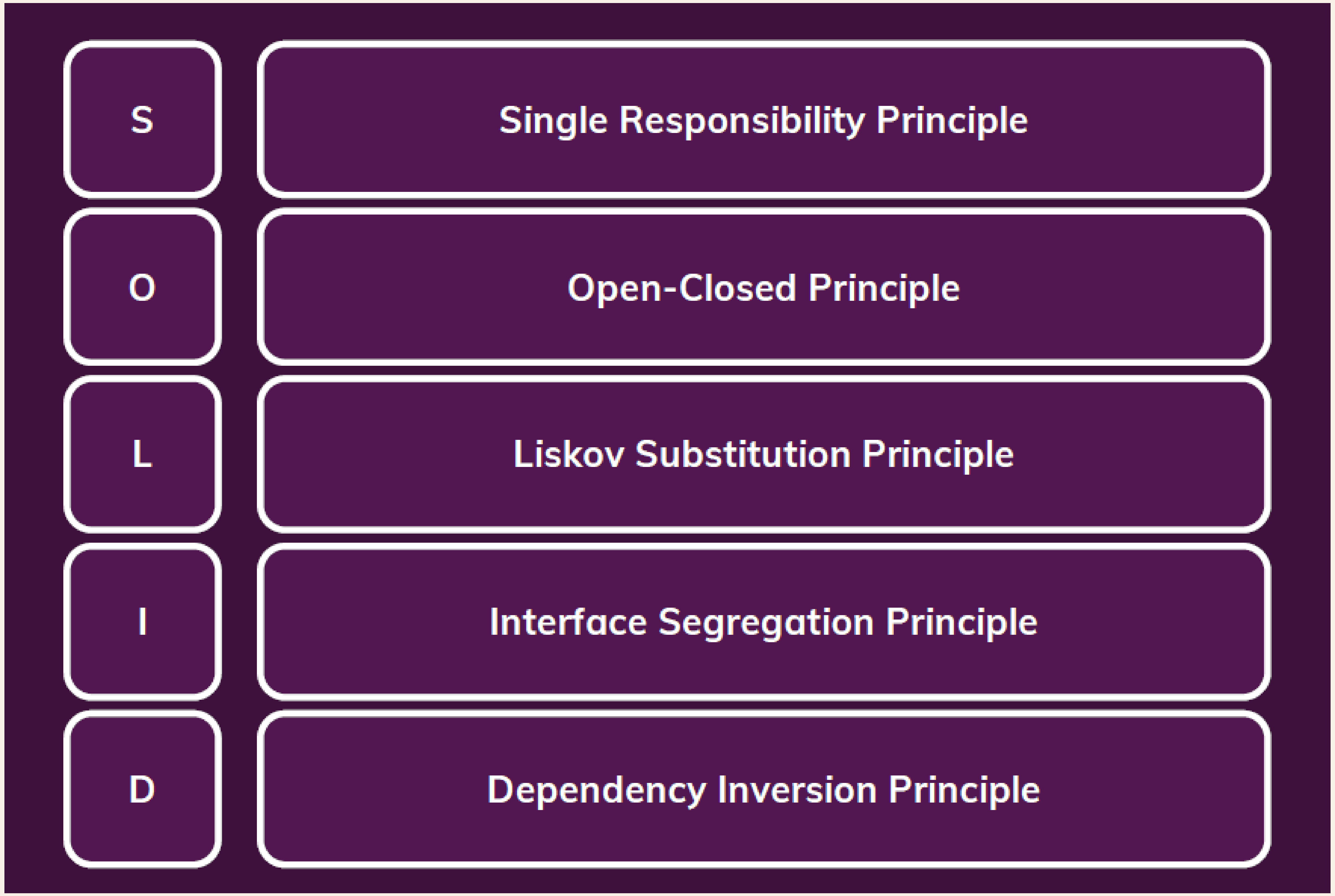
SOLID PRINCIPLES

SOLID PRINCIPLES



S	Single Responsibility Principle
O	Open-Closed Principle
L	Liskov Substitution Principle
I	Interface Segregation Principle
D	Dependency Inversion Principle

Single Responsibility Principle

Classes should have a single responsibility – a class shouldn’t change for more than one reason.

Single Responsibility Principle

```java
public class ProfileAuthenticationManager{

        public void profileLogout(Profile profile) {
                System.out.println("Logged out");
        }


        public Profile profileLogin(ArrayList<Profile> profileList, String name) {
                //login logic
        }

}
```

```java
public class ProfileContentManager{

        public void addToWatchlist(Profile profile, VodContent content) {
                //watchList addition logic
        }

        public void addToWatchHistory(Profile profile, VodContent content) {
                //watchhistory addition logic
        }

}
```

```java
public abstract class Profile {

        private String profileId;
        private String profileName;
        private String parentalControlEnabled;
        private ArrayList<VodContent> VodContentWatchList;
        private ArrayList<VodContent> VodContentWatchHistory;

        public Profile( parameters... ) {
                super();
                this.profileId = profileId;
                this.profileName = profileName;
                this.parentalControlEnabled = parentalControlEnabled;
                VodContentWatchList = vodContentWatchList;
                VodContentWatchHistory = vodContentWatchHistory;
        }


        //getters and setters

}
```

Single Responsibility Principle

- Profile class: The primary responsibility of Profile class is to represent a user profile, and it manages the profile-related data and methods. So, it adheres to the SRP principle.

- CreateProfileService Class: This class is responsible for creating different types of profiles based on user input. It takes user input, creates profiles, and ensures that each profile type is created correctly.

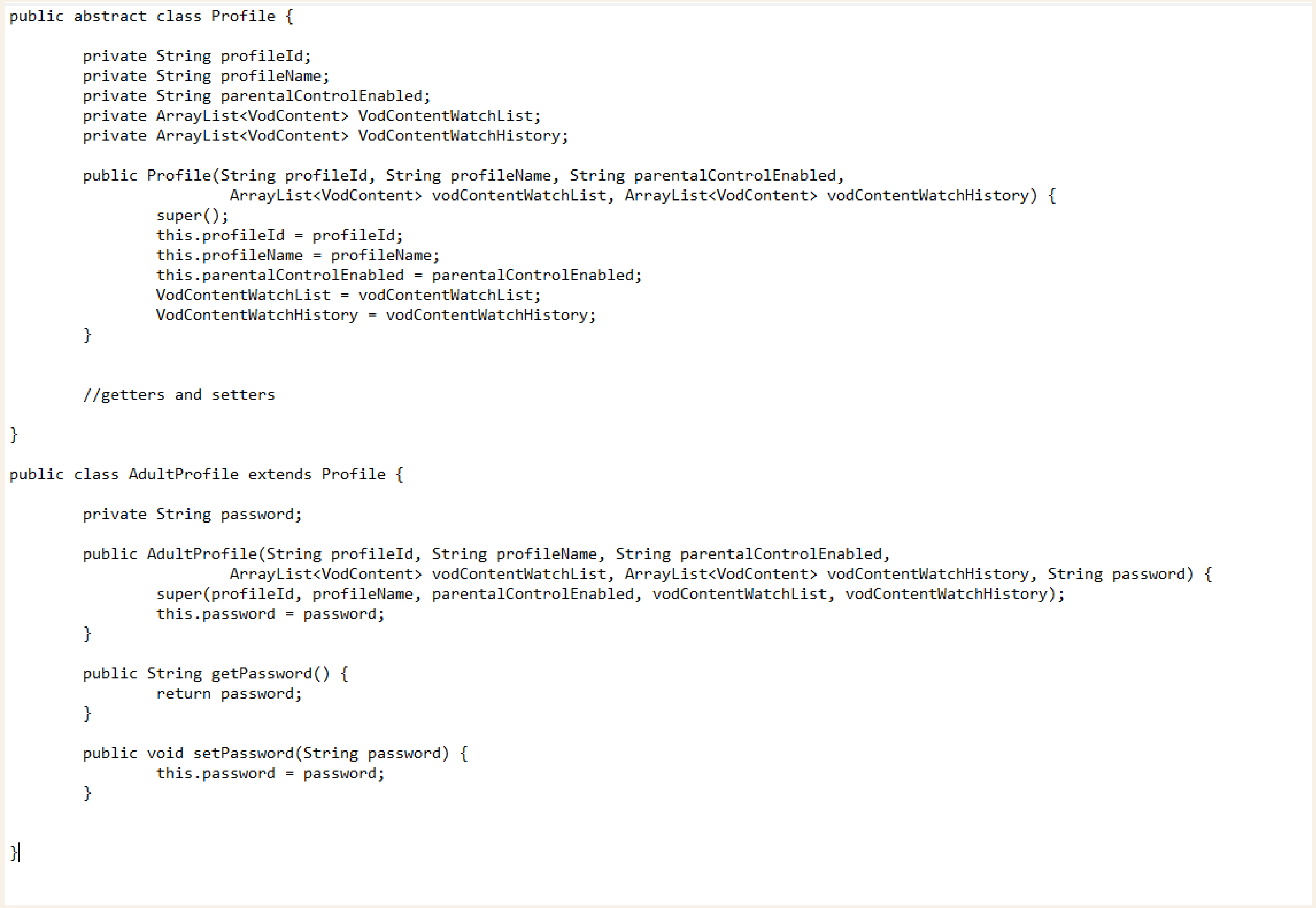
- ProfileAuthenticationManager Class: Manages the authentication process for user profiles. It validates login credentials and logs out profiles. It follows SRP by focusing on authentication-related tasks.

