<>();
    for (int i = 0; i < NUM_THREADS; i++) {
        final int taskId = i;
        futures.add(executor.submit(() -> {
            System.out.println("Task " + taskId + " running");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                System.out.println("Task " + taskId + " interrupted: " + e.getMessage());
            }
        }));
    }
    for (Future<?> future : futures) {
        try {
            future.get();
        } catch (InterruptedException | ExecutionException e) {
            System.out.println("Error in task: " + e.getMessage());
        }
    }
    executor.shutdown();
    break;

case 7:
    System.out.println("Demonstrating locks and conditions...");
    Thread lockThread1 = new Thread(new LockConditionTask());
    Thread lockThread2 = new Thread(new LockConditionTask());
    lockThread1.start();
    lockThread2.start();
    try {
        lockThread1.join();
        lockThread2.join();
    } catch (InterruptedException e) {
        System.out.println("Thread interrupted: " + e.getMessage());
    }
    break;

case 8:
    System.out.println("Demonstrating deadlock...");
    Thread deadlockThread1 = new Thread(new DeadlockTask(true));
    Thread deadlockThread2 = new Thread(new DeadlockTask(false));
    deadlockThread1.start();
    deadlockThread2.start();
    try {
        deadlockThread1.join();
        deadlockThread2.join();
    } catch (InterruptedException e) {
        System.out.println("Thread interrupted: " + e.getMessage());
    }
    break;

case 9:
```

```
            System.out.println("Using thread-local variables...");
            Thread threadLocalThread1 = new Thread(() -> {
                threadLocalValue.set((int) (Math.random() * 100));
                System.out.println("Thread-local value: " + threadLocalValue.get());
            });
            threadLocalThread1.start();
            try {
                threadLocalThread1.join();
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted: " + e.getMessage());
            }
            break;

        case 10:
            System.out.println("Implementing producer-consumer problem...");
            List<Thread> producerThreads = new ArrayList<>();
            List<Thread> consumerThreads = new ArrayList<>();
            for (int i = 0; i < PRODUCER_COUNT; i++) {
                Thread producer = new Thread(new Producer());
                producerThreads.add(producer);
                producer.start();
            }
            for (int i = 0; i < CONSUMER_COUNT; i++) {
                Thread consumer = new Thread(new Consumer());
                consumerThreads.add(consumer);
                consumer.start();
            }
            for (Thread producer : producerThreads) {
                try {
                    producer.join();
                } catch (InterruptedException e) {
                    System.out.println("Producer thread interrupted: " + e.getMessage());
                }
            }
            for (Thread consumer : consumerThreads) {
                try {
                    consumer.join();
                } catch (InterruptedException e) {
                    System.out.println("Consumer thread interrupted: " + e.getMessage());
                }
            }
            break;

        case 11:
            System.out.println("Using Executors and Callable...");
            ExecutorService executorService =
Executors.newFixedThreadPool(NUM_THREADS);
            List<Future<Integer>> results = new ArrayList<>();
            for (int i = 0; i < NUM_THREADS; i++) {
                final int taskId = i;
                results.add(executorService.submit(() -> {
```

```
                System.out.println("Callable task " + taskId + " running");
                Thread.sleep(2000);
                return taskId * 2;
            }));
        }
        for (Future<Integer> future : results) {
            try {
                System.out.println("Result: " + future.get());
            } catch (InterruptedException | ExecutionException e) {
                System.out.println("Error getting result: " + e.getMessage());
            }
        }
        executorService.shutdown();
        break;

    case 0:
        System.out.println("Exiting...");
        break;

    default:
        System.out.println("Invalid choice. Please try again.");
        break;
    }

  } while (choice != 0);

  scanner.close();
}

static class RunnableTask implements Runnable {
  private final String name;

  public RunnableTask(String name) {
    this.name = name;
  }

  @Override
  public void run() {
    System.out.println(name + " is running.");
  }
}

// Synchronization Task
static class SyncTask implements Runnable {
  private static int count = 0;
  private static final Object lock = new Object();

  @Override
  public void run() {
    synchronized (lock) {
      count++;
```

```
            System.out.println("Count: " + count);
        }
    }
}

// Wait and Notify Task
static class ThreadWaitNotify {
    private boolean flag = false;

    public synchronized void producer() {
        System.out.println("Producer started");
        try {
            Thread.sleep(1000);
            flag = true;
            notify();
            System.out.println("Producer finished");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public synchronized void consumer() {
        System.out.println("Consumer started");
        while (!flag) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Consumer finished");
    }
}

