In this assignment you will gain experience with generics, protected access, and perhaps abstract methods. In addition, you will take a few additional steps towards having a full simulation.

**Multiple Shape Packages and Protected Access.**
Several of you have put all of your shapes, both atomic and composite, in one package. Distribute the shapes into two or more packages – for instance one package could hold all of the atomic shapes and the other the composite ones. There are several other useful ways to also divide the shapes.

Once you do this, you may realize that certain variables are no longer visible in a subclass, because they are declared in a superclass in some other package. Declare such variables as protected.

**Generic Collections**
So far, you have not been able to share code among collections whose elements are of different types. Use generics to share as much code as possible between collections with different element types. You should not have to change code that uses these collections, as you can replace your existing collections with (direct or indirect) extension of generic collections.

**Implementing Simulation Constraints**
You should add the following constraints to your simulation:

1. Do not allow an approaching knight to move past the knight crossing area unless he has been passed. Make some reasonable interpretation of the term “moving past the knight crossing area”.
2. Allow an approaching knight to be passed if either (a) the guard has failed and thus fallen down, or (b) knight has answered N questions, where N is some named constant or scene parameter.
3. The say command has no effect if the approaching knight is not in the standing area. Again make some reasonable assumption of what it means for a knight to be in the standing area.
4. If the collection of approaching knight is empty, then the move knight command should apply to the collection of departing knights. Of course, if both collections are empty, then the command has no visible effect.

Abstract Methods (Extra Credit)
Find opportunities to make “reasonable” use of abstract methods. In particular, see if you can unite all avatar classes into a common abstract avatar class, with the concrete classes responsible only for implementing factory abstract methods – these are abstract methods that instantiate classes and returning these instances.

Fixing Earlier Problems (Optional and Ongoing)
As mentioned in class, you will get a chance to get back points you lost for making mistakes or being late. In the spirit of the 50% Greek debt write-off, you get back 50% of the lost points. This means:

1. If you lost $N$ points on some mistake, and you correct it completely, you get back $N/2$ points. The correction should be part of the code you submit for the last assignment. If you lost points on some code that is not part (in some form) of the last assignment, then it is too much work for the TAs to verify your correction.
2. If you lost $N$ points for being late, you will get back $N/2$ points.
3. If you moved your assignment submission dates by $K$ weeks, and you manage to do $N$ points worth of work on the last $K$ assignments, then you get $N/2$ points rather than 0.

With the last assignment, you will provide a document explaining what you lost and what you have done do get back these points. The exact format of the document will be communicated to you by the TAs. In the meantime, you may start working on fixing problems.