GTU Solved Question Answer

Unit-6

1) Explain about callback. [3 Marks]

A **callback** in Java is a mechanism where a method is passed as an argument to another method, allowing it to be executed later. This is commonly used for **asynchronous programming**, event handling, and interface-based communication.

Example:
Using an interface to implement a callback:

```
interface Callback {
    void onComplete();
}

class Task {
    void execute(Callback callback) {
        System.out.println("Task is running...");
        callback.onComplete(); // Invoking the callback
    }
}

class Main {
    public static void main(String[] args) {
        Task task = new Task();
        task.execute(() -> System.out.println("Task completed!"));
    }
}
```

Key Points:

- Callbacks help in event handling and asynchronous execution.
- Implemented using interfaces, anonymous classes, or lambda expressions.
- Commonly used in multi-threading, GUI programming, and API requests.

Hint

If we are not going to use call back

```
interface Callback {
   void onComplete();  // No body, just a declaration
}
class MyCallback implements Callback {
   @Override
   public void onComplete() {
      System.out.println("Task completed!");  // Now it prints something
   }
}
class Task {
   void execute(Callback callback) {
      System.out.println("Task is running...");
      callback.onComplete();  // Calls the method implemented in MyCallback
   }
}

public class DemoCallBack {
   public static void main(String[] args) {
      Task task = new Task();
      MyCallback myCallback = new MyCallback();
      task.execute(myCallback);  // This will print "Task is running..." and "Task completed!"
   }
}
```

2) Explain about Proxy class, Interface and Methods. (3 Marks)

[You can Write this explanation for proxy class whereas interface and method are already given in note book. Example is only just for your reference.]

A **Proxy Class** in Java is used to **control access to another object**. It acts as an **intermediary (middleman)** between the client and the actual object. This is useful when you want to:

Add **extra functionality** (like logging, security, or caching)
Control access to an object without changing its original code
Modify method behavior dynamically

Hint

Without Proxy (Direct Access to Object)

```
interface Internet {
   void connectTo(String site);
}

class RealInternet implements Internet {
   public void connectTo(String site) {
      System.out.println("Connecting to " + site);
   }
}
```

```
public class Main {
    public static void main(String[] args) {
        Internet internet = new RealInternet();
        internet.connectTo("example.com");  // Directly calling the method
    }
}
```

Output:
Connecting to example.com

Using a Proxy to Control Access

```
import java.util.*;

interface Internet {
    void connectTo(String site);
}
class RealInternet implements Internet {
    public void connectTo(String site) {
        System.out.println("Connecting to " + site);
    }
}

class ProxyInternet implements Internet {
    private RealInternet realInternet = new RealInternet();
    private static List<String> blockedSites = Arrays.asList("blocked.com",
"badsite.com");

    public void connectTo(String site) {
        if (blockedSites.contains(site.toLowerCase())) {
            System.out.println("Access Denied to " + site);
        } else {
            realInternet.connectTo(site);
        }
    }
}

public class ProxyExample {
    public static void main(String[] args) {
        Internet internet = new ProxyInternet();
        internet.connectTo("example.com");   // Allowed
        internet.connectTo("blocked.com");   // Blocked
    }
}
```

Output:
Connecting to example.com
Access Denied to blocked.com

4) Write a java program to read students details from console and write that students details into emp.txt file. (4 Marks)

```
import java.io.FileWriter;
import java.io.IOException;
```

```
import java.util.Scanner;

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

        // Get student details
        System.out.print("Enter Student Name: ");
        String name = scanner.nextLine();

        System.out.print("Enter Age: ");
        int age = scanner.nextInt();

        System.out.print("Enter Roll Number: ");
        int rollNumber = scanner.nextInt();

        // Writing to file without try-catch
        FileWriter writer = new FileWriter("emp.txt", true);
        writer.write("Student Name: " + name + "\n");
        writer.write("Age: " + age + "\n");
        writer.write("Roll Number: " + rollNumber + "\n");
        writer.write("-------------------------\n"); // Separator for multiple entries
        writer.close();

        System.out.println("Student details saved to emp.txt successfully.");

        // Close scanner
        scanner.close();
    }
}
```