// Interrupt Task
static class ThreadInterrupt implements Runnable {
    @Override
    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                System.out.println("Running...");
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread was interrupted!");
        }
    }
}

// Locks and Conditions Task
static class LockConditionTask implements Runnable {
```

```
    @Override
    public void run() {
        lock.lock();
        try {
            System.out.println("Thread is waiting...");
            condition.await();
            System.out.println("Thread is resuming...");
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        } finally {
            lock.unlock();
        }
    }
}

// Deadlock Task
static class DeadlockTask implements Runnable {
    private final boolean flag;
    private static final Object lock1 = new Object();
    private static final Object lock2 = new Object();

    public DeadlockTask(boolean flag) {
        this.flag = flag;
    }

    @Override
    public void run() {
        if (flag) {
            synchronized (lock1) {
                System.out.println("Lock 1 acquired by " + Thread.currentThread().getName());
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized (lock2) {
                    System.out.println("Lock 2 acquired by " + Thread.currentThread().getName());
                }
            }
        } else {
            synchronized (lock2) {
                System.out.println("Lock 2 acquired by " + Thread.currentThread().getName());
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized (lock1) {
                    System.out.println("Lock 1 acquired by " + Thread.currentThread().getName());
                }
            }
```

```
            }
        }
    }

    // Producer-Consumer Problem
    static class Producer implements Runnable {
        @Override
        public void run() {
            for (int i = 0; i < 5; i++) {
                try {
                    queue.put(i);
                    System.out.println("Produced: " + i);
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    System.out.println("Producer interrupted: " + e.getMessage());
                }
            }
        }
    }

    static class Consumer implements Runnable {
        @Override
        public void run() {
            while (true) {
                try {
                    Integer item = queue.take();
                    System.out.println("Consumed: " + item);
                } catch (InterruptedException e) {
                    System.out.println("Consumer interrupted: " + e.getMessage());
                }
            }
        }
    }
}
```

Output :

```
PROBLEMS 1   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE

PS C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question4> cd "c:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Questio
va ThreadDemo }
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
3. Use wait() and notify()
4. Use sleep() to pause threads
5. Demonstrate thread interruption and termination
6. Use thread pools
7. Implement thread synchronization using locks and conditions
8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 1
Creating and starting threads...
Thread-0 is running.
Thread-1 is running.
Thread-2 is running.
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
1 0   0   Connect   Java: Ready   Server not selected   Ln 348, Col 58   Spaces: 4   UTF-8   CRLF
```

```
Enter your choice: 2
Synchronizing threads...
Count: 1
Count: 2
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
3. Use wait() and notify()
4. Use sleep() to pause threads
5. Demonstrate thread interruption and termination
6. Use thread pools
7. Implement thread synchronization using locks and conditions
8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 3
Demonstrating wait() and notify()...
Consumer started
Producer started
Producer finished
Consumer finished
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
3. Use wait() and notify()
4. Use sleep() to pause threads
5. Demonstrate thread interruption and termination
6. Use thread pools
7. Implement thread synchronization using locks and conditions
8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 4
Demonstrating sleep()...
Thread woke up after 2 seconds
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
3. Use wait() and notify()
4. Use sleep() to pause threads
5. Demonstrate thread interruption and termination
6. Use thread pools
7. Implement thread synchronization using locks and conditions
8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 5
Demonstrating thread interruption...
Running...
Running...
Thread was interrupted!
Thread Operations Menu:
1. Create and start multiple threads
```

```
PROBLEMS 1   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE                    Code +

8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 7
Demonstrating locks and conditions...
Thread is waiting...
Thread is waiting...
PS C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question4> cd "c:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Qu
ThreadDemo.java } ; if ($?) { java ThreadDemo }
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
3. Use wait() and notify()
4. Use sleep() to pause threads
5. Demonstrate thread interruption and termination
6. Use thread pools
7. Implement thread synchronization using locks and conditions
8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 8
Demonstrating deadlock...
Lock 2 acquired by Thread-1
Lock 1 acquired by Thread-0
```

