In28 Minutes

Learn Clean Code

Simple Design, Refactoring & TDD

"The man who is going to maintain your code is a psycho who knows where you stay"

Bad Code

```
public boolean isValidRating() {
    if (this.getRating() != null) {
        if (this.getRating().substring(0, 1).equalsIgnoreCase("B")
                && this.getRating().length() == 2) {
            if (StringUtils.isNumeric(this.getRating().substring(1, 2))
                    && Integer.parseInt(this.getRating().substring(1, 2)) > 0
                    && Integer.parseInt(this.getRating().substring(1, 2)) < 5)
                return true;

        } else if (this.getRating().substring(0, 1).equalsIgnoreCase("A")
                && this.getRating().length() == 3
                && StringUtils.isNumeric(this.getRating().substring(1, 3)))
            return true;

    }
    return false;
}
```

Better Code

```
public boolean isValidRating() {

    if (rating == null) {
        return false;
    }

    if (isValidARating())
        return true;

    if (isValidBRating())
        return true;

    return false;
}
```

in28Minutes Official

DevOps, AWS, Docker, Kubernetes, Java & Spring Boot Experts

- 4.4 Instructor Rating
- 109,459 Reviews
- 547,958 Students
- 33 Courses

My courses (33)

Java Programming for Complete Beginners

in28Minutes Official

4.4 ★★★★½ (11,092)

26 total hours • 302 lectures • Beginner

₹455 ~~₹8,640~~

Master Microservices with Spring Boot and Spring Cloud

in28Minutes Official

4.4 ★★★★½ (21,483)

11 total hours • 141 lectures • All Levels

₹455 ~~₹8,640~~

Bestseller

Master DevOps with Docker, Kubernetes and Azure DevOps

in28Minutes Official

4.4 ★★★★½ (1,818)

21 total hours • 215 lectures • All Levels

₹455 ~~₹8,640~~

Spring Framework Master Class - Java Spring the Modern Way

in28Minutes Official

4.3 ★★★★½ (15,780)

12 total hours • 138 lectures • All Levels

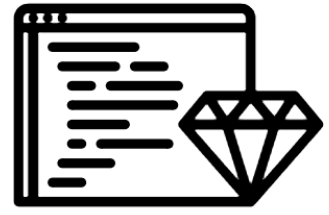
₹455 ~~₹8,640~~

Bestseller

Approach - Clean Code

In28 Minutes

- Hands-on
- Focuses on Basics
- Designed for all levels of programmers
  - Simple examples to start with

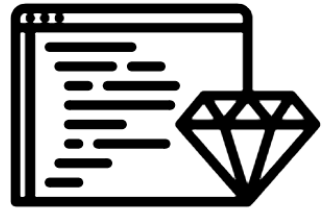


Solve + Watch + Retry = Best Results

Approach - Clean Code

In28 Minutes

- **Section I** : Understand Clarity of Code (Unit Tests)
- **Section II**: Focus on 4 Principles of Simple Design
- **Section III**: Get started with Refactoring
- **Section IV** : Understand TDD



GildedRose

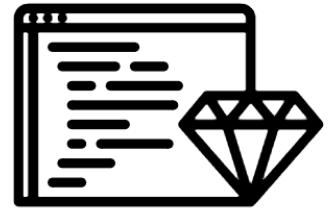
In 28 Minutes

- Goal
  - Understand **Clarity of Code**
  - Give **importance** to Unit Testing
- Exercises
  - `GildedRoseADefaultItemTest.java`
  - `GildedRoseBAgedBrieTest.java`
  - `GildedRoseCBackstagePassesTest.java`

Attitude - This Course

In28 Minutes

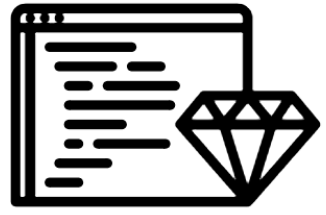
- Designed for all levels of programmers
  - Simple examples to start with
  - Not focusing on Object Orientation
- Provide a Playground
  - Solutions are secondary! Practice!
- Getting to 90% is important
  - Agree to Disagree.
  - Not important to argue about the remaining 10%



