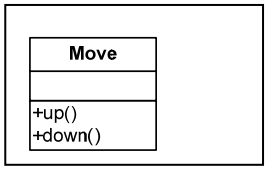
Comments on Homework 9

Overall

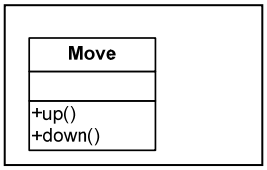
The assignment listed several things that were to be included in this assignment. One of these asked why the particular design approach was chosen. Most people did not answer this question.

Another question asked for the public interface for each class….remember, a classes public interface should be evolved from or based upon the use cases.

While writing something like ….



certainly graphically depicts the public interface and meets the strict letter of the requirement, it really contains no information. It's important to briefly identify what each such function member is to do…



For the class Move, the function member

up()

Moves a player one space forward unless there is a wall. If there is a wall, the player looses a turn.

down()

….

Now, someone other than the person who wrote the specification or designed the architecture can proceed with the details of coding the method.

The design is the difficult, challenging, creative, and fun part of the project…the coding is simply mechanics. Anyone can write code…not everyone has developed the skills for creating a clever, robust, and interesting design….this takes a lot of practice.

As the Final Project requirements state, the final deliverables must include:

1. A requirements specification
2. Your design documentation including: a functional diagram, CRC cards, and UML class diagrams.

3. A test plan.
4. Test cases.
5. Source code.

A number of people submitted a first cut at a portion of those deliverables with Assignment 9. Overall, those looked good. There was nothing glaring missing. Please include the updated versions reflecting any changes in the final delivered package.

Further, it is noted that the design must contain and use

1. At least 5 or 6 classes – these must be legitimate classes….not classes created for the sole purpose of creating classes.
2. Copy constructor
3. Assignment operator
4. Overloaded insertion operator functions as appropriate.
5. Composition
6. Inheritance and polymorphism

Design a First Cut

Looking over the preliminary designs based upon the functional decompositions, CRC cards and class diagrams, several problems stood out.

1. Some people are not showing any inheritance; most who do use it are not showing any inheritance hierarchies utilizing polymorphism in their designs.
2. I do not recall seeing any assignment operators or copy constructors in any of the classes public interface.
3. The specification stipulated that the game was to include things such as gold, fake gold, a teleporter etc. There were a number of designs that did not include any such things; that were only implementing movement around the board by the player, the gnome, and the leprechaun. This is not sufficient.

   In several other cases the things were included, but, none of the classes supported methods to do anything with them. Methods are needed to pick things up or to do something with them.
4. The most serious problem I encountered was that many people are significantly misusing inheritance.

   I found a number of cases such as people or gnomes etc. classes as being subclasses of rooms or walls. These, however, were not the only such examples; there were many others.

   There were other cases in which multiple inheritance was being used, as a illustrative example, to create a child class *person* with the parents such as the *game board* and *rooms*.

   In other cases, circular inheritance was being used…*A* was a child of *B* and *B* was a child of *A*.

When inheritance is used, the child should be '*A Kind Of - AKO*' the parent. The child class should exhibit the '*IS A*' relationship with the parent…*an apple ISA fruit an apple is A Kind Of fruit*.

An apple is *not* a subclass of tree or of garden.

Multiple inheritance can be tricky. In addition to creating object oriented spaghetti code, it's very easy to create some rather dangerous and error prone situations using it. This is one of the reasons that Gossling left it out of Java.

In particular, one needs to be very careful when creating a child class as a subclass from multiple (unrelated) parents. This should not be done simply to get something from such parents. It should only be done after it has been well thought out and there is a specific and constrained advantage to do so. In many many cases, composition is a much better choice.

Composition and aggregation are other ways for extending the class type system. In such cases, we should be using the 'HAS A' relationship. The game board HASA room…*not* the room ISA subclass or child of game board. A room *is not* AKO game board.

5. In their class diagrams, many people are showing all of the data members as public. These should be all be private.