Summary: Fixing SRP Violation Example

Summary: Fixing SRP Violation Example

Problem Description

The example involves a `UserController` class that violates the **Single Responsibility Principle (SRP)**
by handling multiple responsibilities:

1. Receiving and processing client requests.
2. Validating user objects.
3. Persisting user objects in storage.

Identified Issues

- The `UserController` class has multiple reasons to change:

  - Changes to validation logic.

  - Changes to storage methods (e.g., from HashMap to a database).

  For a class to adhere to SRP, it should have only one reason to change.

Refactoring Solution

1. **Extract Validation Logic:**

   - Create a `UserValidator` class to handle user validation.

- Move validation methods from `UserController` to `UserValidator`.

2. **Extract Persistence Logic:**

- Create a `UserPersistenceService` class to manage persistence.

- Move storage methods from `UserController` to `UserPersistenceService`.

3. **Simplify `UserController`:**

- The class now only handles incoming requests and delegates tasks to other classes.

Benefits of Refactoring

- Adheres to **SRP**: Each class now has a single, focused responsibility.
- Improved Maintainability: Changes to validation or persistence affect only their respective classes.
- Reduced Ripple Effect: Modifications are isolated, minimizing side effects.

Real-World Application

- In Spring applications, use **Dependency Injection** to inject `UserValidator` and `UserPersistenceService`
  into `UserController` automatically.
- Use frameworks like JUnit for testing to ensure functionality remains intact after refactoring.

Summary: Fixing SRP Violation Example

By following these steps, the `UserController` now adheres to the Single Responsibility Principle, resulting in cleaner and more maintainable code.