- ProfileContentManager Class: Manages the addition of content to a profile's watchlist and watch history. It adheres to SRP by concentrating on content management tasks.

Open Closed Principle

A class should be open for extension but closed for modification.

Open Closed Principle



```
public abstract class Profile {

        private String profileId;
        private String profileName;
        private String parentalControlEnabled;
        private ArrayList<VodContent> VodContentWatchList;
        private ArrayList<VodContent> VodContentWatchHistory;

        public Profile(String profileId, String profileName, String parentalControlEnabled,
                        ArrayList<VodContent> vodContentWatchList, ArrayList<VodContent> vodContentWatchHistory) {
                super();
                this.profileId = profileId;
                this.profileName = profileName;
                this.parentalControlEnabled = parentalControlEnabled;
                VodContentWatchList = vodContentWatchList;
                VodContentWatchHistory = vodContentWatchHistory;
        }


        //getters and setters

}

public class AdultProfile extends Profile {

        private String password;

        public AdultProfile(String profileId, String profileName, String parentalControlEnabled,
                        ArrayList<VodContent> vodContentWatchList, ArrayList<VodContent> vodContentWatchHistory, String password) {
                super(profileId, profileName, parentalControlEnabled, vodContentWatchList, vodContentWatchHistory);
                this.password = password;
        }

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }


}
```

Open Closed Principle

- Profile Class: The Profile class is open for extension and closed for modification. There are child classes KidsProfile, AdultProfile and ElderProfile extends the Profile class.So, New profile types can be added without modifying the existing Profile class.

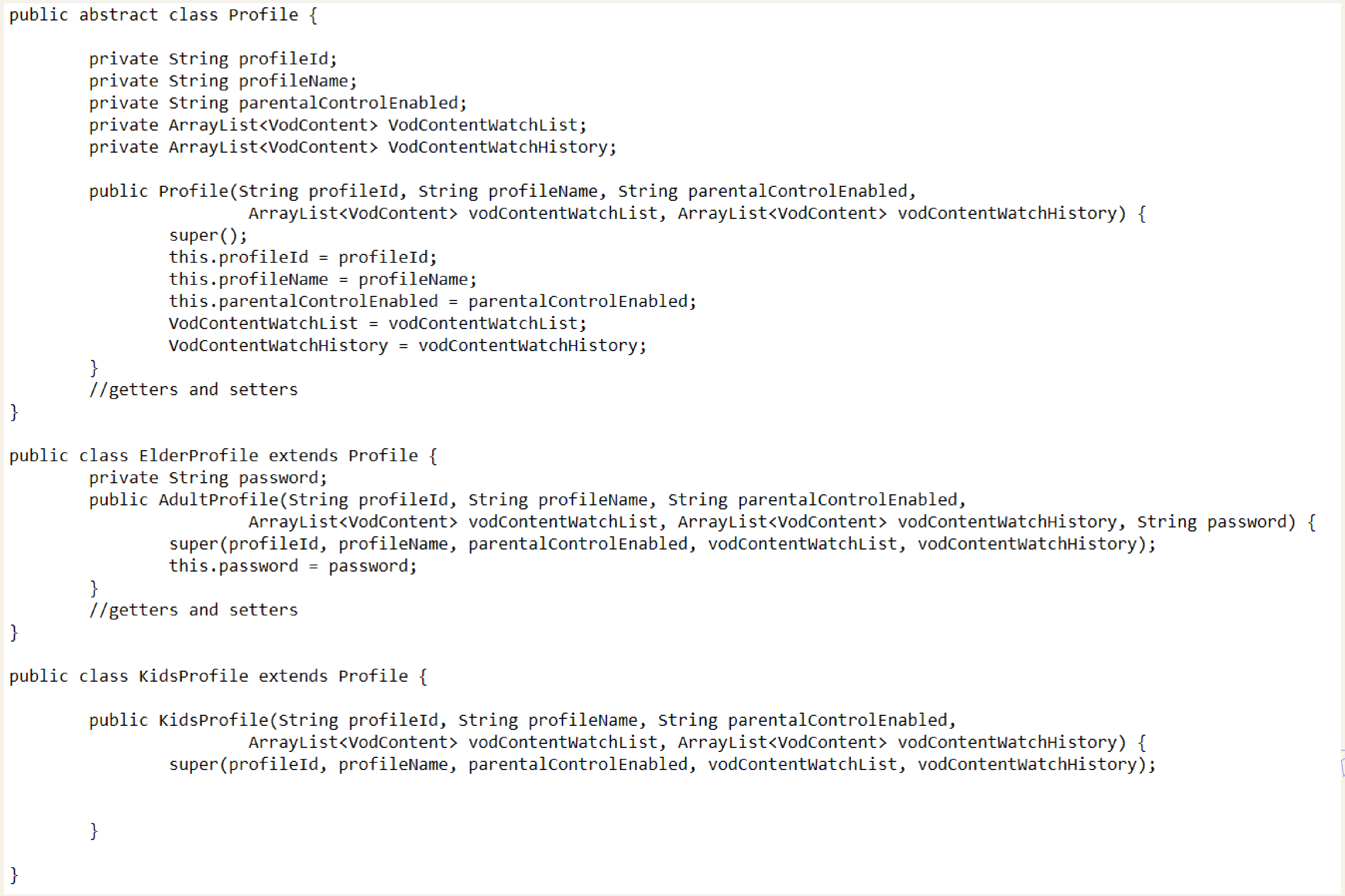
- Interfaces (AuthenticationManager and ContentManager): These interfaces are open for extension and closed for modification, allowing new classes to implement them and provide additional functionalities without changing the interfaces themselves.

- ProfileService Class: The ProfileService class is designed to be open for extension and closed for modification. New methods or features can be added to enhance its functionality without altering the existing code. We can add new features without modifying the existing ProfileService class, but by extending it.

