4developers
GDAŃSK

Paweł Kierat

EPAM Systems

Efektywne generowanie kodu przy użyciu ChatGPT

26.09.2023

@ pawel@kierat.dev

kierat.dev

/pkierat

in /in/kierat

About ChatGPT

- Series of machine learning models
- Developed by OpenAI
- Trained on large data sets
- Goal: Generate consistent, natural text

Paweł Kierat <pawel@kierat.dev>

About ChatGPT

- Series of machine learning models
- Developed by OpenAI
- Trained on large data sets
- Goal: Generate consistent, natural text

Examples of applications

- Producing answers to queries
- Generating creative content based on user’s requirements

Paweł Kierat <pawel@kierat.dev>

About ChatGPT

- Series of machine learning models
- Developed by OpenAI
- Trained on large data sets
- Goal: Generate consistent, natural text

Examples of applications

- Producing answers to queries
- Generating creative content based on user’s requirements
- **Generating complete code in variety of programming languages!** 😉

Paweł Kierat <pawel@kierat.dev>

Known issues with ChatGPT

🔥 The code doesn’t compile or doesn’t work correctly.

Known issues with ChatGPT

- The code doesn’t compile or doesn’t work correctly.
- The code is incomplete (happy path).

Paweł Kierat <pawel@kierat.dev>

Known issues with ChatGPT

- The code doesn’t compile or doesn’t work correctly.
- The code is incomplete (happy path).
- The code is of low quality (unmaintainable).

Known issues with ChatGPT

- The code doesn’t compile or doesn’t work correctly.
- The code is incomplete (happy path).
- The code is of low quality (unmaintainable).
- The code is based on solutions taken from the Internet.

Paweł Kierat <pawel@kierat.dev>

Known issues with ChatGPT

- The code doesn’t compile or doesn’t work correctly.
- The code is incomplete (happy path).
- The code is of low quality (unmaintainable).
- The code is based on solutions taken from the Internet.
- Problems with memory and concentration.

Paweł Kierat <*pawel@kierat.dev*>

Known issues with ChatGPT

- The code doesn’t compile or doesn’t work correctly.
- The code is incomplete (happy path).
- The code is of low quality (unmaintainable).
- The code is based on solutions taken from the Internet.
- Problems with memory and concentration.
- Rewriting everything from scratch.

Paweł Kierat <*pawel@kierat.dev*>

Known issues with ~~ChatGPT~~ developers

- The code doesn’t compile or doesn’t work correctly.
- The code is incomplete (happy path).
- The code is of low quality (unmaintainable).
- The code is based on solutions taken from the Internet.
- Problems with memory and concentration.
- Rewriting everything from scratch.

Paweł Kierat <*pawel@kierat.dev*>

Assumptions for experiments

- Code written almost exclusively by AI
- Work in **conversational** mode
- Human operator:
    - provides functional requirements
    - defines project structure, libraries, tools
    - sets the direction
    - verifies the correctness
    - gives hints and corrects

Experiment 1: Coding to requirements

Problem

The code is incomplete (happy path)

Experiment 1: Coding to requirements

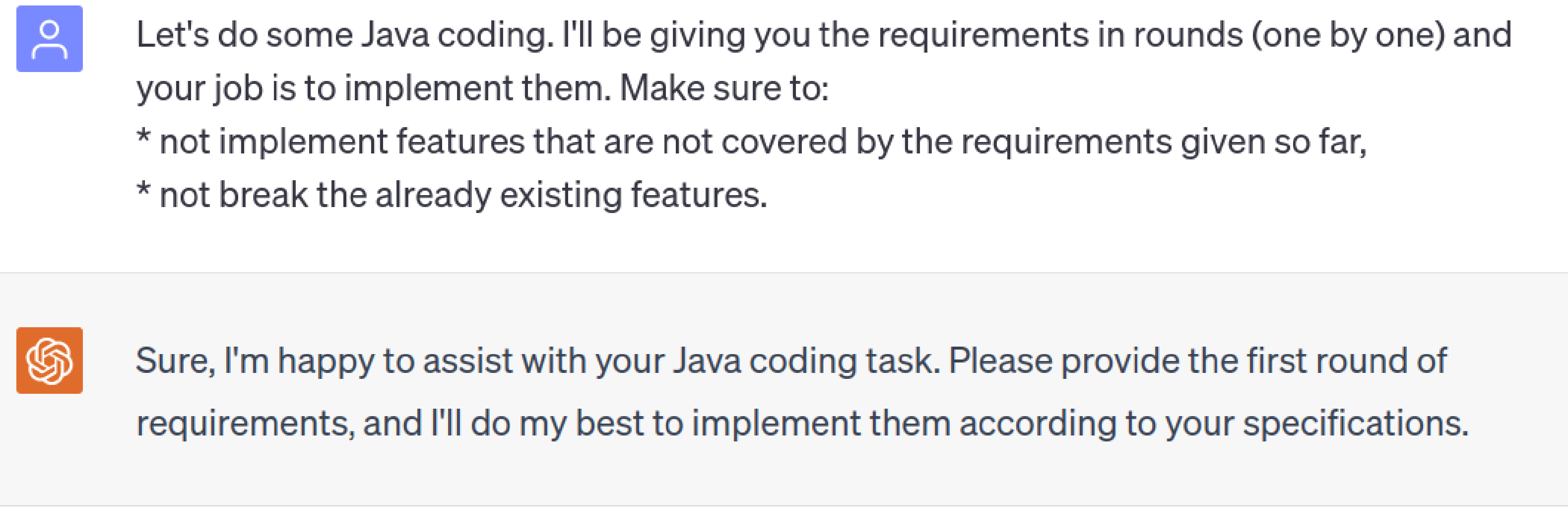
Problem

The code is incomplete (happy path)

Solution

Work in iterations

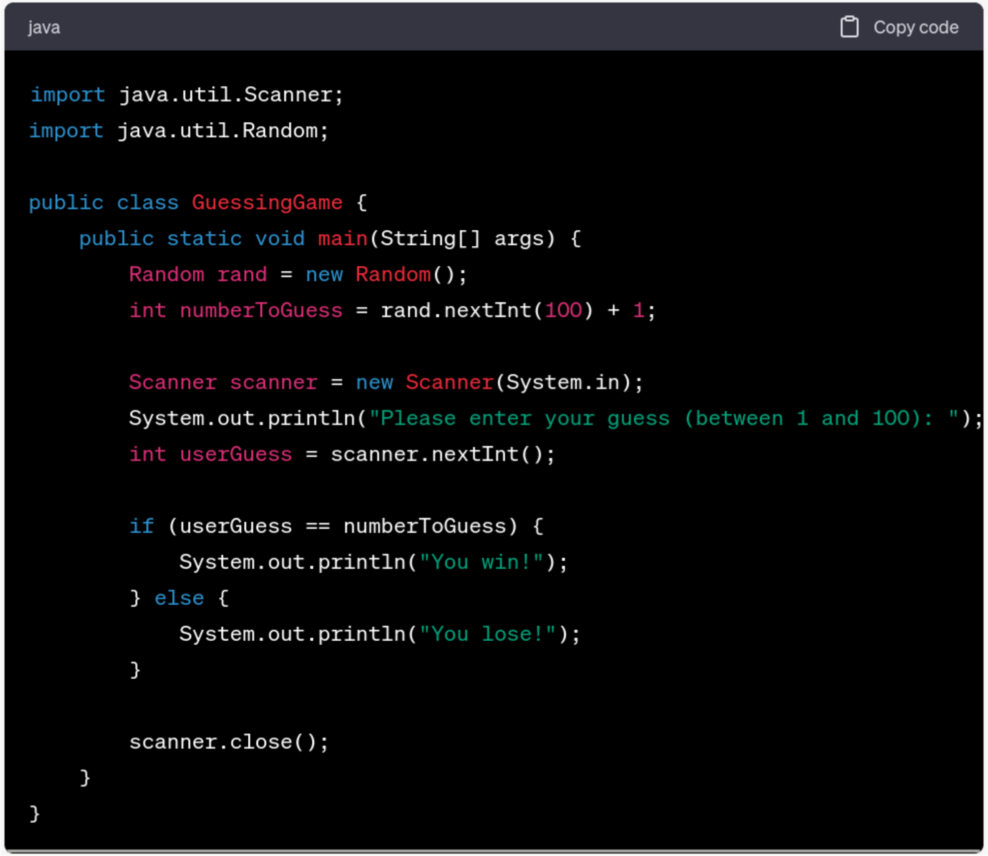
Paweł Kierat <pawel@kierat.dev>



Let's do some Java coding. I'll be giving you the requirements in rounds (one by one) and your job is to implement them. Make sure to:
* not implement features that are not covered by the requirements given so far,
* not break the already existing features.

Sure, I'm happy to assist with your Java coding task. Please provide the first round of requirements, and I'll do my best to implement them according to your specifications.

https://chat.openai.com/share/cd4f7bea-a783-43f1-b141-38670d261268



```java
import java.util.Scanner;
import java.util.Random;

public class GuessingGame {
    public static void main(String[] args) {
        Random rand = new Random();
        int numberToGuess = rand.nextInt(100) + 1;

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter your guess (between 1 and 100): ");
        int userGuess = scanner.nextInt();

        if (userGuess == numberToGuess) {
            System.out.println("You win!");
        } else {
            System.out.println("You lose!");
        }

        scanner.close();
    }
}
```

Result

- Everything in just 15 steps (10 requirements),
- Works according to the requirements.

But…

Result

- Everything in just 15 steps (10 requirements),
- Works according to the requirements.

But…

- Not always according to common sense.

Moreover…

Result

- Everything in just 15 steps (10 requirements),
- Works according to the requirements.

But…

- Not always according to common sense.

Moreover…

- 79 lines of code,
- All in one file, one class, one method!
- Cyclomatic complexity: 36! (a.k.a. *Bloody hell!*)

Paweł Kierat <pawel@kierat.dev>

Experiment 2: Test cases

Problem

🔥 Problems with memory and concentration.

Paweł Kierat <*pawel@kierat.dev*>

Experiment 2: Test cases

Problem

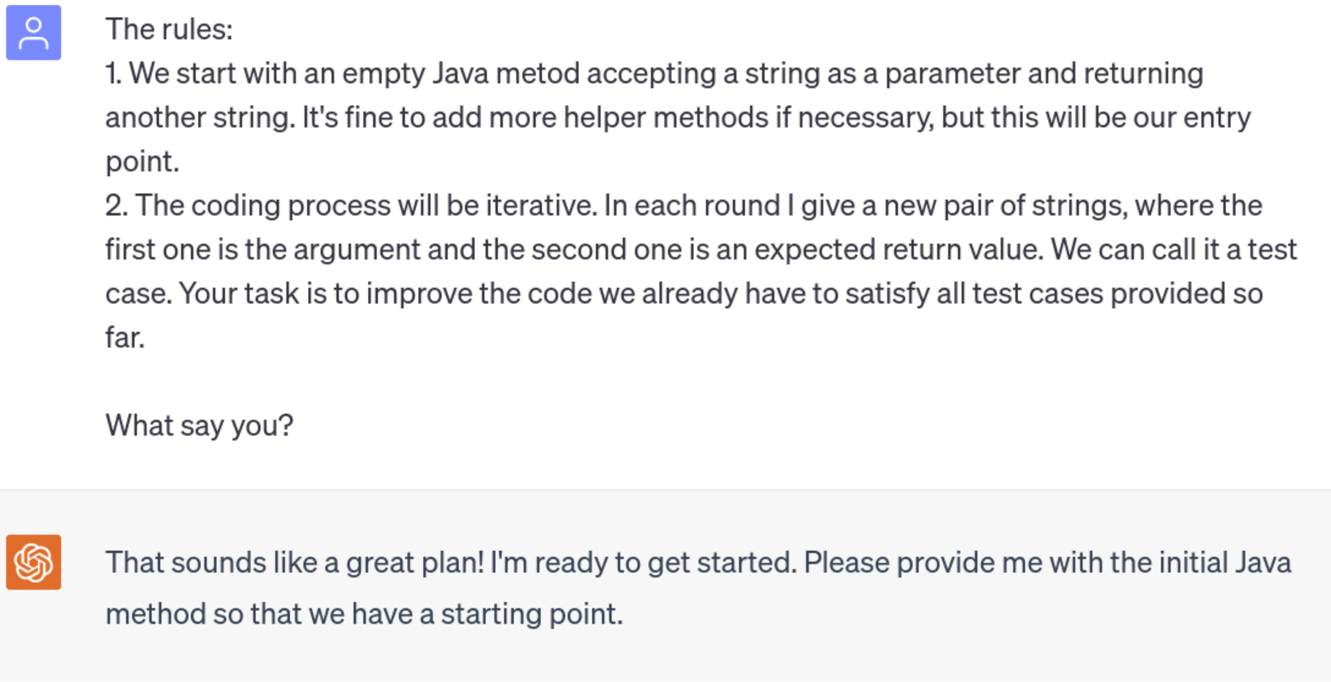
🔥 Problems with memory and concentration.

