Experiment 1

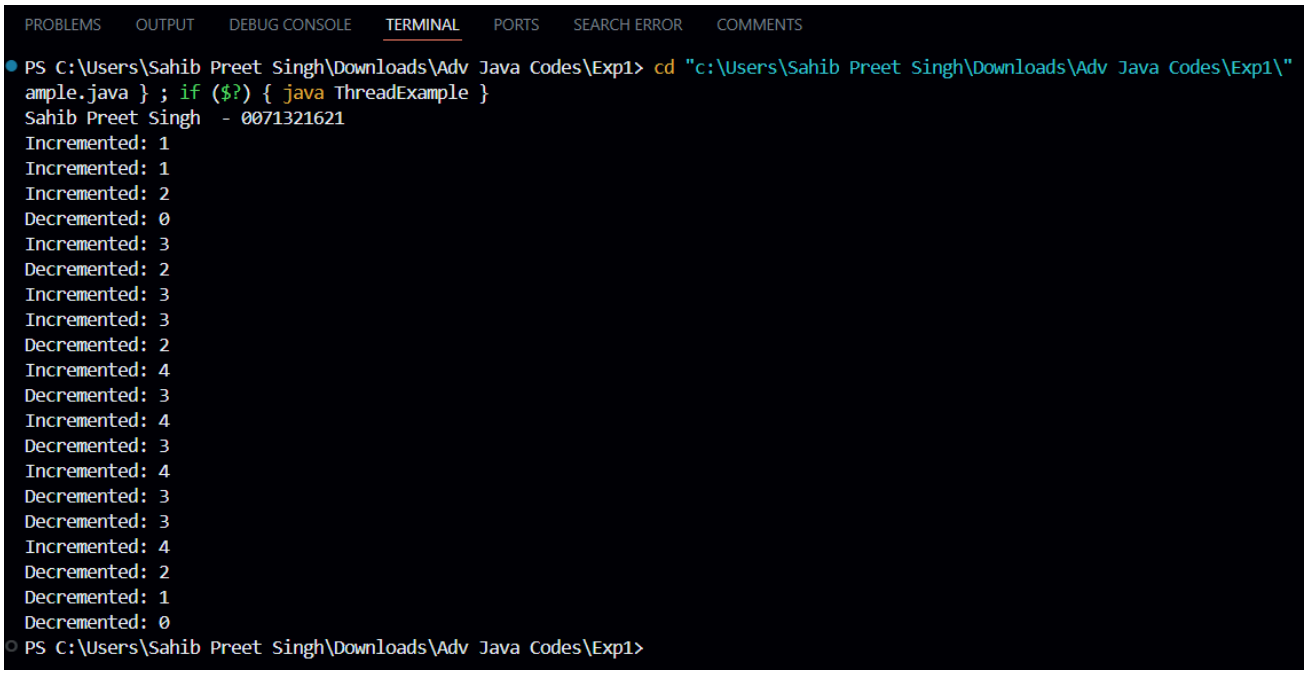
Aim:

Write a java program of thread synchronization, inter-thread communication, and thread pooling.

Code:

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

class Counter {
    private int count = 0;
    public synchronized void increment() {
        count++;
        notify();
    }
    public synchronized void decrement() {
        while (count <= 0) {
            try { wait(); } catch (InterruptedException e) {
Thread.currentThread().interrupt(); }
        }
        count--;
    }
    public int getCount() { return count; }
}

public class ThreadExample {
    public static void main(String[] args) {
        Counter counter = new Counter();
        ExecutorService executor = Executors.newFixedThreadPool(2);

