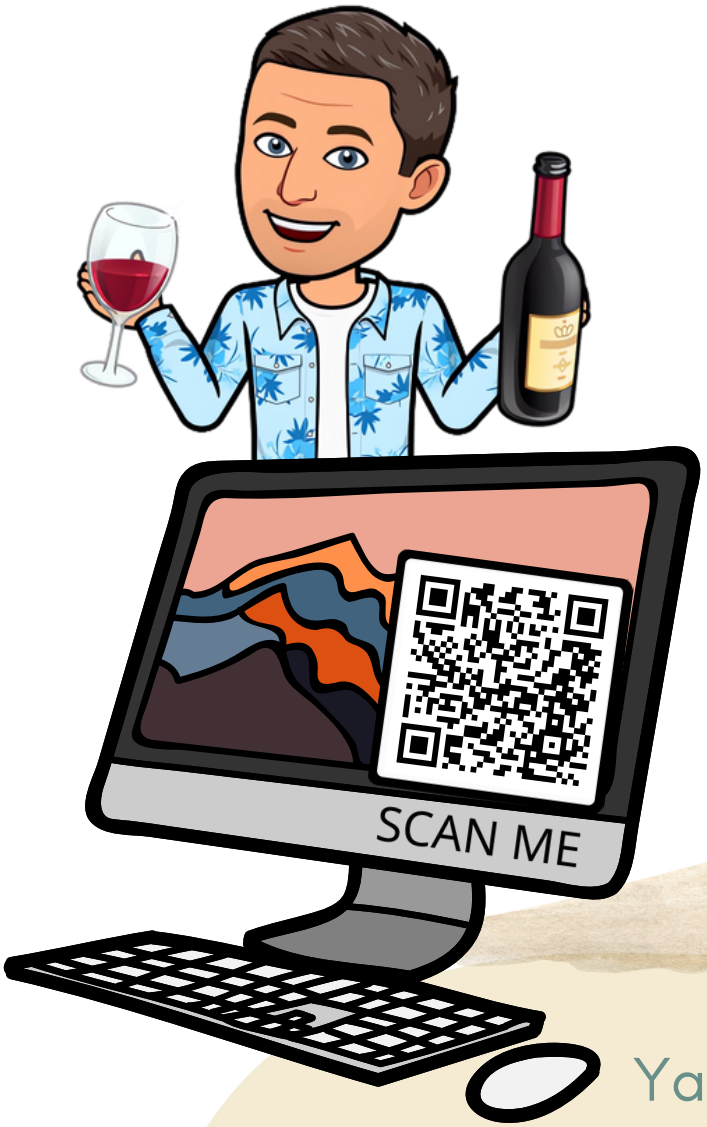
Summer Craft Book

2024 edition

Powered by **Advent of Craft**

Sponsored by
packmind



Yann Courtel & Yoan Thirion

INTRO
Why this book?

Acknowledgements

Before we go any further, the Advent of Craft team would like to thank all the developers, aspiring craftsmen and craftswomen, for the work and engagement done this winter. It was a frank success and as it's our tradition, a gift was in order!

This year, we are launching our version of the summer holiday workbook around the craft called the **Summer Craft Book**.

We hope you really enjoy it! Feel free to print it out and carry it in your holidays.

Special Thanks

A special Thank You to **Laurent Decamps** who came up with the idea of the workbook format, the acceptance testing and the work with the C# exercises.

Join the community

The Advent of Craft 2023 was successful in large part because of the community created around the event. We wanted to emphasize the need to join the community to help us grow together. Scan the QR code in the cover or use the link in the repository to join our discord.

Improving everyday

The software industry is evolving every day and as crafts(wo/)men, we have to evolve also and what a better time to work our craft on a sandy beach. Give up the mojito and dive into the **Summer Craft Book - 2024 edition.**

INTRO

How to use this book

At your own pace

Last year during the Advent of Craft, an exercise everyday was the formal diet.

For this summer holiday workbook, we are advising a section of the book every week but you are free to do it at your own pace.

You have all summer to do it, you can use it after the holidays to get back to speed or during the holidays. After all, you are the master of your own craft.

One theme a week

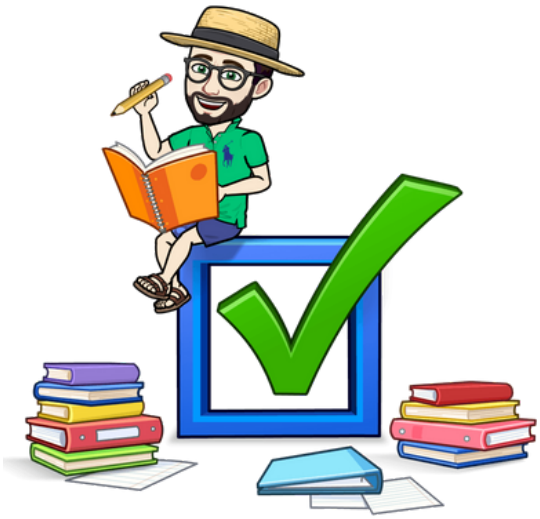
Each week, a specific theme is going to be examined. You will have information, theory, exercises, games and resources to go further.

Set aside a *couple of hours a week* to work on the subject. If you happen to have more time, a coding version will be available so you can dig into each exercise a bit longer.

We wouldn't want your craft skills to go to waste !

Proposed solution

As it's customary, the solution at the end of the book are proposed solution. The real value would be to share your view and your own expertise to the discord community as well.



INTRO

How to use the repository

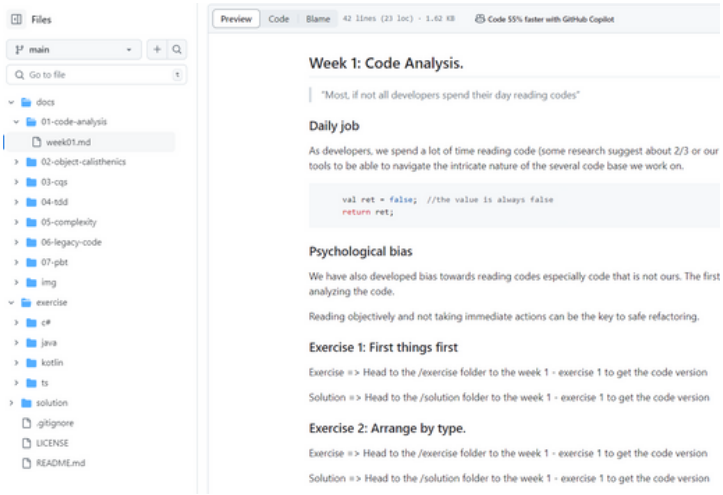
A repository with the book

As with the Advent of Craft edition, the Summer Craft Book has its **own github repository**. Each exercise and most of the games are in it so you could use the repository standalone.

We understand you could be busy in the summer and want to just do the practice so don't hesitate to do just that!

A look at the repository

For each week, there will be a directory that contains information and where to find the exercises & solutions. The QR codes and links in the book will get you to the directory of the week. A channel for each week will be added to the discord to share your implementations.



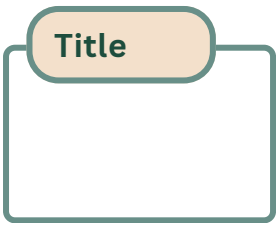
Multiple languages

Practice makes perfect and practice in different stacks makes for perfect crafts(wo)men.

The supported languages of this year's edition are **C#** / **Java** / **Typescript** and **Kotlin**.

In the future, please let us know in the discord what you want to see.

INTRO
Captions



Section

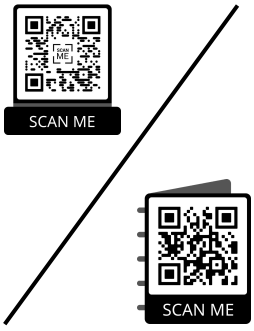
Each page can have different section in the book, the type of sections are *information*, *exercises*, *tips* and *games*.

Invite a pause

Usually when we invite you to do a pause by reflecting and taking notes or reading an article.

Code exercise

This caption represents an exercise that calls for you to get a resource outside of the workbook. (Sometimes as an invitation to go further)



Call to scan & link

Some pages will ask you to scan something outside of the book. We advise to have your phone or computer at arm's length.

Each scan will contain a link to go to the page if you have the digital version.

Proposed Solutions

Each exercise will have a small annotation with the page number where the proposed solution is. These annotations will also have a link to the solution page directly if you use the digital version.

INTRO

Before we start

Set your expectations

At the end of the summer craft book, what would you like to leave with?

.....

.....

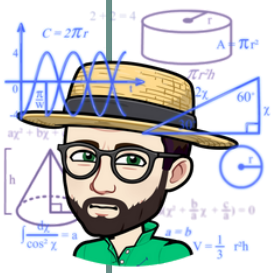
.....

.....

.....

Projecting Results

In the days after completing your summer craft book, how will you see that it has been useful?



.....

.....

.....

.....

.....

Free Comments

What do you like to add / write that has not been covered in this page?

.....

.....

.....

.....

.....

```java
public record Food(LocalDate expirationDate,
                   Boolean approvedForConsumption,
                   UUID inspectorId) {
    public boolean isEdible(Supplier<LocalDate> now) {
        if (this.expirationDate.isAfter(now.get()) &&
                this.approvedForConsumption &&
                this.inspectorId != null) {
            return true;
        } else {
            return false;
        }
    }
```

WEEK 1

Code analysis

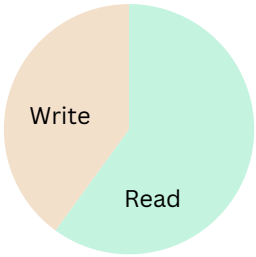
```java
public void roll(int roll) {
        System.out.println(players.get(currentPlayer) + " is the current player");
        System.out.println("They have rolled a " + roll);

        if (inPenaltyBox[currentPlayer]) {
            if (roll % 2 != 0) {
                isGettingOutOfPenaltyBox = true;

                System.out.println(players.get(currentPlayer) + " is getting out of the penalty box");
                places[currentPlayer] = places[currentPlayer] + roll;
                if (places[currentPlayer] > 11) places[currentPlayer] = places[currentPlayer] - 12;

                System.out.println(players.get(currentPlayer)
                        + "'s new location is "
                        + places[currentPlayer]);
                System.out.println("The category is " + currentCategory());
                askQuestion();
            } else {
                System.out.println(players.get(currentPlayer) + " is not getting out of the penalty box");
                isGettingOutOfPenaltyBox = false;
                }
```

"Most, if not all developers spend their day reading codes"



Code readers

Daily job

As developers, we spend a lot of time reading code (some research suggests about two-thirds of our time), and as such, we have polished our own set of tools to be able to navigate the intricate nature of the various codebases we work on.

```
val ret = false;  //the value is always false
return ret;
```

Psychological bias

We have also developed a bias towards reading code, especially code that is not ours. Knowing our own triggers while analyzing code can help us be more mindful of what needs to be changed.

Reading objectively and not taking immediate action can be the key to safe refactoring.

Don't like one liners... Neither switch case hell...

I hate exception! Design pattern overkill here...

Singleton is the worst... Your code looks like a staircase!

We should use English words everywhere...

How do we start analyzing code? What jumps out to us the most? What judgement are we making?

Week 1 - Code Analysis
Exercise

1) Ugly Code Analysis

First things first

You are about to look at a pretty complicated and big code base. Be prepared to look at the big picture before nitpicking every single details you come across.

Scan the code to see the code base.

SCAN ME

What do you see? *Write down all the things you would change and prefix them with a number (add code line number to help).* Solution at page 60

2) Arrange by type

Sort out the actions by category.

Use the numbering from exercise 1 and try to put them in the following category. Solution at page 61

Code Smell	Improvement	Vulnerability

3) Own arrangement

Sort out the actions by your own categories.

Find your own way to arrange the actions and see any similarities between the two models. Solution at page 61

....	....	....

Week 1 - Code Analysis
Game

Spot the Smells

Find the 6 smells in the code below. *Write down the line number and how would you address each smell?* solution page 62

```java
// Define an interface for the event callback
interface MessageReadListener {
    void onMessageRead(String message);
}

@Getter
@Setter
@NoArgsConstructor
public class FileStore {
    public String workingDirectory;
    private MessageReadlistener messageReadListener;

    // Method to save the message into a file
    public String save(int id, String message) throws IOException {
        Path path = Paths.get(workingDirectory, id + ".txt");
        Files write(path, message.getBytes());
        return path.toString();
    }
    // Method to read the message from a file and trigger the event
    public void read(int id) throws IOException {
        Path path = Paths get(workingDirectory, id + ".txt");
        String message = new String(Files.readAllBytes(path));
        messageReadListener.onMessageRead(message);
    }
}
```

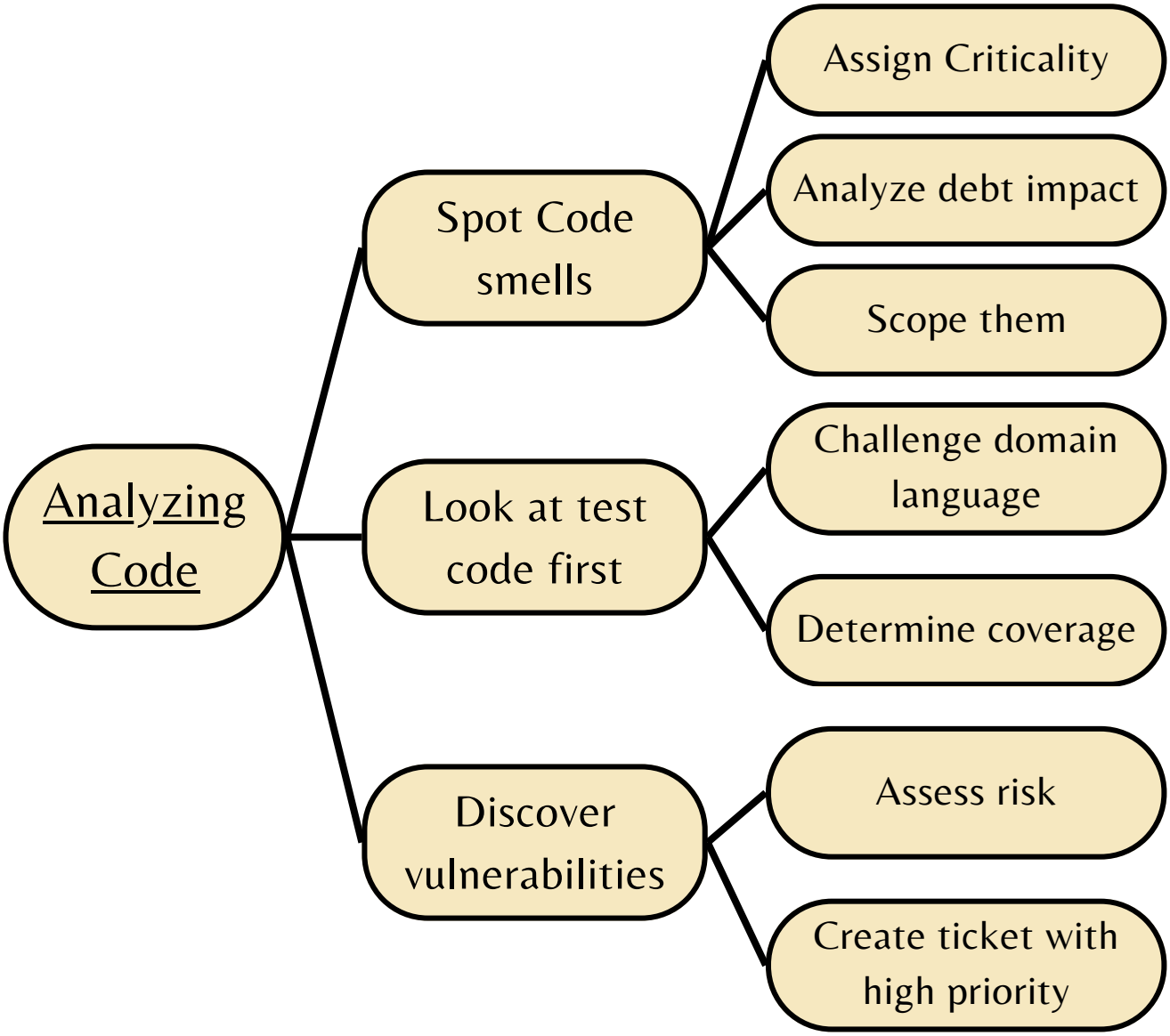
1 ..... 4 .....