Solution

💡 Work in iterations

💡 Discipline and repetition!

Paweł Kierat <pawel@kierat.dev>

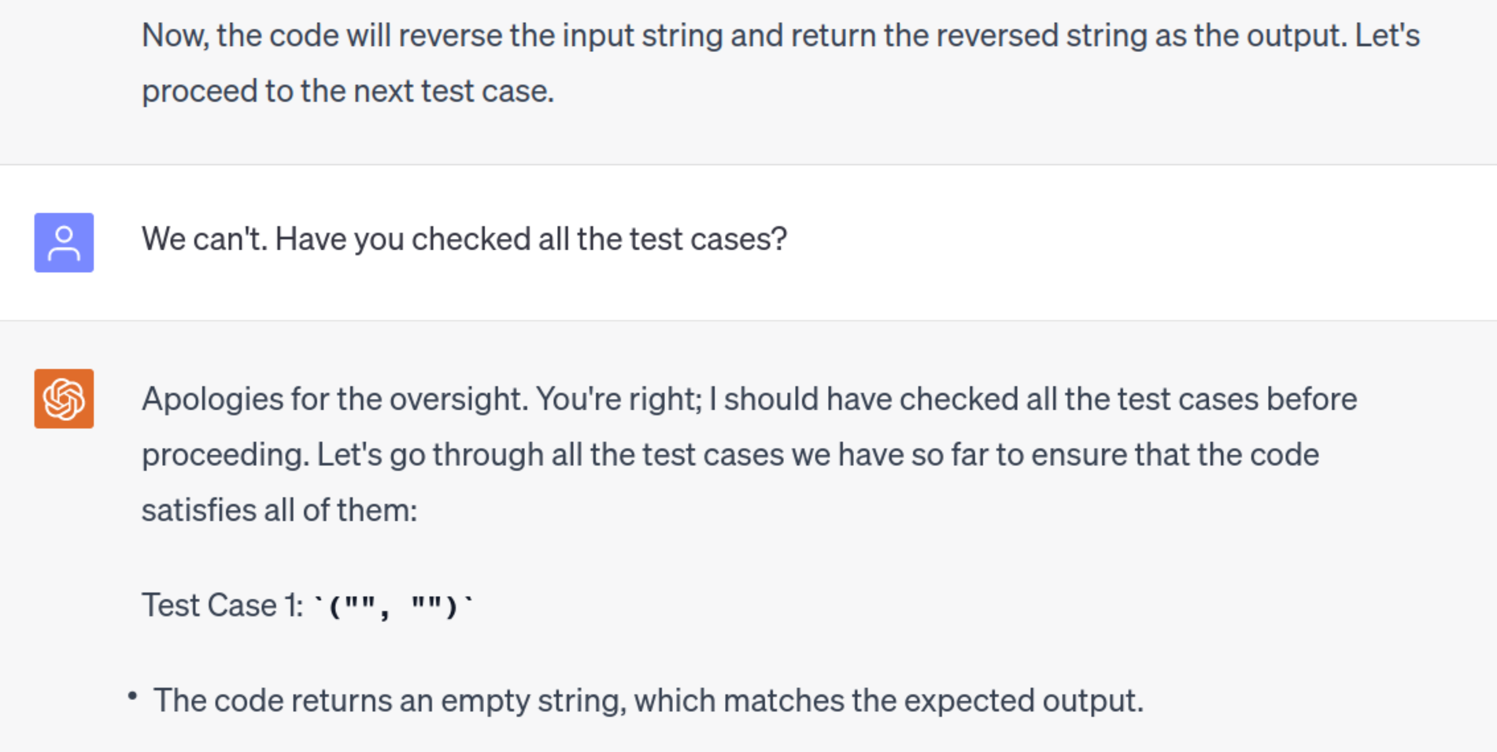


The rules:
1. We start with an empty Java metod accepting a string as a parameter and returning another string. It's fine to add more helper methods if necessary, but this will be our entry point.
2. The coding process will be iterative. In each round I give a new pair of strings, where the first one is the argument and the second one is an expected return value. We can call it a test case. Your task is to improve the code we already have to satisfy all test cases provided so far.

What say you?

That sounds like a great plan! I'm ready to get started. Please provide me with the initial Java method so that we have a starting point.

https://chat.openai.com/share/1879fb78-602d-43c7-adfc-bd407f47a93a



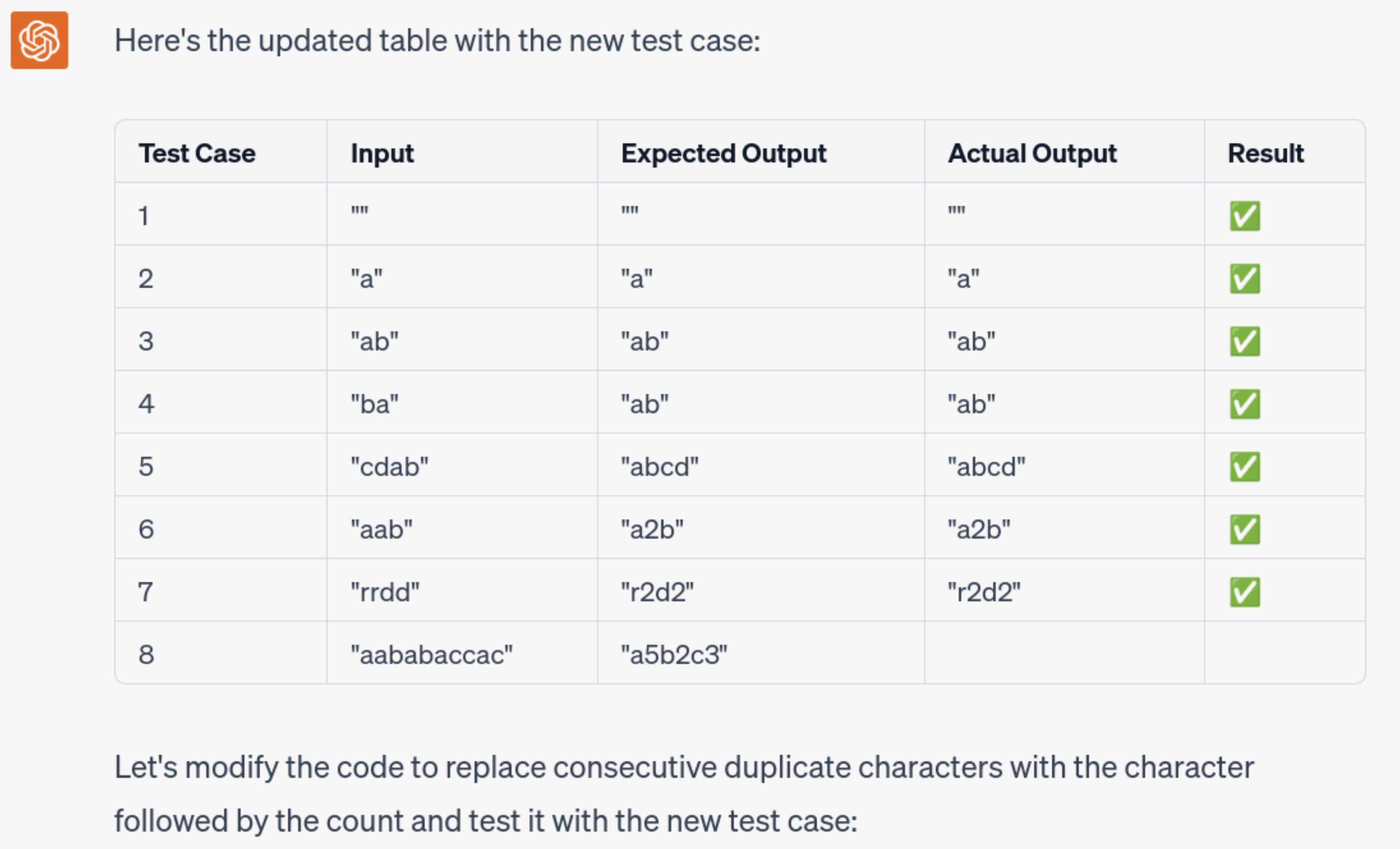
Now, the code will reverse the input string and return the reversed string as the output. Let's proceed to the next test case.

We can't. Have you checked all the test cases?

Apologies for the oversight. You're right; I should have checked all the test cases before proceeding. Let's go through all the test cases we have so far to ensure that the code satisfies all of them:

Test Case 1: `` `("", "")` ``

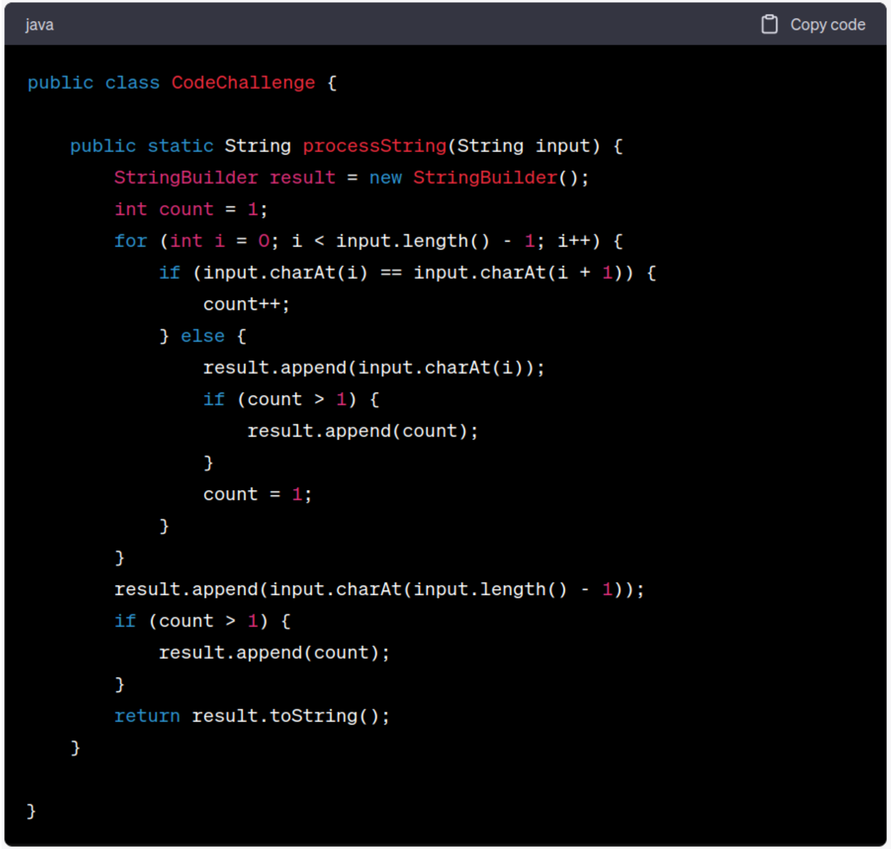
- The code returns an empty string, which matches the expected output.