        Runnable incrementTask = () -> {
            for (int i = 0; i < 10; i++) {
                counter.increment();
                System.out.println("Incremented: " + counter.getCount());
            }
        };
        Runnable decrementTask = () -> {
            for (int i = 0; i < 10; i++) {
                counter.decrement();
                System.out.println("Decremented: " + counter.getCount());
            }
        };
        executor.submit(incrementTask);
        executor.submit(decrementTask);
        executor.shutdown();
    }
}
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SEARCH ERROR
COMMENTS
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp1> cd "c:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp1\"
ample.java } ; if ($?) { java ThreadExample }
Sahib Preet Singh - 0071321621
Incremented: 1
Incremented: 1
Incremented: 2
Decremented: 0
Incremented: 3
Decremented: 2
Incremented: 3
Incremented: 3
Decremented: 2
Incremented: 4
Decremented: 3
Incremented: 4
Decremented: 3
Incremented: 4
Decremented: 3
Decremented: 3
Incremented: 4
Decremented: 2
Decremented: 1
Decremented: 0
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp1>

Experiment 2

Aim:

Implement a client-server application using Java's networking APIs.

Code:

Client-Side Code:

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 5000)) {
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            System.out.println("Server message: " + in.readLine());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Server-Side Code:

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(5000)) {
            System.out.println("Server started...");
            Socket socket = serverSocket.accept();
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.println("Hello from the Server!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp2> javac Server.java Client.java
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp2> java Server
Server started...
```

```
C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp2>javac Server.java Client.java

C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp2>java Client
Server message: Hello from the Server!
```

Experiment 3

Aim:

Design a calculator, a simple text editor, or a graphical game with user interaction and visual components. Explore event handling, layout managers, and UI design principles.

Code:

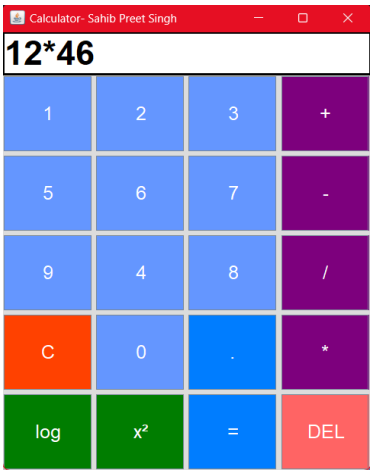
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.DecimalFormat;

public class Calculator extends JFrame implements ActionListener {
    // Components for the calculator
    private JTextField display;
    private JButton[] numberButtons = new JButton[10];
    private JButton addButton, subButton, mulButton, divButton;
    private JButton decButton, equButton, clrButton, delButton;
    private JButton logButton, squareButton;
    // Variables for calculation
    private double num1 = 0, num2 = 0, result = 0;
    private char operator;
    private DecimalFormat df = new DecimalFormat("#.#####"); // Format for display
    public Calculator() {
        // Frame settings
        setTitle("Calculator- Sahib Preet Singh");
        setSize(400, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Display field
        display = new JTextField();
        display.setEditable(false);
        display.setFont(new Font("Arial", Font.BOLD, 36)); // Bigger font size
        display.setBackground(Color.WHITE);
        display.setForeground(Color.BLACK);
           display.setBorder(BorderFactory.createLineBorder(Color.BLACK, 2)); // Border for
display
        add(display, BorderLayout.NORTH);
        // Panel for buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(5, 4, 5, 5)); // 5 rows and 4 columns
        buttonPanel.setBackground(new Color(220, 220, 220));
        // Number buttons
        for (int i = 0; i < 10; i++) {
              numberButtons[i] = createButton(String.valueOf(i), new Color(100, 150, 255));
}
```

