Renier Herba

Professor David Freer

COP 2800 Introduction to Java Programming

04/09/2025

### Make a game with Java and a LLM

GitHub repository: <a href="https://github.com/renierhb/MillionaireGame.git">https://github.com/renierhb/MillionaireGame.git</a>

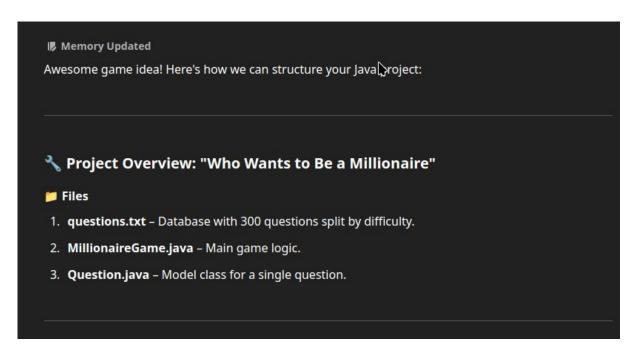
I uploaded the video demonstrating how the game works to the repository. (MillionaireGame.mp4)

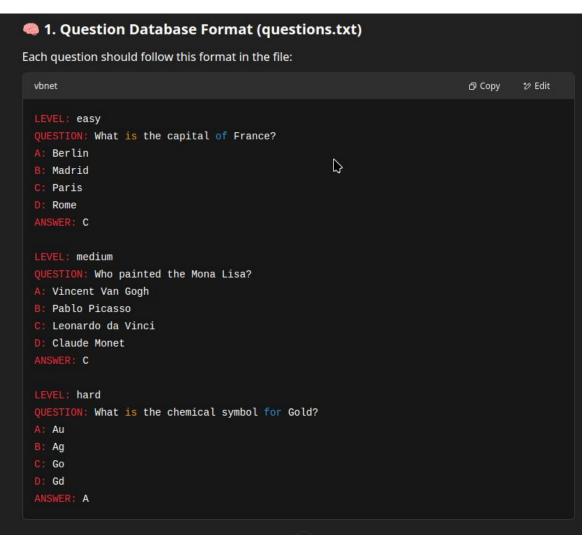
# ChatGPT Chat:

You are a java developer. You want to code the game: "Who want to be a millionaire". The game is a guessing game, where you must to guess 10 questions to get a millon. Everytime you fail, you lost all your money, unless you are in a save step. In the save step, you can decide if you want to leave with all the money you already have or continue.

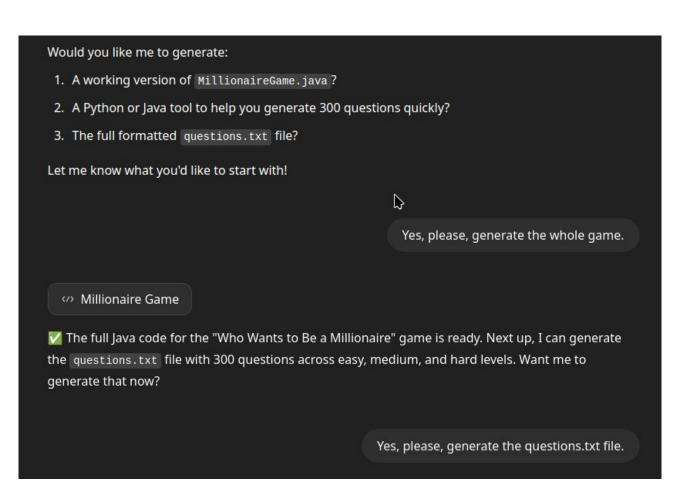
#### Some guidelines:

- Create a database of 300 general questions about science, art, word history, etc. For each question include 4 answer, but just one is the correct. Split these questions in 3 levels: easy, medium, hard. This database must be saved in a txt file.
- When the game start, the code must import the database.
- You ask to the player what is the level of difficulty to play.
- The player must answer 10 questions. In each question the player obtain money if the answer is correct. Split the 1 millon between the 10 questions but not in a linear way.
- Also, add the two save step.
- The player can choose the 50/50 option only 1 time in the game. This option is show in every question as long is not used yet. In this option, the game remove 2 of the incorrect answers and the player have to choose between just two answer, one of them is correct.
- Also, the player can choose the "the call". In this case, the game remove two incorrect answers, and randomly choose between the correct and the incorrect and offer the answer to the player, but as long is random, the question can be correct or not.