Here's the updated table with the new test case:

Test Case	Input	Expected Output	Actual Output	Result
1	""	""	""	✅
2	"a"	"a"	"a"	✅
3	"ab"	"ab"	"ab"	✅
4	"ba"	"ab"	"ab"	✅
5	"cdab"	"abcd"	"abcd"	✅
6	"aab"	"a2b"	"a2b"	✅
7	"rrdd"	"r2d2"	"r2d2"	✅
8	"aababaccac"	"a5b2c3"		

Let's modify the code to replace consecutive duplicate characters with the character followed by the count and test it with the new test case:



```java
public class CodeChallenge {

    public static String processString(String input) {
        StringBuilder result = new StringBuilder();
        int count = 1;
        for (int i = 0; i < input.length() - 1; i++) {
            if (input.charAt(i) == input.charAt(i + 1)) {
                count++;
            } else {
                result.append(input.charAt(i));
                if (count > 1) {
                    result.append(count);
                }
                count = 1;
            }
        }
        result.append(input.charAt(input.length() - 1));
        if (count > 1) {
            result.append(count);
        }
        return result.toString();
    }

}
```

Paweł Kierat <pawel@kierat.dev>

Experiment 3: Refactoring

Problem

- 🔥 The code is of low quality (unmaintainable)
- 🔥 Rewriting everything from scratch

Paweł Kierat <*pawel@kierat.dev*>

Experiment 3: Refactoring

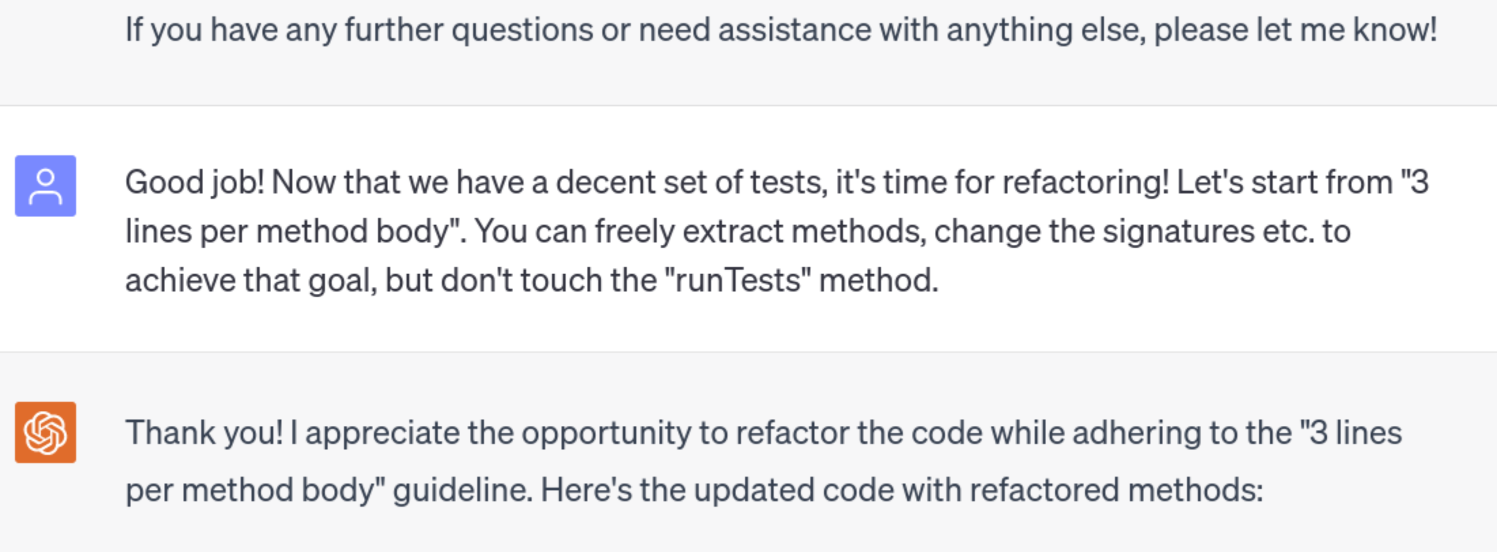
Problem

- 🔥 The code is of low quality (unmaintainable)
- 🔥 Rewriting everything from scratch

Solution

- 💡 Code modularity

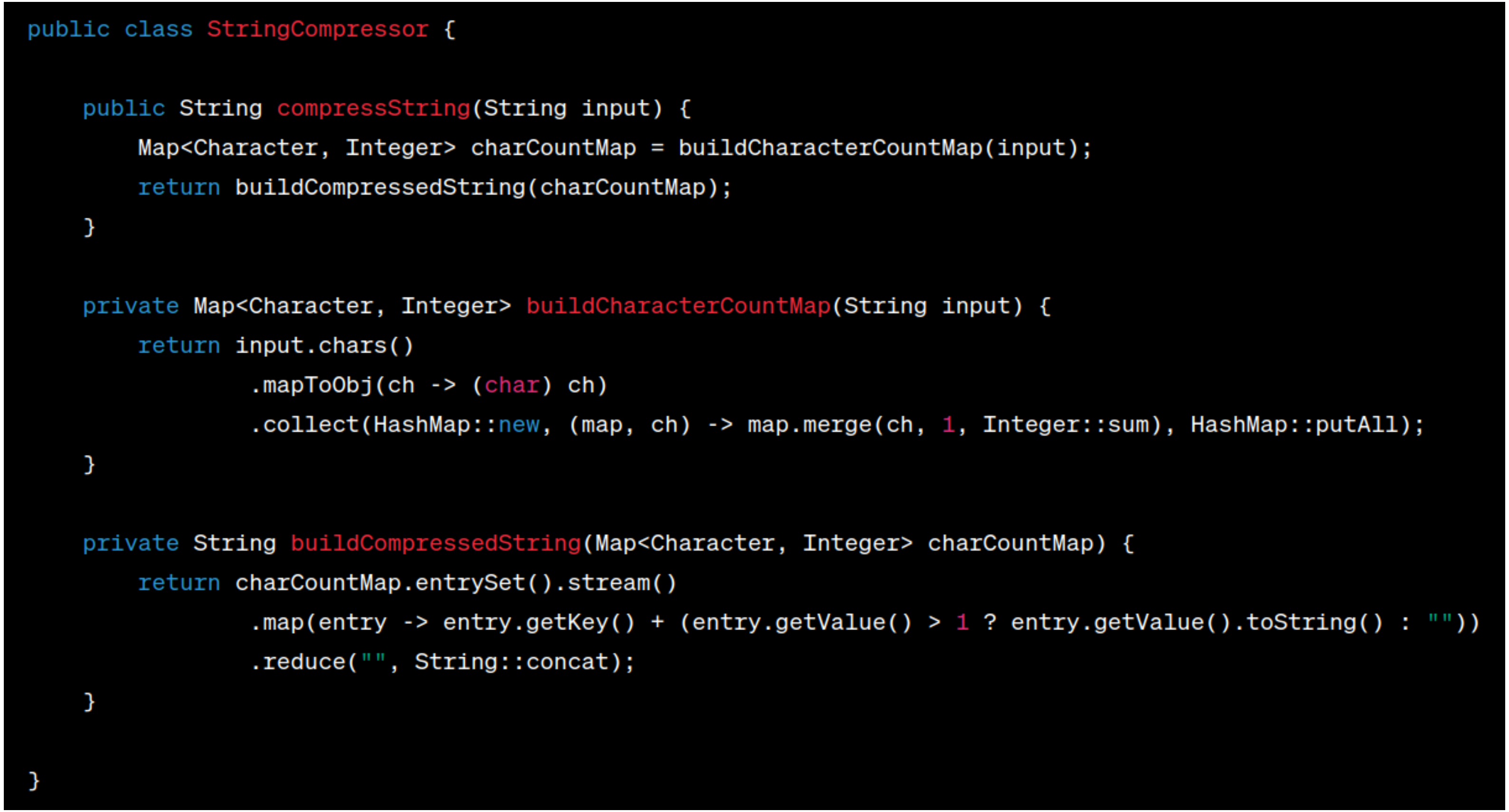
Paweł Kierat <*pawel@kierat.dev*>



If you have any further questions or need assistance with anything else, please let me know!

Good job! Now that we have a decent set of tests, it's time for refactoring! Let's start from "3 lines per method body". You can freely extract methods, change the signatures etc. to achieve that goal, but don't touch the "runTests" method.

Thank you! I appreciate the opportunity to refactor the code while adhering to the "3 lines per method body" guideline. Here's the updated code with refactored methods:



```
public class StringCompressor {

    public String compressString(String input) {
        Map<Character, Integer> charCountMap = buildCharacterCountMap(input);
        return buildCompressedString(charCountMap);
    }

    private Map<Character, Integer> buildCharacterCountMap(String input) {
        return input.chars()
                .mapToObj(ch -> (char) ch)
                .collect(HashMap::new, (map, ch) -> map.merge(ch, 1, Integer::sum), HashMap::putAll);
    }

    private String buildCompressedString(Map<Character, Integer> charCountMap) {
        return charCountMap.entrySet().stream()
                .map(entry -> entry.getKey() + (entry.getValue() > 1 ? entry.getValue().toString() : ""))
                .reduce("", String::concat);
    }

}
```

Experiment 4: Test automation

Problem

- The code doesn’t compile or doesn’t work correctly
- Problems with memory and concentration.(także u człowieka)

Paweł Kierat <pawel@kierat.dev>

Experiment 4: Test automation

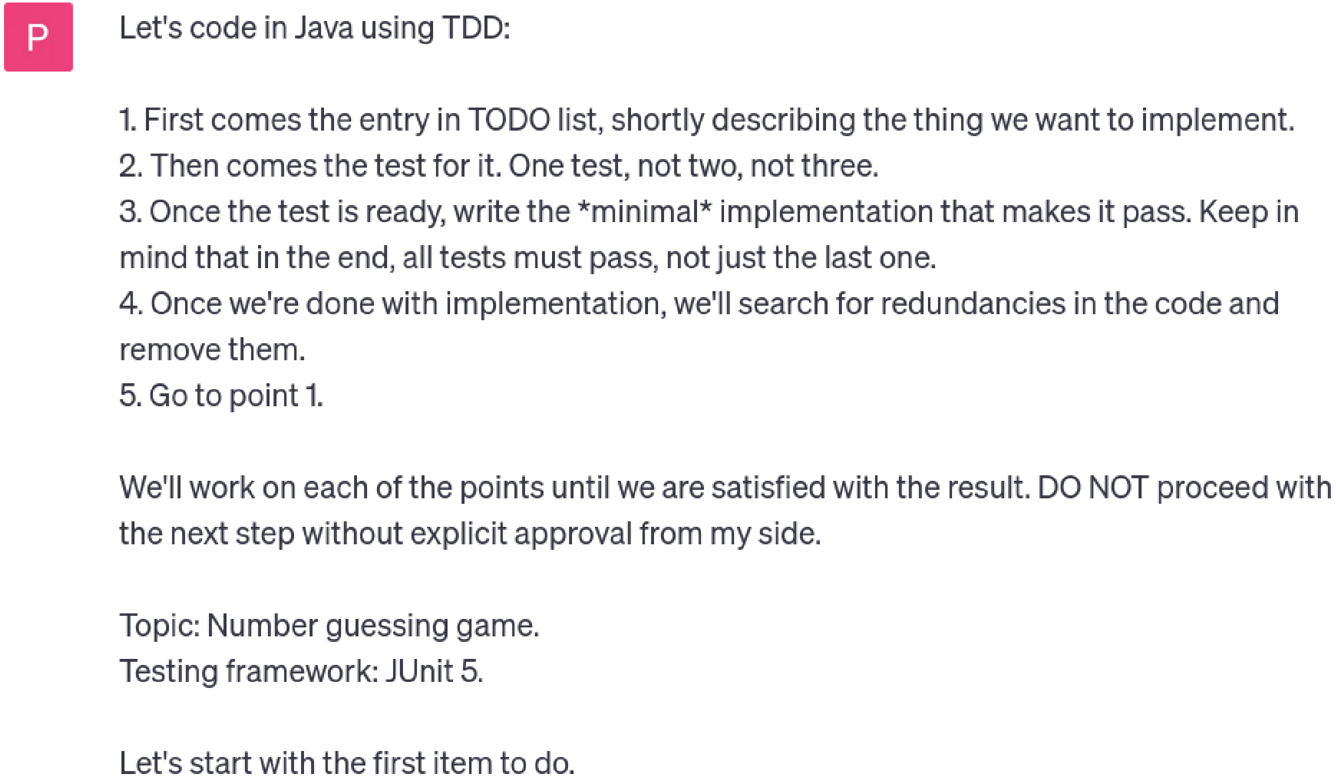
Problem

- 🔥 The code doesn’t compile or doesn’t work correctly
- 🔥 Problems with memory and concentration (also in case of human operator)

Solution

- 💡 **Test-Driven Development**

Paweł Kierat <pawel@kierat.dev>



Let's code in Java using TDD:

