Serialization Assignment-2

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:

As part of a game development team, you need to implement a system for saving and loading game progress. Design a Java program that serializes and deserializes game state objects. Consider scenarios where game levels, achievements, and player inventory need to be preserved.

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.

```
import java.io.*;

import java.util.ArrayList;

import java.util.List;


class PlayerInventory implements Serializable {

  private static final long serialVersionUID = 1L;


  private List<String> items;


  public PlayerInventory() {

    this.items = new ArrayList<>();

  }
```

```java
    public void addItem(String item) {
        items.add(item);
    }

    @Override
    public String toString() {
        return "Inventory: " + items;
    }
}

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

    private String playerName;
    private int level;
    private List<String> achievements;
    private PlayerInventory inventory;

    public GameProgress(String playerName, int level) {
        this.playerName = playerName;
        this.level = level;
        this.achievements = new ArrayList<>();
        this.inventory = new PlayerInventory();
    }

    public void addAchievement(String achievement) {
        achievements.add(achievement);
    }
    public void addItemToInventory(String item) {
        inventory.addItem(item);
    }

    @Override
    public String toString() {
        return "GameProgress{" +
                "playerName='" + playerName + '\'' +
                ", level=" + level +
                ", achievements=" + achievements +
                ", inventory=" + inventory +
                '}';
    }
}

public class GameSerialization {

    private static final String SAVE_FILE = "game_progress.dat";

    public static void main(String[] args) {
        // Simulate creating a game progress
        GameProgress progress = new GameProgress("Player1", 5);
        progress.addAchievement("First Kill");
        progress.addAchievement("Treasure Hunter");
        progress.addItemToInventory("Sword");
        progress.addItemToInventory("Shield");

        // Save game progress
        saveGameProgress(progress);
        // Load game progress
        GameProgress loadedProgress = loadGameProgress();
        if (loadedProgress != null) {
            System.out.println("Loaded Game Progress:");
            System.out.println(loadedProgress);
        }
    }

    // Save game progress to a file
    private static void saveGameProgress(GameProgress progress) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(SAVE_FILE))) {
            oos.writeObject(progress);
            System.out.println("Game progress saved successfully.");
        } catch (IOException e) {
            System.err.println("Error saving game progress: " + e.getMessage());
        }
    }

    // Load game progress from a file
    private static GameProgress loadGameProgress() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(SAVE_FILE)))
{
            return (GameProgress) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error loading game progress: " + e.getMessage());
        }
        return null;
    }
}
```

Explanation

1. Classes Used:
    - PlayerInventory: Represents the player's items, stored as a list.
    - GameProgress: Represents the player's game progress, including:
        - Player name
        - Current level
        - Achievements (list)
        - Player inventory (nested object).
2. Serialization Mechanism:
    - The GameProgress class and nested PlayerInventory class implement the Serializable interface.
    - The game state is serialized and stored in a file (game_progress.dat).
3. Handling Complex Structures:
    - Nested objects like PlayerInventory are serialized seamlessly as long as they are also Serializable.
4. Transient Fields:
    - If there are any temporary or calculated fields, they can be excluded using the transient keyword.

Program Output

Example Run:

```
Game progress saved successfully.
Loaded Game Progress:
GameProgress{playerName='Player1', level=5, achievements=[First Kill, Treasure Hunter],
inventory=Inventory: [Sword, Shield]}
```

Learning Outcomes

1. Proficiency in Serialization:
    - Demonstrates how to save/load game state with nested objects.
    - Proper use of Serializable interface.
2. Handling Complex Structures:
    - Handles nested structures like inventory and achievements seamlessly.
3. Practical Considerations:
    - Provides a reusable mechanism for saving/loading game progress.
    - Can be extended to handle additional game state elements (e.g., player stats, settings).

This program serves as a robust foundation for handling game progress serialization and deserialization in Java.