Liskov Substitution Principle

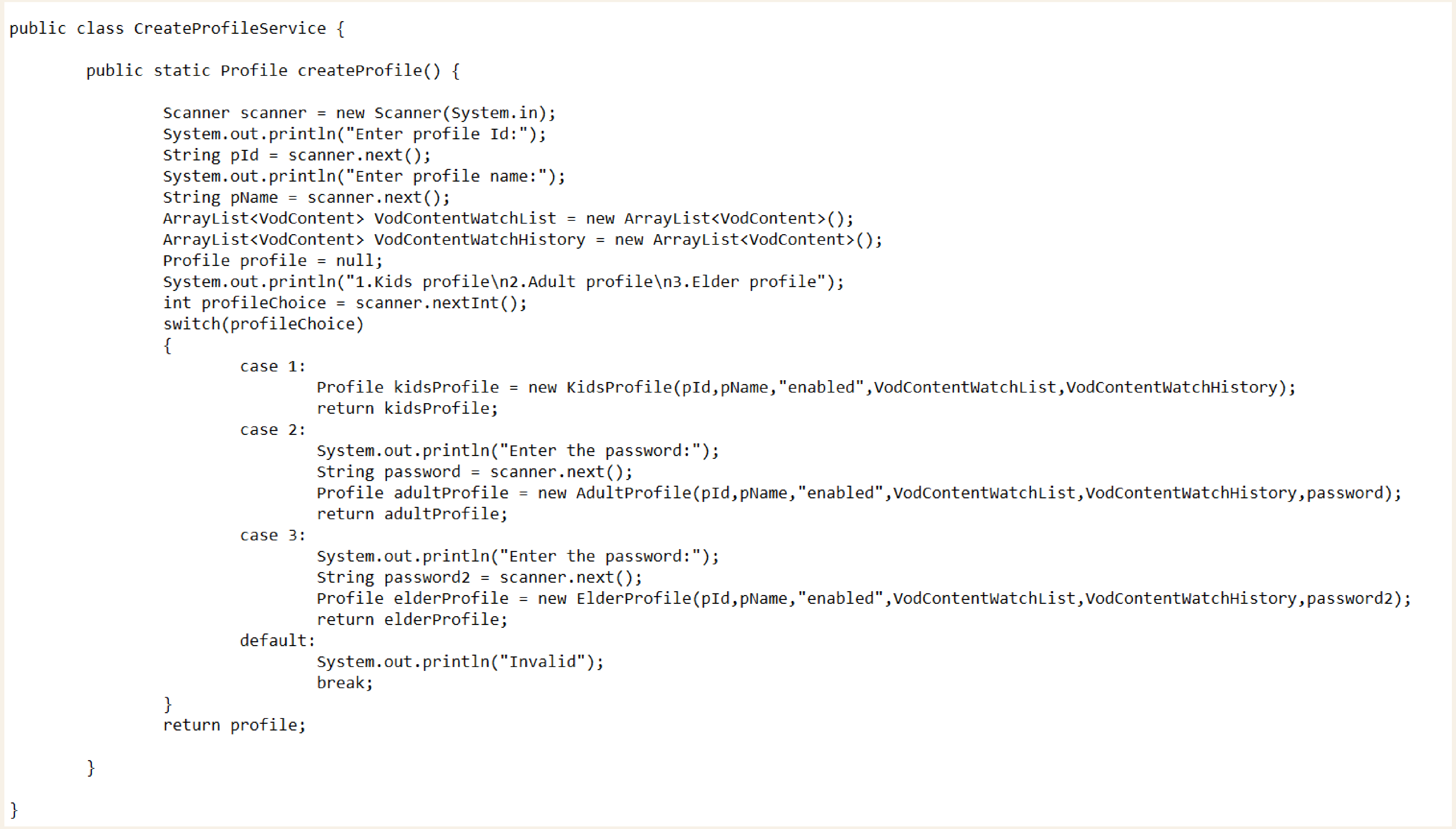
Objects should be replaceable with instances of their subclasses without altering the behavior.

Liskov Substitution Principle



```
public abstract class Profile {

        private String profileId;
        private String profileName;
        private String parentalControlEnabled;
        private ArrayList<VodContent> VodContentWatchList;
        private ArrayList<VodContent> VodContentWatchHistory;

        public Profile(String profileId, String profileName, String parentalControlEnabled,
                        ArrayList<VodContent> vodContentWatchList, ArrayList<VodContent> vodContentWatchHistory) {
                super();
                this.profileId = profileId;
                this.profileName = profileName;
                this.parentalControlEnabled = parentalControlEnabled;
                VodContentWatchList = vodContentWatchList;
                VodContentWatchHistory = vodContentWatchHistory;
        }
        //getters and setters
}

public class ElderProfile extends Profile {
        private String password;
        public AdultProfile(String profileId, String profileName, String parentalControlEnabled,
                        ArrayList<VodContent> vodContentWatchList, ArrayList<VodContent> vodContentWatchHistory, String password) {
                super(profileId, profileName, parentalControlEnabled, vodContentWatchList, vodContentWatchHistory);
                this.password = password;
        }
        //getters and setters
}

public class KidsProfile extends Profile {

        public KidsProfile(String profileId, String profileName, String parentalControlEnabled,
                        ArrayList<VodContent> vodContentWatchList, ArrayList<VodContent> vodContentWatchHistory) {
                super(profileId, profileName, parentalControlEnabled, vodContentWatchList, vodContentWatchHistory);


        }

}
```

Liskov Substitution Principle



```java
public class CreateProfileService {

        public static Profile createProfile() {

                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter profile Id:");
                String pId = scanner.next();
                System.out.println("Enter profile name:");
                String pName = scanner.next();
                ArrayList<VodContent> VodContentWatchList = new ArrayList<VodContent>();
                ArrayList<VodContent> VodContentWatchHistory = new ArrayList<VodContent>();
                Profile profile = null;
                System.out.println("1.Kids profile\n2.Adult profile\n3.Elder profile");
                int profileChoice = scanner.nextInt();
                switch(profileChoice)
                {
                        case 1:
                                Profile kidsProfile = new KidsProfile(pId,pName,"enabled",VodContentWatchList,VodContentWatchHistory);
                                return kidsProfile;
                        case 2:
                                System.out.println("Enter the password:");
                                String password = scanner.next();
                                Profile adultProfile = new AdultProfile(pId,pName,"enabled",VodContentWatchList,VodContentWatchHistory,password);
                                return adultProfile;
                        case 3:
                                System.out.println("Enter the password:");
                                String password2 = scanner.next();
                                Profile elderProfile = new ElderProfile(pId,pName,"enabled",VodContentWatchList,VodContentWatchHistory,password2);
                                return elderProfile;
                        default:
                                System.out.println("Invalid");
                                break;
                }
                return profile;

        }

}
```

Liskov Substitution Principle

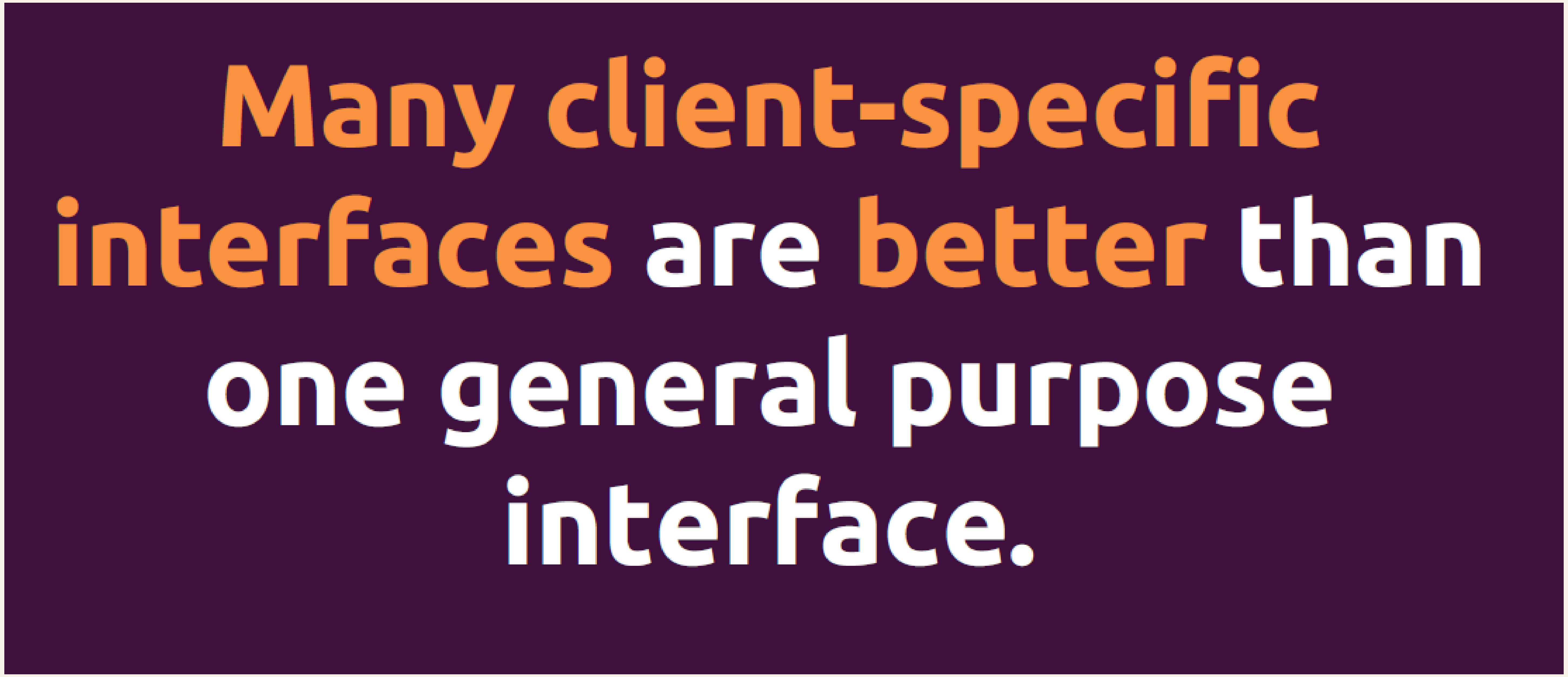
 Subclasses like KidsProfile, AdultProfile, and ElderProfile can be used interchangeably with their base class Profile.