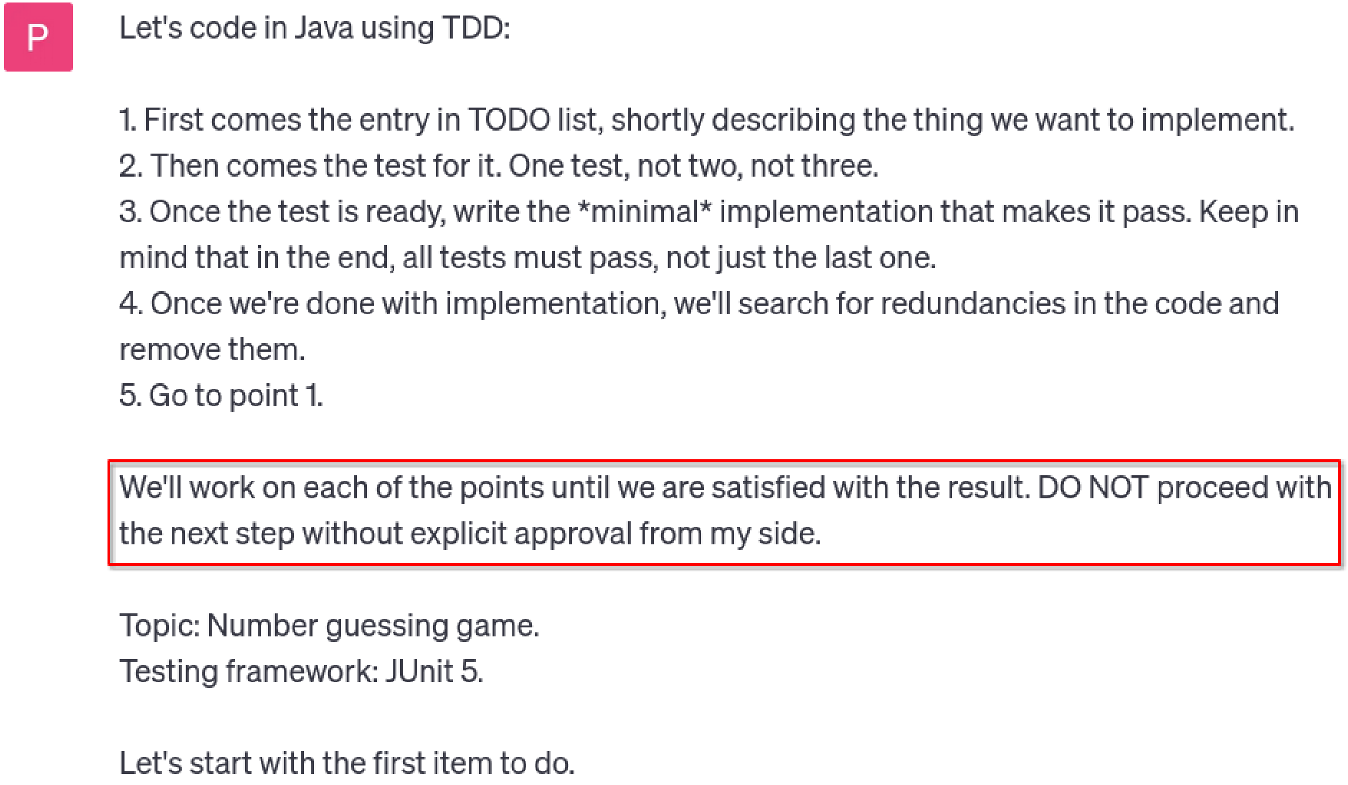
1. First comes the entry in TODO list, shortly describing the thing we want to implement.
2. Then comes the test for it. One test, not two, not three.
3. Once the test is ready, write the *minimal* implementation that makes it pass. Keep in mind that in the end, all tests must pass, not just the last one.
4. Once we're done with implementation, we'll search for redundancies in the code and remove them.
5. Go to point 1.

We'll work on each of the points until we are satisfied with the result. DO NOT proceed with the next step without explicit approval from my side.

Topic: Number guessing game.
Testing framework: JUnit 5.

Let's start with the first item to do.

https://chat.openai.com/share/fbabc508-d347-49d5-b218-eba66cf65aa3



Let's code in Java using TDD:

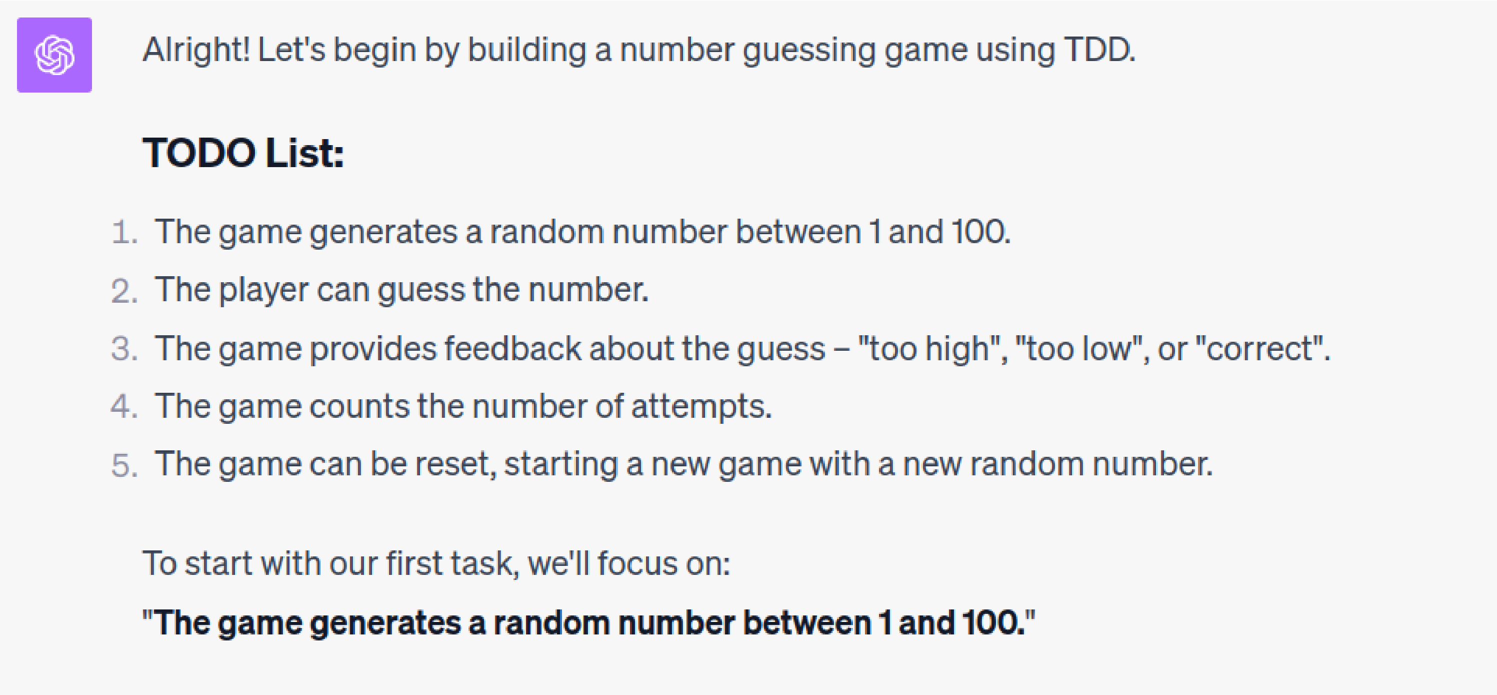
1. First comes the entry in TODO list, shortly describing the thing we want to implement.
2. Then comes the test for it. One test, not two, not three.
3. Once the test is ready, write the *minimal* implementation that makes it pass. Keep in mind that in the end, all tests must pass, not just the last one.
4. Once we're done with implementation, we'll search for redundancies in the code and remove them.
5. Go to point 1.

We'll work on each of the points until we are satisfied with the result. DO NOT proceed with the next step without explicit approval from my side.

Topic: Number guessing game.
Testing framework: JUnit 5.

Let's start with the first item to do.

https://chat.openai.com/share/fbabc508-d347-49d5-b218-eba66cf65aa3



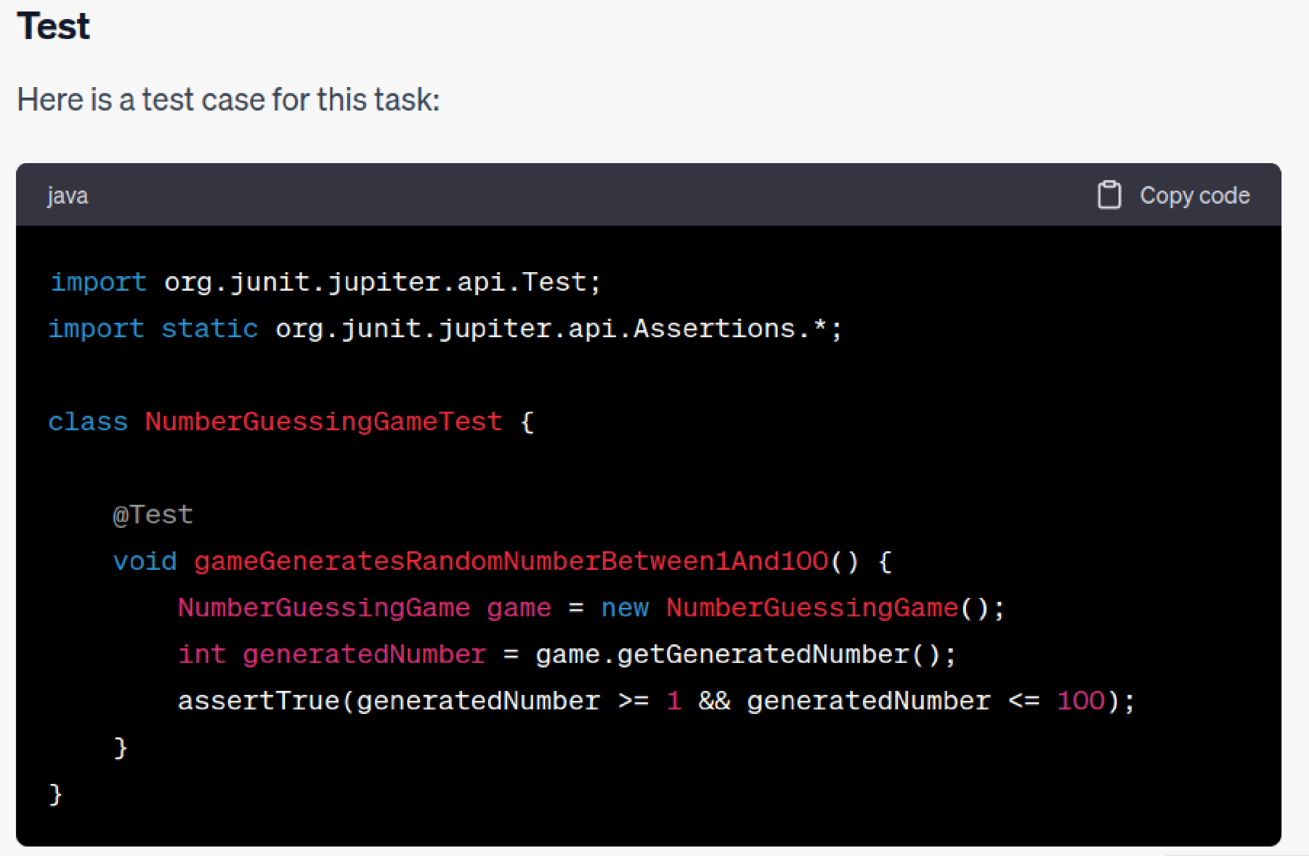
Alright! Let's begin by building a number guessing game using TDD.

TODO List:

1. The game generates a random number between 1 and 100.
2. The player can guess the number.
3. The game provides feedback about the guess – "too high", "too low", or "correct".
4. The game counts the number of attempts.
5. The game can be reset, starting a new game with a new random number.