2 ..... 5 .....

3 ..... 6 .....

Mind-mapping notes

Taking notes as mind-map is a powerful tool to help remember information in a structured way. *Here is an example of notes for a code analysis process.*

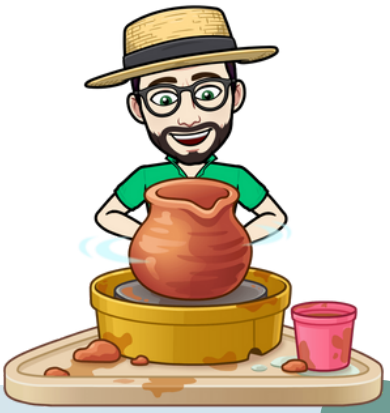


Different mind-map techniques exist to get the most of your map. Scan the code to read the article to discover them.

SCAN ME

Only One Level Of Indentation Per Method

First Class Collections



Don't Use The ELSE Keyword

Wrap All Primitives And Strings

WEEK 2

Object Calisthenics

One Dot Per Line

Don't Abbreviate

Keep All Entities Small

No Classes With More Than Two Instance Variables

No Getters/Setters/Properties

The steps to reading better code

9 Steps to improve code readibility

Object Calisthenics are a set of principles intended to help developers write cleaner, more maintainable code. They were introduced by Jeff Bay in his book "The ThoughtWorks Anthology". The rules focus on maintaining simplicity and clarity in code design. They are as stated below:

1) Only One Level Of Indentation Per Method
2) Don’t Use The ELSE Keyword
3) Wrap All Primitives And Strings
4) First Class Collections
5) One Dot Per Line
6) Don’t Abbreviate
7) Keep All Entities Small
8) No Classes With More Than 2 Instance Variables
9) No Getters/Setters/Properties

Scan to read this article to know more about each step with examples of code.

Which one do you agree and disagree and why? *Write down your observation about the steps above by your own experience*

.....

.....

.....

.....

Week 2 - Object Calisthenics
Exercise

1) Fight indentation

Over indentation is at the core of accidental complexity.
Reduce the indentation level of the following code. The first thing you have to religiously go through is to avoid unnecessary indentation level.

```java
public class FizzBuzz {
    private FizzBuzz() {
    }

    public static String convert(Integer input) throws OutOfRangeException {
        if (input > 0) {
            if (input ⩽ 100) {
                if (input % 3 == 0 && input % 5 == 0) {
                    return "FizzBuzz";
                }
                if (input % 3 == 0) {
                    return "Fizz";
                }
                if (input % 5 == 0) {
                    return "Buzz";
                }
                return input.toString();
            } else {
                throw new OutOfRangeException();
            }
        } else {
            throw new OutOfRangeException();
        }
    }
}
```

List down the steps to simplify this code? *Write down everything you can do to make this code clearer below.* *solution page 63*

.....

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here or clicking.

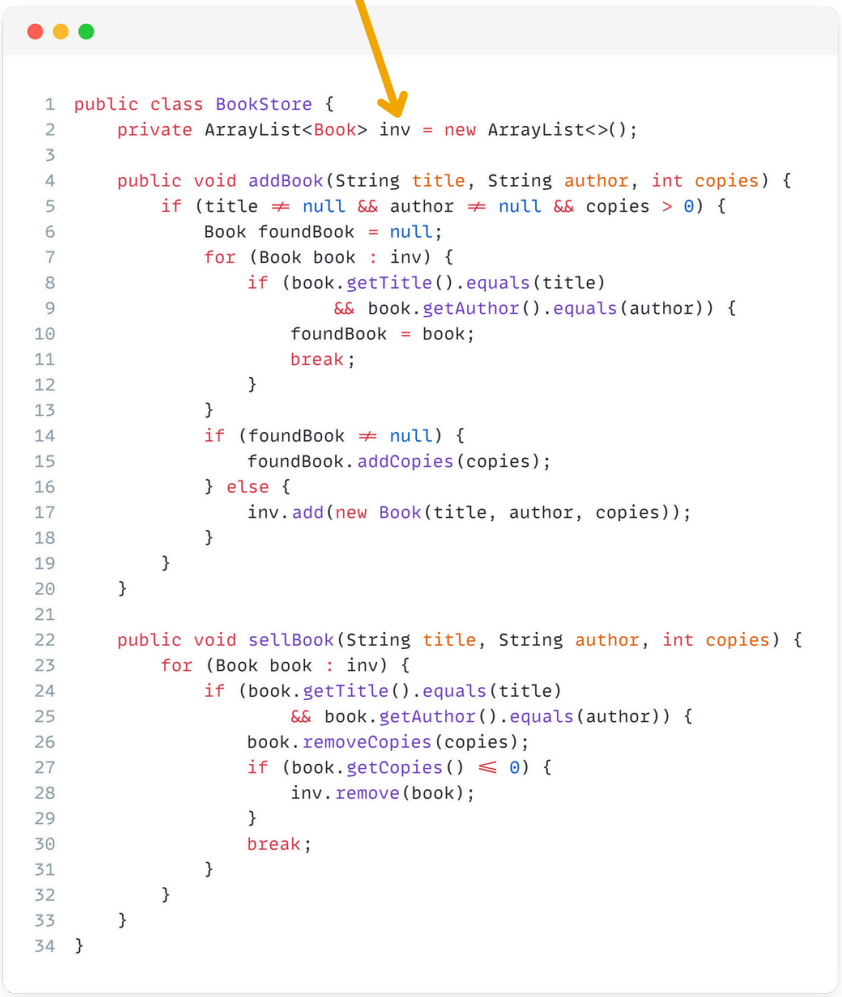
SCAN ME

2) Identify the steps

Identify the Object Calisthenics steps in the code below? *Draw an arrow to where it is in the code that does not respect a step, mark which step it is and the action you would do.* *Solution page 65*

(1) Only One Level Of Indentation
(2) Don't Use The ELSE Keyword
(3) Wrap All Primitives And Strings
(4) First Class Collections
(5) One Dot Per Line
(6) Don't Abbreviate
(7) Keep All Entities Small
(8) 2 Invariants per class max
(9) No Getters/Setters/Properties

Example : **(6) rename variable to inventory**



```java
public class BookStore {
    private ArrayList<Book> inv = new ArrayList<>();

    public void addBook(String title, String author, int copies) {
        if (title ≠ null && author ≠ null && copies > 0) {
            Book foundBook = null;
            for (Book book : inv) {
                if (book.getTitle().equals(title)
                        && book.getAuthor().equals(author)) {
                    foundBook = book;
                    break;
                }
            }
            if (foundBook ≠ null) {
                foundBook.addCopies(copies);
            } else {
                inv.add(new Book(title, author, copies));
            }
        }
    }

    public void sellBook(String title, String author, int copies) {
        for (Book book : inv) {
            if (book.getTitle().equals(title)
                    && book.getAuthor().equals(author)) {
                book.removeCopies(copies);
                if (book.getCopies() ≤ 0) {
                    inv.remove(book);
                }
                break;
            }
        }
    }
}
```

2) Link the steps

Identify the Object Calisthenics steps in the code below? *Draw an arrow to where it is in the code that does not respect a step, mark which step it is and the action you would do.* *Solution page 65*

```java
class Book {
    private String title;
    private String author;
    private int copies;

    public Book(String title, String author, int copies) {
        this.title = title;
        this.author = author;
        this.copies = copies;
    }

    public void addCopies(int additionalCopies) {
        if (additionalCopies > 0) {
            this.copies += additionalCopies;
        }
    }

    public void removeCopies(int soldCopies) {
        if (soldCopies > 0 && this.copies >= soldCopies) {
            this.copies -= soldCopies;
        }
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getCopies() {
        return copies;
    }
}

```

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 2 - Object Calisthenics Game

Complete the sentence

Complete the sentence with the right answer.
Describe which principle it is linked to.

Solution page 66

"Any class that contains a collection should contain _____. Each collection gets wrapped in its own class, so now behaviors related to the collection have a home. You may find that filters become a part of this new class."
[.....]

"Whenever you have a _____ or a string that has some business meaning, encapsulate it within a class to ensure it is not just a raw data but represents a _____."
[.....]

"To promote greater modularity and focus within classes, ensure that no class has more than _____ invariants, which encourages classes to be more _____."
[.....]

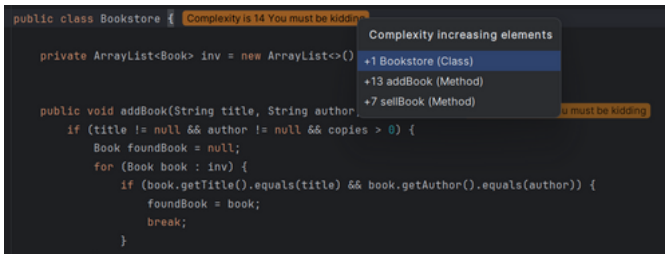
The IDE at the rescue

You can use your IDE to help you figure out the complexity level of your code.

Here is a list of plugin you can use with various IDE

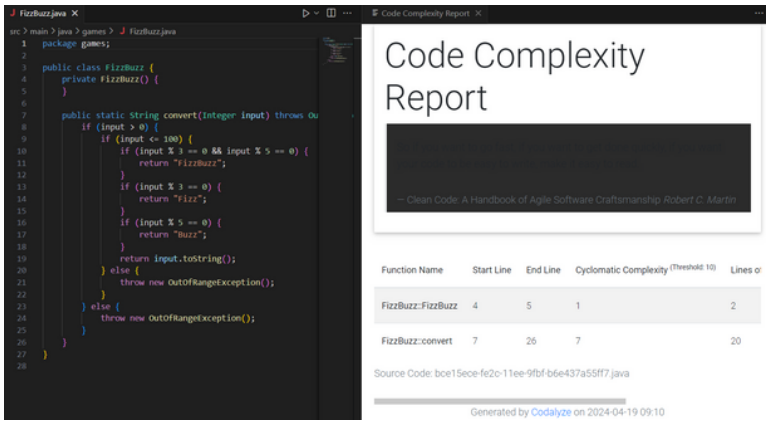
Intellij

Plugin: code complexity



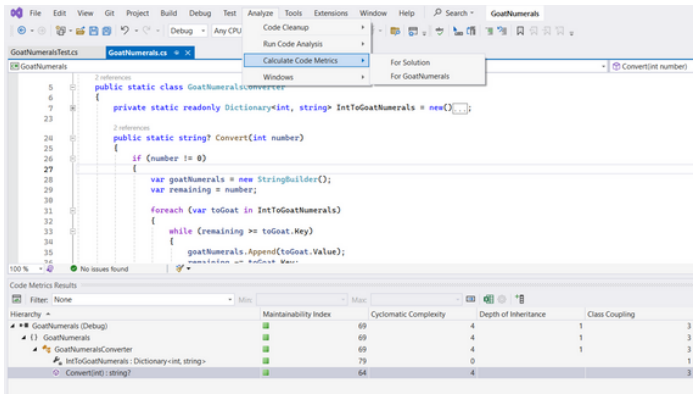
VS Code

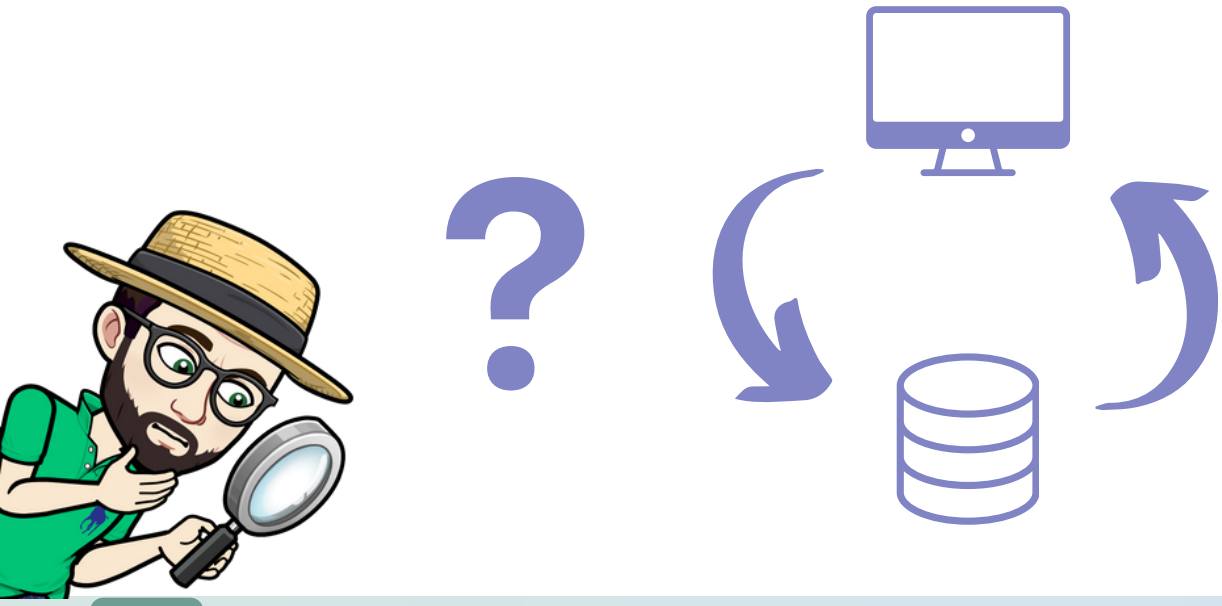
Plugin: Codalize



Visual Studio

Tool: Analyze > Calculate Code Metrics





WEEK 3

Command Query Separation

- [x] Command ?
- [ ] Query ?

```java
import java.util.ArrayList;
import java.util.List;

public class CustomList {
    private List<Integer> items;

    public CustomList() {
        this.items = new ArrayList<>();
    }

    public int addAndGetSize(int item) {
        items.add(item);
        return items.size();  // Modifies state by adding an item and returns the size
    }
}
```

Defining the responsibilities

Whose responsibility is it?

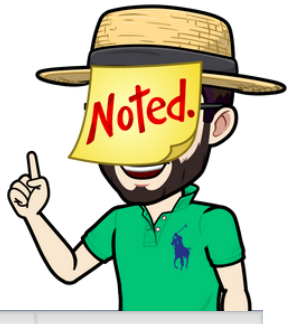
In object-oriented programming (OOP), responsibility refers to the specific roles or behaviors assigned to an object. Each object in a system is responsible for managing its data and behaviors, known as methods, which interact with other objects.

These behaviors or methods are identified as either commands or queries.

- **Commands** change the state of the system.
- **Queries** return information without altering the state.