 We create an instance of the KidsProfile class and we can assign it to a variable of type Profile (the base class).

 This substitution is permissible because KidsProfile is a subtype of Profile, and we can use it wherever a Profile is expected.

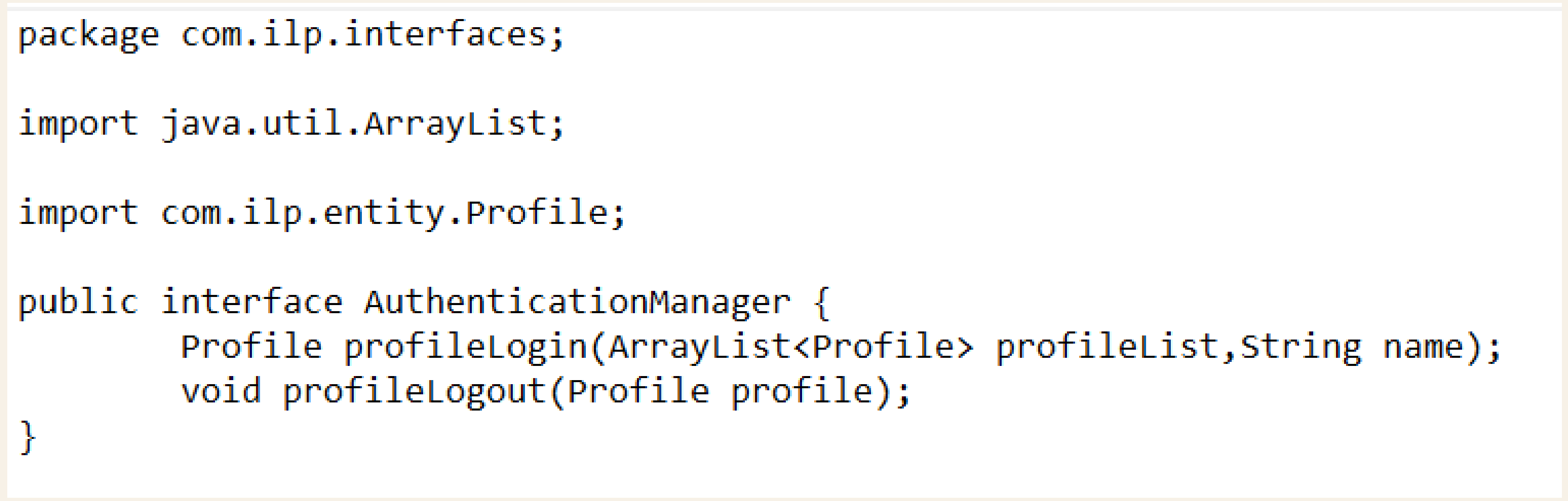
 Instances of subclasses (KidsProfile, AdultProfile, ElderProfile) can be substituted wherever an instance of the base class (Profile) is expected.

Interface Segregation Principle

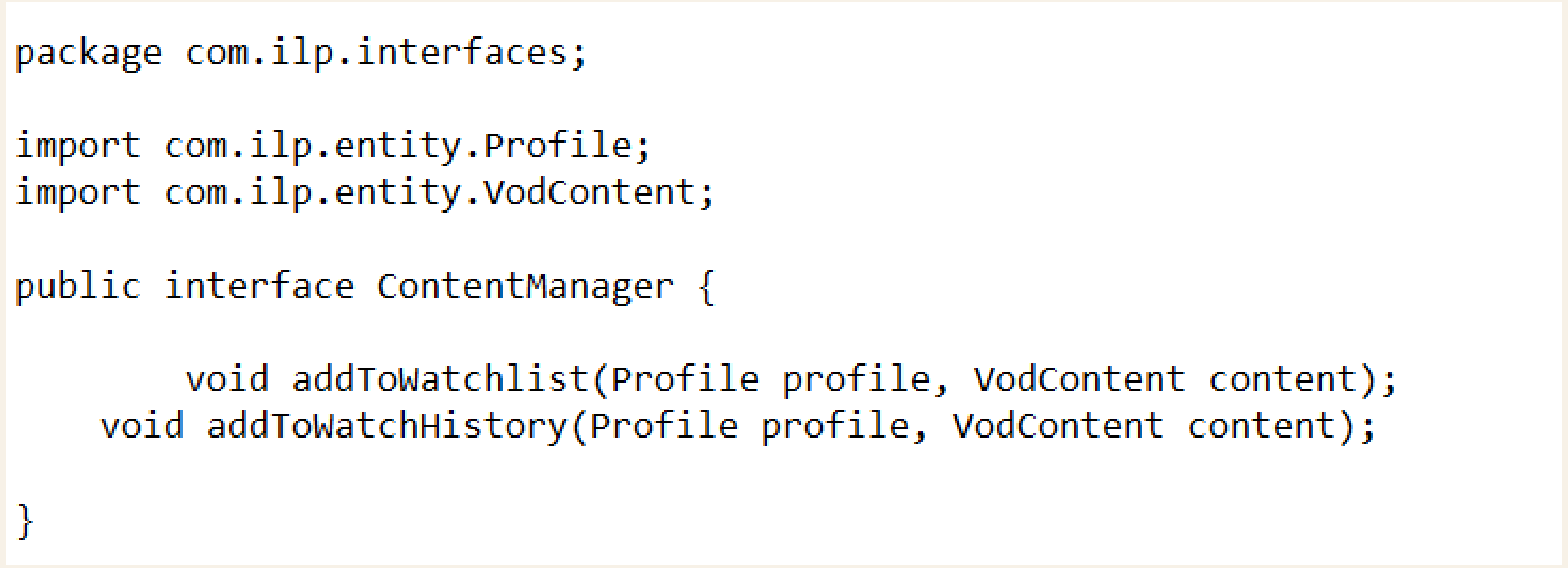


Many client-specific interfaces are better than one general purpose interface.

Interface Segregation Principle

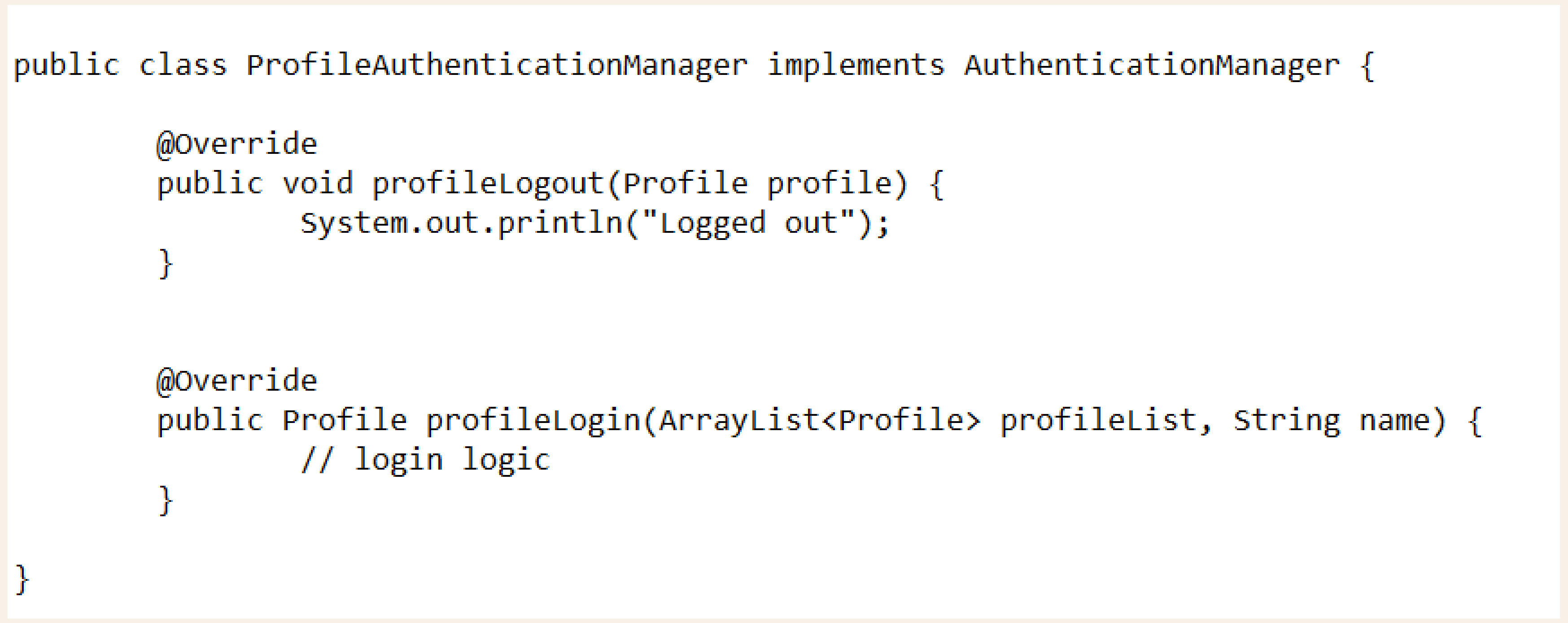


```java
package com.ilp.interfaces;

import java.util.ArrayList;

import com.ilp.entity.Profile;

public interface AuthenticationManager {
        Profile profileLogin(ArrayList<Profile> profileList,String name);
        void profileLogout(Profile profile);
}
```