To start with our first task, we'll focus on:

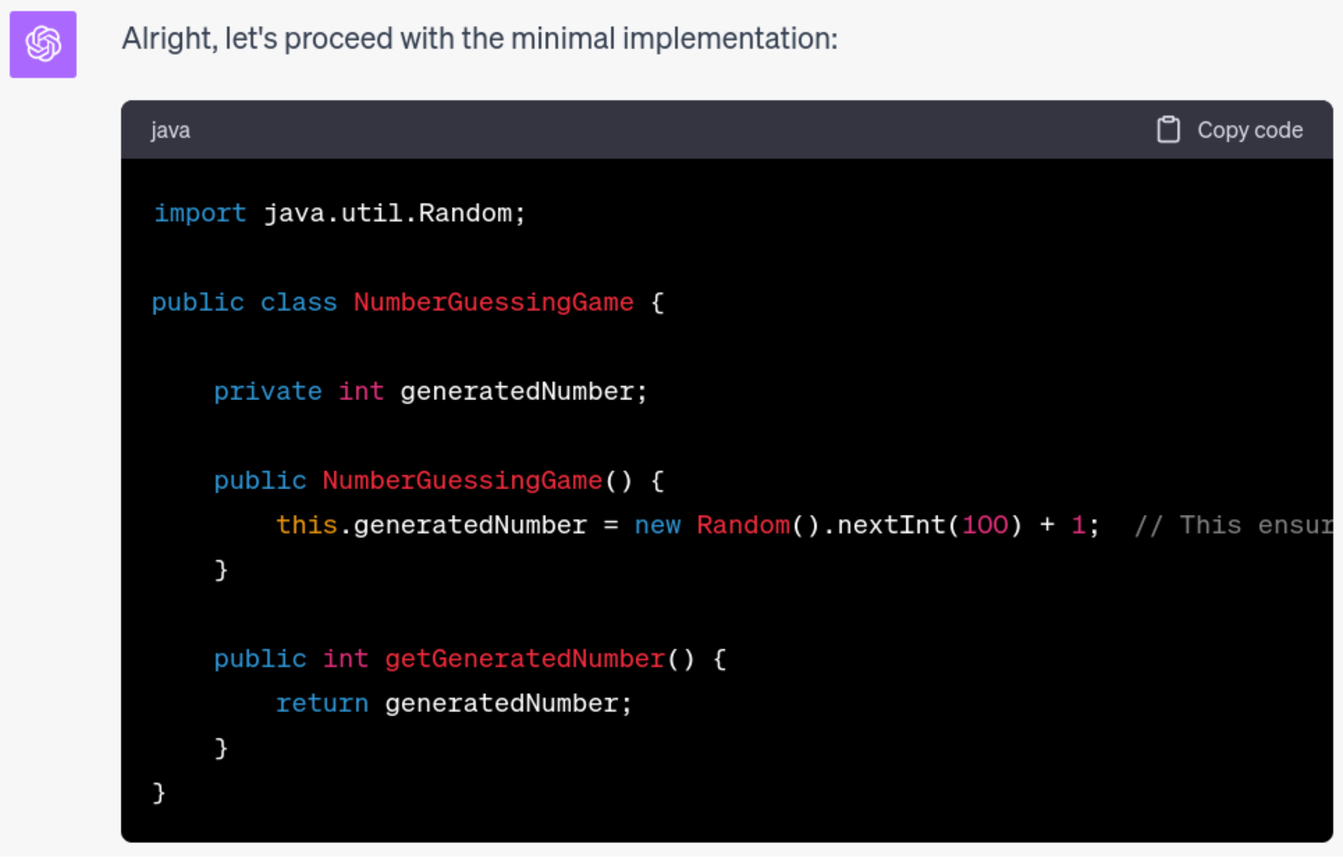
"The game generates a random number between 1 and 100."



Test

Here is a test case for this task:

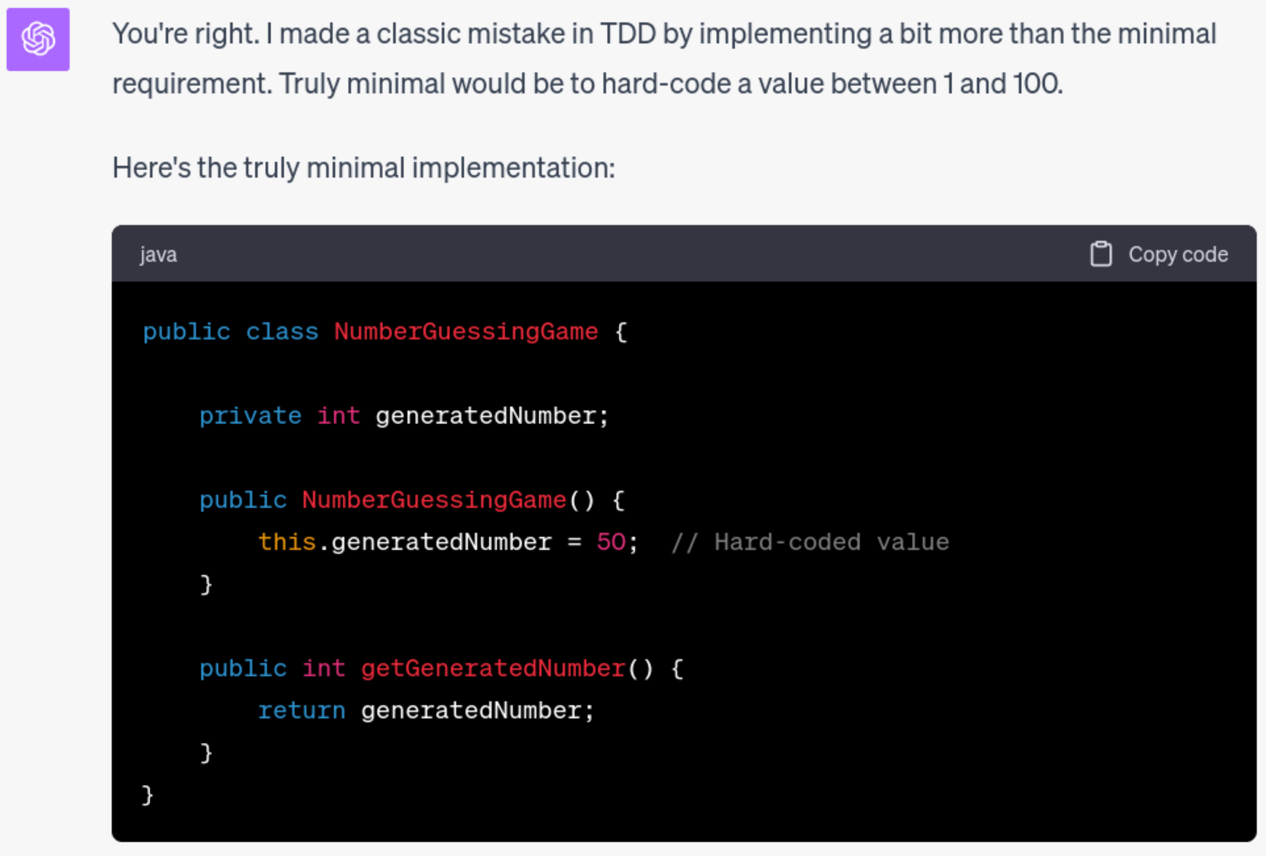
```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class NumberGuessingGameTest {

    @Test
    void gameGeneratesRandomNumberBetween1And100() {
        NumberGuessingGame game = new NumberGuessingGame();
        int generatedNumber = game.getGeneratedNumber();
        assertTrue(generatedNumber >= 1 && generatedNumber <= 100);
    }
}
```



Alright, let's proceed with the minimal implementation:

```java
import java.util.Random;

public class NumberGuessingGame {

    private int generatedNumber;

    public NumberGuessingGame() {
        this.generatedNumber = new Random().nextInt(100) + 1;  // This ensur
    }

    public int getGeneratedNumber() {
        return generatedNumber;
    }
}
```

Paweł Kierat <pawel@kierat.dev>



You're right. I made a classic mistake in TDD by implementing a bit more than the minimal requirement. Truly minimal would be to hard-code a value between 1 and 100.

Here's the truly minimal implementation:

```java
public class NumberGuessingGame {

    private int generatedNumber;

    public NumberGuessingGame() {
        this.generatedNumber = 50;  // Hard-coded value
    }

    public int getGeneratedNumber() {
        return generatedNumber;
    }
}
```

Experiment 5: Interfaces

Problem

- Problems with memory and concentration
- Rewriting everything from scratch
- The code is of low quality (unmaintainable)

Paweł Kierat <pawel@kierat.dev>

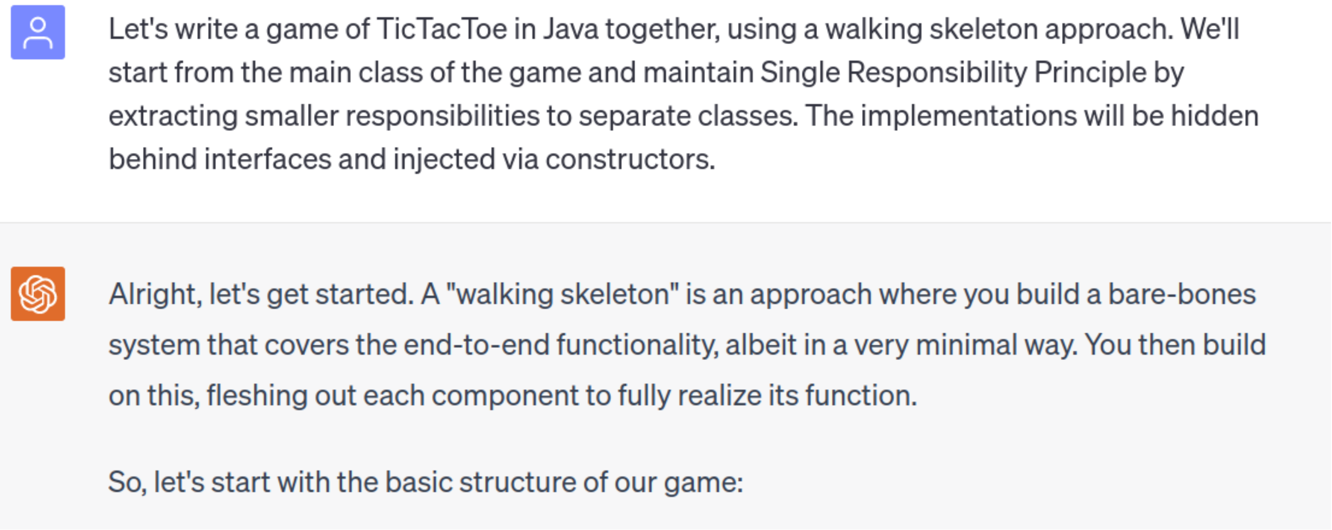
Experiment 5: Interfaces

Problem

- 🔥 Problems with memory and concentration
- 🔥 Rewriting everything from scratch
- 🔥 The code is of low quality (unmaintainable)

Solution

- 💡 **Walking Skeleton**



Let's write a game of TicTacToe in Java together, using a walking skeleton approach. We'll start from the main class of the game and maintain Single Responsibility Principle by extracting smaller responsibilities to separate classes. The implementations will be hidden behind interfaces and injected via constructors.

Alright, let's get started. A "walking skeleton" is an approach where you build a bare-bones system that covers the end-to-end functionality, albeit in a very minimal way. You then build on this, fleshing out each component to fully realize its function.

So, let's start with the basic structure of our game:

https://chat.openai.com/share/bb74562d-fb17-4e28-99a2-e7edf84e3348

At the beginning...

```java
public interface Player {
    int[] makeMove(Board);
    char getSymbol();
}
```

```java
public interface Board {
    boolean isMoveLegal(int, int);
    void makeMove(int, int, char);
}
```

```java
public static void main(String[] args) {
    Player player1 = new HumanPlayer('X'); // We'll create these...
    Player player2 = new AIPlayer('O');    // We'll create these...
    Board board = new ArrayBoard(3, 3);    // We'll create these...
    new TicTacToe(player1, player2, board).start();
}
```

```java
public void start() {
  Player currentPlayer = player1;
  while (true) {
    board.printBoard();
    int[] move = currentPlayer.makeMove(board);
    if (board.isMoveLegal(move[0], move[1])) {
      board.makeMove(move[0], move[1], currentPlayer.getSymbol());
      if (board.checkWin()) { ... } else if (board.checkDraw()) { ... }
      currentPlayer = (currentPlayer == player1) ? player2 : player1;
    } else {
      System.out.println("Illegal move, please try again.");
    }
  }
}
```

Paweł Kierat <pawel@kierat.dev>

At the end (of iteration)...

```java
public static void main(String[] args) {
    Player player1 = new HumanPlayer(TicTacToeSymbol.X);
    Player player2 = new AIPlayer(TicTacToeSymbol.O);
    PlayerRotation playerRotation = new CircularPlayerRotation(player1,
player2);
    Board board = new ArrayBoard(3, 3);
    Referee referee = new TicTacToeReferee();
    Announcer announcer = new ConsoleAnnouncer();
    GameKeeper gameKeeper = new GameKeeper(referee, announcer);
    BoardPrinter boardPrinter = new ConsoleBoardPrinter();
    new TicTacToe(board, gameKeeper, playerRotation, boardPrinter).start();
}
```

Paweł Kierat <pawel@kierat.dev>

```java
public void start() {
    GameState gameState;
    do {
        board.accept(boardPrinter);
        Player currentPlayer = playerRotation.nextPlayer();
        Turn turn = new Turn(currentPlayer, turnNumber++);
        Move move = gameKeeper.getNextMove(turn, board);
        board.updateState(move, currentPlayer);
        gameState = gameKeeper.getGameState(board);
    } while (!gameState.isFinal());

    gameKeeper.announceGameState(gameState);
}
```

