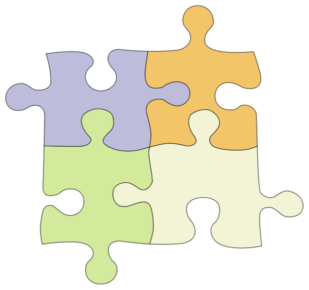
Note that this is the first component of the Final Project and will be counted as 2/5 of the weighted Final Project/Exam grade.

Overview

The purpose of this lab is to stop, think, and integrate what you have done this semester up through now. In addition, you will flesh out your classes to form a more complete basis for an actual, playable game.


Put it together!

Background

This semester has covered the following topics to date:

- **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.

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 single class and interface hierarchy that will serve as a basis for player and non-player characters and creatures.

Part I

Create a class and interface hierarchy that includes the following. **Note that the addition of interfaces and the requirements that accompany them makes this project more involved than previous labs.**

The following are provided for you.

1. The following interfaces:
   (a) `ILocatable`, for any objects that may have a location.
   (b) `IMobile`, for any `ILocatables` that should have the ability to move.
   (c) `IActionable`, for any class that will be able to take an action, such as attack or defend.
   (d) `IUsable`, for any object that can be used, such as items.
2. The `Direction` **enum**, which will contain the eight cardinal or intercardinal directions.
3. An *incomplete* `Creature` class, which you will need to complete. It:
   (a) implements the interfaces `IMobile` and `IActionable`. *Note that* `IMobile` *already implements* `ILocatable`.
   (b) includes the code from Lab 3.

Part II

Add the following classes to your hierarchy.

The following are NOT provided for you.

1. At least three creature *type* classes. These classes will inherit directly from `Creature`. You may use the ones from Lab 3 as a starting point if you wish.
2. At least five monster classes. These classes will inherit directly from the creature types specified above. You may use the ones from Lab 3 as a starting point if you wish.
3. An `Item` class. This class should implement the interfaces `ILocatable` and `IUsable`.
4. At least two item *type* classes (*e.g.,* `Potion`). These classes will inherit directly from `Item`.
5. At least three individual items. These classes will inherit directly from the item types specified above (*e.g.,* `HealingPotion`).
6. A `PlayerCharacter` class. This class will extend the `Creature` class. It should include all of the necessary components of the classes it derives from, and:
   (a) The `Name` property.
   (b) The `Race` property, which consists of an **enum** outlining all of the possible races from Lab 2.
   (c) An `Inventory` property, which is an array of `Items`.
   (d) A `RollStats()` method, which will generate stats according to the "4d6, drop the lowest rule." Additionally, let the player choose the order of the stats, so that may assign each value to the desired attribute.

Finally, note that all constructors necessary to instantiate these classes should be included as well.

Part III

Create a program driver that allows players to create and manipulate all of the above.

Submission

Code should be organized as a `repl.it` project. Alternatively, you may use Visual Studio to create this project, as it will grow to be quite large.