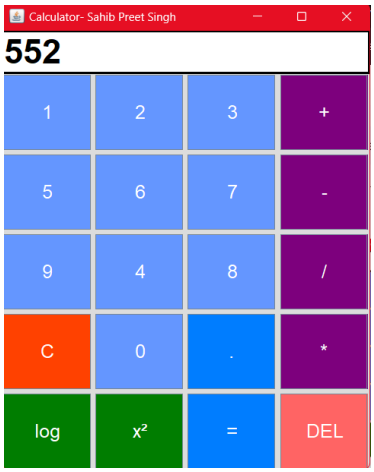
```
        addButton = createButton("+", new Color(128, 0, 128)); // Purple for addition
        subButton = createButton("-", new Color(128, 0, 128)); // Purple for subtraction
        mulButton = createButton("*", new Color(128, 0, 128)); // Purple for multiplication
        divButton = createButton("/", new Color(128, 0, 128)); // Purple for division
        logButton = createButton("log", new Color(0, 128, 0));
        squareButton = createButton("x²", new Color(0, 128, 0));
        // Decimal, Equals, Clear buttons
        decButton = createButton(".", new Color(0, 128, 255));
        equButton = createButton("=", new Color(0, 128, 255));
        clrButton = createButton("C", new Color(255, 69, 0));
        delButton = createButton("DEL", new Color(255, 100, 100)); // Create the DEL button
        // Add buttons to the panel in specified layout
        buttonPanel.add(numberButtons[1]);
        buttonPanel.add(numberButtons[2]);
        buttonPanel.add(numberButtons[3]);
        buttonPanel.add(addButton);
        buttonPanel.add(numberButtons[5]);
        buttonPanel.add(numberButtons[6]);
        buttonPanel.add(numberButtons[7]);
        buttonPanel.add(subButton);
        buttonPanel.add(numberButtons[9]);
        buttonPanel.add(numberButtons[4]);
        buttonPanel.add(numberButtons[8]);
        buttonPanel.add(divButton);
        buttonPanel.add(clrButton);
        buttonPanel.add(numberButtons[0]);
        buttonPanel.add(decButton);
        buttonPanel.add(mulButton);
        buttonPanel.add(logButton);
        buttonPanel.add(squareButton);
        buttonPanel.add(equButton);
        buttonPanel.add(delButton);
        // Add button panel to frame
        add(buttonPanel, BorderLayout.CENTER);
        setVisible(true);
    }
    private JButton createButton(String text, Color color) {
        JButton button = new JButton(text);
        button.setFont(new Font("Arial", Font.PLAIN, 20));
        button.setBackground(color);
        button.setForeground(Color.WHITE); // White text color
        button.setFocusPainted(false);
        button.addActionListener(this);
        return button;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // Number button action
        for (int i = 0; i < 10; i++) {
            if (e.getSource() == numberButtons[i]) {
                display.setText(display.getText() + i);
```

```
    }
}
// Decimal point action
if (e.getSource() == decButton) {
    if (!display.getText().contains(".")) {
        display.setText(display.getText() + ".");
    }
}
// Operator actions
if (e.getSource() == addButton || e.getSource() == subButton ||
        e.getSource() == mulButton || e.getSource() == divButton) {
    num1 = Double.parseDouble(display.getText());
    operator = ((JButton) e.getSource()).getText().charAt(0);
    display.setText(display.getText() + operator);
}
// Equals action
if (e.getSource() == equButton) {
    String[] parts = display.getText().split("\\+|\\-|\\*|\\/");
    if (parts.length < 2)
        return; // Ensure we have two numbers
    num1 = Double.parseDouble(parts[0]);
    num2 = Double.parseDouble(parts[1]);

    switch (operator) {
        case '+':
            result = num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
        case '*':
            result = num1 * num2;
            break;
        case '/':
            if (num2 != 0) {
                result = num1 / num2;
            } else {
                display.setText("Error");
                return;
            }
            break;
    }
    display.setText(df.format(result));
}
// Clear action
if (e.getSource() == clrButton) {
    display.setText("");
}
if (e.getSource() == delButton) {
    String text = display.getText();
    if (text.length() > 0) {
              display.setText(text.substring(0, text.length() - 1)); // Remove last
```

```
character
            }
        }
        // Logarithm action
        if (e.getSource() == logButton) {
            num1 = Double.parseDouble(display.getText());
            result = Math.log(num1);
            display.setText(df.format(result));
        }
        // Square action
        if (e.getSource() == squareButton) {
            num1 = Double.parseDouble(display.getText());
            result = Math.pow(num1, 2);
            display.setText(df.format(result));
        }
    }
    public static void main(String[] args) {
        new Calculator();
    }
}
```

Output:





Experiment 4

Aim:

Implement functionalities like data retrieval, insertion, deletion, and updating records. Explore concepts like JDBC, SQL queries, and database transactions.

