Jasmine Lisondra

Chapter 15: JUnit Internals

The Build-Operate-Check Pattern

The Build-Operate-Check pattern is a way to write tests. First, **Build** the objects you need. Then, **Operate** on them by calling methods or changing their state. Finally, **Check** if the results are what you expected. This pattern helps make your tests clear and easy to understand.

Organizing Tests

Organize your tests to make them easy to read and maintain. Group related tests together and use descriptive names for your test methods. This helps others understand what each test does and why it's important.

The Three Parts of a Test

Each test should have three parts:

1. **Arrange**: Set up the objects and state needed for the test.
2. **Act**: Perform the action you want to test.
3. **Assert**: Verify that the action produced the expected result.

Chapter 16: Refactoring SerialDate

Understanding Legacy Code

Refactoring is about improving existing code without changing its behavior. It's like cleaning up a messy room. Legacy code is old code that might not follow modern practices, making it hard to work with. Refactoring helps make this code easier to understand and maintain.

Step-by-Step Refactoring

When refactoring, take small steps. Make one change at a time and test it to make sure it works. This way, you can gradually improve the code without introducing new bugs.

Importance of Tests in Refactoring

Having good tests is crucial when refactoring. Tests help ensure that your changes don’t break anything. Write tests if they don’t already exist before you start refactoring. This safety net allows you to improve the code with confidence.

Chapter 17: Smells and Heuristics

Understanding Code Smells

Code smells are signs that something might be wrong with your code. They don’t always mean there's a problem, but they suggest that you should look closer. Common code smells include:

- **Duplicated Code**: The same code appears in multiple places.
- **Long Methods**: Methods that are too long and do too much.

- **Large Classes**: Classes that have too many responsibilities.
- **Divergent Change**: When one class is changed in many different ways for different reasons.

Refactoring to Remove Smells

When you find a code smell, consider refactoring to improve the code. For example:

- **Extract Method**: Break long methods into smaller, more focused ones.
- **Extract Class**: Split large classes into smaller ones with single responsibilities.
- **Remove Duplication**: Combine duplicated code into a single method or class.

Heuristics for Clean Code

Heuristics are guidelines that help you write clean code. Some useful heuristics include:

- **Keep It Simple, Stupid (KISS)**: Simple code is easier to understand and maintain.
- **Don't Repeat Yourself (DRY)**: Avoid duplication to reduce complexity.
- **Single Responsibility Principle (SRP)**: Each class and method should have one responsibility.
- **Open/Closed Principle**: Code should be open for extension but closed for modification.