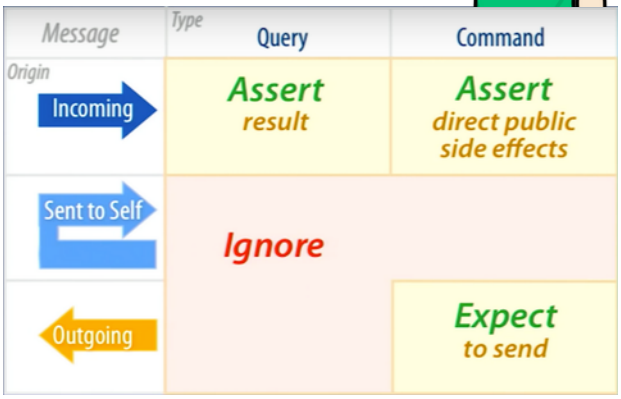
This distinction helps in designing predictable and reliable interactions within the system. The encapsulation of responsibility with the clear distinction between commands and queries ensures a modular, maintainable codebase.

<u>A method is NOT supposed to do both.</u>



How to assert properly.

Assertions in unit testing insures the system is properly validated. It's important the assertions reflects the responsibility of the method and validates it accordingly. Look at the graphic on the right to help you assert correctly your code.



Week 3 - Command Query Separation
Exercise

1) Fix the issue

Identify the issue and its consequences

Please read the code well and try to understand the issue and the underlying bad consequences such a code can have. *Solution page 68*

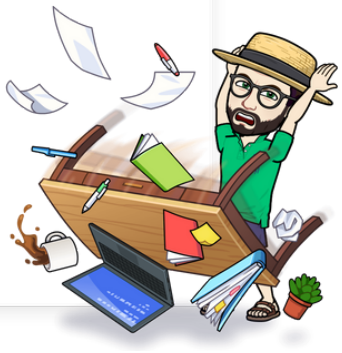
```java
public class Client {
    private final Map<String, Double> orderLines;
    private double totalAmount;

    public Client(Map<String, Double> orderLines) {
        this.orderLines = orderLines;
    }

    public String toStatement() {
        return orderLines.entrySet().stream()
                .map(entry -> formatLine(entry.getKey(), entry.getValue()))
                .collect(Collectors.joining(System.lineSeparator()))
                .concat(System.lineSeparator() + "Total : " + totalAmount + "€");
    }

    private String formatLine(String name, Double value) {
        totalAmount += value;
        return name + " for " + value + "€";
    }

    public double getTotalAmount() {
        return totalAmount;
    }
}
```



Describe the main issue and what changes you would make below.

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

2) Double edge method

Some methods can be both query & command and are especially hard to separate. Analyse the code below and see how you can refactor it to show the real intention.

Solution page 69

```java
public class Cache {
    private Map<String, Integer> map = new HashMap<String, Integer>();

    public int getOrInsert(String key, int value) {
        if (!map.containsKey(key)) {
            map.put(key, value);
        }
        return map.get(key); // Modifies state if key not present and returns value
    }
}
```

Try to write your unit tests and see the changes you want to emerge from them.

.....

.....

.....

.....

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 3 - Command Query Separation
Game

Command or Query ?

Guess the right answer!

Solution page 70

```java
public class UniqueCollection {
    private Set<Integer> elements = new HashSet<>();

    public boolean addIfMissing(int element) {
        return elements.add(element); // Attempts to add an element if it's not already present
    }
}
```

?

☐ Command
☐ Query

?

Command ☐
Query ☐

```java
public class History {
    private LinkedList<String> events = new LinkedList<>();

    public String getLast() {
        return events.getLast(); // Retrieves the last event without removing it
    }
}
```

```java
public class LightSwitch {
    private boolean isOn = false;

    public boolean toggleStatus() {
        isOn = !isOn; // Toggles the state of the light switch
        return isOn; // Also returns the new state
    }
}
```

?

☐ Command
☐ Query

?

Command ☐
Query ☐

```java
public class NetworkService {
    private String serviceUrl; //injected in constructor

    public ConnectionStatus checkConnectivity() {
        try {
            URL url = new URL(serviceUrl);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("HEAD");
            int responseCode = connection.getResponseCode();
            if (responseCode >= 200 && responseCode < 400) {
                return new ConnectionStatus(true, "Service is reachable");
            } else {
                return new ConnectionStatus(false, "Service returned error code: " + responseCode);
            }
        } catch (IOException e) {
            return new ConnectionStatus(false, "Network error: " + e.getMessage());
        }
    }

    public static class ConnectionStatus {
        private boolean isSuccess;
        private String message;
    }
}
```

CQS Heuristics

The CQS principle can bring you several advantages such as code clarity, better testability and maintainability.

CQS Heuristics

Let's look at a formula to how we would do refactoring with CQS in mind.

- Identify Commands and Queries: Review your methods and categorize them as either commands (methods that change state) or queries (methods that return data).
  - Look at the signature to see if it returns a void or something.
  - Look at the code if there is any invariant change or external call.

- Refactor Mixed Methods: If you find methods that both change state and return data, refactor them into separate methods.

- Enforce Separation in New Code: When writing new code, ensure that you adhere to CQS by clearly distinguishing commands from queries.

- Use Clear Naming Conventions: Name your methods in a way that makes it clear whether they are commands or queries. Use this standard naming convention in all your code base.

Exceptions: Fluent APIs.

Fluent APIs returns its instance as well as change states in order to do chain calls for readability. It's a strategic decision, not a CQS breach.

Other patterns use such decision to mix state change and returning data.

A great article on CQS in designing an API can be found here. You can see that CQS can help make design decision.

WEEK 4

TDD

A new methodology all together

Understanding Test-Driven Development (TDD)

Test-Driven Development or TDD is a software development approach that relies on the repetition of a very short development cycle. The primary goal of TDD is to make the code clearer, simple, and bug-free. It is predicated on the test-first method, which emphasizes writing a test before writing the functional code itself.

How TDD Works

TDD can be summarized in 3 main steps:

1. **Write a Test (RED)**: Initially, the developer writes a test that defines a function or improvements of a function, which should initially fail because the function has not yet been developed.
2. **Make it Run (GREEN)**: The next step is to write just enough code to make the test pass. This is where you ensure that the software behaves as expected.
3. **Refactor**: Once the test passes, the resulting code is refactored to acceptable standards. This might involve improving efficiency, removing duplications, and cleaning up poorly written code.

Rules when using TDD (As per Robert Martin)

Three rules can be observed when you are working with TDD:

1. No production code unless it is to make a failing unit test pass.
2. No more of a unit test than is sufficient to fail (compilation also).
3. No more production code than is sufficient to pass the one failing unit test.

Week 4 - TDD
Vocabulary

Some concepts important to know

TDD Concepts

Baby Step: the smallest possible change that can be verified and validated by your test. Meaning a change which does not break any test. The baby step ensures you are **splitting up a big change in smaller more digestible steps** and that you are building up confidence are you are coding your functionality. The resulting implementation should be fully understood and tested. You should not have any more code than necessary.

Emerging design: building your application from the smallest step to incorporate all the changes needed for your functionality will ultimately shape your final implementation. A **specific design** issued **from multiple refactoring** phases should therefore **emerge**.

Example-based testing: involves **specifying concrete values** or examples for inputs to functions or methods, and then **checking** if the **output matches expected results** for those inputs. It is a traditional approach where tests are created based on pre-defined examples that represent typical, boundary, or special case scenarios. The primary focus is to ensure that the software behaves correctly for given instances.

Property-based testing: takes a different approach by **focusing on the properties that a function should satisfy** rather than on individual examples. In this method, the system generates input data automatically, and tests are written to ensure that no matter what the input is, the output should satisfy certain properties. This helps uncover edge cases that example-based tests might miss, offering a more comprehensive assessment of potential bugs or errors in the system.

Triangulation: the triangulation is a TDD technique **used to challenge your implementation by writing more than one example-based test** to test one function of your program. You want your implementation to look a certain way but you lack the test for it. Writing a second failing test to trigger the change you want in your implementation is called *triangulating*.

Double loop: the double loop refers to the practice of **nesting** a smaller, **inner loop of red-green-refactor cycles** within a larger, outer loop that focuses on delivering functional software features incrementally. This approach helps ensure both the correctness of individual units and the integration of those units within the overall system.

Week 4 - TDD
Exercise

1) Password validation

Solution page 71

Design a simple program. Using TDD, craft a simple program to implement a password validation program using the following rules:

- Contains at least 8 characters
- Contains at least one capital letter
- Contains at least one lowercase letter
- Contains at least a number
- Contains at least a special character in this list . * # @ $ % &.
- Any other characters are not authorized.

Write your tests examples and the corresponding conceptualized implementation steps. You might want do it with a pencil to be able to rewrite some of it. *Feel free to jump to the coding version.*

Unit tests examples	Implementation Steps

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 4 - TDD
Exercise

2) Example mapping

Organize your features, scenarios and questions.
A way to get the best out of a requirement or feature to work it with TDD is to do an example mapping exercise (often collaboratively).

We start by **writing the User Story** being discussed on a yellow post-it note and placing it at the top.

User Story

Then, we write each of the **rules** (or acceptance criteria) we already know, on a blue post-it and place them under the yellow post-it of the User Story.

Rules

For each rule, we may need **one or more examples** to illustrate it. We write them on a green post-it and place them under the concerned rule.

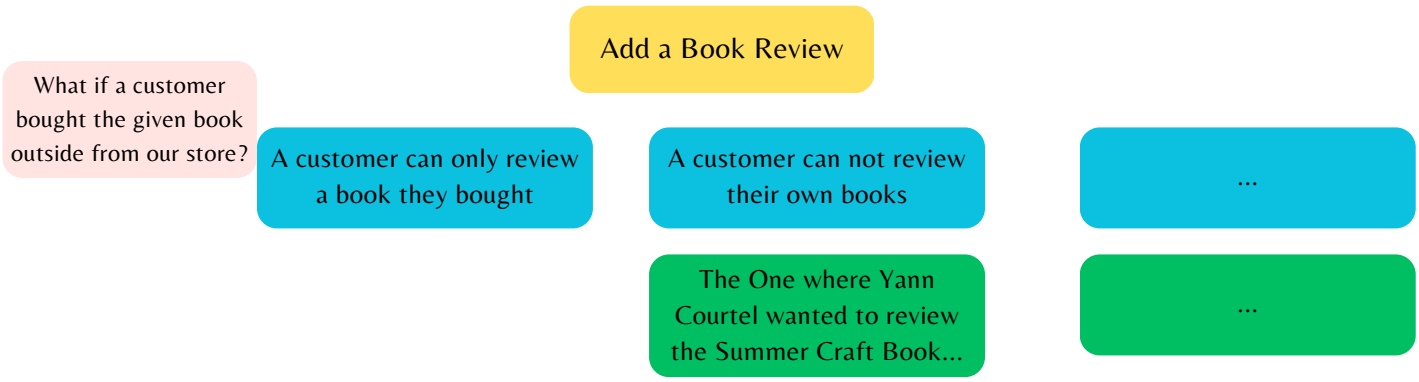
Examples

While discussing, we may discover **questions** that no one in the room can answer.
These are written on a red post-it and the conversation continues.

Questions

We continue until the group is convinced that the scope of the User Story is clear, or we run out of time.

As an example, think of an online bookstore that wants to implement a review system to gather the feedback on the books.

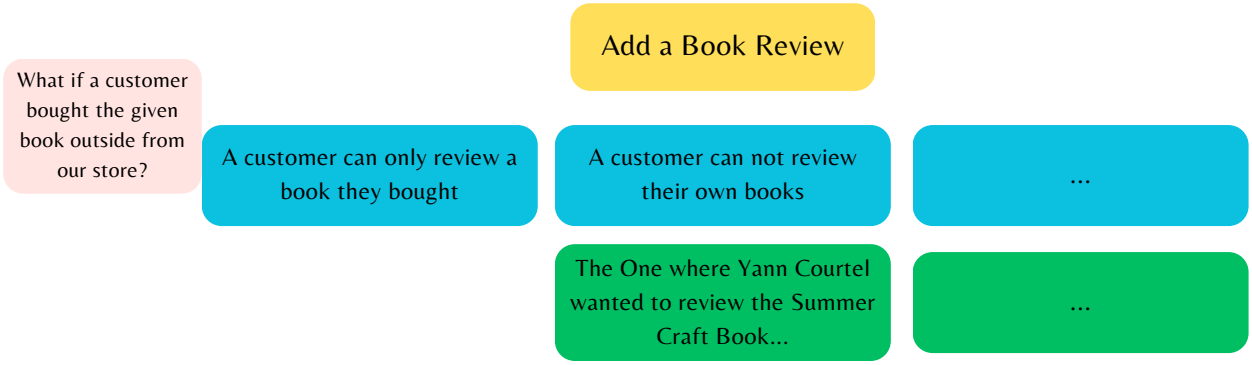


Scan the code below to see another feature broken down using example mapping

SCAN ME

2) Example mapping

Continue the model below.
As an example, think of an online bookstore that wants to implement a review system to gather the feedback on the books.



Week 4 - TDD
Game

The right path

Solution page 72

Scrambled steps. Arrange the TDD steps in the correct sequence to successfully develop the calculator functionality using TDD principles. Some steps can be repeated.

Below are the steps involved in developing a **calculator** using TDD. They are currently in the wrong order. Your task is to reorder them to reflect the correct TDD cycle. Write down the number sequence that you think correctly represents the TDD process from start to finish.

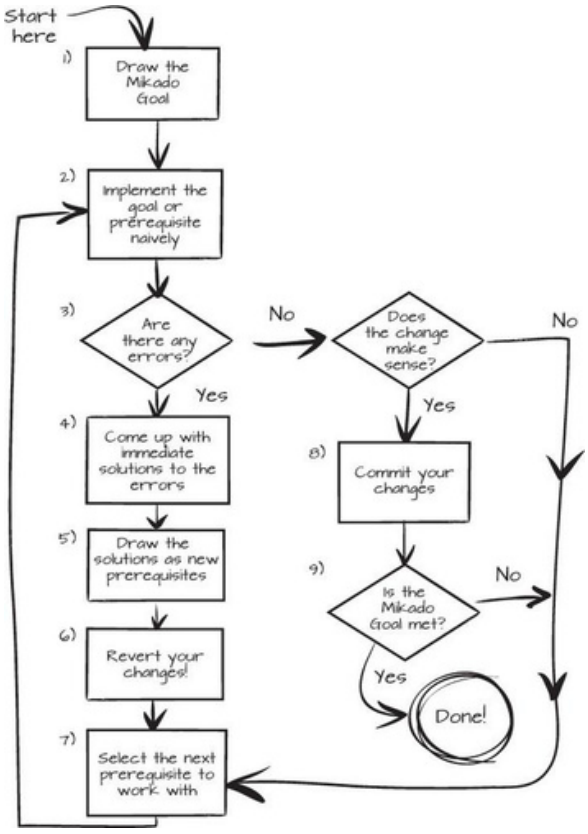
Steps	Order
A. Refactor the code to simplify and improve efficiency without changing its behavior.	
B. Run the test and see it fail (Red phase).	
C. Write a test for a subtraction feature.	
D. Implement the addition feature to pass the test.	
E. Write a test for the addition feature.	
F. Implement the subtraction feature to pass the test.	
G. Run the test and see it pass (Green phase).	
H. Refactor the subtraction implementation to handle edge cases.	
I. Review all tests to ensure coverage and clarity.	
J. Write tests for multiplication and division features.	
K. Implement the multiplication and division features to pass their tests.	