Code:

```
import java.sql.*;

public class Database {
    private static final String URL = "jdbc:mysql://localhost:3306/EmployeeGTBIT";
    private static final String USER = "Sahib007";
    private static final String PASSWORD = "root";

    private Connection connection;

    public Database() {
        try {
            connection = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("Database connected successfully.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Insert a new record into the users table
    public void insertRecord(String name, String email, int age) {
        String query = "INSERT INTO users (name, email, age) VALUES (?, ?, ?)";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setString(1, name);
            statement.setString(2, email);
            statement.setInt(3, age);
            int rowsInserted = statement.executeUpdate();
            System.out.println(rowsInserted + " row(s) inserted.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Retrieve all records from the users table
    public void retrieveRecords() {
        String query = "SELECT * FROM users";
        try (Statement statement = connection.createStatement();
             ResultSet resultSet = statement.executeQuery(query)) {
            System.out.println("ID | Name | Email | Age");
            while (resultSet.next()) {
                System.out.printf("%d | %s | %s | %d%n",
                        resultSet.getInt("id"),
```

```
                    resultSet.getString("name"),
                    resultSet.getString("email"),
                    resultSet.getInt("age"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Update a user's details based on their ID
    public void updateRecord(int id, String name, String email, int age) {
        String query = "UPDATE users SET name = ?, email = ?, age = ? WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setString(1, name);
            statement.setString(2, email);
            statement.setInt(3, age);
            statement.setInt(4, id);
            int rowsUpdated = statement.executeUpdate();
            System.out.println(rowsUpdated + " row(s) updated.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Delete a record based on the user ID
    public void deleteRecord(int id) {
        String query = "DELETE FROM users WHERE id = ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setInt(1, id);
            int rowsDeleted = statement.executeUpdate();
            System.out.println(rowsDeleted + " row(s) deleted.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Use a transaction to insert multiple records atomically
    public void insertMultipleRecordsWithTransaction() {
        String query1 = "INSERT INTO users (name, email, age) VALUES ('Shubh',
'shubh00@gmail.com', 20)";
        String query2 = "INSERT INTO users (name, email, age) VALUES ('Goel', 'goel@eg.com',
23)";

        try {
            connection.setAutoCommit(false);  // Begin transaction

            try (Statement statement = connection.createStatement()) {
                statement.executeUpdate(query1);
                statement.executeUpdate(query2);

                connection.commit();  // Commit transaction if successful
                System.out.println("Transaction committed successfully.");
```

```
            } catch (SQLException e) {
                connection.rollback();  // Rollback transaction if any error occurs
                System.out.println("Transaction rolled back due to error.");
                e.printStackTrace();
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                connection.setAutoCommit(true);  // Restore default commit behavior
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        Database dbOps = new Database();

        // Example usage
        dbOps.insertRecord("Sahib Preet Singh", "sahib61003@ieee.org", 007);
        dbOps.retrieveRecords();
        dbOps.updateRecord(1, "Prabhjot Singh", "prabhleg@gmail.com", 050);
        dbOps.deleteRecord(1);
        dbOps.insertMultipleRecordsWithTransaction();
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp4> cd "c
$?) { java Database }
Database connected successfully.
1 row(s) inserted.
ID | Name             | Email                | Age
1  | Sahib Preet Singh | sahib61003@ieee.org  | 7
1 row(s) updated.
1 row(s) deleted.
Transaction committed successfully.
ID | Name   | Email              | Age
2  | Shubh  | shubh00@gmail.com  | 20
3  | Goel   | goel@eg.com        | 23
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp4>
```

Experiment 5

Aim:

Utilize third-party libraries or frameworks in Java programming. Choose a popular library (e.g., Apache Commons, Gson, Log4j) and develop programs that showcase its features and functionality.

