CS 158 Lab 9 February 16, 2010
Using Collections and Iterators

1. Go to http://faculty.valpo.edu/jcaristi/cs158 and follow the link called "Containers". There you will find the code for the textbook author's **BasicCollection.java**. Create a new project called "Chapter 4", and in that project, create a new package called **containers**, and in that package, put the **BasicCollection** class that the author has built. Fix up any incorrect package references necessary to make everything compile. Also, please download the author's JUnit test class for **BasicCollection**. You should rename it so that it is called "**BasicCollectionTest**", and you should delete his steps 7 and 8, because you will use Netbeans to run all the tests. Add that class to the **containers** test package and fix everything up so that the tests run and you get a green bar.

2. Create another package called **applications**. In this package create a new class called **Asteroid**. Add protected integer instance variables to this class called x, y, dx, and dy. Create another class in this package called **Board**. Add a static integer constant to **Board** called **SIZE** and set it to 100 for now. This will represent a board that is 100x100. In this class, add a protected **BasicCollection** called **asteroids** that can contain only **Asteroid** objects. Add a constructor that instantiates **asteroids**.

3. Back in the **Asteroid** class, add a constructor that initializes x and y to random integers that are nonnegative and less than Board.SIZE. Add methods called **getX** and **getY** that do the usual things.

4. In the **Board** class, add a new method called **addAsteroid** that creates an **Asteroid** object and causes it to be inserted into the **asteroids** collection. This method should have no arguments, and the **Asteroid** that it creates should be "anonymous". Create a JUnit test class and method for testing **addAsteroid**. In the test, simply test the size of the collection before and after adding a few asteroid objects to it.

5. Create a new method of the **Board** class called **destroy**. This method should return a boolean and take two integer arguments that represent a position on the board. In this method, create an **Iterator** for the **asteroids** collection, and use it to go through the collection looking for an asteroid that is located at the same x and y as the arguments that come in. If at least one asteroid is found to be located at that point, remove it from the collection, and the method must return true. If no such asteroid is found, the method simply returns false.

6. Write a test method for **destroy**. Start by adding 10 asteroid objects to the collection. Then get an iterator for the collection. Now, since in this test we know there are 10 asteroids in the collection, and since the **Asteroid** class is in this package and the **Asteroid** instance variables are protected, we can change the location of any asteroid in this test. So use a loop that runs 10 times and has the iterator give us each **Asteroid** one by one and sets the location of the asteroid to (i, i) so that we know exactly where it is. Then assert that you cannot destroy an asteroid located at (1, 0), but you can destroy one at (3, 3), and the list changes size accordingly.

7. Add a method in the **Asteroid** class called **setVelocity** that takes two integer arguments for the horizontal and vertical components of the velocity of the asteroid, and sets dx and dy to those values mod the size of the board (so if someone supplies an argument value larger than the size of the board, it will automatically be reduced).

8. Add a method to the **Asteroid** class called **move** that simply causes the asteroid to change its position by adding the amounts stored in dx and dy to x and y respectively. Also, you will have to take the results mod the size of the board, so that the asteroid will "wrap around" if it ends up going off the end of the board. Create a test method for **move** that sets an asteroid's velocity to (2,-1) (for example), gets the asteroid's current location, tells it to move, and then asserts that its new location is where it should be. Don't forget to "mod" the computations so that locations are on the board. You might want to create a LOT of asteroids (maybe about 300 of them) and make sure they all are where they're supposed to be after they move.