

## Assignment 1: Board Game Tracker

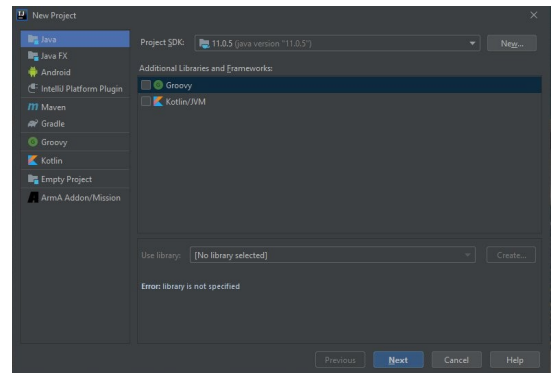
- ◆ See website for due date and late penalty info.
- ◆ Submit deliverables to CourSys: <https://coursys.sfu.ca>
- ◆ This assignment is to be done **individually**. Do not show another student your code, do not copy code from another person/online. Do not reuse your previous work (even if retaking the course).
- ◆ You may use general ideas you find online and from others, but your solution must be your own.
- ◆ See the marking guide for details on how each part will be marked.

### 1. Board Game Tracker

The course website has a capture of some sample output showing how the entire application operates.

#### 1.1 Requirements

- ◆ In IntelliJ, create a **plain “Java” project** (not Groovy, Gradle, Maven, or Kotlin/JVM). See image on right.
- ◆ You must have (at least) the following three classes.
  - A class to represent a board game:
    - ▶ Game name may be more than one word (like “Ticket to Ride”)
    - ▶ Game weight, between 1 and 5 (such as 2.35).
    - ▶ Number of times played
    - ▶ Your game class must correctly implement `toString()`, as discussed in lecture. Remember, this is for debug output, not to format it for the user interface!
  - A class for a **text user interface (UI)**:
    - ▶ In this class, put all code that interacts with the screen and the keyboard.
    - ▶ Be careful not to have much duplicate code in your application! Use functions.
    - ▶ Limit the levels of control structures you nest! Your functions should be no more than a screen-full of code.
    - ▶ You are welcome to add extra classes as you see fit! How could you make it clean and maintainable code?
  - A class to start your application
    - ▶ Contains a `main()` method instantiates the text UI and starts the application running.
    - ▶ You get to decide where you store the collection of games.
- ◆ For this assignment, it is fine if all your classes are in one package.
- ◆ Hint: Can you think of a way to extract some code out of the text UI class into its own class? Doing so will reduce the size of the text UI? For example, what about a general text menu class?
  - Could your text UI instantiate your general menu when it needs to do menu things?
  - **Could you put as much *general* functionality in the menu class as you can so that there is good code reuse?**
  - This menu class will be part of your UI, so it’s OK for it to interact with the screen and the keyboard.



## 1.2 Text Interface Requirements

- ◆ When you prompt the user to choose a menu option, if the user enters an invalid number you must re-ask the user to enter a valid value.
  - You may assume user always enters correct *type* of data: when asked for an `int`, it is OK if the program crashes when the user enters a non-`int` such as 'A'.
  - *Hint (optional): Avoid duplicate code; make a method that does this.*
- ◆ **Main Menu Option: List games**
  - List the name, weight, and number of plays for each game.
  - Number the games from 1.
- ◆ **Main Menu Option: Add a new game**
  - Create a new game.
  - Ask user for required game info:
    - ▶ Name must be 1 or more characters long; may be multiple words
    - ▶ Weight must be between 1.0 and 5.0, such as 2.5
- ◆ **Main Menu Option: Remove game**
  - List the games currently in the system.
  - Allow user to select a game (by number), or 0 to cancel.
  - Entering an invalid number (like -3) handled by application. Entering invalid data *type* ("hello") need *not* be handled.
- ◆ **Main Menu Option: Add game played**
  - Similar to "remove game", user selects a game to work with (or 0 to cancel).
  - Increment the number times the game has been played.
- ◆ **Main Menu Option: Debug dump of game details**
  - For each game in the game-list, call its `toString()` and print the result to the screen.
- ◆ **Main Menu Option: Exit**
  - Exit the application.
- ◆ Your text UI need not match the sample *exactly*, but it should be of equal quality.

## 1.3 Coding Requirements

- ◆ Your code must conform to the programming style guide for this course; see course website.
- ◆ All classes must have a class-level JavaDoc comment describing the purpose of the class.
- ◆ Functions should not be longer than about 30 lines long.
- ◆ Lines of code should not be longer than about 120 characters long.
- ◆ Code should not be more than 3 control structures deep (ex: `if` in `loop` in an `if`).
- ◆ Your class' `toString()` method may only be printed to the screen as part of the "Debug dump" option; otherwise it should be your UI code that is generating screen output.

## 1.4 Suggestions

- ◆ Think about the design before you start coding.
  - List the classes you expect to create.
  - For each class, decide what its responsibilities will be.
  - Think through some of the required features. How will each of your classes work to implement this feature? Can you think of design alternatives?

## 2. Deliverables

Submit a ZIP file of your project to CourSys. See course website for directions on creating and testing your ZIP file for submission. All submissions will automatically be compared for cheating.

### 3. Versions

1.0 Initial release

1.1 Jan 15: Corrected game weight to be “Weight must be between 1.0 and 5.0, such as 2.5”