Code:

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonParseException;

class User {
    private int id;
    private String name;
    private String email;

    // Constructor
    public User(int id, String name, String email) {
        this.id = id;
        this.name = name;
        this.email = email;
    }

    // Getters
    public int getId() { return id; }
    public String getName() { return name; }
    public String getEmail() { return email; }

    @Override
    public String toString() {
        return "User{id=" + id + ", name='" + name + "', email='" + email + "'}";
    }
}

public class GsonDemo {
    public static void main(String[] args) {
        // Initialize Gson with pretty printing
        Gson gson = new GsonBuilder().setPrettyPrinting().create();

        // Create a User object
        User user = new User(1, "Sahib Preet Singh", "sahib61003@ieee.org");

        // Serialize the User object to JSON
        String jsonString = gson.toJson(user);
        System.out.println("Serialized JSON:");
        System.out.println(jsonString);
```

```
        // Deserialize JSON string back to User object
        String jsonInput = "{\"id\":1,\"name\":\"Sahib Preet
Singh\",\"email\":\"sahib61003@ieee.org\"}";

        try {
            User deserializedUser = gson.fromJson(jsonInput, User.class);
            System.out.println("\nDeserialized User object:");
            System.out.println(deserializedUser);
        } catch (JsonParseException e) {
            System.out.println("Failed to parse JSON: " + e.getMessage());
        }

        // Handling more complex JSON (additional data)
        String complexJson = "{"
                + "\"id\":2,"
                + "\"name\":\"Goel\","
                + "\"email\":\"goel@eg.com\""
                + "}";

        try {
            User complexUser = gson.fromJson(complexJson, User.class);
            System.out.println("\nDeserialized Complex User object:");
            System.out.println(complexUser);
        } catch (JsonParseException e) {
            System.out.println("Failed to parse complex JSON: " + e.getMessage());
        }
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp5> cd "c:\Users\Sahib Preet Singh\Downloads\Adv Java Codes
$?) { java GsonDemo }
Serialized JSON:
{
  "id": 1,
  "name": "Sahib Preet Singh",
  "email": "sahib61003@ieee.org"
}

Deserialized User object:
User{id=1, name='Sahib Preet Singh', email='sahib61003@ieee.org'}

Deserialized Complex User object:
User{id=2, name='Goel', email='goel@eg.com'}
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp5>
```

Experiment 6

Aim:

Write a java program to writes objects to a file in a serialized format and then reads and reconstructs the objects from the file.

Code:

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;

// Define a User class that implements Serializable
class User implements Serializable {
    private static final long serialVersionUID = 1L; // For version control
    private int id;
    private String name;
    private String email;

    public User(int id, String name, String email) {
        this.id = id;
        this.name = name;
        this.email = email;
    }

    @Override
    public String toString() {
        return "User{id=" + id + ", name='" + name + "', email='" + email + "'}";
    }
}

public class SerializationExperiment {

    // Method to serialize the list of users to a file
    public static void serializeUsers(List<User> users, String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(users);
            System.out.println("Serialization successful. Users written to " + filename);
        } catch (IOException e) {
            System.err.println("Serialization error: " + e.getMessage());
        }
    }

    // Method to deserialize users from a file
    @SuppressWarnings("unchecked")
    public static List<User> deserializeUsers(String filename) {
        List<User> users = null;
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {
```

```
            users = (List<User>) ois.readObject();
            System.out.println("Deserialization successful. Users read from " + filename);
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Deserialization error: " + e.getMessage());
        }
        return users;
    }

    public static void main(String[] args) {
        String filename = "Serializeddata.ser"; // File name for serialized data

        // Create a list of users
        List<User> users = new ArrayList<>();
        users.add(new User(61, "Sahib", "sahib61003@gmail.com"));
        users.add(new User(62, "Preet", "sahib61003@ieee.org"));
        users.add(new User(63, "Singh", "singhgtbit@gmail.com"));

        // Serialize the users to a file
        serializeUsers(users, filename);

        // Deserialize the users from the file
        List<User> deserializedUsers = deserializeUsers(filename);

        // Print the deserialized users
        if (deserializedUsers != null) {
            System.out.println("\nDeserialized Users:");
            for (User user : deserializedUsers) {
                System.out.println(user);
            }
        }
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp6> cd "c:\Users\Sahib Preet Singh\E
t.java } ; if ($?) { java SerializationExperiment }
Serialization successful. Users written to Serializeddata.ser
Deserialization successful. Users read from Serializeddata.ser

Deserialized Users:
User{id=61, name='Sahib', email='sahib61003@gmail.com'}
User{id=62, name='Preet', email='sahib61003@ieee.org'}
User{id=63, name='Singh', email='singhgtbit@gmail.com'}
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp6>
```

