Код до рефакторинга

```java

import java.util.HashMap;

import java.util.Map;

public class LibrarySystem {

private Map<String, String> users = new HashMap<>();

public String registerUser(String username, String password, String confirmPassword, String email) {

if (username == null || password == null || confirmPassword == null || email == null) {

return "Error: All fields are required.";

}

if (!password.equals(confirmPassword)) {

return "Error: Passwords do not match.";

}

if (users.containsKey(username)) {

return "Error: Username already exists.";

}

users.put(username, password);

return "Registration successful.";

}

public String loginUser(String username, String password) {

if (username == null || password == null) {

return "Error: All fields are required.";

}

if (!users.containsKey(username)) {

return "Error: User not found.";

}

if (!users.get(username).equals(password)) {

return "Error: Incorrect password.";

}

return "Login successful.";

}

public static void main(String[] args) {

LibrarySystem librarySystem = new LibrarySystem();

System.out.println(librarySystem.registerUser("newuser", "password123", "password123", "newuser@example.com"));

System.out.println(librarySystem.loginUser("newuser", "password123"));

}

}

```

Рефакторинг кода

После рефакторинга код становится более читаемым и поддерживаемым, следуя принципам DRY (Don't Repeat Yourself) и SOLID.

Код после рефакторинга

```java

import java.util.HashMap;

import java.util.Map;

public class LibrarySystem {

private Map<String, String> users = new HashMap<>();

public String registerUser(String username, String password, String confirmPassword, String email) {

if (isAnyFieldNull(username, password, confirmPassword, email)) {

return "Error: All fields are required.";

}

if (!password.equals(confirmPassword)) {

return "Error: Passwords do not match.";

}

if (users.containsKey(username)) {

return "Error: Username already exists.";

}

users.put(username, password);

return "Registration successful.";

}

public String loginUser(String username, String password) {

if (isAnyFieldNull(username, password)) {

return "Error: All fields are required.";

}

if (!isUserValid(username, password)) {

return "Error: Incorrect username or password.";

}

return "Login successful.";

}

private boolean isAnyFieldNull(String... fields) {

for (String field : fields) {

if (field == null) {

return true;

}

}

return false;

}

private boolean isUserValid(String username, String password) {

return users.containsKey(username) && users.get(username).equals(password);

}

public static void main(String[] args) {

LibrarySystem librarySystem = new LibrarySystem();

System.out.println(librarySystem.registerUser("newuser", "password123", "password123", "newuser@example.com"));

System.out.println(librarySystem.loginUser("newuser", "password123"));

}

}

```

Описание рефакторинга

1. Избавление от дублирования кода:

- В исходном коде проверка на `null` для полей `username`, `password`, `confirmPassword` и `email` была дублирована в методах `registerUser` и `loginUser`. Мы вынесли эту логику в отдельный метод `isAnyFieldNull`.

2. Метод для проверки правильности пользователя:

- Проверка на корректность пользователя также была дублирована. Мы создали отдельный метод `isUserValid`, который проверяет наличие пользователя и правильность пароля.

3. Улучшение читаемости и поддержки:

- Разделив логические части на отдельные методы, мы сделали код более читаемым и удобным для поддержки. Теперь каждый метод выполняет одну конкретную задачу.

4. Следование принципам SOLID:

- Принцип единственной ответственности (SRP) соблюден, так как каждый метод теперь выполняет одну конкретную задачу.

- Принцип DRY соблюден, так как дублирование кода устранено.

Вывод

Рефакторинг улучшил структуру и читаемость кода, упростил его поддержку и расширение, а также уменьшил вероятность возникновения ошибок из-за дублирования логики.  
  
