OOP Assignment (Jan 2022)

Specification

In this assignment, you are asked to develop an object-oriented library that allows users to play the game of Monopoly. A PDF version of the rules has been provided (ignore the short game rules at the end). It is not expected that you provided a full, working implementation of the game: rather, the intent is to expose you to numerous opportunities to showcase your object-oriented design and implementation skills on a concrete, relatable example.

Your library should expose a class `Game` to its users, which can be instantiated to create a new game (with given initial parameters, if relevant). The public instance properties and methods of `Game` should allow the game to be played (according to the rules) and all necessary game information to be accessed at all times. More precisely, the `Game` class must provide the API to your library:

1. **It must be possible to play the game and access game information in a programmatic way. This should follow an object-oriented design, delegating responsibilities to suitable sub-components.** As a rule of thumb, imagine whether a suitably well-crafted algorithm would be able to fully play the game through your public `Game` interface. Common issues:
    - Printing information to console and/or requiring user input from console (or other UI).
    - Exposing gameplay methods, but not giving access to the full game information.
    - Inadequately delegating responsibility for action execution and/or information exposure to sub-components.
2. **Code executing illegal actions must not compile (preferred), or it must otherwise raise error at runtime. This includes playing illegal moves and other illegal modification of the game state/information.** In other words, the public methods and properties of the `Game` class and any sub-components which it exposes must not allow illegal actions to be performed. Common issues:
    - Methods can be successfully invoked with illegal parameters, or in illegal order.
    - Mutable public properties allow illegal values to be set.
    - Read-only properties expose objects which can themselves be illegally modified.

The user interface—console-based or otherwise—is out of scope for this assignment and will not contribute to its mark. If you wish to provide one, it should be built entirely on top of the `Game` API. It must be possible to create multiple independent instances of `Game` at the same time (e.g. to use the library as part of an online game server).

AI players and other strategy-related features are also out of scope for this assignment.

Task

Design and implement a library with the functionality described above, using an object oriented approach:

- Low-level data used by the library should be structured by means of suitable types (try to avoid classes for lightweight data) and validated where necessary. As much data validation as possible should be delegated to the static type-checking.
- High-level components used by the library should be structured through interfaces and classes, with public methods providing access to external functionality and private/protected

methods providing access to internal functionality.
- Your design should make use of adequately chosen types, interfaces and access control to ensure that the library cannot be misused, while allowing individual components to be safely exposed to the users.
- Where relevant, you should use advanced type features (e.g. polymorphism, advanced types, type operators), idiomatic language features (e.g. destructuring, `for..of` loops, iterators, `map`/`filter` on arrays), and object-oriented patterns (e.g. factory, flyweight, façade, global object). You are advised not to use the singleton pattern unless you find a very compelling reason to do so (in which case, you should explain it).
- Where relevant, you should implement reusable generic data structures (e.g. trees, graphs, queues/stacks).

Your code should be clear and concise: long or complex method/function bodies should be avoided whenever possible (e.g. by delegating to helper methods/functions, or to other components). Where long or complex code is unavoidable, the individual steps should be concisely documented using single-line comments.

Submission

You should submit your TypeScript code, including the `tsconfig.json` file and a brief readme file (plain-text or markdown) explaining how to compile your code (it can be as simple as "Run `tsc` in this folder.").

You should clearly document your code to explain its design and implementation. You should not include a separate report for this purpose: the code documentation should be enough on its own. In particular, all classes, methods, functions, interfaces and types should be suitably documented using TSDoc or ESDoc.

Assessment Criteria

Assessment will be based on the following criteria:

- Can you use an object-oriented, type-driven approach to design libraries, algorithms and data structures?
- Can you implement libraries, algorithms and data structures using object-oriented languages in a structured, type-safe, re-usable and maintainable way?
- Can you explain the rationale behind your object-oriented design and implementation?

Clear code and idiomatic usage of language features will be additional considerations for distinction.