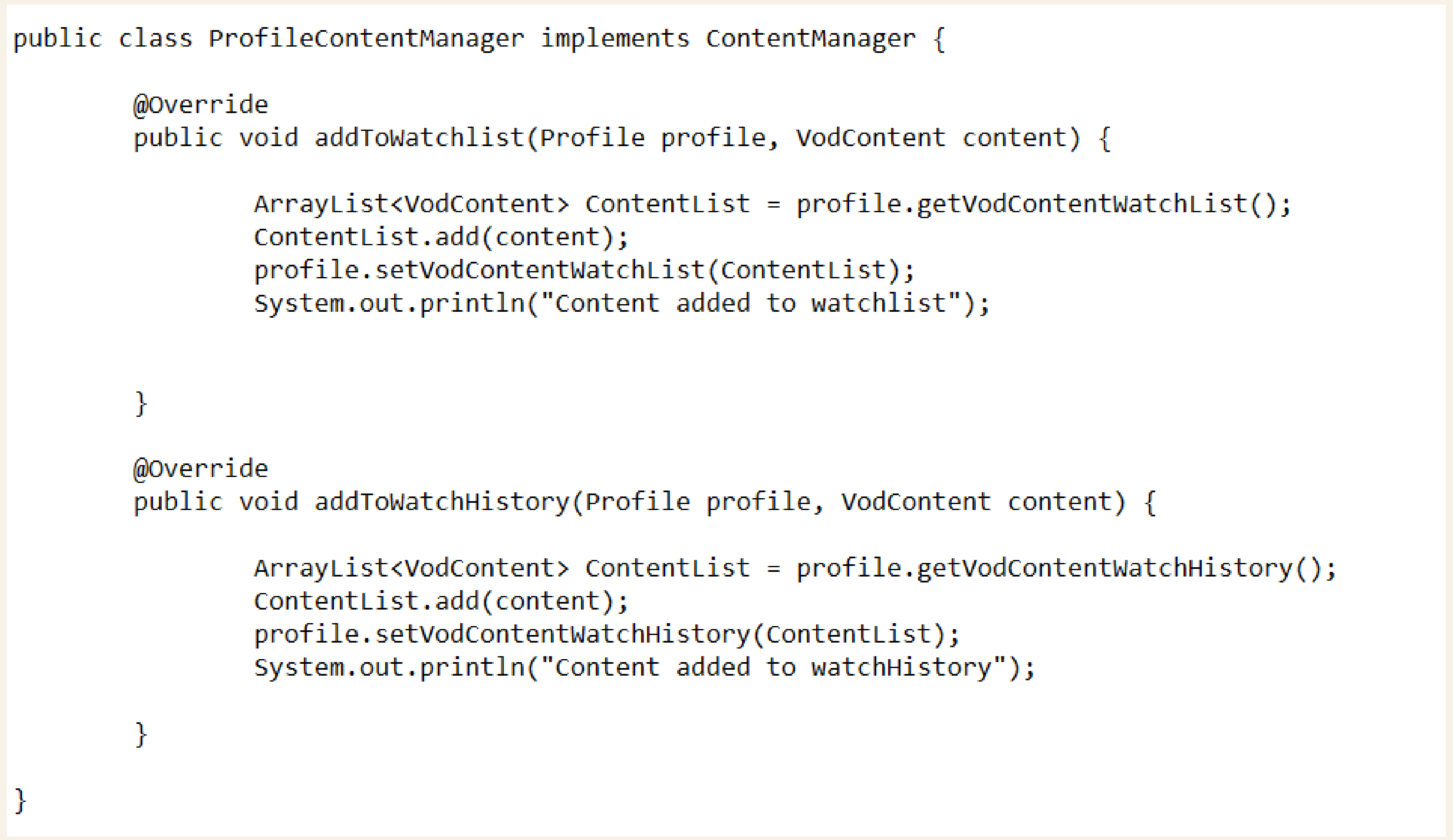


```java
package com.ilp.interfaces;

import com.ilp.entity.Profile;
import com.ilp.entity.VodContent;

public interface ContentManager {

        void addToWatchlist(Profile profile, VodContent content);
    void addToWatchHistory(Profile profile, VodContent content);

}
```

Interface Segregation Principle



```java
public class ProfileAuthenticationManager implements AuthenticationManager {

	@Override
	public void profileLogout(Profile profile) {
		System.out.println("Logged out");
	}


	@Override
	public Profile profileLogin(ArrayList<Profile> profileList, String name) {
		// login logic
	}

}
```



```java
public class ProfileContentManager implements ContentManager {

	@Override
	public void addToWatchlist(Profile profile, VodContent content) {

		ArrayList<VodContent> ContentList = profile.getVodContentWatchList();
		ContentList.add(content);
		profile.setVodContentWatchList(ContentList);
		System.out.println("Content added to watchlist");


	}

	@Override
	public void addToWatchHistory(Profile profile, VodContent content) {

		ArrayList<VodContent> ContentList = profile.getVodContentWatchHistory();
		ContentList.add(content);
		profile.setVodContentWatchHistory(ContentList);
		System.out.println("Content added to watchHistory");

	}

}
```

Interface Segregation Principle

 The AuthenticationManager interface defines two methods: profileLogin and profileLogout.

 This interface is relatively small and specific, focusing on methods related to profile authentication.

The ProfileAuthenticationManager class implements the AuthenticationManager interface.

It provides specific implementations for both profileLogin and profileLogout methods.

The ContentManager interface defines two methods: addToWatchlist and addToWatchHistory.

This interface is small and specific, providing methods related to managing content for a profile.

The ProfileContentManager class implements the ContentManager interface.

