Lab 8

Goals:

- Learn about `abstract` classes via the ProjectileWorld class
- Learn about developing your own frameworks

Using the Konqueror browser, go to the course web-page: http://www.cs.uni.edu/~fienup/cs062s06 and click on the Lab8.zip link and "Save to disk" in the cs2 subdirectory created in lab 1. In a shell window, change to the cs2 subdirectory. Use the "ls" command to list the files in the directory and look for the file "Lab8.zip". Decompress the "Lab8.zip" using the command "unzip Lab8.zip", then change to the Lab8 subdirectory. Compile and run the CannonGameDriver.java.

Part A: Using the Projectile Interface

From lecture yesterday, we talked about modifying the Cannon world so it could fire any object that satisfied the Projectile interface:

```
import java.awt.Graphics;
import java.awt.Point;

public interface Projectile {
   public void    move   ();
   public void    paint  ( Graphics g );
   public boolean northOf( Point referencePoint );
   public boolean southOf( Point referencePoint );
   public boolean westOf ( Point referencePoint );
   public boolean eastOf ( Point referencePoint );
} // end Projectile interface
```

Steps:

1) Refactor the CannonBall class so that it also implements the Projectile interface.

2) Refactor the CannonWorld class to use the Projectile interface - This causes something of a problem, because we cannot create an instance of an interface. However, we can use our refactored CannonBall that satisfies the Projectile interface. (We'll explore a better alternative in part B of the lab) These changes are actually pretty straightforward...

- Change the instance variable to be a Projectile.
- Leave all null assignments and move/paint messages the same.
- Change requests for the CannonBall's instance variables (and the calculations on those values!) into requests to a Projectile.

Part B: Using a ProjectileWorld Framework

What we did in part A works, but if we want to fire some other type of projectile, then we would need to copy the CannonWorld.java file and edit it to construct the different type of projectiles. Remember that duplicate code is hard to maintain! A better alternative is to make the CannonWorld class into a ProjectileWorld framework. You can do this by:

- renaming the CannonWorld class to be ProjectileWorld to reflect its more general nature (Don't forget to change the file name too.)
- replacing the code that creates the CannonBall with a call to an abstract method `makeProjectile` that is defined as:

```
abstract protected Projectile makeProjectile( int startAngle );
```

- this requires that the `abstract` modifier be add to the class definition of the ProjectileWorld class too.

ProjectileWorld is not just one program now. It is either *zero programs* or *as many programs as you can imagine*. We can no longer run the ProjectileWorld program by itself. But, our new ProjectileWorld allows us to build a working program that fires the projectile of our choice in just two steps. To implement a working ProjectileWorld, a client programmer need only:

1. Write a class that implements the Projectile interface, e.g., our refactored CannonBall from part A.
2. Write a subclass of ProjectileWorld that includes a makeProjectile(...) method to create an instance of the new Projectile class.

Since you already have step 1 done, you only need to write a subclass of ProjectileWorld, called RevisedCannonWorld that includes a makeProjectile(...) method to create an instance of the CannonBall. Finally, modify the CannonWorldDriver to use the RevisedCannonWorld class.

Turn In:

After you complete the lab, turn in a print outs of ProjectileWorld.java and your refectored CannonBall.java. (If the printer stops working again, raise your hand and you can demo your modified program for me or the TA.)