Recitation Guide for January 28th, 2008

I. Housekeeping items
   a. Quiz 1
      i. Quiz 1 will be held in class on Wednesday January 30th.
      ii. Pre-quiz reflects topics all topics necessary for the quiz but not the length! There will be no coding from scratch on the quiz.
      iii. No review session.
      iv. KNOW YOUR GRADING TA! WORTH MAJOR POINTS ON THE QUIZ!!!!!
   b. Exam 1
      i. Exam 1 date has been pushed back. Exam 1 will be held in class on Friday February 8th.
      ii. Exam 1 review session Wednesday February 6th. Time and location TBA.

II. Pre-Quiz

III. More topics (if there is time)
   a. super
      i. super constructors
         1. Essentially a constructor for a class creates and sets up the instance. Certain instance variables are set to incoming parameter value or default value within the constructor.
            a. If a class is a blueprint, then the constructor builds the model based off the blueprint.
         2. When a child class extends a parent class, the child class inherits the non-private variables and methods from the parent. Calling `super()` or a super that takes in parameters will go and set up those inherited variables.
      ii. super methods and method overriding
         1. Any method inherited from a super class can be used without any mention of the super class because it is implied. The following lines are equivalent when calling them within the child class:

```
super.method();
method();
```

         2. When a method is overridden, it means a method with the exact same name, return type and number, type and location of the parameters are the same.
         3. In the provided classes, `Major` and `EngineeringMajor,` the method `toString` of `Major` is overridden by `EngineeringMajor`. Thus if we were within `EngineeringMajor` and wanted to call the `toString` located in `Majo`r, we must use the `super`.
         4. Real life example: inherited names

a. It was more prevalent to see a particular name carried down through the generations by a male heir. However, it would get pretty confusing pretty fast legally if both father and son had the EXACT same name. Thus it was necessary to append a Sr. or a Jr. or Roman numeral numbering.

b. Back to our provided example, `Major` and `EngineeringMajor`, and the `toString` method. Both classes have the same method with all of the extra specifications described previously so if you try to call `toString` inside `EngineeringMajor`, the compiler will assume that you are talking about the `toString` inside of `EngineeringMajor`. Thus is necessary to tag on the `super` to call `toString` Sr.

b. Casting

A definition of casting is a type conversion allows programs to treat `Object`s of one type as one of their ancestor types to simplify interacting with them. However, because `int` and `double` are not really `Object`s, but are primitives, we cannot apply this definition. A more general way is to just think of casting is to see it as a type conversion which is only allowed with compatible types where compatibility may be determined through inheritance or belonging to the same category of primitives.

Here is a line of code:

```
double value = Math.cos(Math.PI * 0);
```

What if value was actually an `int` like so:

```
int value = Math.cos(Math.PI * 0);
```

Just looking at the code, you know you would get an error, because the types on either side of equal sign do not match. Currently we have:

```
int = double
```

We need the same type on both sides to get past the compiler. Because we need `int` for our `LetterTurtle` method, we are going to cast the right side of the equal sign to `int`:

```
int value = (int)(Math.cos(Math.PI * 0));
```

Here we finally have the same type on both sides:

```
int = int
```

It is important to note that to change a `double` to an `int`, the `double` is truncated at the decimal point. Essentially this means the decimal and everything to the right of it is thrown away. It is very important to note that does casting will not round the value up or down.

Examples (done in Dr. Java's interactions pane):

```
> (int)5.123
5
> (int)6.523
6
> (int)-9.791
-9
```