The Mikado method

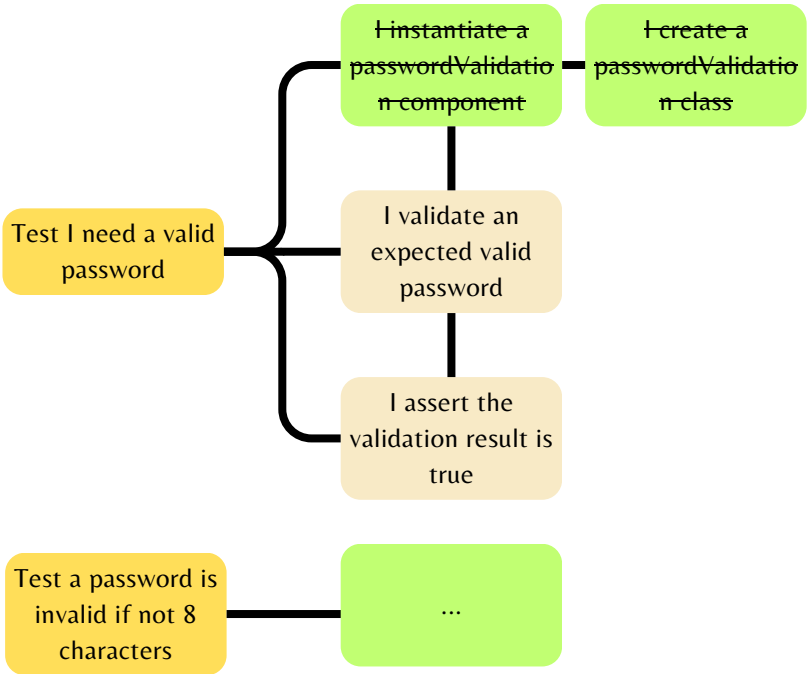
Using the Mikado method with TDD can be a great way to map out the dependencies for a functionality. Let's imagine we are taking the password validation exercise from earlier and try to map out using the Mikado method. You first need to write down the functionality or test you want to iterate.

Here are the steps to implement such method:

- Try to do your change
- Check for regression
- Try trivial changes
- Check non regression tests pass
  - Commit if it passes
  - If the goal is achieved, close the tree
- If non regression tests fails
  - Add a dependency branch to your graph
  - Revert
  - Recurse on dependency



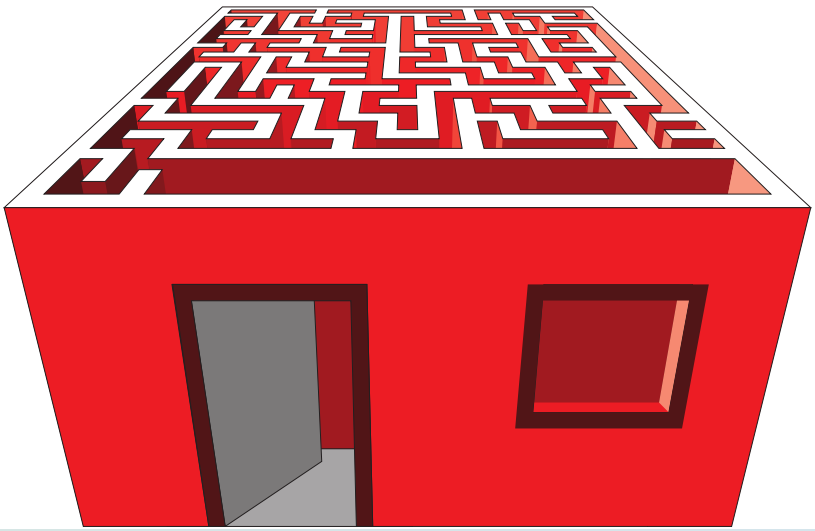
An example of how you could start for the password validation:



Scan the code here to understand the basics of the Mikado method.

Scan the code here if you want to go further and do an entire refactoring kata using the Mikado method.

SCAN ME



WEEK 5

Accidental complexity

Conditionals increase complexity

Ways to reduce accidental complexity

Number of code bases suffer from accidental complexity. One of the cause for such complexity is the way we overuse conditionals.

Avoiding conditionals in code can significantly enhance its readability, maintainability, and error-resistance. Traditional conditional statements like if and else often make code verbose and complex, which can lead to errors and difficulties when debugging or extending the code.

To avoid accidental complexity, several techniques are at your disposal:

- **Inheritance & polymorphism**: if 2 algorythms behave almost the same, you might want to consider creating a component with 2 implementations.
- **Design patterns**: Like Strategy and Command patterns. They are basically a more complex version of the inheritance / polymorphism approach.
- **Nullable objects**: returning nullable objects can help reduce the conditionals.
- **Functional Programming Goodies**: Some FP techniques like Monads can help you handle an alternative state outside of the implementation itself reducing greatly the conditionals in your code.
- **Functional Maps**: a technique leveraging dictionaries or hash maps to associate keys directly with functions or values. It simplifies the logic by abstracting control flow into data, making the code more declarative and easier to understand

Making your code predictable

Striving for predictable code is crucial on your journey. Doing so offers significant advantages, such as easier debugging, reduced maintenance costs, and enhanced scalability.

Predictable code ensures consistent behavior, making it simpler to understand, test and modify. This reliability also improves team collaboration and the onboarding process for new developers, leading to more efficient development cycles.

This practice is fundamental for maintaining high quality and scalability in software projects.

Week 5 - Accidental complexity
Exercise

1) Do not use if... else

Get rid of conditionals in the code below. From the techniques described in the previous section, try to get rid of the conditionals in this code to prevent complexity. *Solution page 73*

```
1     public static String convert(Integer input) throws OutOfRangeException {
2         if (isOutOfRange(input)) {
3             throw new OutOfRangeException();
4         }
5         return convertSafely(input);
6     }
7
8     private static String convertSafely(Integer input) {
9         if (is(FIZZBUZZ, input)) {
10            return "FizzBuzz";
11        }
12        if (is(FIZZ, input)) {
13            return "Fizz";
14        }
15        if (is(BUZZ, input)) {
16            return "Buzz";
17        }
18        return input.toString();
19    }
```

Describe which techniques you would use and where.

.....

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 5 - Accidental complexity
Exercise

2) Above and beyond conditionals

Solution page 74

Optimize the code below. We want to remove all conditionals and make a few adjustments for the future. How easy it is to add a new command? How would you tackle the TODOs?
Remember, we want to avoid complexity. Other things might be missing...

```java
public interface Command {
    void execute();
}

public class CommandProcessor {
    private Map<String, Command> commandMap;

    public CommandProcessor() {
        commandMap = new HashMap<>();
        commandMap.put("greet", () → System.out.println("Hello, World!"));
        commandMap.put("exit", () → System.out.println("Exiting application..."));
    }

    public void processCommand(String command) {
        if (commandMap.containsKey(command)) {
            commandMap.get(command).execute();
        } else {
            System.out.println("Unknown command");
        }
    }

    public static void main(String[] args) {
        CommandProcessor cp = new CommandProcessor();

        // TODO: Should be able to pass my name to the command to say Hello to me!
        cp.processCommand("greet");  // Outputs: Hello, World!
        cp.processCommand("exit");   // Outputs: Exiting application...
        // TODO: Should display all commands available
        cp.processCommand("help");    // Outputs: Unknown command
    }
}
```

Describe here how you would do it.

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 5 - Accidental complexity
Game

Detect the pattern

Solution page 75

Find the design pattern in this code. In the code below, a pattern has been used to reduce complexity, would you be able to find which one?

```java
public class TaskExecutor {
    private Task task;

    public TaskExecutor(Task task){
        this.task = task;
    }

    public void executeTask(){
        task.performTask();
    }
}

interface Task {
    void performTask();
}

class OperationA implements Task {
    public void performTask() {
        System.out.println("Executing Operation A");
    }
}

class OperationB implements Task {
    public void performTask() {
        System.out.println("Executing Operation B");
    }
}
```

?

- [ ] Command
- [ ] Observer
- [ ] Strategy
- [ ] Singleton

Would you suggest another pattern instead? If so, which one and why?

.....

.....

.....

Ask another human

Why collaborative programming ?

Pair, mob, team or just in general collaborative programming involves at least 2 developers working on a single machine, on a single piece of code.

This is by far the best way to avoid accidental complexity. Not only will you have at least 2 brains working on a single subject but you won't have any opportunity to digress when coding. You won't take shortcut because you have a teammate and you likely won't stay stuck on a piece of code and miss something.

Different formats

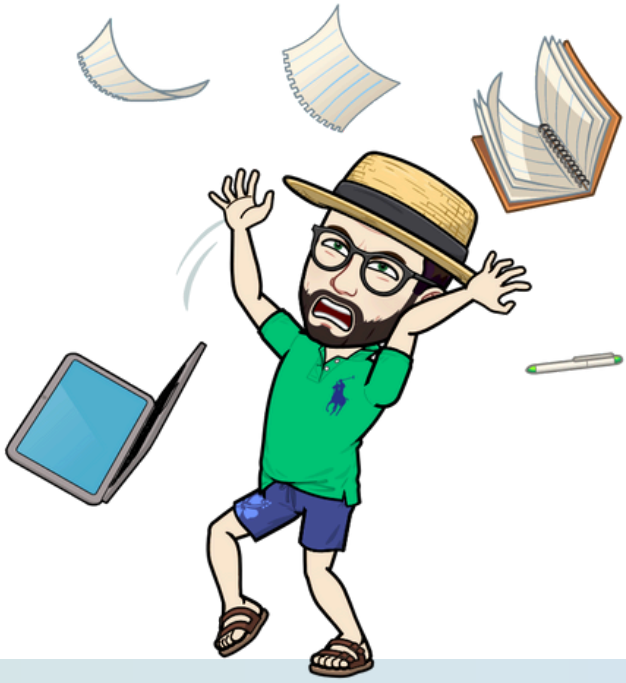
Pair programming involves two developers working together at one workstation. One developer, the "driver," writes the code, while the other, the "navigator," reviews each line of code as it is written. The two programmers switch roles frequently (ideally every 10-15 minutes).
Benefits: Enhances code quality, improves knowledge sharing, and helps in problem-solving by leveraging two perspectives.

Mob or team programming extends the idea of pair programming to a whole team. All team members work together on the same task, at the same time, in the same space (or virtually), using a single computer.

Peer review: one or more developers review the code written by another developer. This process usually happens after the code is written and before it is merged into the main codebase.
Benefits: Identifies bugs, ensures adherence to coding standards, and facilitates knowledge transfer across the team.

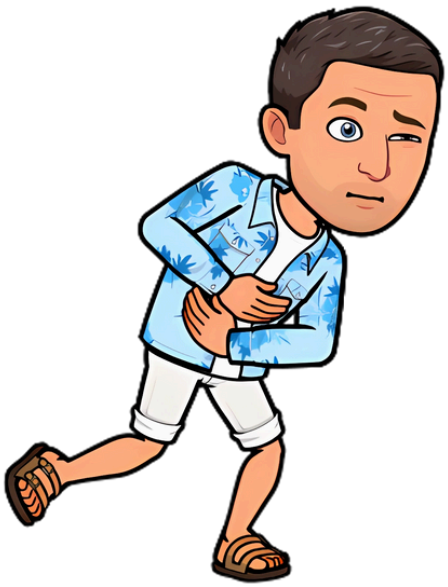
Plugins

Tools such as the CodeWithMe plugin with the Intellij IDE can help you share a coding session with colleagues for an easy collaborative programming workshop.



WEEK 6

Legacy Code



Your code might be legacy

Identifying how a code becomes a debt

When we think of legacy code, we usually think of an old technology, monolith architecture and crazy indentations levels, abbreviated variables and implementation leaks all over the place.

Yet, we all do hot-fixes, sometimes on a daily basis without tests and instantly add to the technical debt of our project. Wouldn't you say this is legacy code?

As Martin Fowler puts it in his bool, *Working effectively with Legacy Code*:

> "To me, legacy code is simply code without tests."

A well-written code with the best patterns and naming without the proper tests adds instantly to the amount of potential unsafe refactoring to your code. It becomes what we call a technical debt and, if not tended to, can fester and may need a considerable rework.

The boy scout rule

One way to look at your code (or any code) is to say: How can I make this code better than when I found it?

That is the *Boy Scout Rule*.

You don't have to refactor everything, all at once. Just scope small steps, tested, that will make your code better for the future developers. Better, share your changes with your colleagues and indulge in pair programming to lesser the code review time and have shorter feedback loops.

Trust in your tests

When refactoring legacy code, you should trust your tests to ensure modifications don't introduce new bugs. Proper tests validate code behavior, offering a safety net that enables confident alterations. Trusting these tests allows you to maintain system integrity while improving code structure and performance. And remember the golden rule of refactoring:

<u>Do NOT start a refactoring without the proper tests</u>

Week 6 - Legacy Code
Approval testing

How to tackle legacy code

Getting out of the mess.

Testing the code before you refactor might not be an option. Some circumstances might force you to rework the code without the proper tests suite. Here are some exceptions to the rule of thumb:

- Total inability to test my code (manual testing ?)
- Technical code that only has manual or semi-manual tests (a batch for instance)
- Auto-generated actions by the IDE that does not break the design (renaming, extracting, inlining, etc)
- Test code

How about your own experience, do you have any other scenarios not covered here? How did you achieve the lack of testability?

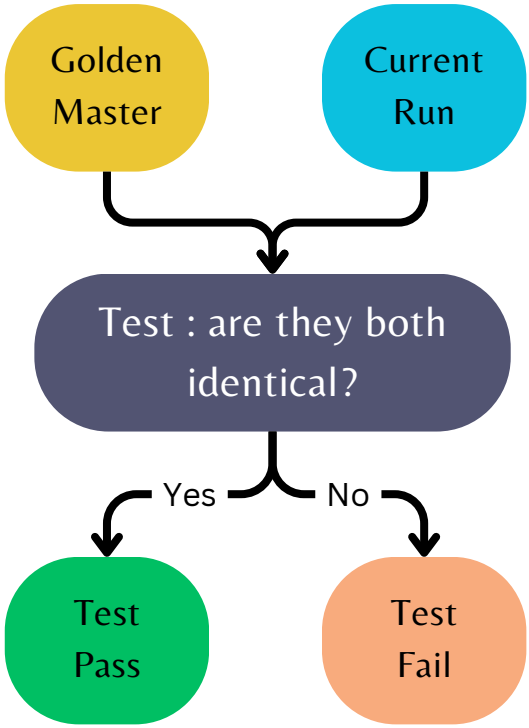
Approval testing

A last resort solution.

A technique used to bypass the lack of testability is to use the approval testing or most commonly called the Golden Master tests.

The idea is to be able to make “snapshots” of your code as an expected and run the same snapshot of your code at a different time and compare them. If they are not the same, the test will fail.

The snapshot can be generated in many different way using the console output, a logging system or just the output of a method.



Week 6 - Legacy Code
Exercise

1) Put the code under tests

Solution page 76

Try the approval testing approach. Add tests for the code below. You can use the Golden Master technique with an expected file.

```java
public enum DocumentTemplateType {
    DEERPP("DEER", RecordType.INDIVIDUAL_PROSPECT),
    DEERPM("DEER", RecordType.LEGAL_PROSPECT),
    AUTP("AUTP", RecordType.INDIVIDUAL_PROSPECT),
    AUTM("AUTM", RecordType.LEGAL_PROSPECT),
    SPEC("SPEC", RecordType.ALL),
    GLPP("GLPP", RecordType.INDIVIDUAL_PROSPECT),
    GLPM("GLPM", RecordType.LEGAL_PROSPECT);

    private final String documentType;
    private final RecordType recordType;

    DocumentTemplateType(String documentType, RecordType recordType) {
        this.documentType = documentType;
        this.recordType = recordType;
    }

    public static DocumentTemplateType fromDocumentTypeAndRecordType(String documentType, String recordType) {
        for (DocumentTemplateType dtt : DocumentTemplateType.values()) {
            if (dtt.getDocumentType().equalsIgnoreCase(documentType)
                    && dtt.getRecordType().equals(RecordType.valueOf(recordType))) {
                return dtt;
            } else if (dtt.getDocumentType().equalsIgnoreCase(documentType)
                    && dtt.getRecordType().equals(ALL)) {
                return dtt;
            }
        }
        throw new IllegalArgumentException("Invalid Document template type or record type");
    }

    private RecordType getRecordType() {
        return recordType;
    }

    private String getDocumentType() {
        return documentType;
    }
}
```

Write down the combinations you would add to your golden master file.