Experiment 7

Aim:

Write a java program that uses reflection to inspect and modify the behavior of objects based on user input or external configuration

Code:

```
import java.lang.reflect.Field;
import java.util.Scanner;

// Sample class with private fields
class Person {
    private String name;
    private int age;

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

    // Getters
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

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

        // Create an instance of Person
        Person person = new Person("CRofClass", 21);
        System.out.println("Original Person: " + person.getName() + ", Age: " +
person.getAge());
        // Get the class of the Person object
        Class<?> personClass = person.getClass();
        // Modify the fields based on user input
        System.out.print("Enter new name: ");
        String newName = scanner.nextLine();
        System.out.print("Enter new age: ");
        int newAge = scanner.nextInt();

        try {
            // Access the private fields using reflection
```

```
            Field nameField = personClass.getDeclaredField("name");
            Field ageField = personClass.getDeclaredField("age");
            // Make the fields accessible
            nameField.setAccessible(true);
            ageField.setAccessible(true);
            // Modify the values of the fields
            nameField.set(person, newName);
            ageField.set(person, newAge);
            // Print the modified Person object
            System.out.println("Modified Person: " + person.getName() + ", Age: " +
person.getAge());

        } catch (NoSuchFieldException | IllegalAccessException e) {
            e.printStackTrace();
        }
        scanner.close();
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp7> cd "c:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp7\
 if ($?) { java Reflectionlib }
Original Person: CRofClass, Age: 21
Enter new name: Sahib Preet Singh
Enter new age: 22
Modified Person: Sahib Preet Singh, Age: 22
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp7>
```

Experiment 8

Aim:

Implement generic methods to perform operations like sorting, searching, or filtering on generic collections.

Code:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.function.Predicate;

public class GenericsCollection {

    // Generic method for sorting a list
    public static <T extends Comparable<T>> void sortList(List<T> list) {
        Collections.sort(list);
    }

    // Generic method for searching an item in a list
    public static <T> int searchItem(List<T> list, T item) {
        return list.indexOf(item);
    }

    // Generic method for filtering a list based on a predicate
    public static <T> List<T> filterList(List<T> list, Predicate<T> predicate) {
        List<T> filteredList = new ArrayList<>();
        for (T item : list) {
            if (predicate.test(item)) {
                filteredList.add(item);
            }
        }
        return filteredList;
    }

    public static void main(String[] args) {
        // Create a list of integers
        List<Integer> numbers = new ArrayList<>();
        numbers.add(6);
        numbers.add(1);
        numbers.add(0);
        numbers.add(3);
        numbers.add(7);
        numbers.add(5);
        numbers.add(8);

        // Sort the list
        System.out.println("Original list: " + numbers);
        sortList(numbers);
```

```
        System.out.println("Sorted list: " + numbers);

        // Search for an item
        int searchItem = 7;
        int index = searchItem(numbers, searchItem);
        System.out.println("Item " + searchItem + " found at index: " + index);

        // Create a list of strings
        List<String> names = new ArrayList<>();
        names.add("Sahib");
        names.add("Aman");
        names.add("Divyajeet");
        names.add("Gursimar");
        names.add("Rohan");