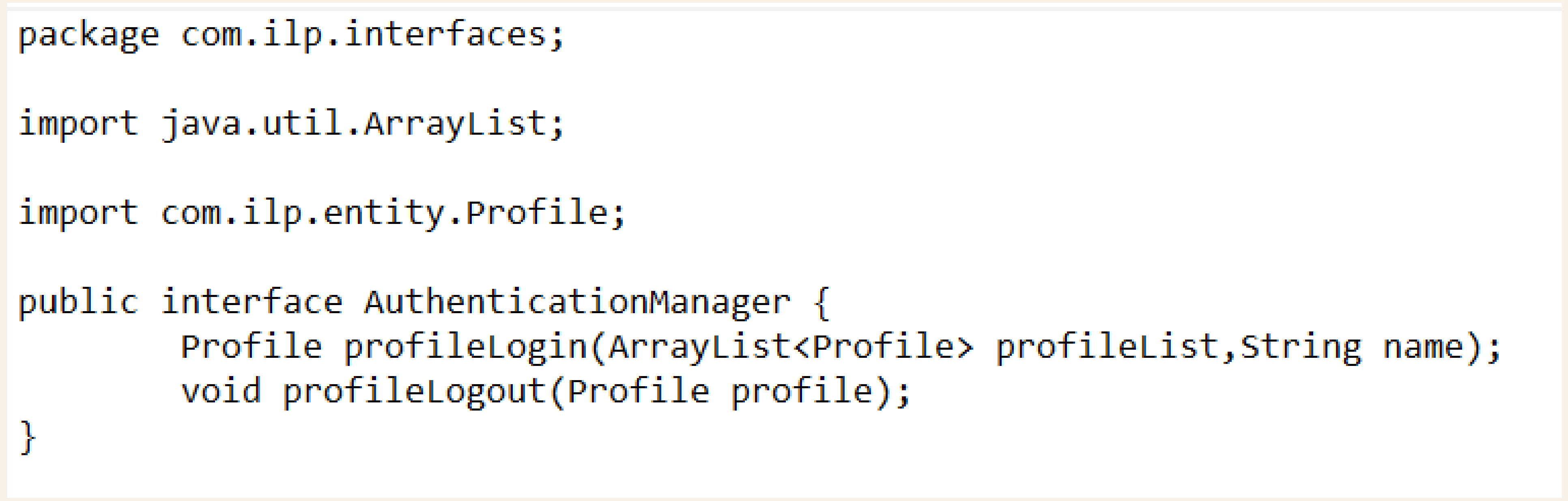
It provides specific implementations for both addToWatchlist and addToWatchHistory methods.

So this follows ISP.

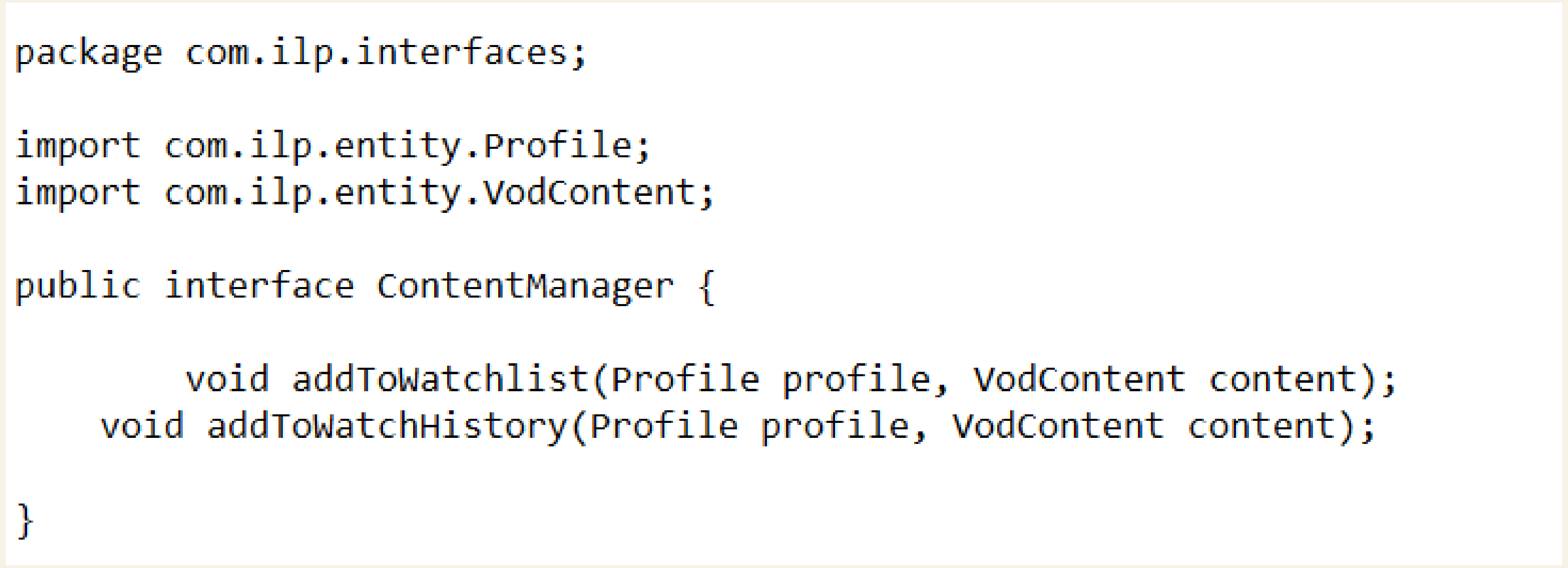
Dependency Inversion Principle

You should depend upon abstractions, not concretions.

Dependency Inversion Principle



```
package com.ilp.interfaces;

import java.util.ArrayList;

import com.ilp.entity.Profile;

public interface AuthenticationManager {
        Profile profileLogin(ArrayList<Profile> profileList,String name);
        void profileLogout(Profile profile);
}
```



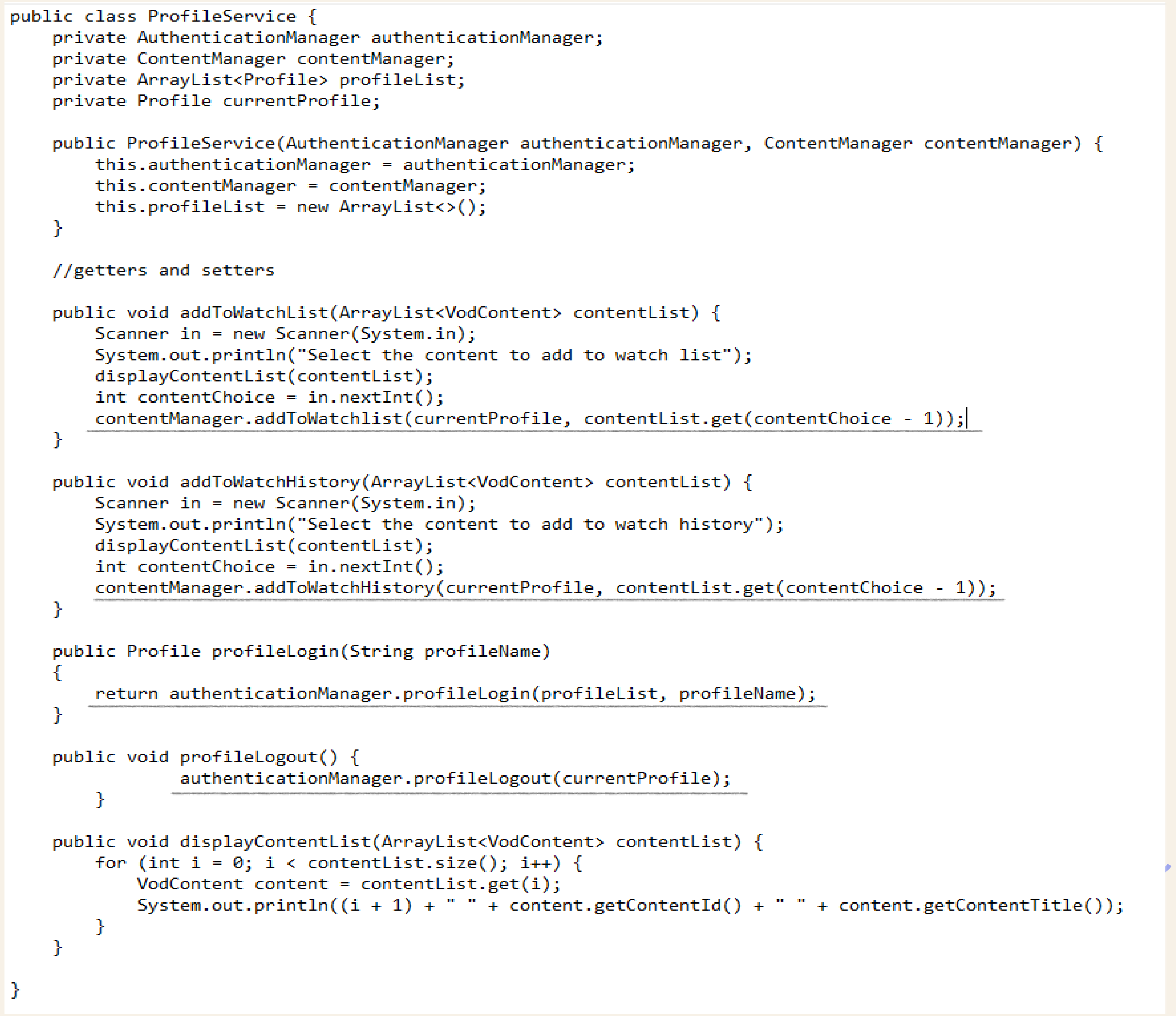
```
package com.ilp.interfaces;

import com.ilp.entity.Profile;
import com.ilp.entity.VodContent;

public interface ContentManager {

        void addToWatchlist(Profile profile, VodContent content);
    void addToWatchHistory(Profile profile, VodContent content);

}
```