.....

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 6 - Legacy Code
Exercise

2) What's wrong here

Solution page 77

Could you explain what's incorrect about the implementation below? Explain what is wrong with this legacy code and why.

.....

.....

.....

```java
class ReportGenerator {
    public void generateReport(String reportType, List<ReportData> data) {
        if ("CSV".equals(reportType)) {
            System.out.println("Starting CSV Report Generation...");
            System.out.println("CSV Header: ID, Value, Description");
            for (ReportData d : data) {
                System.out.println(d.getId() + "," + d.getValue() + "," + d.getDescription());
            }
            System.out.println("CSV Report Generated Successfully.");
        } else if ("PDF".equals(reportType)) {
            System.out.println("Starting PDF Report Generation...");
            System.out.println("PDF Report Title: Comprehensive Data Report");
            System.out.println("-----------------------------------------------");
            for (ReportData d : data) {
                System.out.println("Data ID: " + d.getId() + " | " + "Data Value: " + d.getValue() + " | " + "Description: " + d.getDescrip
            }
            System.out.println("-----------------------------------------------");
            System.out.println("PDF Report Generated Successfully.");
        } else {
            System.out.println("Report type " + reportType + " not supported.");
        }
    }
}

```

```java
public class ReportData {
    private int id;
    private double value;
    private String description;

    public ReportData(int id, double value, String description) {
        this.id = id;
        this.value = value;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public double getValue() {
        return value;
    }

    public String getDescription() {
        return description;
    }
}
```

Highlight all the smells you can find in the code and provide a solution to how you would fix them.

.....

.....

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 6 - Legacy Code Game

Smell to score

Using the grid of score below, reach the minimum score. For each code snippet, try to detect as many smells as possible and a potential fix.

Write down the smells and explain how you would fix them to get the score for each smells.

```java
public class OrderProcessor {
    public void processOrders(List<Order> orders) {
        for (Order item : orders) {
            if (item.getStatus() == Order.UNPROCESSED) {
                if (item.getItems() > 5) {
                    // apply bulk discount
                    item.setTotal(item.getTotal() * 0.9);
                }
                item.setStatus(Order.PROCESSED);
            }
        }
    }
}
```

```java
public class Order {
    public static final int UNPROCESSED = 0;
    public static final int PROCESSED = 1;
    private int status;
    private int items;
    private double total;

    public Order(int status, int items, double total) {
        this.status = status;
        this.items = items;
        this.total = total;
    }

    public int getStatus() {
        return status;
    }

    public void setStatus(int status) {
        this.status = status;
    }

    public int getItems() {
        return items;
    }

    public double getTotal() {
        return total;
    }

    public void setTotal(double total) {
        this.total = total;
    }
}
```

Each smell is worth points. You need to find enough smells to reach the required minimum score.

Smell score

remove bug / risk (6), extract / correct dependency (5), increase domain language (4), simplify code (3), fix naming (2), adjust readability (1)

Smell	Type	Line #	Potential fix	Score
wrong variable name	2	5	rename item name for ‘order’	2

Required score = 15

Start the exercise on the next page

Help yourself with the code version

SCAN ME

Week 6 - Legacy Code
Game

Smell to score

Solution page 78

Using the grid of score below, reach the minimum score. For each code snippet, try to detect as many smells as possible and a potential fix.

Required score = 15

Smell score

remove bug / risk (6), extract / correct dependency (5), increase domain language (4), simplify code (3), fix naming (2), adjust readability (1)

Smell	Type	Line #	Potential fix	Score

Feel free to make the changes to your own solution by using the coding version of the exercise.

Total score = ____

Other Techniques for Testing Legacy Code

Often, you won't be able to test your code without making changes to it. Thus, you want to identify the steps you can take in your code that won't change its behavior.

Extending Classes (Seams)

When you identify a method that accesses the outside, such as getting information from a database or a singleton, it often involves hard-wired dependencies that make testing difficult. To address this, make the method overridable (virtual with some mocks) and override it inside the tests with a test double. This way, the new test class won't call the database, and you can mock your return. You can also add your own implementation details used for testing.

Extract & Overloading Methods

If your code is too tightly coupled to the implementation of its dependencies, you might not have a method that calls the outside, but everything will be procedural. One way to change that is to extract part of your code in a method that accesses the outside or a singleton, and then change the accessibility of the method so you can override it. This is similar to the technique mentioned above.

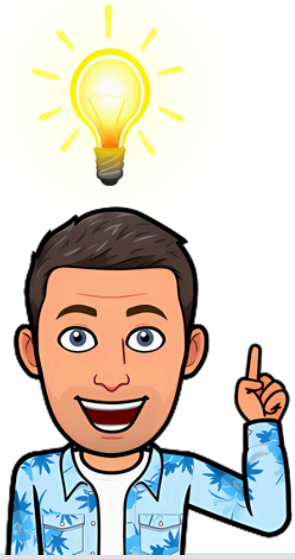
Sprout Methods

The sprout method involves adding new code by "sprouting" it into small, isolated methods or classes without altering the existing code structure. This approach minimizes the risk of introducing bugs into the legacy code and allows for new features to be implemented in a more controlled and manageable way.

- Identify where the new functionality is needed.
- Create a new method or class to implement the new functionality.
- Call the new method or class from the existing code at appropriate points.

All technics can be done automatically using your favorite IDE's shortcuts.

Some technics are illustrated in this article. If you have any legacy code on hands, it could be worth trying.



WEEK 7

Property-Based Testing

Different type of testing

Example-based testing

By default, we test using example-based expectations. We pass into our system an input by calling a behavior of a component and we check the output, whether it be a data we can read or verify a task from a mock that is being triggered. This approach is straightforward and effective for functions or methods where the outputs are predictable and can be defined in advance. Anyone who is familiar with unit tests and has written one knows what example-based testing is.

```
@Test
void testAdd() {
    Calculator calculator = new Calculator();
    int expected = 5;

    int actual = calculator.add(3, 2);

    assertEquals(expected, actual);
}
```

```
class FizzBuzzTests {

    private static Stream<Arguments> invalidInputs() {
        return Stream.of(
                Arguments.of(0),
                Arguments.of(-1),
                Arguments.of(101)
        );
    }

    @ParameterizedTest
    @MethodSource("invalidInputs")
    void parse_fail_for_numbers_out_of_range(int input) {
        assertThat(FizzBuzz.convert(input).isEmpty())
                .isTrue();
    }
}
```

Parameter-based testing

Parameter-based testing allows you to run the same test with different values without duplicating code. It is useful for covering a range of scenarios and ensuring that your method behaves correctly with various inputs.

Property-based testing

Property-based testing involves checking that certain properties hold true for all valid inputs. It uses algorithms to generate inputs dynamically, testing the robustness of the function beyond fixed data sets. Different libraries can be used to achieve that. Here an example is the vavr library.

```
import io.vavr.test.Arbitrary;
import io.vavr.test.Property;
import io.vavr.test.Gen;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {

    private final Calculator calculator = new Calculator();

    @Test
    public void additionAlwaysReturnsTheSumOfTwoIntegers() {
        Arbitrary<Integer> integers = Gen.choose(Integer.MIN_VALUE / 2, Integer.MAX_VALUE / 2).arbitrary();

        Property.def("addition always returns the sum of two integers")
                .forAll(integers, integers)
                .suchThat((a, b) -> {
                    int expectedSum = a + b;
                    int actualSum = calculator.add(a, b);
                    return expectedSum == actualSum;
                })
                .check()
                .assertIsSatisfied();
    }
}
```

Week 7 - Property-Based Testing
Exercise

1) Thousand tests in one

Refactor the Fizzbuzz tests from parameterized tests to property-based tests. Look at the tests and implementation to see how you can extract common behavior and see if you can find properties. *Solution page 79*

```java
public class FizzBuzz {
    public static final int MIN = 1;
    public static final int MAX = 100;
    private static final Map<Integer, String> mapping = LinkedHashMap.of(
            15, "FizzBuzz",
            3, "Fizz",
            5, "Buzz"
    );

    public static Option<String> convert(int input) {
        return isOutOfRange(input)
                ? none()
                : some(convertSafely(input));
    }

    private static String convertSafely(Integer input) {
        return mapping
                .find(p -> is(p._1, input))
                .map(p -> p._2)
                .getOrElse(input.toString());
    }

    private static boolean is(Integer divisor, Integer input) {
        return input % divisor == 0;
    }

    private static boolean isOutOfRange(Integer input) {
        return input < MIN || input > MAX;
    }
}
```

```java
class FizzBuzzTests {
    private static Stream<Arguments> validInputs() {
        return Stream.of(
                Arguments.of(1, "1"),
                Arguments.of(67, "67"),
                Arguments.of(82, "82"),
                Arguments.of(3, "Fizz"),
                Arguments.of(66, "Fizz"),
                Arguments.of(99, "Fizz"),
                Arguments.of(5, "Buzz"),
                Arguments.of(50, "Buzz"),
                Arguments.of(85, "Buzz"),
                Arguments.of(15, "FizzBuzz"),
                Arguments.of(30, "FizzBuzz"),
                Arguments.of(45, "FizzBuzz")
        );
    }

    private static Stream<Arguments> invalidInputs() {
        return Stream.of(
                Arguments.of(0),
                Arguments.of(-1),
                Arguments.of(101)
        );
    }

    @ParameterizedTest
    @MethodSource("validInputs")
    void parse_successfully_numbers_between_1_and_100(int input, String expectedResult) {
        assertThat(FizzBuzz.convert(input))
                .isEqualTo(Some(expectedResult));
    }

    @ParameterizedTest
    @MethodSource("invalidInputs")
    void parse_fail_for_numbers_out_of_range(int input) {
        assertThat(FizzBuzz.convert(input).isEmpty())
                .isTrue();
    }
}
```

Write down the properties you see from the snippet of code. Refactor the tests using PBT.

.....

.....

.....

.....

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 7 - Property-Based Testing Game

Find the property

Solution page 80

Identify the valid properties for the code samples. Look at the code snippets below and find out what properties are valid in the proposed list on the right.

1) Testing a Maximum Function

```
public int max(int a, int b) {
    return a > b ? a : b;
}
```

- [ ] **Commutativity**: max(a, b) should equal max(b, a).
- [ ] **Idempotence**: max(a, a) should return a.
- [ ] **Non-Negativity**: max(a, b) should not be negative.
- [ ] **Greater or Equal**: max(a, b) is always greater than or equal to a and b.
- [ ] **Additive Property**: max(a+1, b) is greater than max(a, b).

2) Testing an Array Sorting Function

```
public int[] sort(int[] array) {
    Arrays.sort(array);
    return array;
}
```

- [ ] **Idempotence**: Applying sort twice results in the same output as applying it once.
- [ ] **Element Preservation**: The sorted array contains exactly the same elements as the input array.
- [ ] **Non-Decreasing Order**: Every element in the array is less than or equal to the next element.
- [ ] **Length Alteration**: The length of the sorted array is different from the input array.
- [ ] **First Element**: The first element is always the smallest in the array.

3) Testing a Method that Trims Whitespace

```
public String trimString(String input) {
    return input.trim();
}
```

- [ ] **Idempotence:** Applying trim twice results in the same output as applying it once.
- [ ] **Length Reduction:** The trimmed string is always shorter than the input.
- [ ] **Empty String Result:** Trimming an empty string results in an empty string.
- [ ] **Whitespace Preservation:** Inner whitespace is preserved.
- [ ] **Non-Whitespace Characters:** Non-whitespace characters are never removed.

You can go further and write the tests using the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 7 - Property-Based Testing Game

2) Find the property

Exercise page 81

Identify the valid properties for the code samples. Look at the code snippets below and find out what properties are valid in the proposed list on the right.

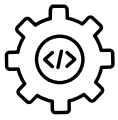
4) Testing a Simple Array Filter Function

```java
public String trimString(String input) {
    return input.trim();
}
```

- [ ] **Size Reduction:** The output array is always smaller than the input array.
- [ ] **Element Consistency:** All elements in the output array are even.
- [ ] **Non-Duplication:** The output array contains no duplicate elements.
- [ ] **Non-Negativity:** The output array contains only non-negative numbers.
- [ ] **Order Preservation:** The order of elements in the output array is the same as their order in the input array.

5) Testing an Asynchronous Data Loading Method

```java
public CompletableFuture<String> loadDataAsync(String url) {
    return CompletableFuture.supplyAsync(() -> {
        try {
            // Simulated delay
            Thread.sleep(1000);
            return "Data from " + url;
        } catch (InterruptedException e) {
            return "Failed to load";
        }
    });
}
```



- [ ] **Non-Null Result:** The CompletableFuture always completes with a non-null value.
- [ ] **Consistency:** The same URL always results in the same data.
- [ ] **Response Time:** The CompletableFuture completes within 1 second.
- [ ] **Error Handling:** Any interruption results in a "Failed to load" message.
- [ ] **Thread Safety:** The method can be safely called from multiple threads simultaneously.

6) Testing JSON Serialization

```java
public String serializeToJson(Object data) {
    return new Gson().toJson(data);
}
```

- [ ] **Reversibility:** Deserializing the serialized JSON string yields an object equal to the original.
- [ ] **Consistency:** Serializing the same object multiple times produces the same JSON string.
- [ ] **Null Handling:** Serializing a null object returns "null".
- [ ] **Exception Safety:** Serialization throws an exception if the object contains non-serializable types.
- [ ] **Format Validity:** The output is always a valid JSON format.

You can go further and write the tests using the coding version of this exercise by scanning the QR code here.

SCAN ME

Property-based and TDD

Using Property to drive your development with TDD can yield great results.

Look at the tests and implementation to see how you can extract common behavior and test them without using examples.

Here are the steps to implement such method:

- **Red** Phase:
  - Write a property that should fail initially.
  - For instance, "a sorted list should remain sorted after sorting again".
- **Green** Phase:
  - Implement the minimal code required to pass the property.
  - Run the property tests to ensure the code passes for a wide range of inputs.
- **Refactor** Phase:
  - Refactor the code while ensuring the properties continue to hold true.
  - Run the property tests again to verify correctness.

Combining Example-Based and Property-Based Testing:

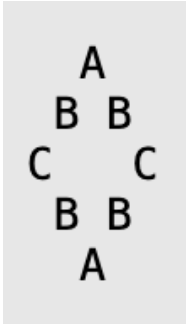
- Use property-based tests to cover a wide range of inputs and general properties.
- Use example-based tests for specific edge cases, known bugs, or critical scenarios.

An example of how you could start for the Diamond kata:

Horizontally symmetric
for all (validCharacter in [A-Z])
such that diamond(validCharacter) == reverse(diamond(validCharacter))

A Diamond is a square (height = width)
for all (validCharacter)
such that diamond(validCharacter) is a square

2 identical letters per line
for all (validCharacter)
such that each line in diamond(validCharacter) contains 2 identical letters except first and last



To try the property driven development on an exercise, you can do the day 21 of the 2023's Advent of Craft here.