5) Write a java program to read employee details from emp.txt file and print on screen. (4 Marks)

```
import java.io.FileReader;
import java.io.IOException;

public class ReadEmployeeDetails {
    public static void main(String[] args) throws IOException {
        FileReader reader = new FileReader("emp.txt");
        int ch;

        // Read and print the file content character by character
        while ((ch = reader.read()) != -1) {
            System.out.print((char) ch);
        }

        reader.close(); // Close the file reader
    }
}
```

6) Write a method for computing x_y doing repetitive multiplication. X and y are of type integer and are to be given as command line arguments. Raise and handle exception(s) for invalid values of x and y. (7 Marks)

```
public class PowerCalculator {
    public static void main(String[] args) {
        try {
            // Check if two arguments are provided
            if (args.length != 2) {
                throw new IllegalArgumentException("Please provide exactly two integer
arguments.");
            }

            // Parse arguments to integers
            int x = Integer.parseInt(args[0]);
            int y = Integer.parseInt(args[1]);

            // Compute x^y using repetitive multiplication
            int result = computePower(x, y);
            System.out.println(x + " ^ " + y + " = " + result);

        } catch (NumberFormatException e) {
            System.out.println("Invalid input! Please enter valid integers.");
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }

    // Method to compute x^y using repetitive multiplication
    public static int computePower(int x, int y) {
        if (y < 0) {
            throw new IllegalArgumentException("Exponent must be non-negative.");
        }

        int result = 1;
        for (int i = 0; i < y; i++) {
            result *= x; // Multiply x repeatedly
        }
        return result;
    }
}
```

7) Write a program to make calculator that accepts input from commandline? Use java's exception handling mechanism (7 Marks)

```
public class CommandLineCalculator {
    public static void main(String[] args) {
        try {
            // Check if exactly three arguments are provided
            if (args.length != 3) {
                throw new IllegalArgumentException("Usage: java CommandLineCalculator <num1>
<operator> <num2>");
            }

            // Parse numbers
            double num1 = Double.parseDouble(args[0]);
            double num2 = Double.parseDouble(args[2]);
            char operator = args[1].charAt(0);
```

```
            // Perform calculation
            double result = calculate(num1, operator, num2);
            System.out.println("Result: " + result);

        } catch (NumberFormatException e) {
            System.out.println("Invalid input! Please enter valid numbers.");
        } catch (ArithmeticException e) {
            System.out.println("Math error: " + e.getMessage());
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }

    // Method to perform calculations
    public static double calculate(double num1, char operator, double num2) {
        switch (operator) {
            case '+': return num1 + num2;
            case '-': return num1 - num2;
            case '*': return num1 * num2;
            case '/':
                if (num2 == 0) throw new ArithmeticException("Cannot divide by zero.");
                return num1 / num2;
            case '%':
                if (num2 == 0) throw new ArithmeticException("Cannot calculate remainder with
zero.");
                return num1 % num2;
            default:
                throw new IllegalArgumentException("Invalid operator! Use +, -, *, /, or %.");
        }
    }
}
```

8) Create a method named Withdraw () in the main function performing exception handling. Example: Balance = 1000; Withdraw = 12000 Here, Balance<Withdraw (throw exception) (7 marks)

```
import java.util.Scanner;

public class BankTransaction {
    public static void main(String[] args) {
        // Initialize balance
        double balance = 1000;

        Scanner scanner = new Scanner(System.in);

        // Get withdrawal amount
        System.out.print("Enter amount to withdraw: ");
        double withdrawAmount = scanner.nextDouble();

        try {
            // Call the withdraw method
            withdraw(balance, withdrawAmount);
            // If no exception, update balance
            balance -= withdrawAmount;
```

```
            System.out.println("Withdrawal successful! Remaining balance: " + balance);
        } catch (Exception e) {
            // Print exception message
            System.out.println("Transaction failed: " + e.getMessage());
        }

        scanner.close();
    }

    // Method to perform withdrawal with exception handling
    public static void withdraw(double balance, double amount) throws Exception {
        if (amount > balance) {
            throw new Exception("Insufficient balance! Available balance: " + balance);
        }
    }
}
```

Unit 7

1) What do you understand by event source and event object? Explain how to register an event handler object and how to implement a handler interface?

(7 Marks)

Event Source and Event Object in JavaFX

In JavaFX, events are generated when a user interacts with UI components.

1. **Event Source:** The component (UI element) that generates an event. For example, a **Button** is an event source when it is clicked.
2. **Event Object:** An instance of the **Event** class that carries information about the event, such as the type of event and the source component.

