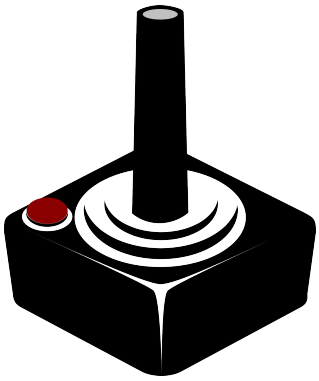
Note that this is the Project Final and will be counted as 35 of the Project/Exam grade.

Overview



Shall we play a game?

The purpose of this project is two-fold:

1. Serve as an assessment of the topics covered this semester—particularly, the topics that illustrate *Object-Oriented Design*. Beyond demonstrating your knowledge of these discrete topics, the project will also test your ability to integrate these topics into a cohesive whole.
2. Give you the opportunity to create a playable game, largely designed by *you*!

Background

Before the Project, Part I, we covered the following topics:

- **Control Flow**, sequencing code, conditionals, and repetition through loops.
- **Functions**, breaking up and modularizing code through abstraction.
- **Classes**, providing the ability to *encapsulate* member data and methods, and separate implementation from use through *information hiding*.
- **Inheritance**, establishing *is·a* relationships among classes, allowing for *polymorphism*.
- **Interfaces**, providing the ability to establish *can·do* contracts for classes, and enabling multiple inheritance.
- **Value Types**, distinguishing between value and reference data types and how they behave differently.

Since the Project, Part I, we have added the following topics:

- **Exception Handling**, providing a way to avoid application crashes, while providing meaningful feedback.
- **Generics**, enabling classes to be parameterized by type(s).

Finally, we have learned how to plan for—and document—these concepts through UML diagramming.

Instructions

You will work to combine the various concepts and previous assignments completed this semester to design a game that utilizes a number of assets.

Part I

Extend the class and interface hierarchy created for the *Project, Part I* to include the following. Note that your final project **must include all requirements outlined in the *Project, part I* handout.**

1. Create a `Container<T>` class that implements a collection with a generic type. The `Container<T>` class must:
   - Include a method, `Size()`, that reflects the *fixed size* of the Container.
   - Store the objects of generic type `T` in a member array named `Entries`.

- Two constructors—one that assigns a default `Size` of 16, and one that includes a `Size` parameter.
- An overridden `ToString()` method, which returns a string representation of the container's contents. Empty slots should be designated as such.
- An `public bool Add(T entry)` method, that adds the given entry to the `Entries` array in the first slot available. If no slots are available, then the entry is not added. This method returns a `bool` indicating whether or not the entry was successfully added to the `Container`.
- A `public T Remove(int slot)` method, that removes the entry from the given slot in the `Entries` array. If the entry does not exist, then the method returns `null`. If the entry does exist, it is removed and returned.

2. Rework the `PlayerCharacter` class to use `Container<Item>` as its `Inventory`.

Part II

Create a program driver that allows serves as a playable game incorporating the assets you have defined as a class and interface hierarchy. The game should include *at least the following components*:

- A menu-driven character creation process.
- Player-choosable actions, including:
  - Those actions defined in the `ILocatable` and `IMobile` interfaces.
  - Attacks or other interactions with the environment.
  - The ability to use `Item`s. Note that `Item`s should define all of the methods in the `IUsable` interface. `Item`s should be able to be picked up and dropped.
- An interactive game environment (e.g., combat, movement, *etc.*).
- A visualization of the game environment. This may be done in any way you desire, but it must be a valid interactive game interface, and not just a logging of messages to the console. See *rogue*, *nethack*, or *Zork!* for playable examples of a console-based game environment.

Part III

A UML class diagram detailing the relationship between your classes and interfaces.

Submission

Code should be submitted as a `repl.it` link. Please include your UML diagram in your Canvas submission as well. You will present your game to the class during Finals week!