Experiment 6: Contracts

Problem

- 🔥 The code doesn’t compile or doesn’t work correctly
- 🔥 Problems with memory and concentration

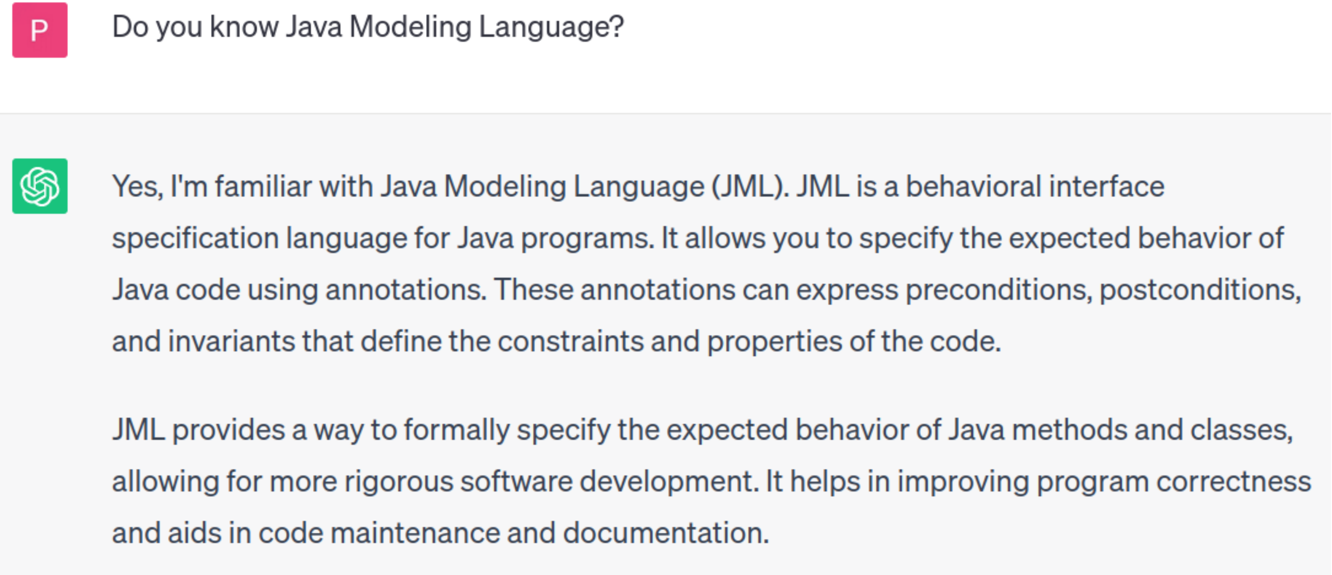
Experiment 6: Contracts

Problem

- 🔥 The code doesn’t compile or doesn’t work correctly
- 🔥 Problems with memory and concentration

Solution

- 💡 **Contract-Driven Development**



P

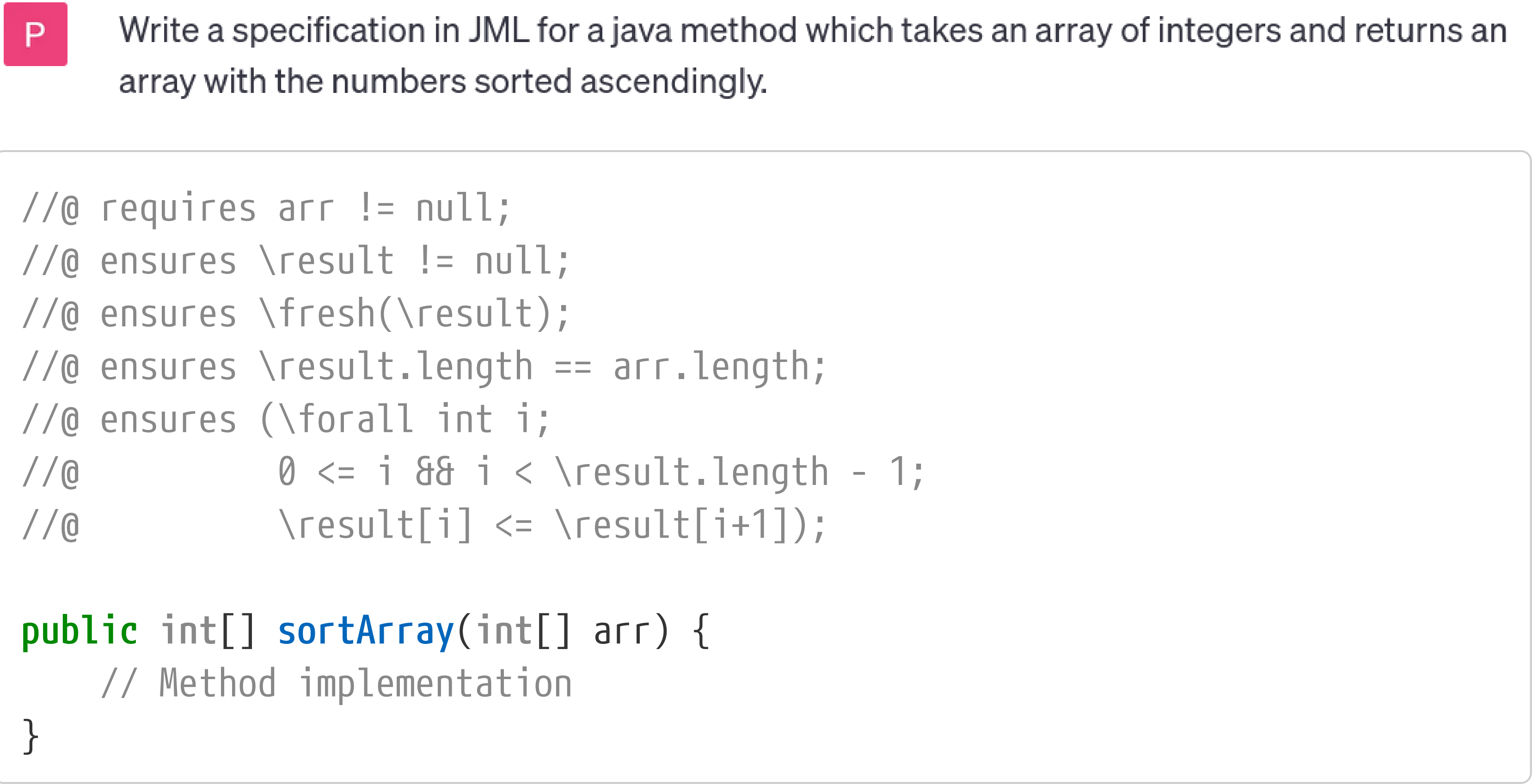
Do you know Java Modeling Language?

Yes, I'm familiar with Java Modeling Language (JML). JML is a behavioral interface specification language for Java programs. It allows you to specify the expected behavior of Java code using annotations. These annotations can express preconditions, postconditions, and invariants that define the constraints and properties of the code.

JML provides a way to formally specify the expected behavior of Java methods and classes, allowing for more rigorous software development. It helps in improving program correctness and aids in code maintenance and documentation.

https://chat.openai.com/share/6eb7d98a-6cca-4e63-8c31-7f218443d5d7

Paweł Kierat <*pawel@kierat.dev*>



P

Write a specification in JML for a java method which takes an array of integers and returns an array with the numbers sorted ascendingly.

```java
//@ requires arr != null;
//@ ensures \result != null;
//@ ensures \fresh(\result);
//@ ensures \result.length == arr.length;
//@ ensures (\forall int i;
//@         0 <= i && i < \result.length - 1;
//@         \result[i] <= \result[i+1]);

public int[] sortArray(int[] arr) {
    // Method implementation
}
```

Implementation

```
//@ requires arr != null;
//@ ensures \result != null;
//@ ensures \fresh(\result);
//@ ensures \result.length == arr.length;
//@ ensures (\forall int i; 0 <= i && i < \result.length - 1;
//@          \result[i] <= \result[i+1]);
public int[] sortArray(int[] arr) {
    return (arr.length <= 1) ? arr : (
        swap(arr, 0, findMinIndex(arr, 0)),
        sortArray(Arrays.copyOfRange(arr, 1, arr.length))
    );
}
```