SCAN ME

WEEK 8

Recap

Week 8 - Recap
Software Craftsmanship

Recap: week per week

Week 1: Code analysis
- Categorizing and prioritizing refactoring actions.
- Choose your own category.
- Assign complexity to refactoring actions.
- Detect hotspots to see where the value is.
- Mind-map your notes to be more organized.
- Scope your refactoring.

Week 2: Object Calisthenics
- Follow the principles to simplify your code.
- Fight indentation and conditionals.
- Avoid primitives and abbreviations.
- Do hesitate to be verbose in your naming.
- One dot per line.

Week 3: Command Query Separation
- A command impacts the system.
- A query returns data.
- A method cannot be both.
- You can enforce these rules at the code review

Week 4: TDD
- A methodology of design & development.
- Small iteration cycle starting with the tests.
- The three steps:
  - Red - write a failing test
  - Green - quickly pass the test
  - Refactor - refactor the code you wrote
- Baby step: the smallest step to drive your code.

Week 5: Accidental Complexity
- Use inheritance, design patterns.
- Functional maps are also a solution.
- Make your code as predictable as possible.
- Remember YAGNI, KISS and other principles.
- Collaborative programming to avoid complexity.

Week 6: Legacy Code
- Legacy code can be code without tests.
- Code can become a technical debt very fast.
- Use approval testing to get out of the mess.
- Do not refactor a code without tests.
- Seams can help you make a code testable.

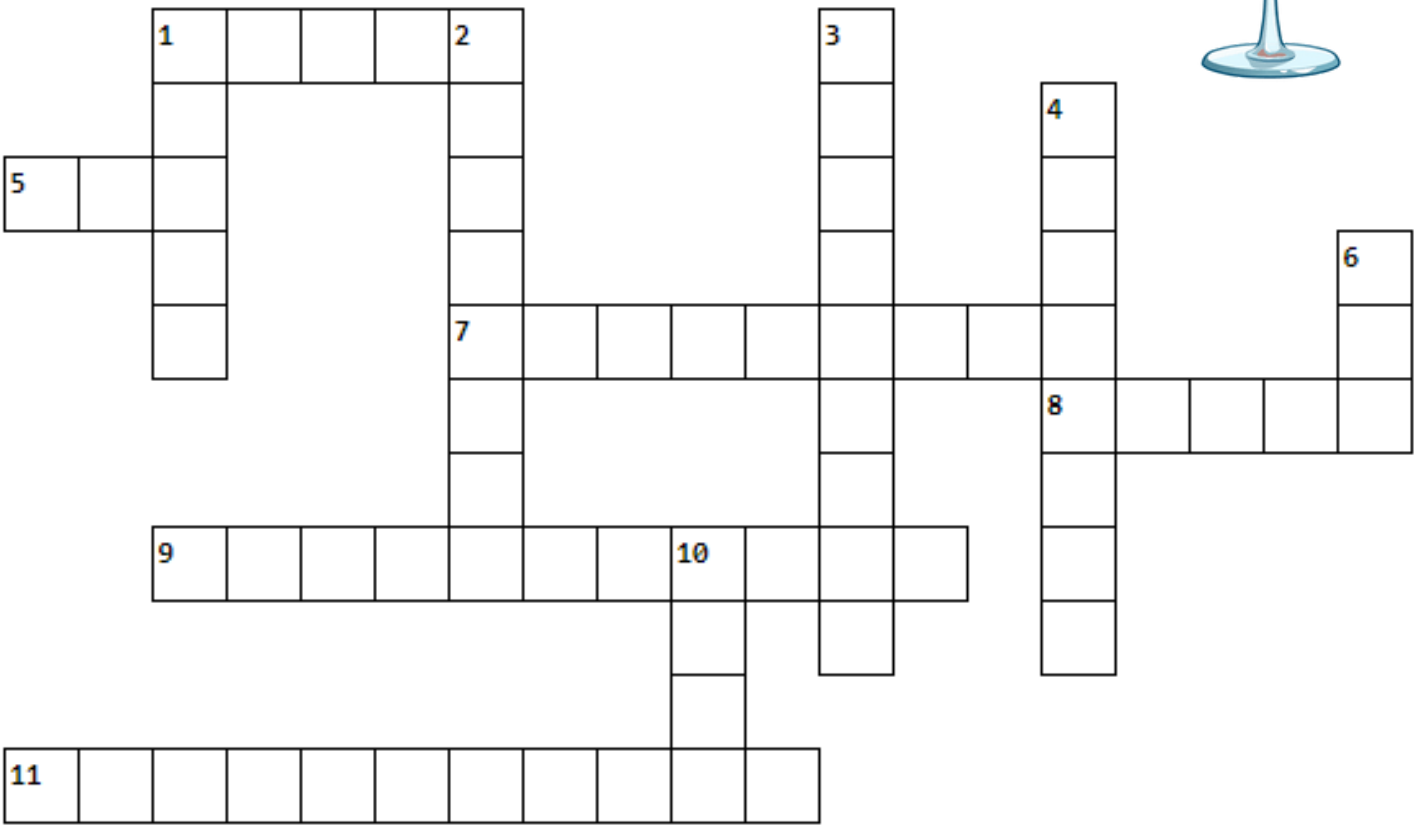
Week 7: Property-based Testing
- Example-based testing tests concrete examples.
- Param testing uses a range of examples.
- PBT defines property and checks they stay true.
- PBT generates 1000 of inputs to check property.
- Use PBT with TDD to boost your code.

Write your own notes about each week in this section

Week 8 - Recap
Game

Crossword

Solution page 82



Across

1. Agile Framework
5. Where the magic happens
7. if (true)
8. Set of good engineering principles
9. Level of danger
11. Allow mocking

Down

1. A bad code
2. Testing tests
3. One instance
4. TDD's core concept
6. A modeling centered around the business
10. I will do it later

Week 8 - Recap
Game Within the Game

Find the hidden sentence

Find all the words that composes the hidden sentence. Clues are located throughout the entire book.

Solution page 83

“ 1 _ _ _ 2 _ _ _ 3 _ _ 4 _ _ 5 _ _

6 _ _ _ _ _ 7 _ _ _ _ 8 _ _ _ 9 _ _ _ _

10 _ _ _ 11 _ _ _ _ - 12 _ _ _ _ _ _

13 _ _ _ _ 14 _ _ _ 15 _ _ _ _ 16 _ _ _

17 _ _ _ _ _ 18 _ _ . ”

1	Belongs to us	7	W/	13	Same as #7
2	A program we can test with approval testing	8	Same as #1	11	1st word of week 1 tip
3	Word #5 at the 3rd person	9	What improves with the Boy Scout Rule	12	More than the one
4	1st word, 3rd paragraph, week 2 game	10	!!!	10	Anagram of how
5	The 1st letter of the first two concepts of TDD	11	A word in the principle in week 6 exercise 2	11	TDD is about better .. ?
6	Contrary of loose	12	Inclined	12	Our industry

We are crafters

“Software craftsmanship is to thrive to reach excellence with everybody along with you”

Craftsmanship is a lot of practice, some theory, some sharing and some common sense.

Share what you have learned during this experience and enjoy the journey.

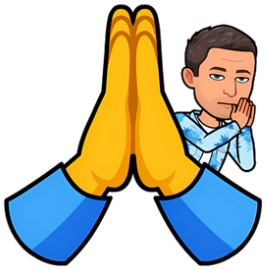
Remember to be humble as you can learn from anybody.

On a last note, you can enjoy this crossword to test you knowledge. Happy crafting!

Sharing is caring

If you like the experience, please share your own experience on the discord of the Advent of Craft.

We will gladly take all the feedback.



Scan the code here to be linked to our Discord. You can share about specific concepts or exercise with the community.

SCAN ME

ANSWERS

Proposed Solutions

Week 1 - Code Analysis
Exercise - Proposed Solution

1) Ugly Code Analysis

What do you see? *Write down all the things you would change and prefix them with a number (add code line number to help).* *Exercise at page* 9

Class MovieStore

1. L6 & L7: wrong encapsulation.
   -> The field should be private with methods to access info.
2. L6 & L7: inconsistent and confusing names.
   -> Rename fields with a consistent naming convention
3. L10 & L11: dependency mock issue making unit testing hard.
   -> Inject the dependencies as parameters of the constructor.
4. L23: movie store logic tie to the console.
   -> Move the output logic to own class or injected method.
5. L32: Risk: we can add a movie with empty spaces as title?
   -> Add data verification.
6. L32: Confusing name: the name of the method does not say what it does (update copies and add movies). -> rename.
7. L64: Duplication: the research and check if a copy is available is duplicated in the buyMovie method -> extract method.
8. L88: Dead code? the method is never used, never tested
   -> If not use anywhere, remove the code, otherwise, test it.
9. L94: Risk: no early return if, say, title is an empty string
   -> Add a check and an early return. Add a test for it.
10. L97: Bug: the research by title is case-sensitive
    -> find another method of the string to compare. & a test!
11. Class MovieStore. SRP violation: the movie store has too many responsibilities
12. L20: Argument switching between customer and movie. Can lead to error passing arguments.
13. L28: error handling using the console is a bad practice.
    -> Use exception or better a monadic structure.
14. L32: Maintenance overhead: number of parameters is high.
    -> Passing a movie instead could work.

... We can move on to the other classes ...

To check the full list of smells, you can check our coding version with TODOs inside the code itself.

SCAN ME

Week 1 - Code Analysis
Exercise - Proposed Solution

2) Arrange by type

Sort out the actions by category.

Use the numbering from exercise 1 and try to put them in the following category. *Exercise at page* 10

Code Smell	Improvement	Vulnerability
1, 2, 3, 6, 8, 12	4, 7, 11, 13, 14	5, 9, 10

3) Own arrangement

Sort out the actions by your own categories.

Find your own way to arrange the actions and see any similarities between the two models. *Exercise at page* 10

Here we are choosing a model based on the priority of the task and the added-value the given task may bring.

High Risk / High Value	Needed changes	Too complex to do now
Can lead to a bug > require immediate actions!	A task that won't lead to a regression but will greatly help for future evolution.	Complex task for which we won't see the value now.
12, 9, 3, 5, 10	1, 7, 8, 2, 6, 14	4, 11, 13

Week 1 - Code Analysis
Game - Proposed Solution

Spot the Smells

Find the 6 smells in the code below. *How would you address each smell?* *Exercise page 11*

```
1  // Define an interface for the event callback
2  interface MessageReadListener {
3      void onMessageRead(String message);
4  }
5
6  @Getter
7  @Setter
8  @NoArgsConstructor
9  public class FileStore {
10     public String workingDirectory;
11     private MessageReadListener messageReadListener;
12
13     // Method to save the message into a file
14     public String save(int id, String message) throws IOException {
15         Path path = Paths.get(workingDirectory, id + ".txt");
16         Files.write(path, message.getBytes());
17         return path.toString();
18     }
19
20     // Method to read the message from a file and trigger the event
21     public void read(int id) throws IOException {
22         Path path = Paths.get(workingDirectory, id + ".txt");
23         String message = new String(Files.readAllBytes(path));
24         messageReadListener.onMessageRead(message);
25     }
26 }
27
```

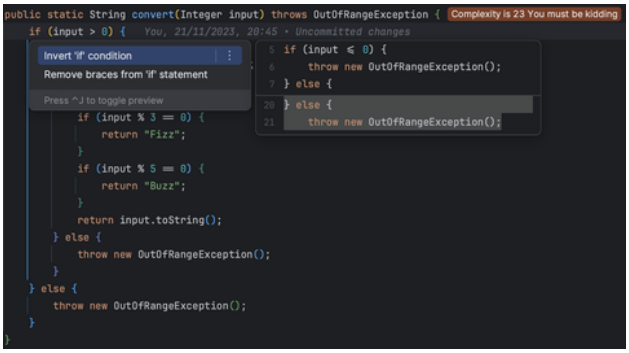
1. L10: wrong encapsulation -> The field should be private
2. L14: A save method returning String -> should return a void
3. L15 & 22: Code duplication -> A single method to get the file path with the directory
4. L16: The save method does more than one responsibility. -> Should be called saveContentToFile
5. L21: A read method returning a void -> should return data (String?)
6. L14 & 21: Confusing name. What does the parameter 'id' represent? -> Should be fileName perhaps.

Week 2 - Object Calisthenics
Exercise - Proposed Solution

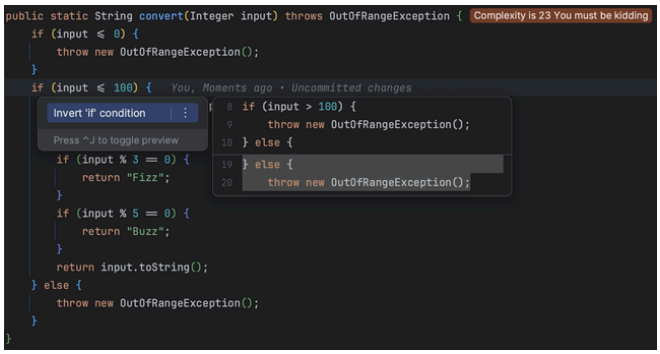
1) Fight indentation

An example of step by step solution for this exercise. *exercise page 15*

Start by
inverting condition



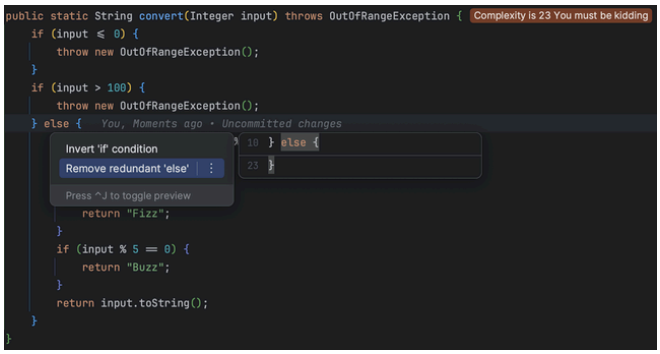
Apply the same ***safe*** refactoring for
the other ***if*** on input



Our code now looks like

```
public static String convert(Integer input) throws OutOfRangeException {
    if (input ⩽ 0) {
        throw new OutOfRangeException();
    }
    if (input > 100) {
        throw new OutOfRangeException();
    } else {
        if (input % 3 == 0 && input % 5 == 0) {
            return "FizzBuzz";
        }
        if (input % 3 == 0) {
            return "Fizz";
        }
        if (input % 5 == 0) {
            return "Buzz";
        }
        return input.toString();
    }
}
```

We can remove the redundant ***else***



We ***group the guards*** in the
same condition

```
public static String convert(Integer input) throws OutOfRangeException {
    if (input ⩽ 0 || input > 100) {
        throw new OutOfRangeException();
    }
    if (input % 3 == 0 && input % 5 == 0) {
        return "FizzBuzz";
    }
    if (input % 3 == 0) {
        return "Fizz";
    }
    if (input % 5 == 0) {
        return "Buzz";
    }
    return input.toString();
}
```

We now have only ***one level of indentation***.
We still have space for improvement...

```
public static String convert(Integer input) throws OutOfRangeException {
    // Encapsulate the condition
    if (input ⩽ 0 || input > 100) {
        throw new OutOfRangeException();
    }
    // Avoid magic numbers
    if (input % 3 == 0 && input % 5 == 0) {
        return "FizzBuzz";
    }
    // Modulo repeated everywhere
    if (input % 3 == 0) {
        return "Fizz";
    }
    if (input % 5 == 0) {
        return "Buzz";
    }
    return input.toString();
}
```

Week 2 - Object Calisthenics
Exercise - Proposed Solution

1) Fight indentation

Here is an ***"improved" version*** of it

```
public class FizzBuzz {
    public static final int MIN = 0;
    public static final int MAX = 100;
    public static final int FIZZ = 3;
    public static final int BUZZ = 5;
    public static final int FIZZBUZZ = 15;

    private FizzBuzz() {
    }

    public static String convert(Integer input) throws OutOfRangeException {
        if (isOutOfRange(input)) {
            throw new OutOfRangeException();
        }
        return convertSafely(input);
    }

    private static String convertSafely(Integer input) {
        if (is(FIZZBUZZ, input)) {
            return "FizzBuzz";
        }
        if (is(FIZZ, input)) {
            return "Fizz";
        }
        if (is(BUZZ, input)) {
            return "Buzz";
        }
        return input.toString();
    }

    private static boolean is(Integer divisor, Integer input) {
        return input % divisor == 0;
    }

    private static boolean isOutOfRange(Integer input) {
        return input <= MIN || input > MAX;
    }
}
```

Beyond: a little feedback

How can refactoring my methods to achieve one level of indentation per method streamline the ***readability and maintainability*** of my code?