Registering an Event Handler Object

An **event handler object** processes events for a component. It is registered using the **setOnAction()** method for action events.

Example:

```
Button btn = new Button("Click Me");

btn.setOnAction(new EventHandler<ActionEvent>() {

   @Override

   public void handle(ActionEvent event) {

      System.out.println("Button Clicked!");

   }

});
```

Here, an event handler is registered to the button using `setOnAction()`.

Implementing a Handler Interface

Instead of using an anonymous class, we can implement the **EventHandler** interface in a separate class.

Example:

```
import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;
```

```
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

class ButtonHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Button Clicked!");
    }
}
public class EventDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button("Click Me");
        btn.setOnAction(new ButtonHandler()); // Registering the event handler

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("JavaFX Event Handling");
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    } }
```

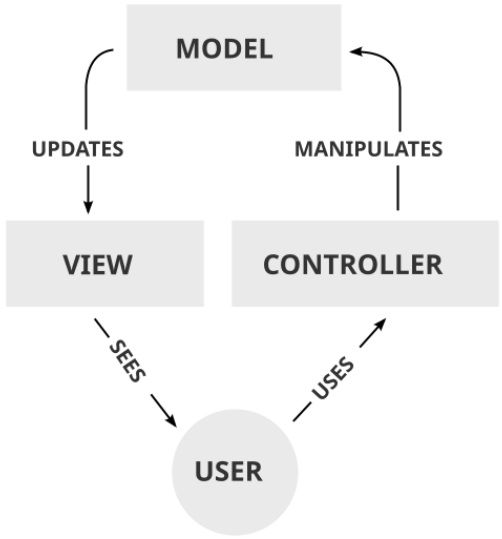
2) With a neat diagram explain the Model view controller design pattern and list out the advantages and disadvantages of using it in designing an application. (7 Marks)

Model-View-Controller (MVC) Design Pattern in JavaFX

The **Model-View-Controller (MVC)** pattern is a software design approach that separates an application into three components:

1. **Model:** Represents the application's data and business logic.
2. **View:** Handles the UI and displays data from the model.
3. **Controller:** Manages user interactions, updates the model, and refreshes the view.

MVC Diagram:



Advantages of MVC in JavaFX:

✅ **Separation of concerns** – Easier to manage, debug, and update.
✅ **Code reusability** – Components can be reused in different parts of the application.
✅ **Scalability** – Suitable for large applications.
✅ **Parallel development** – Developers can work on Model, View, and Controller separately.

Disadvantages of MVC in JavaFX:

❌ **Complexity** – Increases the number of classes and interactions.
❌ **Learning curve** – Requires a good understanding of design patterns.
❌ **Overhead** – Can be overkill for small applications.

```
// Model

class CounterModel {

    private int count = 0;
```

```
    public int getCount() { return count; }
    public void increment() { count++; }
}

// View (UI)
class CounterView {
    Button btn = new Button("Click Me");
    Label lbl = new Label("Count: 0");
    VBox layout = new VBox(10, lbl, btn);
    Scene getScene() { return new Scene(layout, 300, 200); }
}

// Controller
class CounterController {
    private CounterModel model;
    private CounterView view;

    public CounterController(CounterModel model, CounterView view) {
        this.model = model;
        this.view = view;
        view.btn.setOnAction(e -> {
            model.increment();
            view.lbl.setText("Count: " + model.getCount());
        });
    }
}

// Main Application
public class MVCDemo extends Application {
```

```
    @Override
    public void start(Stage primaryStage) {
        CounterModel model = new CounterModel();
        CounterView view = new CounterView();
        new CounterController(model, view);

        primaryStage.setScene(view.getScene());
        primaryStage.setTitle("MVC in JavaFX");
        primaryStage.show();
    }

    public static void main(String[] args) { launch(args); }
}
```

3) Explain the concept of inner classes and explain the types of inner classes with an example program. (7 Marks)

Inner Class in Java

An **inner class** in Java is a class that is **declared inside another class**. It helps in **encapsulating logic**, improving **readability**, and providing **better access control**.

Types of Inner Classes

Java supports four types of inner classes:

1. **Member Inner Class**
2. **Static Nested Class**
3. **Local Inner Class**
4. **Anonymous Inner Class**

---

1. Member Inner Class

A non-static class inside another class. It has access to all members (even private) of the outer class.