```
PROBLEMS 1   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE

9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 9
Using thread-local variables...
Thread-local value: 96
Thread Operations Menu:
1. Create and start multiple threads
2. Synchronize threads
3. Use wait() and notify()
4. Use sleep() to pause threads
5. Demonstrate thread interruption and termination
6. Use thread pools
7. Implement thread synchronization using locks and conditions
8. Demonstrate deadlock and ways to avoid it
9. Use thread-local variables
10. Implement producer-consumer problem
11. Use Executors and Callable
0. Exit
Enter your choice: 10
Implementing producer-consumer problem...
Consumed: 0
Produced: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
```

5) **Design a Java program to implement a Collection Management System that manages different types of collections such as lists, sets, and maps. The program should allow users to perform the following operations for each type of collection:**

   a) **Lists:**
      i) **Add an element: The user can add an element to the list.**
      ii) **Remove an element: The user can remove an element from the list.**
      iii) **Display all elements: The user can view all elements in the list.**
   b) **Sets:**
      i) **Add an element: The user can add an element to the set.**
      ii) **Remove an element: The user can remove an element from the set.**
      iii) **Display all elements: The user can view all elements in the set.**
   c) **Maps:**
      i) **Add a key-value pair: The user can add a key-value pair to the map.**
      ii) **Remove a key-value pair: The user can remove a key-value pair from the map.**
      iii) **Display all key-value pairs: The user can view all key-value pairs in the map.**
   d) **Requirements:**
      i) **Implement separate classes for each type of collection (ListManager, SetManager, MapManager).**
      ii) **Use appropriate collection classes (e.g., ArrayList, LinkedList, HashSet, TreeMap) to store the elements and key-value pairs.**
      iii) **Use inheritance and polymorphism to manage different types of collections.**
      iv) **Implement exception handling to handle possible errors (e.g., element not found in the list/set, duplicate keys in the map).**
      v) **Provide a user-friendly console interface for the user to interact with the Collection Management System.**
   e) **Cover all Java collections topics, including Lists, Sets, and Maps**

code:

```
import java.util.*;
import java.util.Scanner;

abstract class CollectionManager {
   abstract void addElement();
   abstract void removeElement();
   abstract void displayElements();
}

class ListManager extends CollectionManager {
   private List<String> list = new ArrayList<>();
   private Scanner scanner = new Scanner(System.in);
```

```
    @Override
    void addElement() {
        System.out.print("Enter element to add to the list: ");
        String element = scanner.nextLine();
        list.add(element);
        System.out.println("Element added.");
    }

    @Override
    void removeElement() {
        System.out.print("Enter element to remove from the list: ");
        String element = scanner.nextLine();
        if (list.remove(element)) {
            System.out.println("Element removed.");
        } else {
            System.out.println("Element not found.");
        }
    }

    @Override
    void displayElements() {
        System.out.println("List elements: " + list);
    }
}

class SetManager extends CollectionManager {
    private Set<String> set = new HashSet<>();
    private Scanner scanner = new Scanner(System.in);

    @Override
    void addElement() {
        System.out.print("Enter element to add to the set: ");
        String element = scanner.nextLine();
        if (set.add(element)) {
            System.out.println("Element added.");
        } else {
            System.out.println("Element already exists.");
        }
    }

    @Override
    void removeElement() {
        System.out.print("Enter element to remove from the set: ");
        String element = scanner.nextLine();
        if (set.remove(element)) {
            System.out.println("Element removed.");
        } else {
            System.out.println("Element not found.");
        }
    }
```

```
    @Override
    void displayElements() {
        System.out.println("Set elements: " + set);
    }
}

class MapManager extends CollectionManager {
    private Map<String, String> map = new TreeMap<>();
    private Scanner scanner = new Scanner(System.in);

    @Override
    void addElement() {
        System.out.print("Enter key: ");
        String key = scanner.nextLine();
        System.out.print("Enter value: ");
        String value = scanner.nextLine();
        if (map.containsKey(key)) {
            System.out.println("Key already exists. Updating value.");
        }
        map.put(key, value);
        System.out.println("Key-Value pair added.");
    }

    @Override
    void removeElement() {
        System.out.print("Enter key to remove from the map: ");
        String key = scanner.nextLine();
        if (map.remove(key) != null) {
            System.out.println("Key-Value pair removed.");
        } else {
            System.out.println("Key not found.");
        }
    }

    @Override
    void displayElements() {
        System.out.println("Map elements: " + map);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CollectionManager listManager = new ListManager();
        CollectionManager setManager = new SetManager();
        CollectionManager mapManager = new MapManager();

        int choice;
        do {
            System.out.println("Collection Management System:");
```