.....

.....

.....

.....

.....

What strategies can I use to ***break down complex nested structures*** into smaller, more focused methods without sacrificing functionality?

.....

.....

.....

.....

.....

2) Identify the steps

Identify the Object Calisthenics steps in the code below? *Draw an arrow to where it is in the code that does not respect a step, mark which step it is and the action you would do.*

Exercise page 16

Example : **(6) rename variable to inventory**

(4) wrap inventory into its own class

(1) extract method to get book

(2) refactor method not to use else

```java
public class BookStore {
    private ArrayList<Book> inv = new ArrayList<>();

    public void addBook(String title, String author, int copies) {
        if (title != null && author != null && copies > 0) {
            Book foundBook = null;
            for (Book book : inv) {
                if (book.getTitle().equals(title)
                        && book.getAuthor().equals(author)) {
                    foundBook = book;
                    break;
                }
            }
            if (foundBook != null) {
                foundBook.addCopies(copies);
            } else {
                inv.add(new Book(title, author, copies));
            }
        }
    }

    public void sellBook(String title, String author, int copies) {
        for (Book book : inv) {
            if (book.getTitle().equals(title)
                    && book.getAuthor().equals(author)) {
                book.removeCopies(copies);
                if (book.getCopies() <= 0) {
                    inv.remove(book);
                }
                break;
            }
        }
    }
}
```

(5) refactor this check to avoid chaining

Week 2 - Object Calisthenics
Exercise - Proposed Solution

2) Link the steps

Identify the Object Calisthenics steps in the code below? *Draw an arrow to where it is in the code that does not respect a step, mark which step it is and the action you would do.* *Exercise page 16*

```java
class Book {
    private String title;
    private String author;
    private int copies;

    public Book(String title, String author, int copies) {
        this.title = title;
        this.author = author;
        this.copies = copies;
    }

    public void addCopies(int additionalCopies) {
        if (additionalCopies > 0) {
            this.copies += additionalCopies;
        }
    }

    public void removeCopies(int soldCopies) {
        if (soldCopies > 0 && this.copies >= soldCopies) {
            this.copies -= soldCopies;
        }
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getCopies() {
        return copies;
    }
}
```

(9) Using better getters than just exposing each private property. Here something to say if there are any copies instead. boolean hasCopies()

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 2 - Object Calisthenics
Game - Proposed Solution

Complete the sentence

Complete the sentence with the right answer.

Describe which principle it is linked to. *Exercise page 18*

"Any class that contains a collection should contain no other member. Each collection gets wrapped in its own class, so now behaviors related to the collection have a home. You may find that filters become a part of this new class."

[First Class Collection]

"Whenever you have a primitive type or a string that has some business meaning, encapsulate it within a class to ensure it is not just a raw data but represents a meaningful piece of information."

[Wrap All Primitives]

"To promote greater modularity and focus within classes, ensure that no class has more than two invariants, which encourages classes to be more cohesive."

[No Class with More than 2 Invariants]

Week 3 - Command Query Separation

Exercise - Proposed Solution

1) Fix the issue

Identify the issue and its consequences

Please read the code well and try to understand the issue and the underlying bad consequences such a code can have. *Exercise page 22*

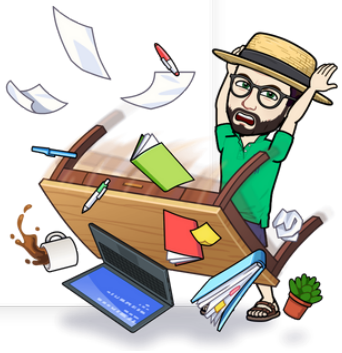
```java
public class Client {
    private final Map<String, Double> orderLines;
    private double totalAmount;

    public Client(Map<String, Double> orderLines) {
        this.orderLines = orderLines;
    }

    public String toStatement() {
        return orderLines.entrySet().stream()
                .map(entry -> formatLine(entry.getKey(), entry.getValue()))
                .collect(Collectors.joining(System.lineSeparator()))
                .concat(System.lineSeparator() + "Total : " + totalAmount + "€");
    }

    private String formatLine(String name, Double value) {
        totalAmount += value;
        return name + " for " + value + "€";
    }

    public double getTotalAmount() {
        return totalAmount;
    }
}
```



Describe the main issue and what changes you would make below.

The toStatement implementation is fishy. It does two things: return the line and update amount

- Each time we call the toStatement method it will mutate the internal state of the object...
- It will update the totalAmount meaning that the second time we call the method the result will be wrong

One way to resolve this is to have a method to get the total amount and remove the assigning of the value.
We will have to adapt all the places where the formatLine is used to get the amount.
Then we can add queries to format and return the lines and another query to get specific info like amount.

You can check the full coding version for the solution of the exercise. Scan the QR code or click on the link here.

SCAN ME

Week 3 - Command Query Separation

Exercise - Proposed Solution

2) Double edge method

Some methods can be both query & command and are especially hard to separate. Analyse the code below and see how you can refactor it to show the real intention.

Exercise page 23

```java
public class Cache {
    private Map<String, Integer> map = new HashMap<String, Integer>();

    public int getOrInsert(String key, int value) {
        if (!map.containsKey(key)) {
            map.put(key, value);
        }
        return map.get(key); // Modifies state if key not present and returns value
    }
}
```

Try to write your unit tests and see the changes you want to emerge from them.

The issue with the unit tests here is that to test everything, you will need to write a confusing inputs and mocks. It makes it confusing to call this method.

One way to address it is to break it down with different method and say exactly what the command is doing --> ensurePresentAndGet(key, value) and command insertIfAbsent(key, value) both publics

Another method get(key) can return null if you don't have any value present.

That way, each method does exactly what it says and no confusion can come of it.

You can check the full coding version for the solution of the exercise. Scan the QR code or click on the link here.

SCAN ME

Week 3 - Command Query Separation
Game - Proposed Solution

Command or Query ?

Guess the right answer!

Exercise page 24

```
public class UniqueCollection {
    private Set<Integer> elements = new HashSet<>();

    public boolean addIfMissing(int element) {
        return elements.add(element); // Attempts to add an element if it's not already present
    }
}
```

Looks like we are just adding an element. The boolean is just indicating us if it worked.

?

- [x] Command
- [ ] Query

?

- [ ] Command
- [x] Query

```
public class History {
    private LinkedList<String> events = new LinkedList<>();

    public String getLast() {
        return events.getLast(); // Retrieves the last event without removing it
    }
}
```

Just a simple read inside a collection.

```
public class LightSwitch {
    private boolean isOn = false;

    public boolean toggleStatus() {
        isOn = !isOn; // Toggles the state of the light switch
        return isOn; // Also returns the new state
    }
}
```

We are changing the state of the switch as well as returning it. The resulting boolean is the new state.

?

- [x] Command
- [ ] Query

?

- [ ] Command
- [x] Query

```
public class NetworkService {
    private String serviceUrl; //injected in constructor

    public ConnectionStatus checkConnectivity() {
        try {
            URL url = new URL(serviceUrl);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("HEAD");
            int responseCode = connection.getResponseCode();
            if (responseCode >= 200 && responseCode < 400) {
                return new ConnectionStatus(true, "Service is reachable");
            } else {
                return new ConnectionStatus(false, "Service returned error code: " + responseCode);
            }
        } catch (IOException e) {
            return new ConnectionStatus(false, "Network error: " + e.getMessage());
        }
    }

    public static class ConnectionStatus {
        private boolean isSuccess;
        private String message;
    }
}
```

While this code does a lot of checks, it does not change internal states.

Week 4 - TDD
Exercise - Proposed Solution

Exercise 1 : Password validation

Exercise page 29

Design a simple program. Using TDD, craft a simple program to implement a password validation program using the following rules:

- Contains at least 8 characters
- Contains at least one capital letter
- Contains at least one lowercase letter
- Contains at least a number
- Contains at least a special character in this list . * # @ $ % &.
- Any other characters are not authorized.

Unit tests examples

- A valid password first -> 'P@ssw0rd'
- A password FROM the valid password that breaks one rule only -> 'P@ssw0r' [it does not contain 8 characters]
- A password FROM the valid password that breaks one rule only -> 'P@ssword' [it does not contain a number]
- I continue with each rules until the very last rule.
- I use a password with an invalid character -> 'P@ssword^'

Implementation Steps

- Create a PasswordValidator class with a validate method that returns true by default (any invalid combinations will be checked separately)
- returns true if the password contains at least 8 characters otherwise false (to pass all tests)
- returns true if the password contains at least 8 characters and a lower letter otherwise false.
- I create private methods for each separate rules at the refactor phase..
- returns true if the password check all the rules except the last one otherwise false..
- I can refactor my tests to use parameterized tests.
- I add a rule to check invalid characters
- I refactor my code to have the invalid characters as a static set of characters in my class.
- I try to simplify my validation public method and have as much simple methods as possible.

You can check the full coding version for the solution of the exercise. Scan the QR code or click on the link here.

SCAN ME

Week 4 - TDD
Game - Proposed Solution

The right path

Exercise page 32

Scrambled steps. Arrange the TDD steps in the correct sequence to successfully develop the calculator functionality using TDD principles. Some steps can be repeated.

Below are the steps involved in developing a **calculator** using TDD. They are currently in the wrong order. Your task is to reorder them to reflect the correct TDD cycle. Write down the number sequence that you think correctly represents the TDD process from start to finish.

Steps	Order
A. Refactor the code to simplify and improve efficiency without changing its behavior.	5
B. Run the test and see it fail (Red phase).	2, 7, 12
C. Write a test for a subtraction feature.	6
D. Implement the addition feature to pass the test.	3
E. Write a test for the addition feature.	1
F. Implement the subtraction feature to pass the test.	8
G. Run the test and see it pass (Green phase).	4, 9, 14
H. Refactor the subtraction implementation to handle edge cases.	15
I. Review all tests to ensure coverage and clarity.	16
J. Write tests for multiplication and division features.	10
K. Implement the multiplication and division features to pass their tests.	13

Week 5 - Accidental complexity
Exercise - Proposed Solution

1) Do not use if... else

Get rid of conditionals in the code below. From the techniques described in the previous section, try to get rid of the conditionals in this code to prevent complexity. *Exercise page 36*

```java
public static String convert(Integer input) throws OutOfRangeException {
    if (isOutOfRange(input)) {
        throw new OutOfRangeException();
    }
    return convertSafely(input);
}

private static String convertSafely(Integer input) {
    if (is(FIZZBUZZ, input)) {
        return "FizzBuzz";
    }
    if (is(FIZZ, input)) {
        return "Fizz";
    }
    if (is(BUZZ, input)) {
        return "Buzz";
    }
    return input.toString();
}
```

Describe which techniques you would use and where.

Here a technic you can use is a Functional Map to avoid the multiple conditionals.

- It would have a signature like this : Predicate<Integer>, Function<Integer, String> -> String
- It contains a Predicate to check if the input ensures it and the corresponding value function in case of a match

You would then call for the filter the transformation of the string.

The map could be static so you can also address the performance with this approach.

You can now safely remove the convertSafely method by adding the OutOfRange as a filter.

note: In the future, the logic can be outside the fizzbuzz class if needs be..

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 5 - Accidental complexity
Exercise - Proposed Solution

2) Above and beyond conditionals

Exercise page 37

Optimize the code below. We want to remove all conditionals and make a few adjustments for the future. How easy it is to add a new command? How would you tackle the TODOs?
Remember, we want to avoid complexity. Other things might be missing...

```java
public interface Command {
    void execute();
}

public class CommandProcessor {
    private Map<String, Command> commandMap;

    public CommandProcessor() {
        commandMap = new HashMap<>();
        commandMap.put("greet", () → System.out.println("Hello, World!"));
        commandMap.put("exit", () → System.out.println("Exiting application..."));
    }

    public void processCommand(String command) {
        if (commandMap.containsKey(command)) {
            commandMap.get(command).execute();
        } else {
            System.out.println("Unknown command");
        }
    }

    public static void main(String[] args) {
        CommandProcessor cp = new CommandProcessor();

        // TODO: Should be able to pass my name to the command to say Hello to me!
        cp.processCommand("greet");  // Outputs: Hello, World!
        cp.processCommand("exit");   // Outputs: Exiting application...
        // TODO: Should display all commands available
        cp.processCommand("help");    // Outputs: Unknown command
    }
}
```

Describe here how you would do it.

Here to add a new command, you need to modify this class to add to the map. While it's better than a switch case, it does not make our code open for extension as in the O of SOLID.
Besides, our CommandProcessor holds all the commands logic and the dependency to the console.
Creating an implementation for each command and using polymorphism would help remove that.
As well, it would add a flexibility to add first and lastname and other specific logic..
While you are at it, add some tests!

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 5 - Accidental complexity
Game - Proposed Solution

Detect the pattern

Exercise page 38

Find the design pattern in this code. In the code below, a pattern has been used to reduce complexity, would you be able to find which one?

```
public class TaskExecutor {
    private Task task;

    public TaskExecutor(Task task){
        this.task = task;
    }

    public void executeTask(){
        task.performTask();
    }
}

interface Task {
    void performTask();
}

class OperationA implements Task {
    public void performTask() {
        System.out.println("Executing Operation A");
    }
}

class OperationB implements Task {
    public void performTask() {
        System.out.println("Executing Operation B");
    }
}
```

?

- [ ] Command
- [ ] Observer
- [x] Strategy
- [ ] Singleton

Would you suggest another pattern instead? If so, which one and why?

An alternative design pattern could be the Command pattern. By encapsulating each operation in its command class, we can simplify the TaskInvoker management of execution, enhance flexibility, and make it easier to introduce new commands without altering existing code.

