Dice Game- Documentation

User Guide

The program will run and print onto the terminal console and does not have a GUI. Expected inputs are made with a keyboard, utilising both letters and numbers as well as the enter key.

When starting the game up, you will be prompted to input your name, tell the program how many sides you want your specific die to have, and how many rounds you want to play. The game will then start, and you will get prompts for inputs such as guessing what number the die landed on.

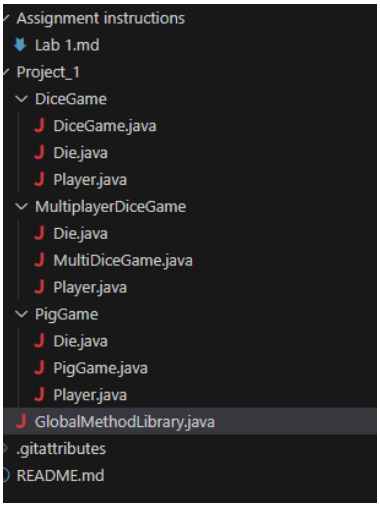
A few fail-safe checks have been made to avoid complication, such as making sure that the user inputs the correct values and letters to work with the program. If invalid inputs are given, then the user will be prompted to enter correct ones until they do.

I will summarise the flow in a bullet point list:

- Program starts.
- Game starts with introduction to the game.
- Asks the user for name.
- Asks the user to assign the number of sides on their die.
- Asks the user for the number of rounds to be played.
- --- Game of guessing starts ---
    - Game rolls the die the user has created.
    - Game asks the user to guess what number it rolled.
    - Game tells the user if its correct or wrong.
    - Repeat above process for the number of rounds declared.
    - Game ends and tells the user their total correct answers.
- Asks if the player wants to play again
    - If **yes –** Go back to bullet point 5: Set the number of rounds.
    - If **no –** Thanks the player for playing and ends the game + program.

Structure

My structure segment will include file specific structure and project structure due to me having done the 2 extra assignments because they affected how I arranged the files.



I decided to keep the individual projects inside their own folder to avoid potential complications and conflicting needs from each assignment. The assignments being dice game, multiplayer dice game, and the pig game. I've divided each class into its own file for easier management and visual clarity over what belongs where. There are duplicates of files such as "die.java" or "player.java" in each specific assignment, I see this a necessary structure since all three of these are technically their own game. It will also help with clarity when getting into the code knowing that all the files in their respective folder is dedicated to that specific assignment. In other words, there's no second-guessing if anything is part of another game in said files.

There is a single outlier in all of this and it's the GlobalMethodLibrary. This is where the fail-safe methods are kept, I will dive deeper into it later in the document.

There are general code structures that I try to follow across the board. One example being an extra spacing line to create a more visually pleasing look and it at least helps me segment the process more and create an invisible "box" that puts related code near each other.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    boolean replay = false;

    System.out.println(
            x:"Welcome to this simple Dice game!\nThe game is b
    System.out.print(s:"To start off. ");
    Player player = setupUser(sc);

    do {
        coreGameProcess(sc, player);

        System.out.print(s:"Do you want to play again? Y/N: ");
        replay = GlobalMethodLibrary.checkYesOrNo(sc);
        if (replay) {
            System.out.println(x:"----------------------------
        }
    } while (replay);

    System.out.println(x:"-------------------------Thanks for
    sc.close();
}
```

To start off, I will explain the structure of the main file "DiceGame.java" first, then work outwards from there.

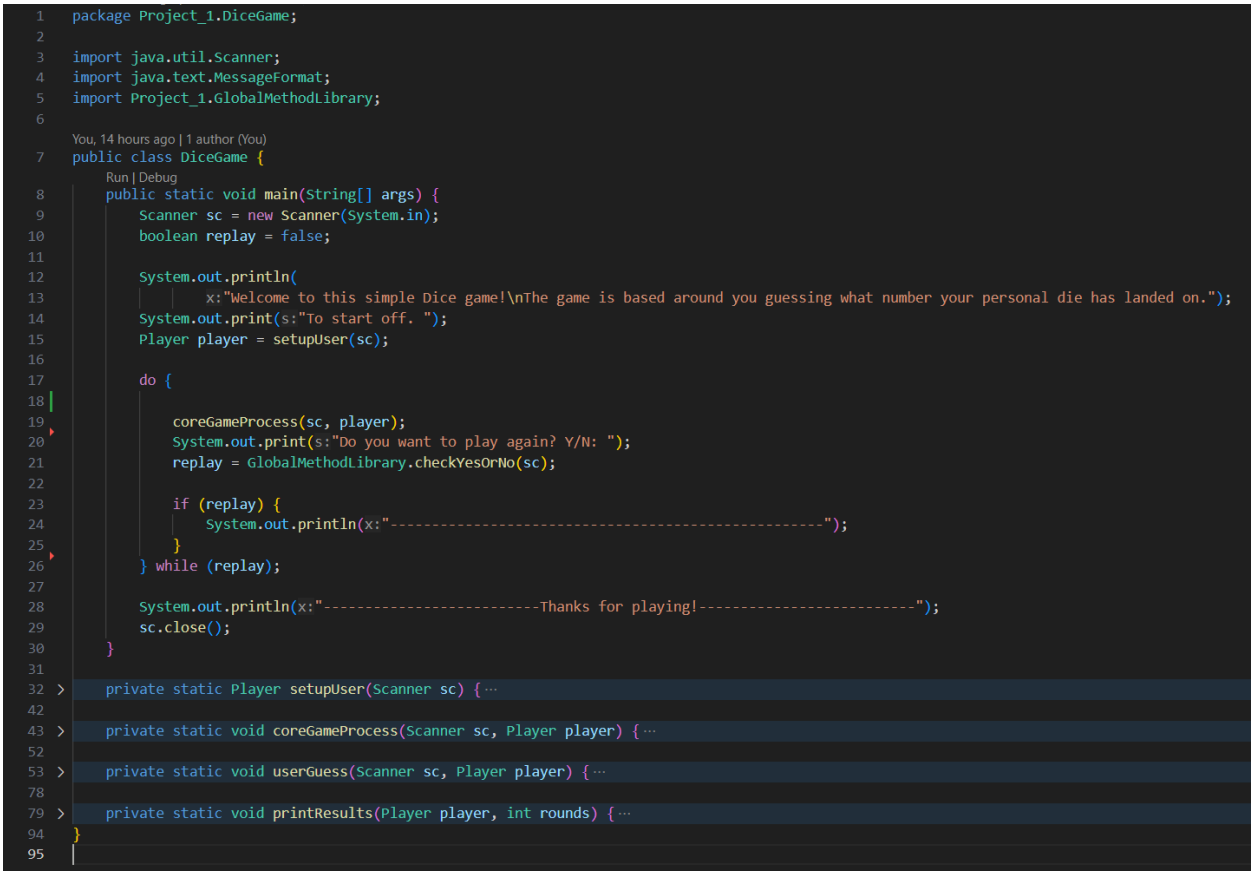
DiceGame.Java

To avoid a larger cluster of text I opted to break out multiple snippets into their own methods. Outside of easier readability, it also comes with the benefit of modularity and reusability. It may seem excessive at times, but the principle will prove beneficial in the long run, especially larger projects.

It is currently only extrapolated into 4 extra methods outside of the main method:

- main – Introduction, functions for game loop and exiting the game.
- setupUser – Where the user creates their player(object).
- coreGameProcess – The core method that contains the game logic.
- userGuess – Where the guessing and dice rolling is executed.
- printResults – Prints out the results to the user.

Within these methods are calls to other methods that are from the Player class, the Die class and the GlobalMethodLibrary class.