```
        System.out.println("1. Manage Lists");
        System.out.println("2. Manage Sets");
        System.out.println("3. Manage Maps");
        System.out.println("0. Exit");
        System.out.print("Enter your choice: ");

        while (!scanner.hasNextInt()) {
            System.out.println("Invalid input. Please enter a number.");
            scanner.next();
            System.out.print("Enter your choice: ");
        }

        choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                manageCollection(listManager);
                break;
            case 2:
                manageCollection(setManager);
                break;
            case 3:
                manageCollection(mapManager);
                break;
            case 0:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }

    } while (choice != 0);

    scanner.close();
}

private static void manageCollection(CollectionManager manager) {
    Scanner scanner = new Scanner(System.in);
    int choice;

    do {
        System.out.println("Collection Management Menu:");
        System.out.println("1. Add element        ");
        System.out.println("2. Remove element      ");
        System.out.println("3. Display elements    ");
        System.out.println("0. Back to main menu   ");
        System.out.print("Enter your choice: ");

        while (!scanner.hasNextInt()) {
```

```
            System.out.println("Invalid input. Please enter a number.");
            scanner.next();
            System.out.print("Enter your choice: ");
        }

        choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                manager.addElement();
                break;
            case 2:
                manager.removeElement();
                break;
            case 3:
                manager.displayElements();
                break;
            case 0:
                System.out.println("Returning to main menu...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }

    } while (choice != 0);
  }
}
```

Output:

```
PROBLEMS 1   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE

PS C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question
5> cd "c:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Ques
tion5\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Collection Management System:
1. Manage Lists
2. Manage Sets
3. Manage Maps
0. Exit
Enter your choice: 1
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter element to add to the list: orange
Element added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter element to add to the list: Mango
Element added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 3
List elements: [orange, Mango]
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 2
Enter element to remove from the list: Mango
Element removed.
Collection Management Menu:
1. Add element
```

```
PROBLEMS 1   OUTPUT   TERMINAL   PORTS   SQL CONSOLE   COMMENTS   DEBUG CONSOLE

Returning to main menu...
Collection Management System:
1. Manage Lists
2. Manage Sets
3. Manage Maps
0. Exit
Enter your choice: 2
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter element to add to the set: Orange
Element added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter element to add to the set: Orange
Element already exists.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 3
Set elements: [Orange]
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 2
Enter element to remove from the set: Orange
Element removed.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
⊗ 0 ⚠ 1   0   Connect   Java: Ready   Server not selected   Ln 195, Col 1   Spaces: 4   U
```

PROBLEMS 1 OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEBUG CONSOLE

```
Enter element to remove from the set: Orange
Element removed.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 3
Set elements: []
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 0
Returning to main menu...
Collection Management System:
1. Manage Lists
2. Manage Sets
3. Manage Maps
0. Exit
Enter your choice: 3
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter key: fruit1
Enter value: Apple
Key-Value pair added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter key: Fruit2
Enter value: Mango
Key-Value pair added.
Collection Management Menu:
1. Add element
```

0 1 0 Connect Java: Ready Server not selected

PROBLEMS 1 OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEB

```
Enter value: Apple
Key-Value pair added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter key: Fruit2
Enter value: Mango
Key-Value pair added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 1
Enter key: Fruit3
Enter value: Orange
Key-Value pair added.
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 3
Map elements: {Fruit2=Mango, Fruit3=Orange, fruit1=Apple}
Collection Management Menu:
1. Add element
2. Remove element
3. Display elements
0. Back to main menu
Enter your choice: 0
Returning to main menu...
Collection Management System:
1. Manage Lists
2. Manage Sets
3. Manage Maps
0. Exit
Enter your choice:
```

6) **Add new employees: The user can add details like employee ID, name, department, and salary.**
    a) **Update employee details: The user can update the name, department, or salary of an existing employee based on their employee ID.**
    b) **Delete an employee: The user can delete an employee from the system based on their employee ID.**
    c) **Display all employees: The user can view a list of all employees and their details.**
    d) **Search for an employee: The user can search for an employee by their employee ID and view their details.**
    e) **Requirements:**
        i) **Use Object-Oriented Programming (OOP) principles and create an Employee class with appropriate attributes and methods.**
        ii) **Use appropriate data structures (e.g., ArrayList, HashMap) to store the employee data.**
        iii) **Implement exception handling to handle possible errors (e.g., invalid employee ID, input validation).**
        iv) **Provide a user-friendly console interface for the user to interact with the Employee Management System.**

Code:

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

class Employee {
    private String id;
    private String name;
    private String department;
    private double salary;

    public Employee(String id, String name, String department, double salary) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.salary = salary;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }
```