Adjust the Choose difficulty: you can enter lower or upper case, or just the first letter

To support difficulty input as **full word or just the first letter**, in **upper or lower case**, we'll normalize the user input.

# What to change:

### Original:

```
java

System.out.print("Choose difficulty (easy/medium/hard): ");
String level = scanner.nextLine().trim().toLowerCase();
```

### Replace with:

```
java

System.out.print("Choose difficulty (easy/medium/hard or E/M/H): ");
String input = scanner.nextLine().trim().toLowerCase();
String level;
switch (input) {
   case "e", "easy" -> level = "easy";
   case "m", "medium" -> level = "medium";
   case "h", "hard" -> level = "hard";
   default -> {
        System.out.println("Invalid difficulty. Defaulting to 'easy'.");
   }
} **Copy ** Edit***

System.out.println("Loose difficulty (easy/medium/hard or E/M/H): ");

**The copy **Defaulting to 'easy'.");

**The copy **Defaulting to 'easy'.");
```

## Benefits:

- Users can enter: easy, E, e, Medium, m, HARD, H, etc.
- Falls back to "easy" if the input is unrecognized.

Would you like me to apply this change to the full code for you?

Adjust the choose the answer: use lower and upper

To allow the player to enter their answer in **upper or lower case** (like a, b, C, etc.), we just need to **normalize the input to uppercase** before processing.

# What to change:

# Original:

This already handles input in a case-insensitive way because it converts the user input to uppercase.

#### So no code change is needed here.

But to ensure **correctness throughout**, make sure all later references assume uppercase as well — which is the case in your current code (e.g., choice \_harAt(0) != q.getCorrectAnswer() compares uppercase letters).

### Why this works:

- .trim().toUpperCase() normalizes input.
- Matches only A/B/C/D, 5050, CALL, or EXIT.
- Keeps prompting until a valid choice is made.

Let me know if you want this applied across the whole code and returned as a file.

you can use CALL or call

Perfect — the current logic already handles CALL correctly because it uses <code>.toUpperCase()</code> on the input. But just to be **extra clear and consistent**, we'll ensure it supports both "call" and "CALL" (and any mixed casing like "call" ).

# Final Adjusted Input Handling Block:

Here's the improved and **fully validated input loop**, with support for:

- Upper/lower/mixed case answers ( a , B , Call 5050 , etc.)
- · Re-prompting on invalid input

In the save step, you can use lower and upper, or the first letter

Great! Let's update the save step decision logic so the user can enter:

- continue or c
- exit or e
- In any case (Exit, C, E, Continue, etc.)

### P Now:

- User can type C, c, continue, Continue, EXIT, e, etc.
- Invalid input is handled gracefully.

Want me to plug this into your full MillionaireGame.java file and send it back?

Also, show how much money you could have?

Nice idea! It adds tension to the moment — showing the **next prize** when the user is deciding whether to **continue or exit** at a safe step.

Suggest to start again or leave at all

Great idea — giving the player a chance to **start over** or **exit the game entirely** if they lose adds a nice UX touch.

\cdot\
could you add ASCII art for the questions?

Absolutely! **Q** Adding a bit of **ASCII art** at the start of each question adds a fun and dramatic feel — just like a game show.

Here's a simple, readable example you can insert before showing each question:

I want that the question look more realistic, like order 2:2

Ah, gotcha — you're aiming for the **question and answers to look more like a real multiple-choice layout**, like:

```
vbnet

Q: What is the capital of France?

A: Berlin B: Paris
C: Madrid D: Rome
```

This is cleaner and feels more like the actual game show style — let's make that happen!

put more ascii art for is the player win

Absolutely! ME Let's celebrate the win with some dramatic **ASCII art** when the player reaches \$1,000,000!