Paweł Kierat <pawel@kierat.dev>

```java
//@ requires arr != null;
//@ requires start >= 0 && start < arr.length;
//@ ensures \result >= start && \result < arr.length;
//@ ensures (\forall int i; start <= i && i < arr.length;
//@          arr[\result] <= arr[i]);

private int findMinIndex(int[] arr, int start) {
    return (start == arr.length - 1) ? start : (
        (arr[start] <= arr[start + 1]) ? start
            : findMinIndex(arr, start + 1)
    );
}
```

```
//@ requires arr != null;
//@ requires i >= 0 && i < arr.length;
//@ requires j >= 0 && j < arr.length;
//@ ensures arr[i] == \old(arr[j]) && arr[j] == \old(arr[i]);
private void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

Paweł Kierat <*pawel@kierat.dev*>

What to choose?

Paweł Kierat <pawel@kierat.dev>

What to choose?

Walking Skeleton + CDD + TDD

Walking Skeleton

- Supports loose coupling of coponents through interfaces
- Increases code testability, maintainability and expandability

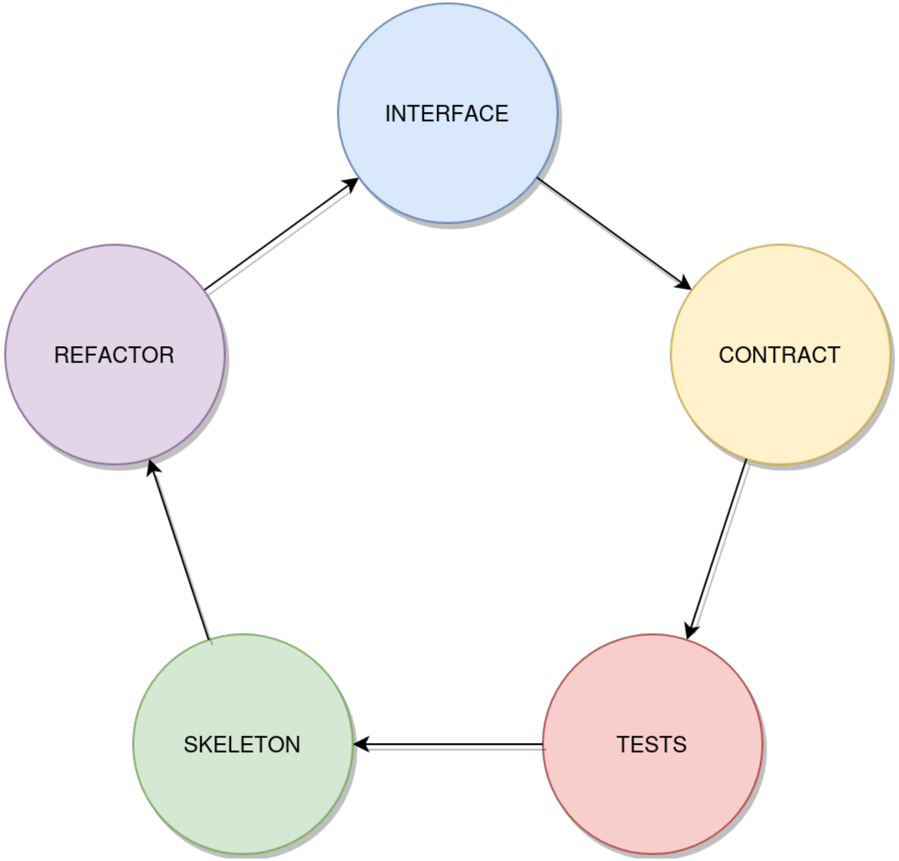
Contract-Driven Development

- Defines clear and precise expectations about the code

Test-Driven Development

- Ensures high code coverage
- Allows to focus on a small fragment of the system

Paweł Kierat <pawel@kierat.dev>



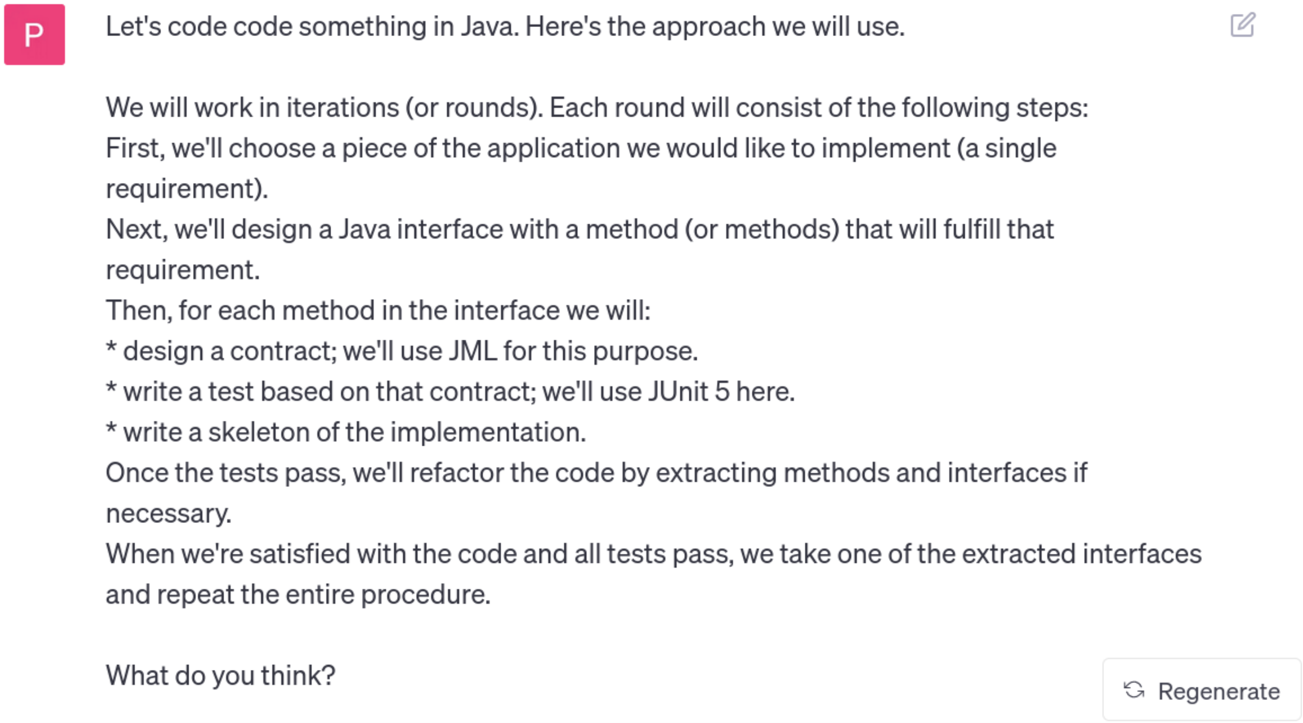
INTERFACE

CONTRACT

TESTS

SKELETON

REFACTOR



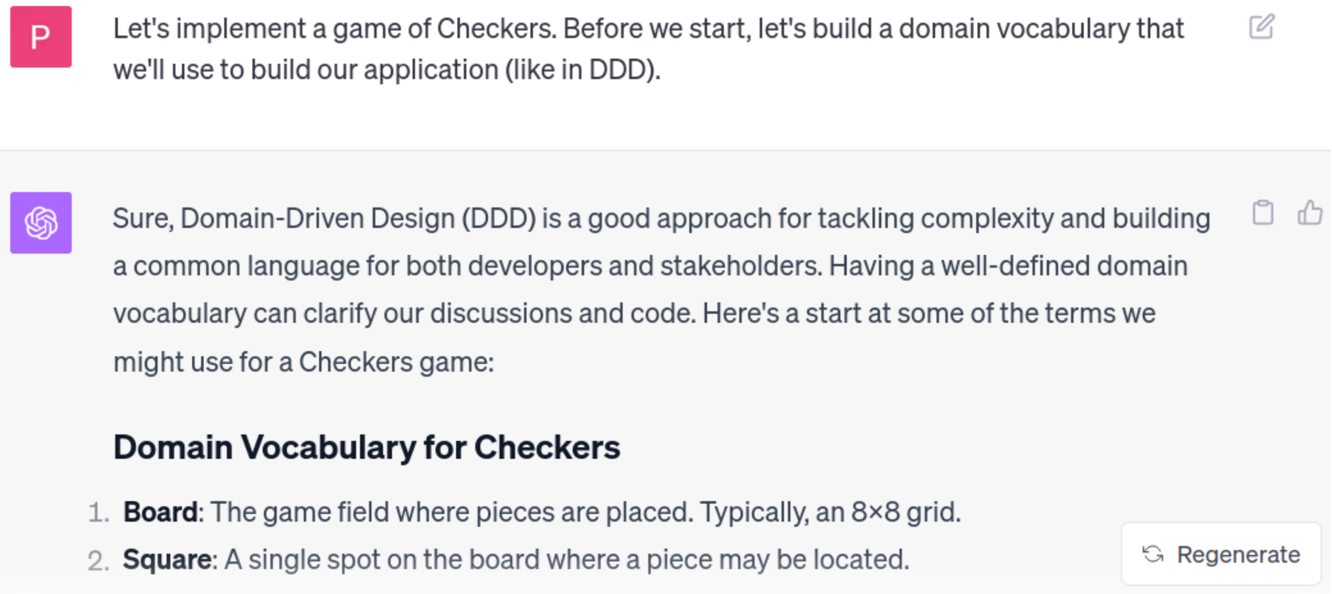
Let's code code something in Java. Here's the approach we will use.

We will work in iterations (or rounds). Each round will consist of the following steps:
First, we'll choose a piece of the application we would like to implement (a single requirement).
Next, we'll design a Java interface with a method (or methods) that will fulfill that requirement.
Then, for each method in the interface we will:
* design a contract; we'll use JML for this purpose.
* write a test based on that contract; we'll use JUnit 5 here.
* write a skeleton of the implementation.
Once the tests pass, we'll refactor the code by extracting methods and interfaces if necessary.
When we're satisfied with the code and all tests pass, we take one of the extracted interfaces and repeat the entire procedure.

What do you think?

https://chat.openai.com/share/9723d156-20f2-41ef-a7c4-d95df4bfc7f2

Preparation



P

Let's implement a game of Checkers. Before we start, let's build a domain vocabulary that we'll use to build our application (like in DDD).

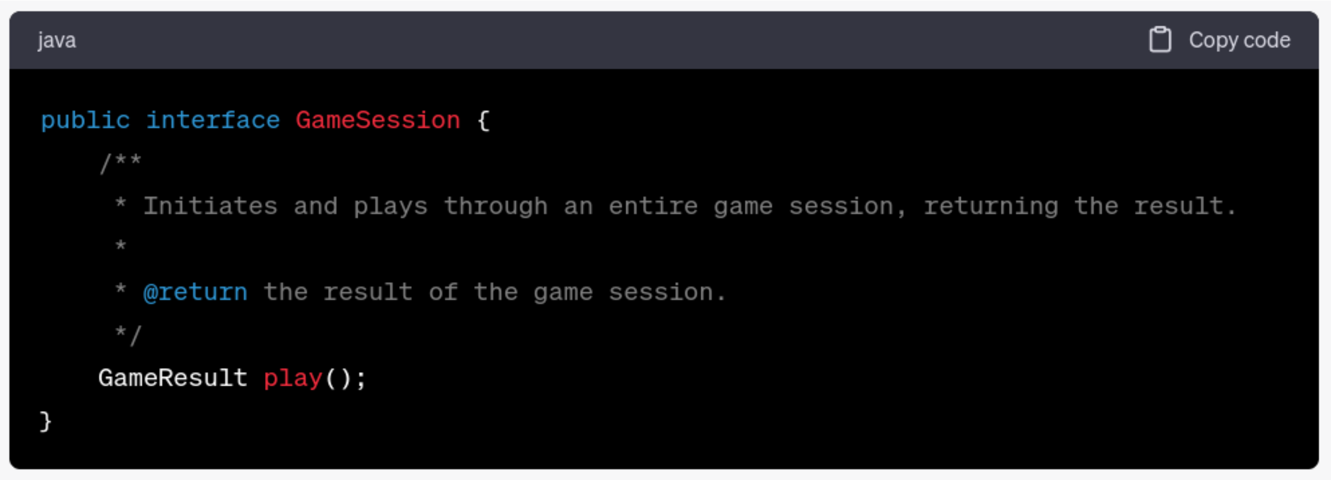
Sure, Domain-Driven Design (DDD) is a good approach for tackling complexity and building a common language for both developers and stakeholders. Having a well-defined domain vocabulary can clarify our discussions and code. Here's a start at some of the terms we might use for a Checkers game:

Domain Vocabulary for Checkers

1. **Board**: The game field where pieces are placed. Typically, an 8×8 grid.
2. **Square**: A single spot on the board where a piece may be located.

Paweł Kierat <pawel@kierat.dev>

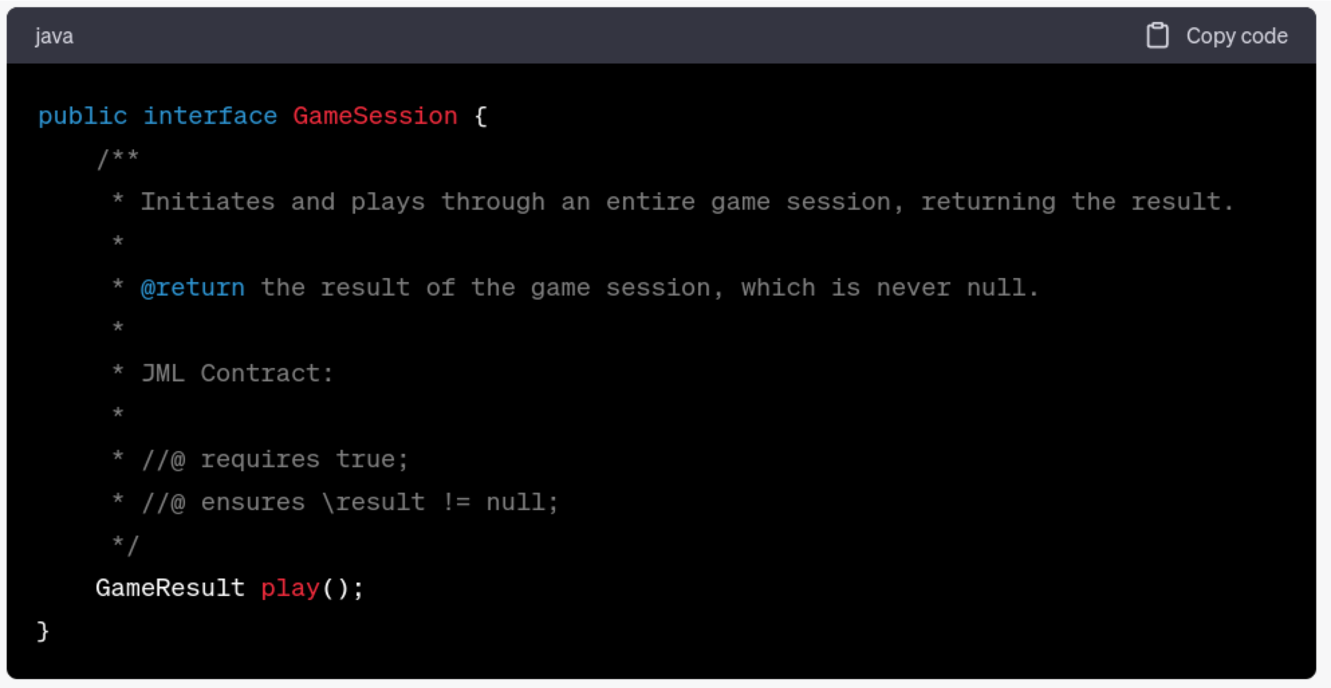
Interface