Example:

```
java
CopyEdit
```

```
class Outer {
    private String message = "Hello from Outer class";

    class Inner {
        void display() {
            System.out.println(message);  // Accessing outer class private
data
        }
    }

    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();  // Creating an inner class
object
        inner.display();
    }
}
```

🔹 **Output:** `Hello from Outer class`

---

2. Static Nested Class

A `static` inner class that **does not require an instance of the outer class**.

Example:

```
java
CopyEdit
class Outer {
    static class Inner {
        void display() {
            System.out.println("Inside Static Inner Class");
        }
    }

    public static void main(String[] args) {
        Outer.Inner inner = new Outer.Inner();  // No need to create an
Outer class object
        inner.display();
    }
}
```

🔹 **Output:** `Inside Static Inner Class`

---

3. Local Inner Class

A class defined **inside a method**. It can only be accessed within that method.

Example:

```
java
CopyEdit
class Outer {
    void outerMethod() {
        class Inner {
            void display() {
```

```
                System.out.println("Inside Local Inner Class");
            }
        }
        Inner inner = new Inner();
        inner.display();
    }

    public static void main(String[] args) {
        Outer outer = new Outer();
        outer.outerMethod();
    }
}
```

🔹 **Output:** `Inside Local Inner Class`

---

4. Anonymous Inner Class

A class **without a name** that is used for **one-time use**, usually for implementing interfaces or abstract classes.

Example: *(Using an interface)*

```
java
CopyEdit
interface Greeting {
    void sayHello();
}

public class AnonymousExample {
    public static void main(String[] args) {
        Greeting obj = new Greeting() {  // Anonymous class implementing
Greeting
            public void sayHello() {
                System.out.println("Hello from Anonymous Inner Class!");
            }
        };
        obj.sayHello();
    }
}
```

🔹 **Output:** `Hello from Anonymous Inner Class!`

Advantages of Inner Classes

✅ **Encapsulation:** Keeps related classes together.
✅ **Improved Readability:** Reduces unnecessary class files.
✅ **Access to Outer Class Members:** Can access `private` members of the outer class.

Disadvantages of Inner Classes

❌ **Increases Complexity:** Code might be harder to understand.
❌ **Less Reusable:** Inner classes are tightly coupled with the outer class.

4)What is reflection and how does it help to manipulate java code. (4 Marks)

Reflection in Java is a powerful feature that allows a program to inspect and manipulate classes, methods, and fields at runtime, even if their names are not known at compile time.

Why Use Reflection?

- To examine class structures dynamically.
- To access private fields and methods.
- To create instances of classes dynamically.
- To call methods dynamically.

Key Classes in Reflection (from `java.lang.reflect` package)

1. `Class<?>` – Represents a class or interface.
2. `Field` – Represents a field (variable) of a class.
3. `Method` – Represents a method of a class.
4. `Constructor<?>` – Represents a constructor.