        // Filter names that start with 'S'
        List<String> filteredNames = filterList(names, name -> name.startsWith("S"));
        System.out.println("Names starting with 'S': " + filteredNames);
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp8> cd "c:\Users\Sahib Preet Singh\Downloads
a } ; if ($?) { java GenericsCollection }
Original list: [6, 1, 0, 3, 7, 5, 8]
Sorted list: [0, 1, 3, 5, 6, 7, 8]
Item 7 found at index: 5
Names starting with 'S': [Sahib]
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp8>
```

Experiment 9

Aim:

Design custom annotations and use them in a Java program to provide additional metadata and define behavior.

Code:

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.reflect.Method;

// Define a custom annotation
@Retention(RetentionPolicy.RUNTIME)
@interface OperationInfo {
    String description();
}

// Class with methods that use the custom annotation
class Calculator {

    @OperationInfo(description = "This method performs addition of two values.")
    public int calculateSum(int x, int y) {
        return x + y;
    }

    @OperationInfo(description = "This method performs subtraction of two values.")
    public int calculateDifference(int x, int y) {
        return x - y;
    }
}

public class CustomAnnotationExample {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();

        // Reflectively inspect methods for annotations
        Method[] methods = calculator.getClass().getDeclaredMethods();
        for (Method method : methods) {
            // Check if the method has the custom annotation
            if (method.isAnnotationPresent(OperationInfo.class)) {
                OperationInfo annotation = method.getAnnotation(OperationInfo.class);
                System.out.println("Operation: " + method.getName() + " - " +
annotation.description());

                // Call the method for demonstration
                try {
                    if (method.getName().equals("calculateSum")) {
                        System.out.println("Result of calculateSum: " +
method.invoke(calculator, 10, 7));
```

```
                    } else if (method.getName().equals("calculateDifference")) {
                        System.out.println("Result of calculateDifference: " +
method.invoke(calculator, 10, 7));
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        System.out.println("Sahib Preet Singh 00713211621 AI-ML");
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp9> cd "c:\Users\Sahib Pr
e.java } ; if ($?) { java CustomAnnotationExample }
Operation: calculateSum - This method performs addition of two values.
Result of calculateSum: 17
Operation: calculateDifference - This method performs subtraction of two values.
Result of calculateDifference: 3
Sahib Preet Singh 00713211621 AI-ML
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp9>
```

Experiment 10

Aim:

Write a java program to Integrate Java with native code by using the JNI (with native libraries written in C/C++).

Code:

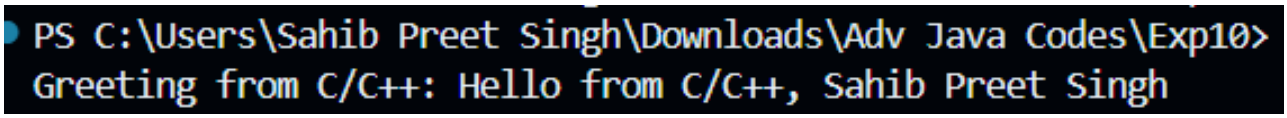
JniExample.Java file:

```
public class JniExample {
    // Declare the native method
    public native String sayHello(String name);

    // Load the native library
    static {
        System.loadLibrary("NativeLibrary");  // Loads NativeLibrary.dll (Windows) or
libNativeLibrary.so (Linux/Mac)
    }
    public static void main(String[] args) {
        JniExample example = new JniExample();
        String greeting = example.sayHello("Sahib Preet Singh");
        System.out.println("Greeting from C/C++: " + greeting);
    }
}
```

NativeLibrary.C

```
#include <jni.h>
#include <stdio.h>
#include <string.h>

// Import the JniExample header file generated by javah (JniExample.h) if available
#include "JniExample.h"
// JNI function implementation of `sayHello`
JNIEXPORT jstring JNICALL Java_JniExample_sayHello(JNIEnv *env, jobject thisObj, jstring
name) {
    const char *nameStr = (*env)->GetStringUTFChars(env, name, NULL);
    char message[60] = "Hello from C/C++, ";
    strcat(message, nameStr);
    (*env)->ReleaseStringUTFChars(env, name, nameStr);
    return (*env)->NewStringUTF(env, message);
}
```

Output:


PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp10>
Greeting from C/C++: Hello from C/C++, Sahib Preet Singh

