SOLID PRINCIPLES

DOCUMENTATION

Overview

This Java application, named GameStore, serves as a simple representation of a game store. The application employs several classes and interfaces to demonstrate the usage of SOLID principles. Below is the documentation for the provided code.

Classes and Interface Used

- GameStore – Main Class
- GameProcessor
- GameProcessorInterface
- Downloadable
- NormalDownload
- FTPDownload
- Installable
- Game
- OnlineGame
- OfflineGame
- Verification
- OnlineVerification
- OnlineVerification
- SystemBasedVerification

Single Responsibility Principle (SRP)

Game Class

```
public class Game {
    private String name;
    private String version;
    private int price;
    public Game(String name, String version, int price) {
        super();
        this.name = name;
        this.version = version;
        this.price = price;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getVersion() {
        return version;
    }
    public void setVersion(String version)
        {
        this.version = version;
    }
    public int getPrice() {
        return price;
    }
    public void setPrice(int price) {
        this.price = price;
    }
}
```

Single Responsibility Principle (SRP)

- The Game class follows the Single Responsibility Principle (SRP).
- SRP states that a class should have only one reason to change.
- The class manages properties and behavior related to a game (name, version, price).
- Its single responsibility is to handle the attributes of a game entity.
- Adhering to SRP promotes maintainability and isolates changes to game-related aspects within the class

Single Responsibility Principle (SRP)

The wrong method In SRP would be to add a function like ApplyDiscount() to the Class Game.

```
ApplyDiscount(){
    double discount;
    dicsount = price*0.3;
}
```

This violates SRP as discount is a business logic part, not related to the actual game.

Open/Closed Principle (OCP)

Downloadable Interface

There are several implementions of Downloadable Interface, such as FTPDownloader, NormalDownload. This makes Downlaod function open for extentions, without modifying existing code.

```
public interface Downloadable {
   void downloadGame(Game game);
}
public class FTPDownloader implements Downloadable {
   @Override
   public void downloadGame(Game game) {
      // Simulate FTP download logic
      System.out.println("Downloading game " + game.getName() + " via FTP...");
   }
}
```

Open/Closed Principle (OCP)

The Wrong method to do this would be to Implement all the fuctionality in a single class. This means, when we need to add a new fuctionality. We have to modify existing code. Which violates the principles of Open for extention and Closed for modification.

Liskov Substitution Principle (LSP)

OnlineGame Class and OfflineGame Class

The **OnlineGame and OfflineGame** class is an extension of the **Game** class, demonstrating the Liskov Substitution Principle. It inherits from **Game** and introduces additional properties specific to online games.

These two Classes can be substituted interchangable, and the behaviour of the program will not change, and even and instance of Game can also be used, thus completed the Liskov Substitution Principle (LSP)

Same fuctionality is implemented by the child classes of Verification Class, that are OnlineVerification , SystemBasedVerification and KeyBasedVarification.

Liskov Substitution Principle (LSP)

```
public class OfflineGame extends Game {
   boolean storyMode;
   public OfflineGame(String name, String version, int price, boolean storyMode) {
      super(name, version, price);
      this.storyMode = storyMode;
   }
}
public class OnlineGame extends Game {
   boolean multiplayer;
   public OnlineGame(String name, String version, int price, boolean multiplayer) {
      super(name, version, price);
      this.multiplayer = multiplayer;
   }
}
```

Liskov Substitution Principle (LSP)

Can be used in code like:

```
Game game = new OnlineGame(gameName, gameVersion,price,true);
Game game2 = new OfflineGame(gameName, gameVersion,price,true);
```

Interface Segregation Principle (ISP)

Downloadable Interface and Installable Interface

The Downloadable and Installable interface are interfaces defining two fuctionality of idownloading and intsalling games. These fuction whould normally be used together. But since it can be used individually, both the functions are defined in interfaces and then segregated.

```
public interface Downloadable {
   void downloadGame(Game game);
}
public interface Installable {
   void installGame(Game game);
}
```

Interface Segregation Principle (ISP)

The wrong method would be to have an interface that has both fuctionalities. But this mean that if a class implements that interface, and doesn’t need a function, still it will have to prove an implementation for it.

Dependancy Inversion

Verification Interface

There are 3 verification classes that are implementing the Verification Interface, that are OnlineVerification , SystemBasedVerification and KeyBasedVarification. SO normally we can create objects of these classes by using the class as data type.

eg OnlineVerification newVerification = new OnlineVerification() .

This will be the program or the current class dependant on a low-level module such as OnlineVerification. Instead, we invert the dependency to the parent of the classes.

Eg

Verification newVerification = new OnlineVerification()