SOLID Principles Implementation - Pranav S R - ILP Batch 2

Single Responsibility Principle (SRP)

Entity Classes (*Movie, User, PremiumUser, Review, WatchList*). Each entity class has a single responsibility:

- Movie: Represents a movie with details like title, genre, and release date.

```
public class Movie {
    private String title;
    private String genre;
    private String releaseDate;
    public Movie(String title, String genre, String releaseDate) {
        this.title = title;
        this.genre = genre;
        this.releaseDate = releaseDate;
    }
            public String getTitle() {
                        return title;
            }
            public void setTitle(String title) {
                        this.title = title;
            }
            public String getGenre() {
                        return genre;
            }
            public void setGenre(String genre) {
                        this.genre = genre;
            }
            public String getReleaseDate() {
                        return releaseDate;
            }
            public void setReleaseDate(String releaseDate) {
                        this.releaseDate = releaseDate;
            }
 }
```

- User: Represents a user with a username.

```
public class User {
            private String username;

   public User(String username) {
      this.setUsername(username);
   }

            public String getUsername() {
                        return username;
            }

            public void setUsername(String username) {
                        this.username = username;
            }

}
```

- PremiumUser: Represents a premium user, inheriting from *User*.

```
public class PremiumUser extends User implements PremiumUserFeatures{
            public PremiumUser(String username) {
      super(username);
   }

            @Override
            public boolean CanWriteJournal() {
                        return true;
            }

}
```

- Review: Represents a review with content, associated user, and movie.

```
public class Review {
            private String content;
            private User user;
   private Movie movie;
   public String getContent() {
                        return content;
            }


   public Review(String content, User user, Movie movie) {
      this.content = content;
      this.user = user;
      this.movie = movie;
   }

            public void setContent(String content) {
                        this.content = content;
            }

            public User getUser() {
                        return user;
            }

            public void setUser(User user) {
                        this.user = user;
            }

            public Movie getMovie() {
                        return movie;
            }

            public void setMovie(Movie movie) {
                        this.movie = movie;
            }

}
```

- WatchList: Represents a watchlist, managing a list of movies.

```
public class WatchList {
        private List<Movie> movies;

  public WatchList() {
    this.movies = new ArrayList<>();
  }

  public void addMovie(Movie movie) {
    movies.add(movie);
  }

  public void removeMovie(Movie movie) {
    movies.remove(movie);
  }

}
```

Open/Closed Principle (OCP)

The User class is open for extension but closed for modification

- User

```
public class User {
        private String username;

  public User(String username) {
    this.setUsername(username);
  }

        public String getUsername() {
                return username;
        }

        public void setUsername(String username) {
                this.username = username;
        }

}
```

- PremiumUser

```
public class PremiumUser extends User implements PremiumUserFeatures{
        public PremiumUser(String username) {
    super(username);
  }

        @Override
        public boolean CanWriteJournal() {
                return true;
        }

}
```

Interface Classes *(ReviewOperations, WatchListOperations)*. The interfaces are open for extension (new methods can be added), but closed for modification (existing methods remain unchanged).

- ReviewOperations

```
public interface ReviewOperations {
   void addReview(Review review);
   void deleteReview(Review review);
   void editReview(Review review, String newContent);
}
```

- WatchListOperations

```
public interface WatchListOperations {
   void addToWatchlist(WatchList watchList, Movie movie);
   void removeFromWatchlist(WatchList watchList, Movie movie);
}
```

Service Classes *(ReviewService, WatchListService).* Service classes implement the interfaces, and if new operations need to be added, new methods can be introduced without modifying existing code.

- ReviewService

```
public class ReviewService implements ReviewOperations{
          @Override
          public void addReview(Review review) {
                    System.out.println("Review Added");
          }
          @Override
          public void deleteReview(Review review) {
                    System.out.println("Review Deleted");
          }
```

```
        @Override
        public void editReview(Review review, String newContent) {
                System.out.println("Review Edited");
        }
}
```

- WatchLsitService

```
public class WatchListService implements WatchListOperations{

        @Override
        public void addToWatchlist(WatchList watchList, Movie movie) {
                watchList.addMovie(movie);

        }

        @Override
        public void removeFromWatchlist(WatchList watchList, Movie movie) {
                watchList.removeMovie(movie);

        }


}
```

Liskov Substitution Principle (LSP)

Inheritance (*PremiumUser*)

The *PremiumUser* class properly inherits from the base *User* class, extending its behaviour. Objects of the base class (*User*) can be replaced with objects of the derived class (*PremiumUser*) without affecting the correctness of the program.

```
public class PremiumUser extends User implements PremiumUserFeatures{
        public PremiumUser(String username) {
    super(username);
  }

        @Override
        public boolean CanWriteJournal() {
                return true;
        }

}
```

<u>Interface Segregation Principle (ISP)</u>

Interface Classes (*ReviewOperations*, *WatchListOperations*,*PremiumUserFeatures*)

Interfaces are segregated, with each declaring methods specific to its purpose. Clients (like the *Main* class) are not forced to implement methods they don't need.

- PremiumUserFeatures

```
public interface PremiumUserFeatures {
        boolean CanWriteJournal();

}
```

- ReviewOperations

```
public interface ReviewOperations {
    void addReview(Review review);
    void deleteReview(Review review);
    void editReview(Review review, String newContent);
}
```

- WatchListOperations

```
public interface WatchListOperations {
    void addToWatchlist(WatchList watchList, Movie movie);
    void removeFromWatchlist(WatchList watchList, Movie movie);
}
```

Dependency Inversion Principle (DIP)

The Main Class *ImdbSystem* depends on abstractions (interfaces) rather than concrete implementations. Instances of service classes are injected through interfaces, promoting flexibility and allowing for easy replacement of implementations.

```
    public class ImdbSystem {

        public static void main(String[] args) {

            Movie movie = new Movie("Inception", "Sci-Fi", "2010");
      User user = new User("JohnDoe");
      WatchList watchList = new WatchList();
      Review review = new Review("Amazing movie!", user, movie);

      WatchListOperations watchListOperation = new WatchListService();
      ReviewOperations reviewOperation = new ReviewService();

      watchListOperation.addToWatchlist(watchList, movie);
      reviewOperation.addReview(review);

        }

    }
```

Service classes (*ReviewService, WatchListService*) depend on abstractions by implementing the operations declared in interfaces (*ReviewOperations, WatchListOperations*).

- ReviewService

```
public class ReviewService implements ReviewOperations{
        @Override
        public void addReview(Review review) {
                System.out.println("Review Added");
        }
        @Override
        public void deleteReview(Review review) {
                System.out.println("Review Deleted");
        }

        @Override
        public void editReview(Review review, String newContent) {
                System.out.println("Review Edited");
        }
}
```

- WatchLsitService

```
public class WatchListService implements WatchListOperations{
        @Override
        public void addToWatchlist(WatchList watchList, Movie movie) {
                watchList.addMovie(movie);
        }
        @Override
        public void removeFromWatchlist(WatchList watchList, Movie movie) {
                watchList.removeMovie(movie);
        }
}
```

- ReviewOperations

```
public interface ReviewOperations {
    void addReview(Review review);
    void deleteReview(Review review);
    void editReview(Review review, String newContent);
}
```

- WatchListOperations

```
public interface WatchListOperations {
    void addToWatchlist(WatchList watchList, Movie movie);
    void removeFromWatchlist(WatchList watchList, Movie movie);
}
```

This structure and design support the SOLID principles, providing a modular, maintainable, and extensible system.