Week 6 - Legacy Code
Exercise - Proposed Solution

1) Put the code under tests

Exercise page 43

Try the approval testing approach. Add tests for the code below. You can use the Golden Master technique with an expected file.

```java
public enum DocumentTemplateType {
    DEERPP("DEER", RecordType.INDIVIDUAL_PROSPECT),
    DEERPM("DEER", RecordType.LEGAL_PROSPECT),
    AUTP("AUTP", RecordType.INDIVIDUAL_PROSPECT),
    AUTM("AUTM", RecordType.LEGAL_PROSPECT),
    SPEC("SPEC", RecordType.ALL),
    GLPP("GLPP", RecordType.INDIVIDUAL_PROSPECT),
    GLPM("GLPM", RecordType.LEGAL_PROSPECT);

    private final String documentType;
    private final RecordType recordType;

    DocumentTemplateType(String documentType, RecordType recordType) {
        this.documentType = documentType;
        this.recordType = recordType;
    }

    public static DocumentTemplateType fromDocumentTypeAndRecordType(String documentType, String recordType) {
        for (DocumentTemplateType dtt : DocumentTemplateType.values()) {
            if (dtt.getDocumentType().equalsIgnoreCase(documentType)
                    && dtt.getRecordType().equals(RecordType.valueOf(recordType))) {
                return dtt;
            } else if (dtt.getDocumentType().equalsIgnoreCase(documentType)
                    && dtt.getRecordType().equals(ALL)) {
                return dtt;
            }
        }
        throw new IllegalArgumentException("Invalid Document template type or record type");
    }

    private RecordType getRecordType() {
        return recordType;
    }

    private String getDocumentType() {
        return documentType;
    }
}
```

Write down the combinations you would add to your golden master file.

In this code, to add combinations to your Golden Master, we need to add every single combination of documentType and recordType.

The approval library allows us to use the VerifyAllCombinations method that would help us do that.

```
CombinationApprovals.verifyAllCombinations(
DocumentTemplateType::fromDocumentTypeAndRecordType,
new String[]{"AUTP", "AUTM", "DEERPP", "DEERPM", "SPEC", "GLPP", "GLPM"},
new String[]{"INDIVIDUAL_PROSPECT", "LEGAL_PROSPECT", "ALL"});
```

Then, making the test dynamic with enum values would be enough.

For refactoring, a couple of steps:
1) Create a map of string with the DocumentTemplateType
2) a query to get the specific key
3) a simple query to get the documentTemplateType

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 6 - Legacy Code
Exercise - Proposed Solution

2) What's wrong here

Exercise page 44

Could you explain what's incorrect about the implementation below? Explain what is wrong with this legacy code and why.

The main issue in this legacy code is that the ReportGenerator contains all the logic to render a report. As well as for all types which ends up in a complex if ... else.

It also has a hard dependency on the console for the output.
It makes the following code very difficult to test and change.

```java
class ReportGenerator {
    public void generateReport(String reportType, List<ReportData> data) {
        if ("CSV".equals(reportType)) {
            System.out.println("Starting CSV Report Generation...");
            System.out.println("CSV Header: ID, Value, Description");
            for (ReportData d : data) {
                System.out.println(d.getId() + "," + d.getValue() + "," + d.getDescription());
            }
            System.out.println("CSV Report Generated Successfully.");
        } else if ("PDF".equals(reportType)) {
            System.out.println("Starting PDF Report Generation...");
            System.out.println("PDF Report Title: Comprehensive Data Report");
            System.out.println("---------------------------------------------");
            for (ReportData d : data) {
                System.out.println("Data ID: " + d.getId() + " | " + "Data Value: " + d.getValue() + " | " + "Description: " + d.getDescrip
            }
            System.out.println("---------------------------------------------");
            System.out.println("PDF Report Generated Successfully.");
        } else {
            System.out.println("Report type " + reportType + " not supported.");
        }
    }
}

```

Highlight all the smells you can find in the code and provide a solution to how you would fix them.

```java
public class ReportData {
    private int id;
    private double value;
    private String description;

    public ReportData(int id, double value, String description) {
        this.id = id;
        this.value = value;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public double getValue() {
        return value;
    }

    public String getDescription() {
        return description;
    }
}
```

The reportGenerator class should only render a report for a specific type.
Each report type will have its own logic via inheritance.

BEFORE we do that though, we need to add the appropriate tests. Let's use the approval testing again for each report type. We will have two files for it (CSV and PDF).

To avoid the if, a functional map will be added and a error case will be handle in a monad structure. Finally, the output will be handled by the Main class and not the ReportGenerator.
At the end we can convert the ReportData into a record.

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 6 - Legacy Code Game - Proposed Solution

Smell to score

Exercise page 46

Using the grid of score below, reach the minimum score. For each code snippet, try to detect as many smells as possible and a potential fix.

Smell score

remove bug / risk (6), extract / correct dependency (5), increase domain language (4), simplify code (3), fix naming (2), adjust readability (1)

Smell	Type	Line #	Potential fix	Score
wrong variable name	2	5	rename item name for 'order'	2
magic number for status	4	8	Replace integer constants with an enum called Status with values UNPROCESSED and PROCESSED	4
no null check for order	6	7	Add a null check at the beginning to check if order is null. Add an exception or return a monadic structure.	6
processing order and discount logic in the same place	3	9	extract the logic of applying the discount in a separate method	3
Loop to process orders makes code complicated	1	7	Replace for loop with collection iteration logic	1
10% discount for order with 5 or more items with magic numbers	4	9 & 11	Add named constants for magic numbers.	4
item name for each order can be confusing	1	7	Rename item by order	1
Order should not expose a setter for the total	6	32	Replace the accessibility of the setTotal method	6
Order should have the logic to change its total if there is a discount. Not the orderProcessor	5	13	Move the logic to the Order class and also move the magic numbers and the assignment of the status. All accessors are now private.	5
There is no information on what orders have been processed	5	6	Add a monadic structure with a list of order that have been processed. We can always add an error code later if the process fails.	5

Use the coding version with tests to check if you removed a risk and check how to address a specific smell.

Total score = 37

Week 7 - Property-Based Testing
Exercise - Proposed Solution

1) Thousand tests in one

Refactor the Fizzbuzz tests from parameterized tests to property-based tests. Look at the tests and implementation to see how you can extract common behavior and see if you can find properties. *Exercise page 50*

```java
public class FizzBuzz {
    public static final int MIN = 1;
    public static final int MAX = 100;
    private static final Map<Integer, String> mapping = LinkedHashMap.of(
            15, "FizzBuzz",
            3, "Fizz",
            5, "Buzz"
    );

    public static Option<String> convert(int input) {
        return isOutOfRange(input)
                ? none()
                : some(convertSafely(input));
    }

    private static String convertSafely(Integer input) {
        return mapping
                .find(p -> is(p._1, input))
                .map(p -> p._2)
                .getOrElse(input.toString());
    }

    private static boolean is(Integer divisor, Integer input) {
        return input % divisor == 0;
    }

    private static boolean isOutOfRange(Integer input) {
        return input < MIN || input > MAX;
    }
}
```

```java
class FizzBuzzTests {
    private static Stream<Arguments> validInputs() {
        return Stream.of(
                Arguments.of(1, "1"),
                Arguments.of(67, "67"),
                Arguments.of(82, "82"),
                Arguments.of(3, "Fizz"),
                Arguments.of(66, "Fizz"),
                Arguments.of(99, "Fizz"),
                Arguments.of(5, "Buzz"),
                Arguments.of(50, "Buzz"),
                Arguments.of(85, "Buzz"),
                Arguments.of(15, "FizzBuzz"),
                Arguments.of(30, "FizzBuzz"),
                Arguments.of(45, "FizzBuzz")
        );
    }

    private static Stream<Arguments> invalidInputs() {
        return Stream.of(
                Arguments.of(0),
                Arguments.of(-1),
                Arguments.of(101)
        );
    }

    @ParameterizedTest
    @MethodSource("validInputs")
    void parse_successfully_numbers_between_1_and_100(int input, String expectedResult) {
        assertThat(FizzBuzz.convert(input))
                .isEqualTo(Some(expectedResult));
    }

    @ParameterizedTest
    @MethodSource("invalidInputs")
    void parse_fail_for_numbers_out_of_range(int input) {
        assertThat(FizzBuzz.convert(input).isEmpty())
                .isTrue();
    }
}
```

Write down the properties you see from the snippet of code. Refactor the tests using PBT.

We can see 2 properties here:
for all (invalidInput) --> such that convert(invalidInput) is none
for all (validInput) -->
such that convert(validInput) is a string looking matching one item from the list: validInput, "Fizz", "Buzz", "FizzBuzz"
When refactoring we need to do one property at the time. Usually the easiest first. Here the invalid one. Then we iterate on the test.
While writing properties, we try with the wider range and we tackle the errors with example based testing.

You can go further and get the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 7 - Property-Based Testing
Game - Proposed Solution

Find the property

Exercise page 51

Identify the valid properties for the code samples. Look at the code snippets below and find out what properties are valid in the proposed list on the right.

1) Testing a Maximum Function

INVALID ANSWERS
Non-Negativity: Invalid unless a and b are non-negative
Additive Property: Invalid, not necessarily true

```
public int max(int a, int b) {
    return a > b ? a : b;
}
```

- [x] **Commutativity**: max(a, b) should equal max(b, a).
- [x] **Idempotence**: max(a, a) should return a.
- [ ] **Non-Negativity**: max(a, b) should not be negative.
- [x] **Greater or Equal**: max(a, b) is always greater than or equal to a and b.
- [ ] **Additive Property**: max(a+1, b) is greater than max(a, b).

2) Testing an Array Sorting Function

INVALID ANSWERS
Length Alteration: Invalid, sorting does not change array length

```
public int[] sort(int[] array) {
    Arrays.sort(array);
    return array;
}
```

- [x] **Idempotence**: Applying sort twice results in the same output as applying it once.
- [x] **Element Preservation**: The sorted array contains exactly the same elements as the input array.
- [x] **Non-Decreasing Order**: Every element in the array is less than or equal to the next element.
- [ ] **Length Alteration**: The length of the sorted array is different from the input array.
- [ ] **First Element**: The first element is always the smallest in the array.

3) Testing a Method that Trims Whitespace

INVALID ANSWERS
Length Reduction: Invalid, not true if input has no leading/trailing whitespace
Whitespace Preservation: Irrelevant for property, as trim() affects only leading/trailing whitespace
Non-Whitespace Characters: Invalid, not a direct concern of trimming function

```
public String trimString(String input) {
    return input.trim();
}
```

- [x] **Idempotence:** Applying trim twice results in the same output as applying it once.
- [ ] **Length Reduction:** The trimmed string is always shorter than the input.
- [x] **Empty String Result:** Trimming an empty string results in an empty string.
- [ ] **Whitespace Preservation:** Inner whitespace is preserved.
- [ ] **Non-Whitespace Characters:** Non-whitespace characters are never removed.

You can go further and write the tests using the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 7 - Property-Based Testing
Game - Proposed Solution

2) Find the property

Exercise page 52

Identify the valid properties for the code samples. Look at the code snippets below and find out what properties are valid in the proposed list on the right.

INVALID ANSWERS
Size Reduction: Invalid, not true if all are even
Non-Duplication: Invalid, filtering doesn't affect duplicates
Non-Negativity: Invalid unless input guarantees it

4) Testing a Simple Array Filter Function

```java
public String trimString(String input) {
    return input.trim();
}
```

- [ ] **Size Reduction:** The output array is always smaller than the input array.
- [x] **Element Consistency:** All elements in the output array are even.
- [ ] **Non-Duplication:** The output array contains no duplicate elements.
- [ ] **Non-Negativity:** The output array contains only non-negative numbers.
- [x] **Order Preservation:** The order of elements in the output array is the same as their order in the input array.

5) Testing an Asynchronous Data Loading Method

INVALID ANSWERS
Consistency: Invalid, can fail if data changes or due to transient failures
Response Time: Invalid, system load or external delays may affect this

```java
public CompletableFuture<String> loadDataAsync(String url) {
    return CompletableFuture.supplyAsync(() -> {
        try {
            // Simulated delay
            Thread.sleep(1000);
            return "Data from " + url;
        } catch (InterruptedException e) {
            return "Failed to load";
        }
    });
}
```

- [x] **Non-Null Result:** The CompletableFuture always completes with a non-null value.
- [ ] **Consistency:** The same URL always results in the same data.
- [ ] **Response Time:** The CompletableFuture completes within 1 second.
- [x] **Error Handling:** Any interruption results in a "Failed to load" message.
- [x] **Thread Safety:** The method can be safely called from multiple threads simultaneously.

6) Testing JSON Serialization

INVALID ANSWERS
Exception Safety: Invalid, Gson handles most types and their exceptions are not typical
Format Validity: True, but generally assumed and not directly testable as a property

```java
public String serializeToJson(Object data) {
    return new Gson().toJson(data);
}
```

- [x] **Reversibility:** Deserializing the serialized JSON string yields an object equal to the original.
- [x] **Consistency:** Serializing the same object multiple times produces the same JSON string.
- [x] **Null Handling:** Serializing a null object returns "null".
- [ ] **Exception Safety:** Serialization throws an exception if the object contains non-serializable types.
- [ ] **Format Validity:** The output is always a valid JSON format.

You can go further and write the tests using the coding version of this exercise by scanning the QR code here.

SCAN ME

Week 8 - Recap
Game - Proposed Solution

Crossword

Exercise page 56

	1 S	C	R	U	2 M				3 S				4 B				6 D	
	M				U				I				A				D	
5 I	D	E			T				N				B				D	
	L				A				G				Y				D	
	L				7 T	A	U	T	O	L	O	G	S	O	L	I	D	
					I				E				T					
					O				T				E					
	9 I	N	D	E	N	T	A	10 T	I	O	N		P					
								O		N								
								D										
11 A	B	S	T	R	A	C	T	I	O	N								

Across

1. Agile Framework
5. Where the magic happens
7. if (true)
8. Set of good engineering principles
9. Level of danger
11. Allow mocking

Down

1. A bad code
2. Testing tests
3. One instance
4. TDD's core concept
6. A modeling centered around the business
10. I will do it later

Week 8 - Recap
Game - Proposed Solution

Find the hidden sentence

Find all the words that composes the hidden sentence. Clues are located throughout the entire book.

Exercise page 57

“ OUR JOB IS TO BE

STRICT WITH OUR CODE

BUT OPEN - MINDED

WITH THE ONES WHO

DESIGN IT .”

1	Belongs to us	7	W/	13	Same as #7
2	A program we can test with approval testing	8	Same as #1	11	1st word of week 1 tip
3	Word #5 at the 3rd person	9	What improves with the Boy Scout Rule	12	More than the one
4	1st word, 3rd paragraph, week 2 game	10	!!!	10	Anagram of how
5	The 1st letter of the first two concepts of TDD	11	A word in the principle in week 6 exercise 2	11	TDD is about better .. ?
6	Contrary of loose	12	Inclined	12	Our industry

Week 8 - Recap
Page of notes

Personal notes

Week 8 - Recap
Page of notes

Personal notes

Week 8 - Recap
Page of notes

Personal notes

Week 8 - Recap
Page of notes

Personal notes

Week 8 - Recap
Page of notes

Personal notes

Week 8 - Recap
A final word

Reflect on your vacation

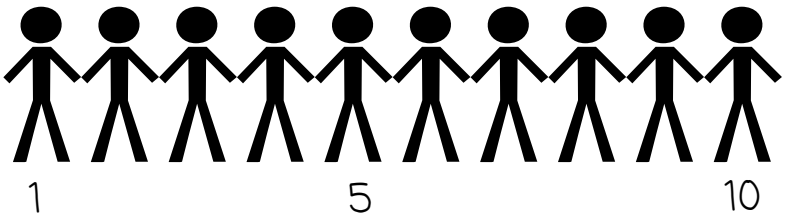
In which way your experience in this year's Summer Craft Book met your expectations? (reflect on your expectations from page 6)

.....

.....

.....

On a scale from 1 to 10, how likely are you to recommend our Summer Craft Book to a friend or colleague? (From ***1: not at all likely to 10: Extremely likely***)



What is your favorite moment, exercise or game of the Summer Craft Book?

.....

.....

.....

.....

.....

Would you have any suggestion for next year's edition?

.....

.....

.....

.....

Summer Craft Book

2024 edition

THANK YOU

See you next summer...