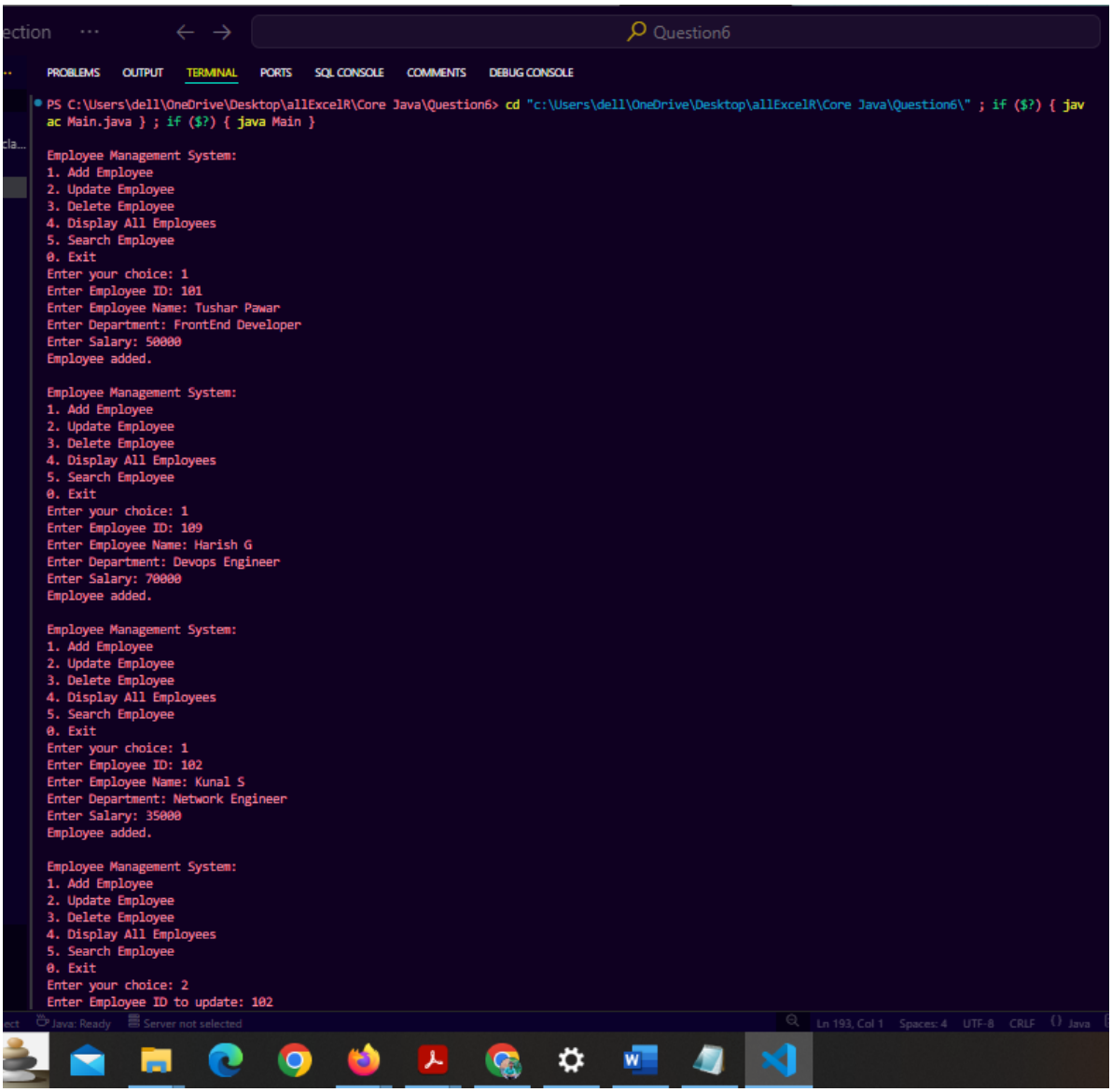
```
    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Department: " + department + ", Salary: $" +
salary;
    }
}

class EmployeeManager {
    private Map<String, Employee> employeeMap = new HashMap<>();
    private Scanner scanner = new Scanner(System.in);

    public void addEmployee() {
        System.out.print("Enter Employee ID: ");
        String id = scanner.nextLine();
        if (employeeMap.containsKey(id)) {
            System.out.println("Employee ID already exists.");
            return;
        }
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Department: ");
        String department = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();
        scanner.nextLine();

        Employee employee = new Employee(id, name, department, salary);
        employeeMap.put(id, employee);
        System.out.println("Employee added.");
    }
```

```
public void updateEmployee() {
    System.out.print("Enter Employee ID to update: ");
    String id = scanner.nextLine();
    Employee employee = employeeMap.get(id);
    if (employee == null) {
        System.out.println("Employee not found.");
        return;
    }
    System.out.print("Enter new Name (leave blank to keep current): ");
    String name = scanner.nextLine();
    if (!name.isEmpty()) {
        employee.setName(name);
    }
    System.out.print("Enter new Department (leave blank to keep current): ");
    String department = scanner.nextLine();
    if (!department.isEmpty()) {
        employee.setDepartment(department);
    }
    System.out.print("Enter new Salary (leave blank to keep current): ");
    String salaryInput = scanner.nextLine();
    if (!salaryInput.isEmpty()) {
        try {
            double salary = Double.parseDouble(salaryInput);
            employee.setSalary(salary);
        } catch (NumberFormatException e) {
            System.out.println("Invalid salary input.");
        }
    }
    System.out.println("Employee details updated.");
}

public void deleteEmployee() {
    System.out.print("Enter Employee ID to delete: ");
    String id = scanner.nextLine();
    if (employeeMap.remove(id) != null) {
        System.out.println("Employee deleted.");
    } else {
        System.out.println("Employee not found.");
    }
}

public void displayAllEmployees() {
    if (employeeMap.isEmpty()) {
        System.out.println("No employees to display.");
    } else {
        for (Employee employee : employeeMap.values()) {
            System.out.println(employee);
        }
    }
}
```

```
    public void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        String id = scanner.nextLine();
        Employee employee = employeeMap.get(id);
        if (employee != null) {
            System.out.println(employee);
        } else {
            System.out.println("Employee not found.");
        }
    }
}

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

        do {
            System.out.println("\nEmployee Management System:");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Delete Employee");
            System.out.println("4. Display All Employees");
            System.out.println("5. Search Employee");
            System.out.println("0. Exit");
            System.out.print("Enter your choice: ");

            while (!scanner.hasNextInt()) {
                System.out.println("Invalid input. Please enter a number.");
                scanner.next();
                System.out.print("Enter your choice: ");
            }

            choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    manager.addEmployee();
                    break;
                case 2:
                    manager.updateEmployee();
                    break;
                case 3:
                    manager.deleteEmployee();
                    break;
                case 4:
                    manager.displayAllEmployees();
                    break;
                case 5:
```

```
                manager.searchEmployee();
                break;
            case 0:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }

    } while (choice != 0);

    scanner.close();
  }
}
```