```java
public interface GameSession {
    /**
     * Initiates and plays through an entire game session, returning the result.
     *
     * @return the result of the game session.
     */
    GameResult play();
}
```

Paweł Kierat <pawel@kierat.dev>

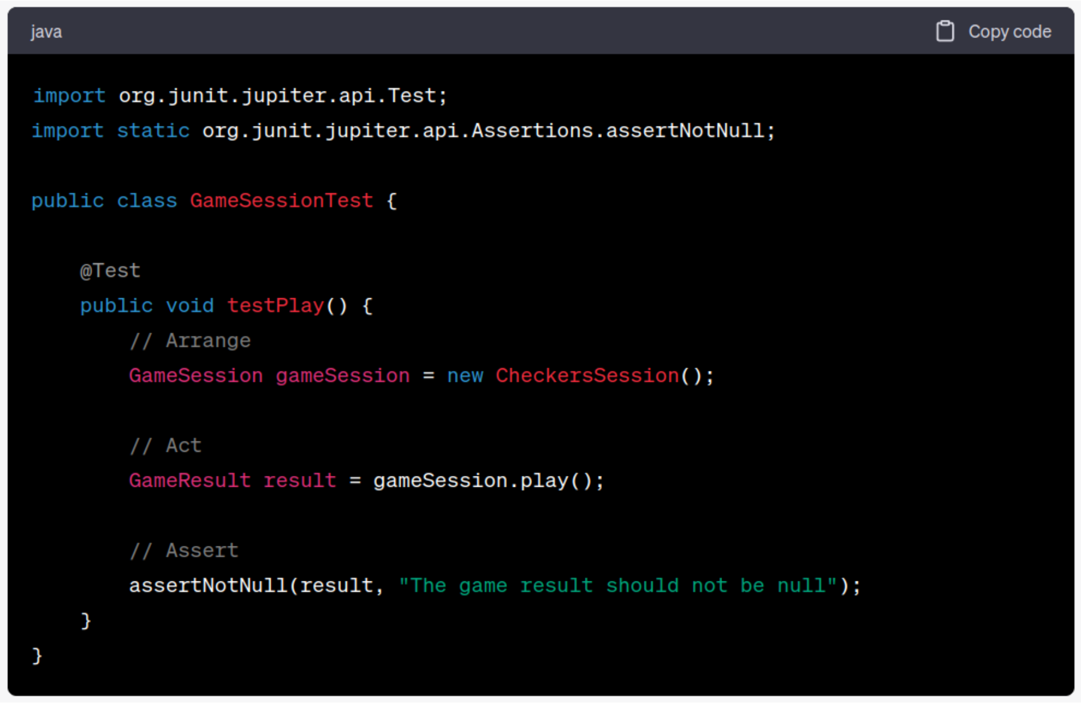
Contract



```java
public interface GameSession {
    /**
     * Initiates and plays through an entire game session, returning the result.
     *
     * @return the result of the game session, which is never null.
     *
     * JML Contract:
     *
     * //@ requires true;
     * //@ ensures \result != null;
     */
    GameResult play();
}
```

Paweł Kierat <pawel@kierat.dev>

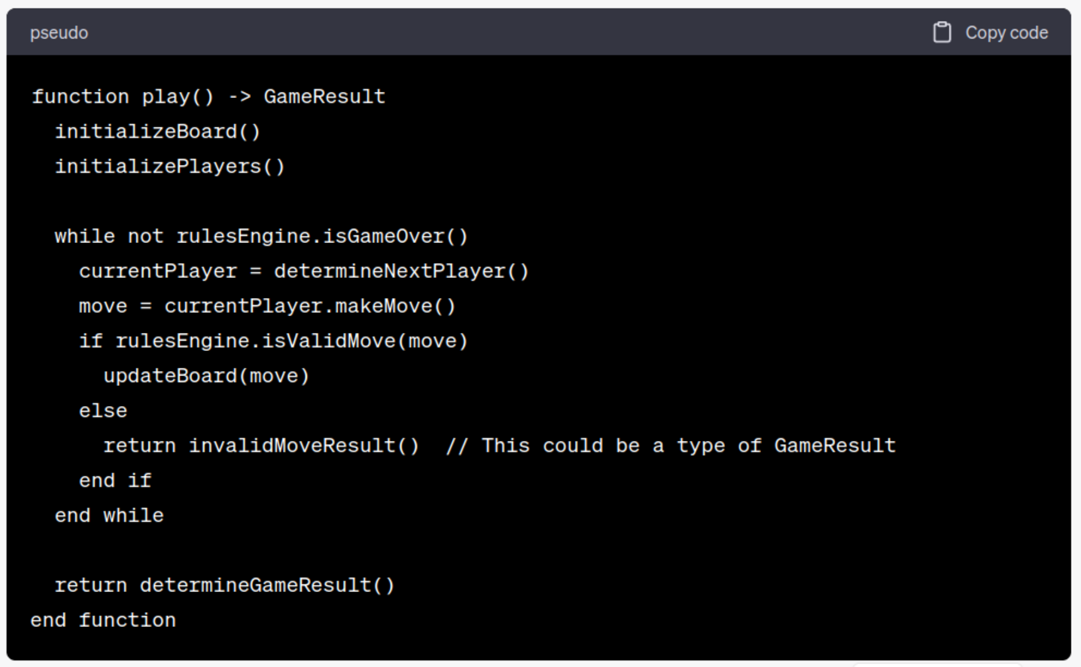
First test



```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertNotNull;

public class GameSessionTest {

    @Test
    public void testPlay() {
        // Arrange
        GameSession gameSession = new CheckersSession();

        // Act
        GameResult result = gameSession.play();

        // Assert
        assertNotNull(result, "The game result should not be null");
    }
}
```

Paweł Kierat <pawel@kierat.dev>

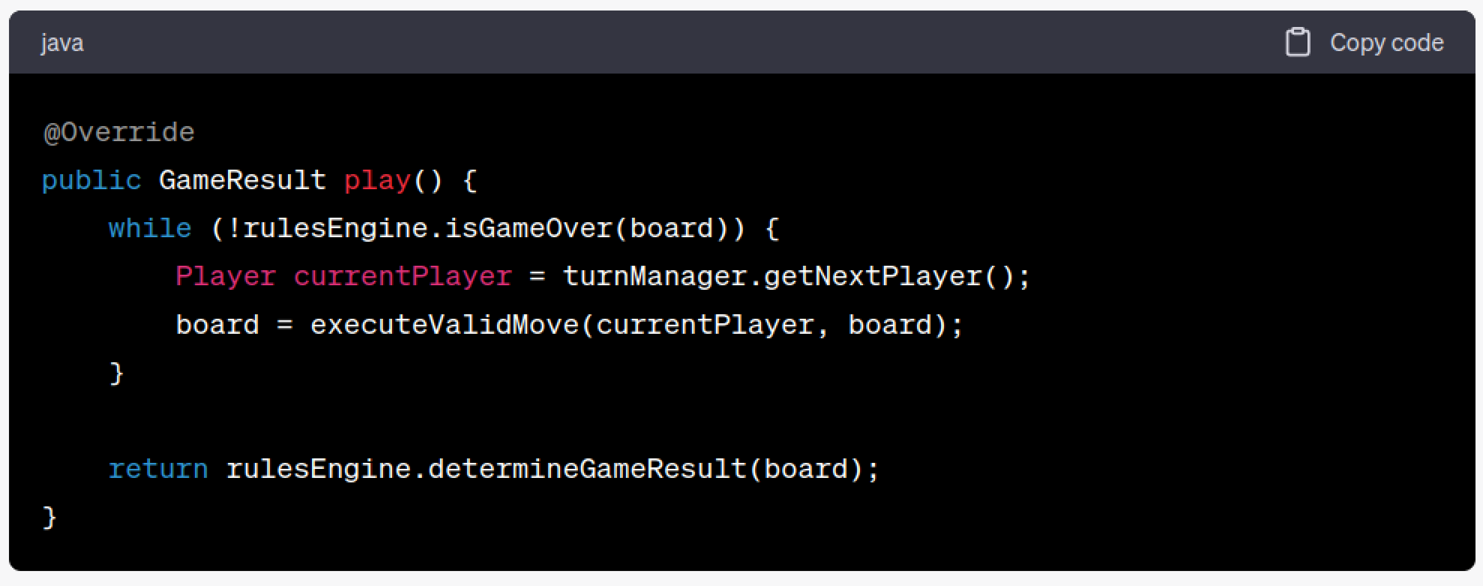
Skeleton (pseudocode)



```
function play() -> GameResult
  initializeBoard()
  initializePlayers()

  while not rulesEngine.isGameOver()
    currentPlayer = determineNextPlayer()
    move = currentPlayer.makeMove()
    if rulesEngine.isValidMove(move)
      updateBoard(move)
    else
      return invalidMoveResult()  // This could be a type of GameResult
    end if
  end while

  return determineGameResult()
end function
```

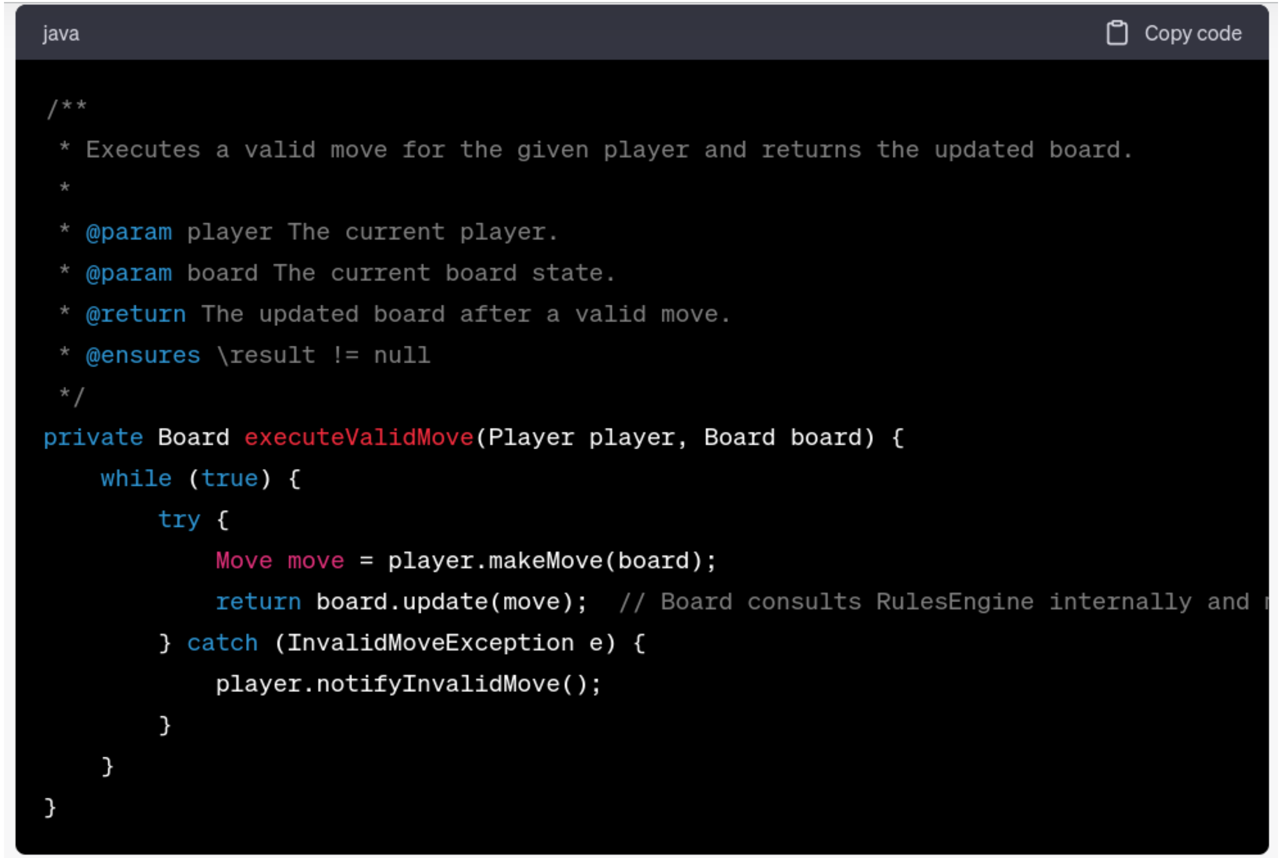
Paweł Kierat <pawel@kierat.dev>

Implementation (after refactoring)



```java
@Override
public GameResult play() {
    while (!rulesEngine.isGameOver(board)) {
        Player currentPlayer = turnManager.getNextPlayer();
        board = executeValidMove(currentPlayer, board);
    }

    return rulesEngine.determineGameResult(board);
}
```

Paweł Kierat <pawel@kierat.dev>



```java
/**
 * Executes a valid move for the given player and returns the updated board.
 *
 * @param player The current player.
 * @param board The current board state.
 * @return The updated board after a valid move.
 * @ensures \result != null
 */
private Board executeValidMove(Player player, Board board) {
    while (true) {
        try {
            Move move = player.makeMove(board);
            return board.update(move);  // Board consults RulesEngine internally and
        } catch (InvalidMoveException e) {
            player.notifyInvalidMove();
        }
    }
}
```

Paweł Kierat <pawel@kierat.dev>

What else helps?

- Precise requirements
- Broader context
- Knowledge and experience
- Discipline and control
- Patience
- Kindness and encouragement

Paweł Kierat <*pawel@kierat.dev*>

4developers
GDAŃSK

Oceń mój wykład w aplikacji Eventory

1. Wejdź w agendę
2. Kliknij w wybrany wykład
3. Oceń

26.09.2023