Most Difficult Challenge in Programming

In28 Minutes

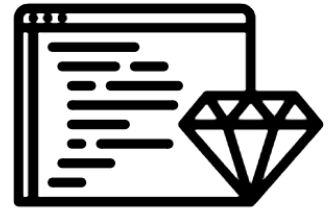
- Naming!
  - Creating good names is hard
  - Make names as long as necessary
  - Length of variable name is inversely proportional to scope
  - Follow conventions
    - Packages, Classes, Interfaces, Methods, Variables, Constants
    - Project specific (get vs retrieve vs ..)



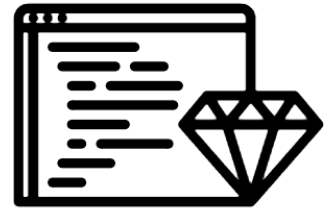
Comments

In28 Minutes

- Different Types
  - Type 1 : API Comments
  - Type 2 : Comments to help other programmers understand your code
  - Type 3 : TODO, FIXME
- Use Type 1 and Type 3 liberally!
- Type 2 should focus on Why but not what?
  - ex: Sorting, Performance Optimizations



10 Tips - Good Programmer



- Pair Programming
- Boy Scout Rule
- Continuous Learning
- Learn Touch Typing
- Use Key Board
- Ask Why?
- Work with smart people
- Understand All Tools - IDE, JShell, Maven, Gradle etc
- Play with other languages and frameworks
- Understand different programming approaches
  - Object Oriented, Functional Programming, Structured etc.

4 Principles of Simple Design

Keep Your Design Simple

Why Simple Design?

In28 Minutes

- Agile & Extreme Programming
  - Focus on **Today's** Requirements => Do them well
  - Change is **expected**
  - No Big Design Up Front
- Good Goals for Starting Programmers
- Design vs Architecture
  - **Architecture** - Difficult to change
  - **Design** - Easier to change

Four Principles of Simple Design

In28 Minutes

- Runs all tests
- Minimize Duplication
- Maximize Clarity
- Keep it small

Runs all tests

In28 Minutes

- Code should work!
- (Recommended) Have automated tests
- Design evolves (NOT created in one day)
  - Without Automation Tests, Developers are reluctant to make changes
- (Recommended) TDD leads to Better Tests and Better Design

Minimize Duplication

In28 Minutes

- Does this need explanation?
- Duplication leads to:
  - More Bugs
  - More Maintenance

Maximize Clarity

In28 Minutes

- Code we write today will be maintained by someone else later
  - That someone might be YOU a couple of months later!
- Good Starting Point
  - Focus on Naming - Methods, Variables, Classes etc

Keep it small

In28 Minutes

- Effort involved in making something perfect is high
  - Effort to Improve Design: 90 to 99 GREATER THAN 60 to 90
- Does that effort give you significant returns?
  - Extra layer => Additional Complexity
  - Extra Code => More code to maintain
- Keep things simple and small

Summary

In 28 Minutes

- Simplest possible testable approach while
  - Maximizing clarity and
  - Reducing duplication

4 Principles of Simple Design - Hands-on

In28 Minutes

- Exercises
  - Refactor Code
    - `StudentHelper.isGradeB()`
    - `StudentHelper.getGrade()`
    - `StudentHelper.willQualifyForQuiz()`
  - Write Code
    - `TextHelper.swapLastTwoCharacters()`
    - `TextHelper.truncateAInFirst2Positions()`
  - Refactor Unit Test
    - `CustomerBOTest`

Refactoring

In28 Minutes

- Altering Structure of Code without affecting "Behavior"
- Toughest part of Refactoring is the order or sequencing of steps
- Continuous Refactoring aided by Tests - Leads to "Clean Code"

Refactoring - Best Practices

In28 Minutes

- Have Unit Tests
- Take small steps
- Run tests at each step

Refactoring - Hands-On

In28 Minutes

- Exercises
  - `CustomerBOImpl.getCustomerProductsSum(List<Product>)`
  - `Movie.isValidRating()`
  - `MenuAccess.setAuthorizationsInEachMenus(List<MenuItem>, Role[])`
  - `UserLoginChecker.isUserAllowedToLogin(long, String, boolean, User, List)`

Test Driven Development

Do the opposite!