Dependency Inversion Principle

```
public class ProfileAuthenticationManager implements AuthenticationManager {

        @Override
        public void profileLogout(Profile profile) {
                System.out.println("Logged out");
        }


        @Override
        public Profile profileLogin(ArrayList<Profile> profileList, String name) {
                // login logic
        }

}
```

```
public class ProfileContentManager implements ContentManager {

        @Override
        public void addToWatchlist(Profile profile, VodContent content) {

                ArrayList<VodContent> ContentList = profile.getVodContentWatchList();
                ContentList.add(content);
                profile.setVodContentWatchList(ContentList);
                System.out.println("Content added to watchlist");


        }

        @Override
        public void addToWatchHistory(Profile profile, VodContent content) {

                ArrayList<VodContent> ContentList = profile.getVodContentWatchHistory();
                ContentList.add(content);
                profile.setVodContentWatchHistory(ContentList);
                System.out.println("Content added to watchHistory");

        }

}
```

Dependency Inversion Principle



```java
public class ProfileService {
    private AuthenticationManager authenticationManager;
    private ContentManager contentManager;
    private ArrayList<Profile> profileList;
    private Profile currentProfile;

    public ProfileService(AuthenticationManager authenticationManager, ContentManager contentManager) {
        this.authenticationManager = authenticationManager;
        this.contentManager = contentManager;
        this.profileList = new ArrayList<>();
    }

    //getters and setters

    public void addToWatchList(ArrayList<VodContent> contentList) {
        Scanner in = new Scanner(System.in);
        System.out.println("Select the content to add to watch list");
        displayContentList(contentList);
        int contentChoice = in.nextInt();
        contentManager.addToWatchlist(currentProfile, contentList.get(contentChoice - 1));
    }

    public void addToWatchHistory(ArrayList<VodContent> contentList) {
        Scanner in = new Scanner(System.in);
        System.out.println("Select the content to add to watch history");
        displayContentList(contentList);
        int contentChoice = in.nextInt();
        contentManager.addToWatchHistory(currentProfile, contentList.get(contentChoice - 1));
    }

    public Profile profileLogin(String profileName)
    {
        return authenticationManager.profileLogin(profileList, profileName);
    }

    public void profileLogout() {
                authenticationManager.profileLogout(currentProfile);
        }

    public void displayContentList(ArrayList<VodContent> contentList) {
        for (int i = 0; i < contentList.size(); i++) {
            VodContent content = contentList.get(i);
            System.out.println((i + 1) + " " + content.getContentId() + " " + content.getContentTitle());
        }
    }

}
```

Dependency Inversion Principle

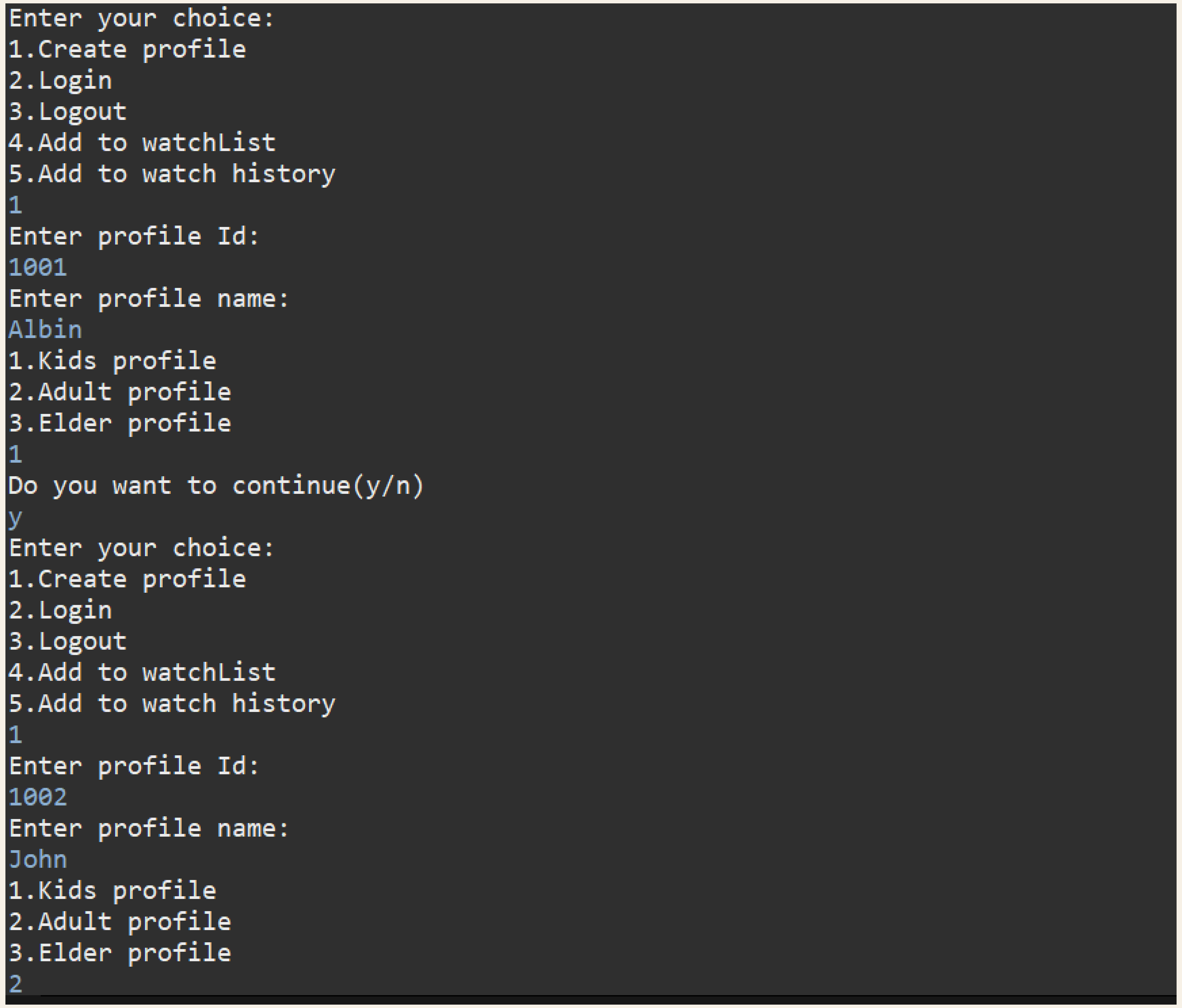
- The AuthenticationManager and ContentManager interfaces serve as abstractions.

- The concrete classes ProfileAuthenticationManager and ProfileContentManager provide specific implementations for the methods defined in the respective interfaces.