Example: Using Reflection in Java

```
import java.lang.reflect.Method;
import java.lang.reflect.Field;
import java.lang.reflect.Constructor;

class Person {
    private String name;

    public Person() {
        this.name = "Default Name";
    }

    public void sayHello() {
        System.out.println("Hello, my name is " + name);
    }
}

public class ReflectionExample {
    public static void main(String[] args) {
        try {
            // Get Class object
            Class<?> personClass = Class.forName("Person");

            // Get constructor and create an instance
            Constructor<?> constructor = personClass.getConstructor();
            Object personInstance = constructor.newInstance();

            // Get and invoke method
            Method method = personClass.getMethod("sayHello");
            method.invoke(personInstance);

            // Get and modify private field
            Field field = personClass.getDeclaredField("name");
            field.setAccessible(true); // Allow access to private field
```

```
            field.set(personInstance, "John Doe");

            // Invoke method again after modification
            method.invoke(personInstance);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

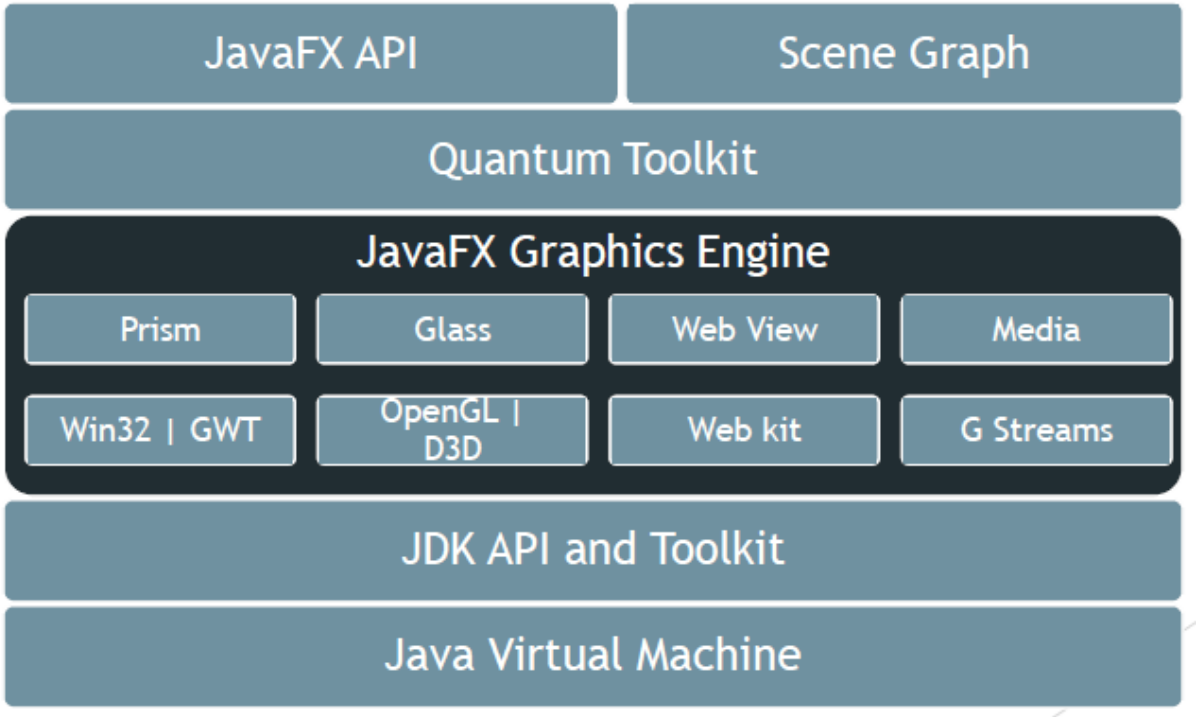
Output:

```
pgsql
CopyEdit
Hello, my name is Default Name
Hello, my name is John Doe
```

Key Takeaways

- **Reflection** is used to inspect and modify classes, methods, and fields at runtime.
- It helps in dynamic behavior (like frameworks and libraries).
- It should be used carefully as it affects performance and security.

5) Explain the architecture of JavaFX. (4 Mark)



- Scene Graph
  - A Scene Graph is the starting point of the construction of the GUI Application. It holds the (GUI) application primitives that are termed as nodes.
  - A node is a visual/graphical object and it may include
    - Geometrical (Graphical) objects
    - UI controls

        - Containers
        - Media elements
- Prism
    - Prism is a high performance hardware–accelerated graphical pipeline that is used to render the graphics in JavaFX. It can render both 2-D and 3-D graphics.
- GWT (Glass Windowing Toolkit)
    - GWT provides services to manage Windows, Timers, Surfaces and Event Queues.
    - GWT connects the JavaFX Platform to the Native Operating System.
- Quantum Toolkit
    - It is an abstraction over the low-level components of Prism, Glass, Media Engine, and Web Engine. It ties Prism and GWT together and makes them available to JavaFX.
- WebView
    - WebView is the component of JavaFX which is used to process HTML content. It uses a technology called Web Kit, which is an internal open-source web browser engine. This component supports different web technologies like HTML5, CSS, JavaScript, DOM and SVG.
- Media Engine
    - The JavaFX media engine is based on an open-source engine known as a Streamer. This media engine supports the playback of video and audio content.

6) Discuss JavaFX benefits? (4 Marks)

JavaFX offers many features designed to meet the requirements of modern application development. These features include:

• **Rich GUI elements:** Offers a wide variety of GUI elements to add to your applications. These elements include buttons, text boxes, tables, graphics, images, and media players.

• **CSS-based style and theme options:** Offers pre-designed style and theme options to provide a modern and aesthetic appearance for your application. These options can be easily customized using CSS (Cascading Style Sheets).

• **Animations and transitions:** Allows you to add animations and transitions to the GUI elements in your application. These features help improve the user experience.

• **2D and 3D graphics:** It is a tool that supports 2D and 3D graphics. This feature is ideal for developing graphic-intensive applications such as games or simulations.

• **Media players:** Provides pre-designed media players to play video and audio media in your application.

• **FXML:** Applications can be designed using an XML-based language called FXML instead of writing Java code. This makes the GUI design and development process easier.

7) Explain mouse and key event handler in JavaFX. (3 Marks)

1. Key Event Handling in JavaFX

- **What is a Key Event in JavaFX?**

A **Key Event** occurs when a user interacts with the keyboard. JavaFX provides the `KeyEvent` class to handle such events.
Common key events include:

- **KEY_PRESSED** → When a key is pressed
- **KEY_RELEASED** → When a key is released
- **KEY_TYPED** → When a character is typed

    - **How to Handle Key Events?**

We can handle key events using **Event Handlers** or **Lambda Expressions** with methods like:

- `setOnKeyPressed()`
- `setOnKeyReleased()`
- `setOnKeyTyped()`

2. Mouse Events in JavaFX:

- JavaFX provides the `MouseEvent` class to handle mouse interactions like:
    - **MOUSE_CLICKED** (when the mouse is clicked)
    - **MOUSE_MOVED** (when the mouse moves)
    - **MOUSE_PRESSED** (when a mouse button is pressed)
    - **MOUSE_RELEASED** (when a mouse button is released)

