Serialization Assignment-3

Skill Description:

"Java Case Study - Serialization" assignment centres around mastering the concept of Java Serialization, covering topics such as serialization basics, implementing the Serializable interface, handling transient variables, and deserialization. Participants will gain hands-on experience in designing, serializing, and deserializing objects, with a focus on creating resilient and efficient serialization mechanisms.

Problem Statement 2:

You are tasked with developing a distributed system where data needs to be transferred between different components. Design a Java application that serializes and deserializes data across networked devices. Implement error handling and recovery mechanisms to ensure data integrity during transmission.

Learning Outcomes:

- Proficiency in applying serialization for saving and loading game progress.
- Skill in handling complex object structures during serialization.
- Understanding the practical considerations for preserving game state.

Design Overview

1. Serialization and Deserialization:
    - Use the Serializable interface to serialize objects into byte streams.
    - Deserialize byte streams back into objects on the receiving side.
2. Network Communication:
    - Use Socket and ServerSocket classes for client-server communication.
    - Data is sent as serialized objects over the network.
3. Error Handling and Recovery:
    - Validate data integrity with checksums.
    - Implement retries and logging mechanisms for failed transmissions.

Java Code

Data Class

```
import java.io.Serializable;

public class DataPacket implements Serializable {
   private static final long serialVersionUID = 1L;

   private String message;
   private int sequenceNumber;

   public DataPacket(String message, int sequenceNumber) {
      this.message = message;
      this.sequenceNumber = sequenceNumber;
   }

   public String getMessage() {
      return message;
   }

   public int getSequenceNumber() {
      return sequenceNumber;
   }

   @Override
   public String toString() {
      return "DataPacket{message='" + message + "', sequenceNumber=" + sequenceNumber
+ "}";
   }
}
```

Server

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.zip.CRC32;

public class Server {
   public static void main(String[] args) {
      try (ServerSocket serverSocket = new ServerSocket(8080)) {
         System.out.println("Server is listening on port 8080...");
         Socket socket = serverSocket.accept();
         System.out.println("Client connected.");
```

```
            try (ObjectInputStream ois = new ObjectInputStream(socket.getInputStream())) {
                DataPacket packet = (DataPacket) ois.readObject();
                long receivedChecksum = ois.readLong();

                // Verify checksum
                long calculatedChecksum = calculateChecksum(packet);
                if (receivedChecksum == calculatedChecksum) {
                    System.out.println("Data received successfully: " + packet);
                } else {
                    System.err.println("Data integrity check failed!");
                }
            } catch (Exception e) {
                System.err.println("Error processing data: " + e.getMessage());
            }
        } catch (IOException e) {
            System.err.println("Server error: " + e.getMessage());
        }
    }

    private static long calculateChecksum(DataPacket packet) throws IOException {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(packet);
        oos.close();

        CRC32 crc = new CRC32();
        crc.update(baos.toByteArray());
        return crc.getValue();
    }
}
```

Client

```
import java.io.*;
import java.net.Socket;
import java.util.zip.CRC32;

public class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 8080)) {
            System.out.println("Connected to server.");

            DataPacket packet = new DataPacket("Hello, Server!", 1);

            try (ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream()))
{
```

```
            // Calculate checksum
            long checksum = calculateChecksum(packet);

            // Send data and checksum
            oos.writeObject(packet);
            oos.writeLong(checksum);
            System.out.println("Data sent to server: " + packet);
        }
    } catch (IOException e) {
        System.err.println("Client error: " + e.getMessage());
    }
  }

  private static long calculateChecksum(DataPacket packet) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(packet);
    oos.close();

    CRC32 crc = new CRC32();
    crc.update(baos.toByteArray());
    return crc.getValue();
  }
}
```

Features and Explanation

1. Serialization and Deserialization:
    - The DataPacket class implements Serializable.
    - Serialized objects are sent from the client and deserialized by the server.
2. Data Integrity:
    - A checksum (CRC32) is calculated for the serialized data on the client.
    - The server recalculates the checksum upon receipt and compares it with the sent checksum.
3. Error Handling:
    - Try-catch blocks handle exceptions during I/O and serialization.
    - Integrity failures are detected by mismatched checksums.
4. Recovery Mechanisms:
    - Retry logic or acknowledgments can be added for failed transmissions.

Execution Steps

1. Run the Server:
    - Start the Server class. It listens on port 8080.
2. Run the Client:
    - Start the Client class. It connects to the server and sends data.
3. Output:
    - Server:

      Server is listening on port 8080...
      Client connected.
      Data received successfully: DataPacket{message='Hello, Server!', sequenceNumber=1}

    - Client:

      Connected to server.
      Data sent to server: DataPacket{message='Hello, Server!', sequenceNumber=1}

Learning Outcomes

1. Understanding Network Serialization:
    - Serialization is critical for sending complex data structures between distributed systems.
2. Implementing Data Integrity:
    - Techniques like checksums ensure secure and reliable communication.
3. Error Handling and Recovery:
    - Proper handling of network failures and data corruption enhances system reliability.