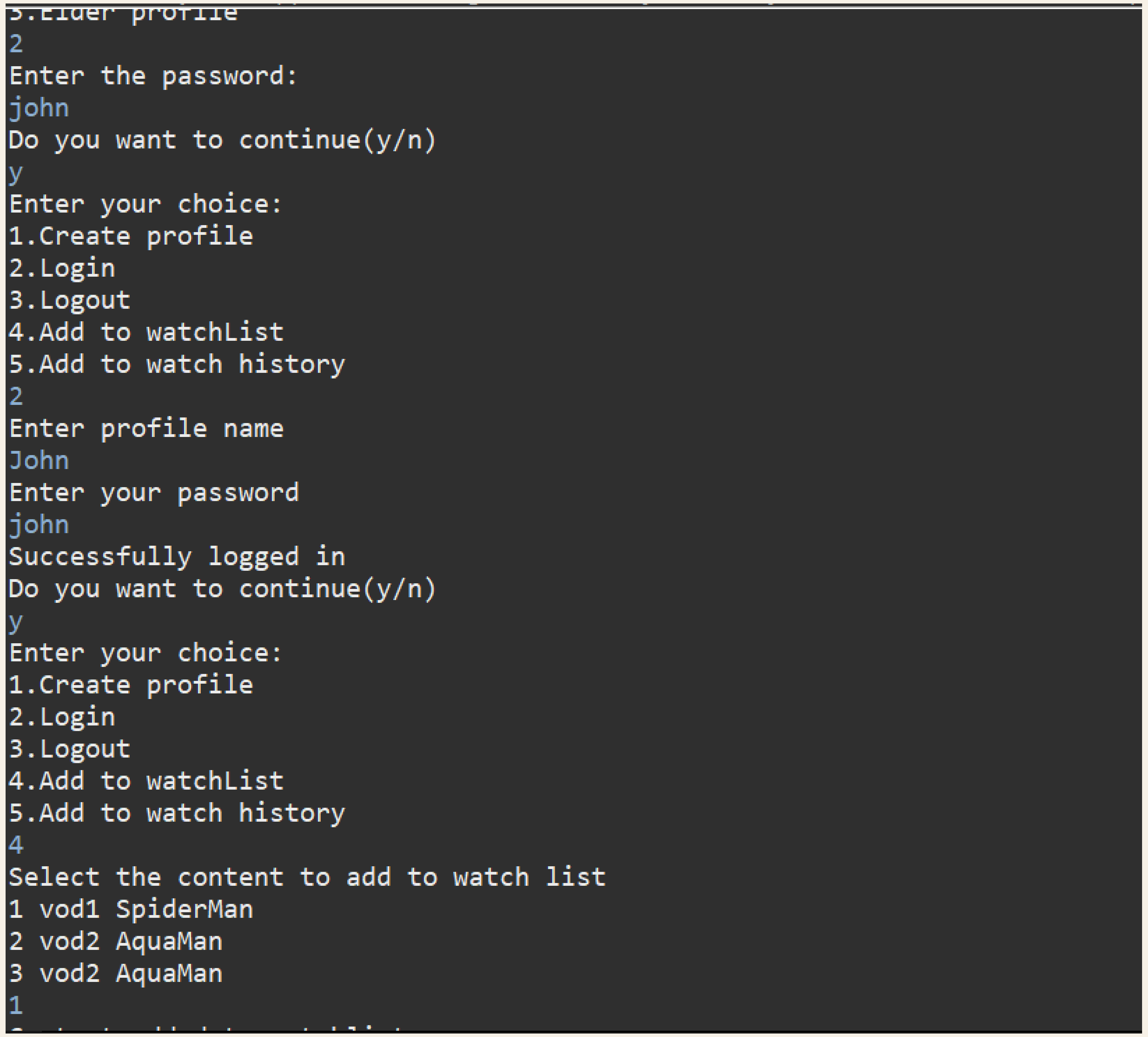
- This allows ProfileService to interact with these implementations without needing to know the specific details of how authentication or content management is performed.

- The ProfileService class represents a high-level module that depends on abstractions rather than concrete implementations.

OUTPUT

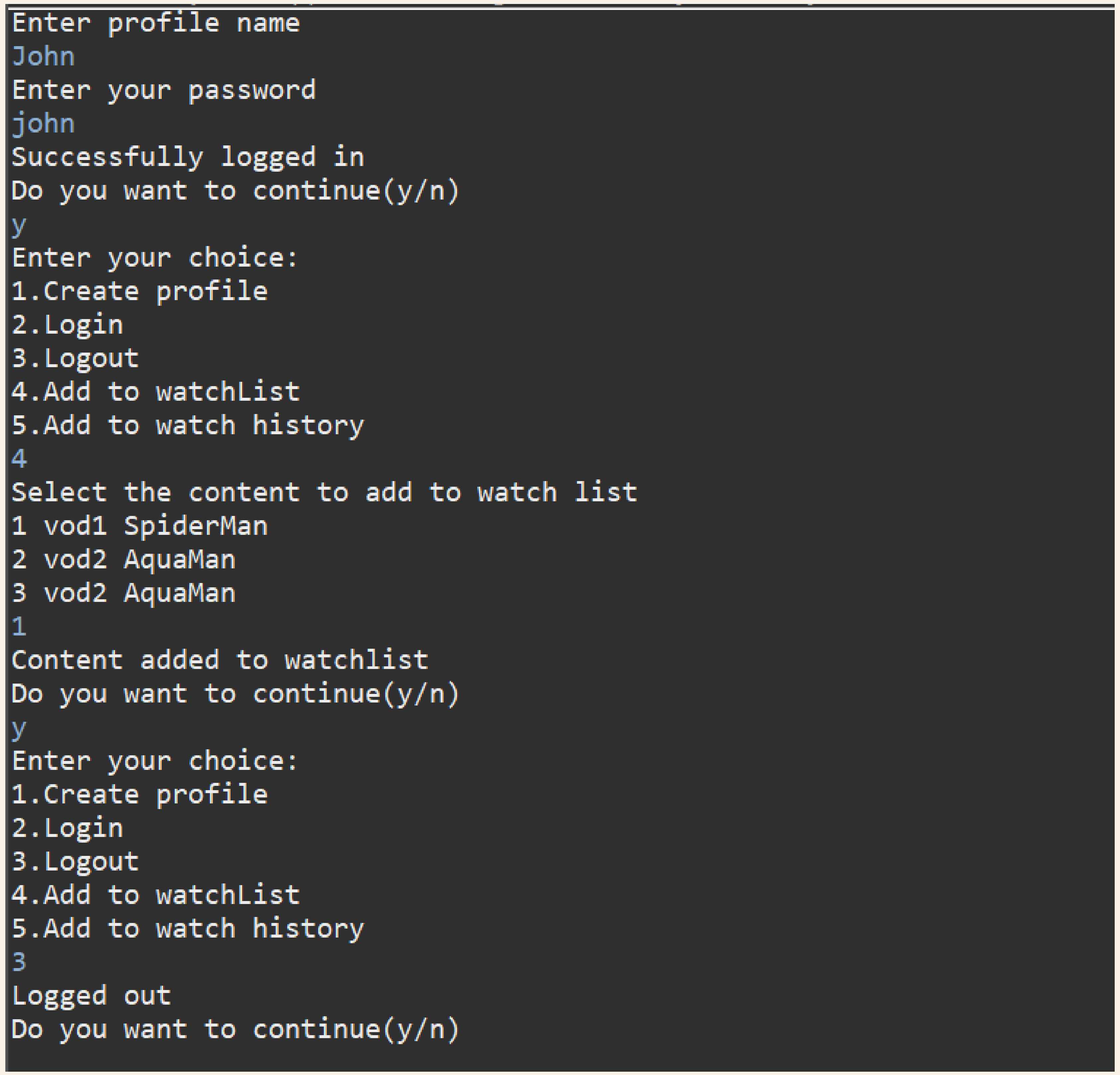


```
Enter your choice:
1.Create profile
2.Login
3.Logout
4.Add to watchList
5.Add to watch history
1
Enter profile Id:
1001
Enter profile name:
Albin
1.Kids profile
2.Adult profile
3.Elder profile
1
Do you want to continue(y/n)
y
Enter your choice:
1.Create profile
2.Login
3.Logout
4.Add to watchList
5.Add to watch history
1
Enter profile Id:
1002
Enter profile name:
John
1.Kids profile
2.Adult profile
3.Elder profile
2
```



```
3.Elder profile
2
Enter the password:
john
Do you want to continue(y/n)
y
Enter your choice:
1.Create profile
2.Login
3.Logout
4.Add to watchList
5.Add to watch history
2
Enter profile name
John
Enter your password
john
Successfully logged in
Do you want to continue(y/n)
y
Enter your choice:
1.Create profile
2.Login
3.Logout
4.Add to watchList
5.Add to watch history
4
Select the content to add to watch list
1 vod1 SpiderMan
2 vod2 AquaMan
3 vod2 AquaMan
1
```

OUTPUT



```
Enter profile name
John
Enter your password
john
Successfully logged in
Do you want to continue(y/n)
y
Enter your choice:
1.Create profile
2.Login
3.Logout
4.Add to watchList
5.Add to watch history
4
Select the content to add to watch list
1 vod1 SpiderMan
2 vod2 AquaMan
3 vod2 AquaMan
1
Content added to watchlist
Do you want to continue(y/n)
y
Enter your choice:
1.Create profile
2.Login
3.Logout
4.Add to watchList
5.Add to watch history
3
Logged out
Do you want to continue(y/n)
```

THANK YOU ...