TDD: Refactoring

Topics

- What is and Why refactoring?
- Things to refactor
- Refactoring legacy code
- How to improve your refactoring skills?

What is and Why Refactoring?

What is refactoring?

- A series of small steps, each of which changes the program's internal structure without changing its external behavior (Martin Fowler)
  - Verify “no change in external behavior” by Unit testing
- Example scenarios that require refactoring
  - Code smalls
  - We have too much duplicate code
  - The class is too big

Why refactoring?

- Helps us deliver more business value faster
- Minimize technical debt
  - To “fix broken windows” (Pragmatic Programmers)
- Improves the design of the our software
  - Easier to maintain and understand
  - Easier to facilitate change
  - Increased re-usability
- Understand unfamiliar code
- To help find bugs

Things to refactor

Remove Complexity: Make future change easier

- Bad names (variables, methods, classes)
- Big class
- Long methods
- Deep conditionals
- Magic numbers
- Local variables
- Improper variable scoping
- Missing encapsulation
- Irrelevant comments

Remove Cleverness: Make code more readable

- Cryptic code
- Abbreviated code
- Hijacking a method (changing its intent for your own purpose)

Refactoring Legacy Code

What is Legacy Code?

- Code without tests
- Code with “code smells”
  - Smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality
  - “I can't test this code”

Technical Challenges of Legacy Code

- Challenges
  - Usage of static method of another class
  - Usage of singleton
  - Creation of dependency inside of the code
- Approach to take
  - Write testing code without chaning the target code – use seam approach
  - Once testing code is done, start refactoring the target code – dependency should be injected (dependency injection)

Best Practices with working with Legacy Code

- Start testing from shortest to deepest branch
  - The shortest branch provides the highest level of abstraction to be tested
- Start refactoring from deepest to the shortes branch
  - The deepest branch provides smallest amount of code that needs to be refactored

How to Improve Your Refactoring Skills?

Tips for improving your refactoring skills

- Pratice with Kana examples
  - Kana is Karate form that can be practiced repeatedly each time making a small progress
- Take some legacy code and start TDD-driven refactoring
  - The more practice you have the better you will be in refactoring

Lab:

Exercise 1: Trip service refactoring
Exercise 2: Movie rental refactoring
1657_tdd_refactoring.zip

Thank You!