  
Обработка книг в библиотеке

Код до рефакторинга

```java

import java.util.ArrayList;

import java.util.List;

class Book {

String title;

String author;

public Book(String title, String author) {

this.title = title;

this.author = author;

}

}

public class LibrarySystem {

private List<Book> books = new ArrayList<>();

public String addBook(String title, String author) {

if (title == null || author == null) {

return "Error: All fields are required.";

}

books.add(new Book(title, author));

return "Book added successfully.";

}

public String removeBook(String title) {

if (title == null) {

return "Error: Title is required.";

}

for (Book book : books) {

if (book.title.equals(title)) {

books.remove(book);

return "Book removed successfully.";

}

}

return "Error: Book not found.";

}

public List<Book> searchBooks(String query) {

List<Book> result = new ArrayList<>();

for (Book book : books) {

if (book.title.contains(query) || book.author.contains(query)) {

result.add(book);

}

}

return result;

}

}

```

Код после рефакторинга

```java

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

class Book {

private String title;

private String author;

public Book(String title, String author) {

this.title = title;

this.author = author;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

}

public class LibrarySystem {

private List<Book> books = new ArrayList<>();

public String addBook(String title, String author) {

if (isNullOrEmpty(title) || isNullOrEmpty(author)) {

return "Error: All fields are required.";

}

books.add(new Book(title, author));

return "Book added successfully.";

}

public String removeBook(String title) {

if (isNullOrEmpty(title)) {

return "Error: Title is required.";

}

Optional<Book> bookToRemove = books.stream().filter(book -> book.getTitle().equals(title)).findFirst();

if (bookToRemove.isPresent()) {

books.remove(bookToRemove.get());

return "Book removed successfully.";

}

return "Error: Book not found.";

}

public List<Book> searchBooks(String query) {

if (isNullOrEmpty(query)) {

return new ArrayList<>();

}

return books.stream()

.filter(book -> book.getTitle().contains(query) || book.getAuthor().contains(query))

.toList();

}

private boolean isNullOrEmpty(String str) {

return str == null || str.isEmpty();

}

}

```

Описание рефакторинга

1. Инкапсуляция полей класса `Book`:

- Сделаны поля `title` и `author` приватными, добавлены геттеры для доступа к ним.

2. Использование стримов для поиска и удаления книг:

- Код стал более лаконичным и читаемым благодаря использованию стримов в методах `removeBook` и `searchBooks`.

3. Проверка на null и пустые строки:

- Введен метод `isNullOrEmpty` для проверки строк на null или пустое значение, что улучшает читаемость и уменьшает дублирование кода.

4. Улучшение читаемости и поддержки:

- Методы стали короче и более понятными, что облегчает их поддержку.

Пример 3: Управление бронированием книг

Код до рефакторинга

```java

import java.util.HashMap;

import java.util.Map;

public class LibrarySystem {

private Map<String, String> reservations = new HashMap<>();

public String reserveBook(String username, String bookTitle) {

if (username == null || bookTitle == null) {

return "Error: All fields are required.";

}

if (reservations.containsKey(bookTitle)) {

return "Error: Book already reserved.";

}

reservations.put(bookTitle, username);

return "Book reserved successfully.";

}

public String cancelReservation(String username, String bookTitle) {

if (username == null || bookTitle == null) {

return "Error: All fields are required.";

}

if (!reservations.containsKey(bookTitle)) {

return "Error: Reservation not found.";

}

if (!reservations.get(bookTitle).equals(username)) {

return "Error: You can only cancel your own reservations.";

}

reservations.remove(bookTitle);

return "Reservation cancelled successfully.";

}

}

```

Код после рефакторинга

```java

import java.util.HashMap;

import java.util.Map;

import java.util.Optional;

public class LibrarySystem {

private Map<String, String> reservations = new HashMap<>();

public String reserveBook(String username, String bookTitle) {

if (isNullOrEmpty(username) || isNullOrEmpty(bookTitle)) {

return "Error: All fields are required.";

}

if (reservations.containsKey(bookTitle)) {

return "Error: Book already reserved.";

}

reservations.put(bookTitle, username);

return "Book reserved successfully.";

}

public String cancelReservation(String username, String bookTitle) {

if (isNullOrEmpty(username) || isNullOrEmpty(bookTitle)) {

return "Error: All fields are required.";

}

return getReservation(bookTitle)

.map(reservedBy -> {

if (!reservedBy.equals(username)) {

return "Error: You can only cancel your own reservations.";

}

reservations.remove(bookTitle);

return "Reservation cancelled successfully.";

})

.orElse("Error: Reservation not found.");

}

private Optional<String> getReservation(String bookTitle) {

return Optional.ofNullable(reservations.get(bookTitle));

}

private boolean isNullOrEmpty(String str) {

return str == null || str.isEmpty();

}

}

```