```java
package Project_1.DiceGame;

import java.util.Scanner;
import java.text.MessageFormat;
import Project_1.GlobalMethodLibrary;

You, 14 hours ago | 1 author (You)
public class DiceGame {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        boolean replay = false;

        System.out.println(
                x:"Welcome to this simple Dice game!\nThe game is based around you guessing what number your personal die has landed on.");
        System.out.print(s:"To start off. ");
        Player player = setupUser(sc);

        do {

            coreGameProcess(sc, player);
            System.out.print(s:"Do you want to play again? Y/N: ");
            replay = GlobalMethodLibrary.checkYesOrNo(sc);

            if (replay) {
                System.out.println(x:"-------------------------------------------------");
            }
        } while (replay);

        System.out.println(x:"-------------------------Thanks for playing!-------------------------");
        sc.close();
    }

    private static Player setupUser(Scanner sc) {···

    private static void coreGameProcess(Scanner sc, Player player) {···

    private static void userGuess(Scanner sc, Player player) {···

    private static void printResults(Player player, int rounds) {···
}
```

Player/Die.java

These are the 2 objects that we create in the program, and they follow a similar structure. For ease of reading and understanding (also best practice from what I can recall?), I've structured the class in the order of: Getters and setters > Constructors > Methods. I also marked with comments above each section for quick viewing. All classes that create objects will follow the same structure, to make sure it keeps being consistent.

We got a rough guideline over the structure and needed code within these classes, and I've tried to keep it streamlined across all files. Some guidelines wanted a specific import of arrays, this was only needed for a single assignment which pushed me to create separate files of the same classes for each game.

```java
package Project_1.DiceGame;
import java.util.Random;

public class Die {
    private int numberOfSides;
    private int currentValue;
    private Random random = new Random();

    // getters, setters
    public int getNumberOfSides() {
        return numberOfSides;
    }
    public int getCurrentValue() {
        return currentValue;
    }

    // Constructors
    public Die(int numberOfSides) {
        this.numberOfSides = numberOfSides;
    }

    // Methods and misc
    public void roll() {
        this.currentValue = random.nextInt(1, numberOfSides + 1);
    }
}
```

GlobalMethodLibrary.java

This is the outlier file that is not part of any specific assignment, but is the one being imported to all of them. This file can be seen as a “general” use case file that you can call to for methods that would be commonly used. One of them being a fail-safe method to check if the user input an int/number and not strings/letters.

```java
public static int checkIfNumber(Scanner sc) {
    int userInput = -1;

    while (true) {
        if (sc.hasNextInt()) {
            userInput = sc.nextInt();
            if (userInput > 0) {
                clearScanner(sc);
                return userInput;
            }
        } else {
            sc.next();
        }
        System.err.print("Invalid input, please enter a number above 0: ");
    }
}
```

In the current version of this code, I only have 5 methods in this file.

- setGameRounds
- checkIfValidString
- checkIfNumber
- checkYesOrNo
- clearScanner

All the methods are quite self-explanatory, but they are used at least twice across multiple assignments. Creating a global methods library can be good to avoid rewriting multiple identical methods. This will also eliminate the potential human error that can occur when rewriting the same method multiple times, for example mistyping a variable. Troubleshooting is also much easier since it all falls back to a single point rather than multiple.

A library like this can also be good for future use, especially when its mostly standalone methods that are there to validate or verify a variable. For example, if something is a number or not. This means that this file is not necessarily bound to a specific project but can be applied to anything the programmer/tester is working on.

Final words

I will not dive into the two other games due to the fact that they follow the same general guidelines with some changes here and there. But there might be some slight differences due to time of writing and if I was in goblin mode or not. More or less, it depends! 😉

Thank you for reading this documentation!

Tan