Output:



PROBLEMS OUTPUT TERMINAL PORTS SQL CONSOLE COMMENTS DEBUG CONSOLE

```
1. Add Employee
2. Update Employee
3. Delete Employee
4. Display All Employees
5. Search Employee
0. Exit
Enter your choice: 3
Enter Employee ID to delete:
Employee not found.

Employee Management System:
1. Add Employee
2. Update Employee
3. Delete Employee
4. Display All Employees
5. Search Employee
0. Exit
Enter your choice: 4
ID: 101, Name: Tushar Pawar, Department: FrontEnd Developer, Salary: $50000.0
ID: 102, Name: Kunal Shirsat, Department: Network Engineer, Salary: $35000.0
ID: 109, Name: Harish G, Department: Devops Engineer, Salary: $70000.0

Employee Management System:
1. Add Employee
2. Update Employee
3. Delete Employee
4. Display All Employees
5. Search Employee
0. Exit
Enter your choice: 1
Enter Employee ID: 101
Employee ID already exists.

Employee Management System:
1. Add Employee
2. Update Employee
3. Delete Employee
4. Display All Employees
5. Search Employee
0. Exit
Enter your choice: 5
Enter Employee ID to search: 102
ID: 102, Name: Kunal Shirsat, Department: Network Engineer, Salary: $35000.0

Employee Management System:
1. Add Employee
2. Update Employee
3. Delete Employee
4. Display All Employees
5. Search Employee
0. Exit
Enter your choice: 0
Exiting...
PS C:\Users\dell\OneDrive\Desktop\allExcelR\Core Java\Question6>
```