Experiment 11

Aim:

Implement functional programming concepts and solve problems related to data manipulation, filtering, or mapping

Code:

```
import java.util.*;
import java.util.stream.Collectors;

class Person {
    private String name;
    private int age;
    private double salary;
    // Constructor
    public Person(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    // Getters
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public double getSalary() {
        return salary;
    }
    // Override toString method to display person information
    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + ", salary=" + salary + '}';
    }
}

public class FunctionalProgrammingExample {
    public static void main(String[] args) {
        // Create a list of Person objects
        List<Person> people = Arrays.asList(
                new Person("Sahib", 28, 55000),
                new Person("Preet", 35, 72000),
                new Person("Gursimar", 45, 80000),
                new Person("Divyajeet", 42, 90000),
```

```
                new Person("Sehdeep", 32, 65000),
                new Person("Rohan", 50, 100000),
                new Person("Monty", 38, 78000)
        );
        // 1. Filter people older than 30
        List<Person> olderThan30 = people.stream()
                .filter(person -> person.getAge() > 30)
                .collect(Collectors.toList());

        System.out.println("People older than 30:");
        olderThan30.forEach(System.out::println);

        // 2. Increase the salary of people older than 40 by 10%
        List<Person> updatedSalaries = people.stream()
                .map(person -> {
                    if (person.getAge() > 40) {
                        double newSalary = person.getSalary() * 1.1;
                        return new Person(person.getName(), person.getAge(), newSalary);
                    }
                    return person;
                })
                .collect(Collectors.toList());
        System.out.println("\nUpdated salaries for people over 40:");
        updatedSalaries.forEach(System.out::println);
        // 3. Get the average salary of people older than 30
        double avgSalaryOver30 = people.stream()
                .filter(person -> person.getAge() > 30)
                .mapToDouble(Person::getSalary)
                .average()
                .orElse(0.0);
        System.out.println("\nAverage salary of people over 30: " + avgSalaryOver30);

        // 4. Sort people by age and print their names
        List<String> sortedNames = people.stream()
                .sorted(Comparator.comparingInt(Person::getAge))
                .map(Person::getName)
                .collect(Collectors.toList());
        System.out.println("\nPeople sorted by age:");
        sortedNames.forEach(System.out::println);
        // 5. Get the person with the highest salary
        Person highestSalaryPerson = people.stream()
                .max(Comparator.comparingDouble(Person::getSalary))
                .orElse(null);
        System.out.println("\nPerson with the highest salary:");
        if (highestSalaryPerson != null) {
            System.out.println(highestSalaryPerson);
        }
        System.out.println("Sahib Preet Singh 00713211621");
    }
}
```

Output:

```
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp11> cd "c:\Users\Sahib Preet Singh\Downloads\
gExample.java } ; if ($?) { java FunctionalProgrammingExample }
People older than 30:
Person{name='Preet', age=35, salary=72000.0}
Person{name='Gursimar', age=45, salary=80000.0}
Person{name='Divyajeet', age=42, salary=90000.0}
Person{name='Sehdeep', age=32, salary=65000.0}
Person{name='Rohan', age=50, salary=100000.0}
Person{name='Monty', age=38, salary=78000.0}

Updated salaries for people over 40:
Person{name='Sahib', age=28, salary=55000.0}
Person{name='Preet', age=35, salary=72000.0}
Person{name='Gursimar', age=45, salary=88000.0}
Person{name='Divyajeet', age=42, salary=99000.00000000001}
Person{name='Sehdeep', age=32, salary=65000.0}
Person{name='Rohan', age=50, salary=110000.00000000001}
Person{name='Monty', age=38, salary=78000.0}

Average salary of people over 30: 80833.33333333333

People sorted by age:
Sahib
Sehdeep
Preet
Monty
Divyajeet
Gursimar
Rohan

Person with the highest salary:
Person{name='Rohan', age=50, salary=100000.0}
Sahib Preet Singh 00713211621
PS C:\Users\Sahib Preet Singh\Downloads\Adv Java Codes\Exp11>
```