```
Text text = new Text(20, 20, "Programming is fun");
pane.getChildren().addAll(text);
text.setOnMouseDragged(e -> {
  text.setX(e.getX());
  text.setY(e.getY());
});
```

8) Explain about adapter classes and mouse events with an example. (3 Marks)

Adapter Class in Java

1. What is an Adapter Class?

- An **adapter class** in Java provides **empty implementations** for all methods of an interface.
- It is useful when a class needs to implement an interface but does not require all its methods.
- Instead of implementing the entire interface, we **extend the adapter class** and override only the required methods.

2. Common Adapter Classes in Java:

- `MouseAdapter` (for `MouseListener`)
- `KeyAdapter` (for `KeyListener`)
- `WindowAdapter` (for `WindowListener`)

Unit 9

1) Create a class called Student. Write a student manager program to manipulate the student information from files by using FileInputStream and FileOutputStream. (7 Marks)

```
import java.io.*;
class Student implements Serializable
{
	String name;
	String regno;
	public Student(String name,String regno)
	{
		this.name=name;
		this.regno=regno;
	}
	public void display()
	{
		System.out.println(name+regno);
	}
}
public class DemoObjectInputStream
{
	public static void main(String args[])throws IOException,ClassNotFoundException
	{
		Student s=new Student("Student1","657");
		//File ob=new File("DataTxt.txt");
		FileOutputStreamfout=new FileOutputStream("ObjTxt.txt");
		ObjectOutputStreamobjout=new ObjectOutputStream(fout);
		objout.writeObject(s);
		objout.close();
		FileInputStream fin=new FileInputStream("ObjTxt.txt");
		ObjectInputStreamobjin=new ObjectInputStream(fin);
		Student instudent=null;
		instudent=(Student)objin.readObject();
		System.out.println(instudent.name);
		System.out.println(instudent.regno);
	}
}
```

2) Write a program that counts the number of words in a text file. The file name is passed as a command line argument. The words in the file are separated by white space characters (7 Marks)

```
import java.io.*;

import java.util.Scanner;



public class WordCount {

   public static void main(String[] args) {

      // Check if a filename is provided
```

```
        if (args.length != 1) {
            System.out.println("Usage: java WordCount <filename>");
            return;
        }

        String filename = args[0];
        int wordCount = 0;

        try {
            File file = new File(filename);
            Scanner scanner = new Scanner(file);

            // Read file word by word
            while (scanner.hasNext()) {
                scanner.next();
                wordCount++;
            }

            scanner.close();
            System.out.println("Total number of words: " + wordCount);
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + filename);
        }
    }
}
```

3) Write a JAVA program to read student.txt file and display the content. (4 Marks)

```
import java.io.*;


public class ReadStudentFile {
   public static void main(String[] args) {
      String filename = "student.txt"; // File to read


      try {
         File file = new File(filename);
         BufferedReader br = new BufferedReader(new FileReader(file));


         String line;
         while ((line = br.readLine()) != null) {
            System.out.println(line); // Print each line
         }


         br.close();
      } catch (FileNotFoundException e) {
         System.out.println("File not found: " + filename);
      } catch (IOException e) {
         System.out.println("Error reading file: " + filename);
      }
   }
}
```

Unit -10

1) Write a java program that evaluates a math expression given in string form command line arguments.