TDD - Three Steps

In28 Minutes

- RED
  - Write a simple test that fails
- GREEN
  - Write simple code to make it succeed
- REFACTOR
  - Make code adhere to "4 Principles of Simple Design"
  - While keeping it Green

TDD - Three Laws

In28 Minutes

1. No Production Code without Failing Test
2. Just enough test to make code fail
3. Just enough code to make test pass

TDD - Three Tips

In28 Minutes

1. Unlearn
2. Practices Makes You Perfect
    - Takes Time (2 to 3 months)
3. Get a Mentor

Unit Testing Organization/Attitude

In28 Minutes

- More important than Code.
  - Lead to Better Design (due to Continuous Refactoring)
- Best written before Code (TDD).
  - TDD improves Design and Code Quality
- Separated from Production Code
- Find Defects Early
  - Continuous Integration

Unit Testing Principles

In28 Minutes

- Easy to understand
  - Test should take no longer than 15 seconds to read
- Test should fail only when there is a problem with production code
- Tests should find all problems with production code
- Tests should have as minimum duplication as possible
- Should run quickly

Principle 1: Easy to understand

In28 Minutes

- Name of the Unit Test
  - Should indicate the condition being tested and (if needed) the result
    - `testClientProductSum_AllProductsSameCurrency` vs `testClientProductSum`
    - `testClientProductSum_DifferentCurrencies_ThrowsException` vs `testClientProductSum1`
- Highlight values important to the test
- One condition per test
- No Exception Handling in a test method.

Principle 2 : Fail only when there is a defect in CUT (Code Under Test)

In 28 Minutes

- No dependencies between test conditions.
  - Don't assume the order in which tests would run.
- Avoid External Dependencies
  - Avoid depending on (db, external interface,network connection,container).. Use Stubs/Mocks.
- Avoid depending on system date and random.
  - Avoid hard-coding of paths (“C:\TestData\dataSet1.dat”);//Runs well on my machine..

Principle 3 : Test's should find all defect's in code

In28 Minutes

- Why else do we write test :)
- Test everything that could possibly break.
  - Test Exceptions.
  - Test Boundary Conditions.
- Use Strong Assertions
  - Do not write “Tests for Coverage”
- Favorite maxim from JUnit FAQ
  - “Test until fear turns to boredom.”

Principle 4 : Less Duplication

In28 Minutes

- No Discussion on this :)

Principle 5 : Test's should run quickly

In28 Minutes

- (FACT) Long running tests are NOT run often
  - Avoid reading from File System or Network
- A temporary solution might be to “collect long running tests into a separate test suite” and run it less often.

Result : Tests as Documentation

In28 Minutes

- Well written tests act as great documentation
- Examples:
  - `testClientProductSum_AllProductsSameCurrency`
  - `testClientProductSum_DifferentCurrencies_ThrowsException`
  - `testClientProductSum_NoProducts`

Try on Your Own (With Solutions)

In28 Minutes

- `UserLoginCheckerTest`
- `MenuAccessTest`

You are all set!

Let's clap for you!

In28 Minutes

- You have a lot of patience! Congratulations
- You have put your best foot forward to get started with Clean Code
- Good Luck!

Do Not Forget!

In28 Minutes

- Clean Code is a Journey
- Recommend the course to your team!
- Your Success = My Success
  - Share your success story with me on LinkedIn (Ranga Karanam)
  - Share your success story and lessons learnt in Q&A with other learners!

Three book I recommend

In28 Minutes

- Code Complete by Steve McConnell.
- The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt
- Effective Java (3rd Edition) by Joshua Bloch