```
import java.util.Stack;


public class MathExpressionEvaluator {
   public static void main(String[] args) {
      if (args.length == 0) {
         System.out.println("Usage: java MathExpressionEvaluator \"expression\"");
         return;
      }

      String expression = String.join("", args).replaceAll("\\s", "");
      System.out.println("Result: " + evaluateExpression(expression));
   }

   public static int evaluateExpression(String expression) {
      Stack<Integer> numbers = new Stack<>();
      Stack<Character> operators = new Stack<>();

      for (int i = 0; i < expression.length(); i++) {
         char c = expression.charAt(i);

         if (Character.isDigit(c)) {
            int num = 0;
            while (i < expression.length() && Character.isDigit(expression.charAt(i))) {
               num = num * 10 + (expression.charAt(i) - '0');
               i++;
```

```
            }
            i--;
            numbers.push(num);
        } else if (c == '(') {
            operators.push(c);
        } else if (c == ')') {
            while (operators.peek() != '(') {
                numbers.push(applyOperation(operators.pop(), numbers.pop(), numbers.pop()));
            }
            operators.pop();
        } else if (isOperator(c)) {
            while (!operators.isEmpty() && precedence(operators.peek()) >= precedence(c)) {
                numbers.push(applyOperation(operators.pop(), numbers.pop(), numbers.pop()));
            }
            operators.push(c);
        }
    }

    while (!operators.isEmpty()) {
        numbers.push(applyOperation(operators.pop(), numbers.pop(), numbers.pop()));
    }

    return numbers.pop();
}

private static boolean isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}
```

```
    private static int precedence(char op) {
        if (op == '+' || op == '-') return 1;
        if (op == '*' || op == '/') return 2;
        return 0;
    }

    private static int applyOperation(char op, int b, int a) {
        switch (op) {
            case '+': return a + b;
            case '-': return a - b;
            case '*': return a * b;
            case '/': return a / b;
            default: throw new IllegalArgumentException("Invalid operator");
        }
    }
}
```

2) Write a java program to take infix expressions and convert it into prefix expressions.

```
import java.util.Stack;

public class InfixToPrefixConverter {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: java InfixToPrefixConverter \"expression\"");
            return;
        }

        String infix = String.join("", args).replaceAll("\\s", "");
```

```
    System.out.println("Prefix Expression: " + infixToPrefix(infix));
}

public static String infixToPrefix(String infix) {
    // Reverse the infix expression
    String reversedInfix = reverseExpression(infix);

    // Convert reversed infix to postfix
    String reversedPostfix = infixToPostfix(reversedInfix);

    // Reverse the postfix to get prefix
    return new StringBuilder(reversedPostfix).reverse().toString();
}

private static String reverseExpression(String expr) {
    StringBuilder reversed = new StringBuilder();
    for (int i = expr.length() - 1; i >= 0; i--) {
        char c = expr.charAt(i);
        if (c == '(') {
            reversed.append(')');
        } else if (c == ')') {
            reversed.append('(');
        } else {
            reversed.append(c);
        }
    }
    return reversed.toString();
}
```

```
private static String infixToPostfix(String infix) {
    Stack<Character> stack = new Stack<>();
    StringBuilder postfix = new StringBuilder();

    for (int i = 0; i < infix.length(); i++) {
        char c = infix.charAt(i);

        if (Character.isLetterOrDigit(c)) {
            postfix.append(c);
        } else if (c == '(') {
            stack.push(c);
        } else if (c == ')') {
            while (!stack.isEmpty() && stack.peek() != '(') {
                postfix.append(stack.pop());
            }
            stack.pop();
        } else if (isOperator(c)) {
            while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(c)) {
                postfix.append(stack.pop());
            }
            stack.push(c);
        }
    }

    while (!stack.isEmpty()) {
        postfix.append(stack.pop());
    }

    return postfix.toString();
```

```
    }

    private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private static int precedence(char op) {
        if (op == '+' || op == '-') return 1;
        if (op == '*' || op == '/') return 2;
        return 0;
    }
}
```