Com S 227
Fall 2015
Miniassignment 1
40 points

Due Date: Monday, September 21, 11:59 pm (midnight)
"Late" deadline (25% penalty): Tuesday, September 22, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Blackboard. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly, you will receive at most half credit. See the section "What to turn in" at the end of this document

Please start the assignment as soon as possible and get your questions answered right away. It is physically impossible for the staff to provide individual help to everyone the night before the assignment is due!

Note that the next homework may be assigned before this one is due.

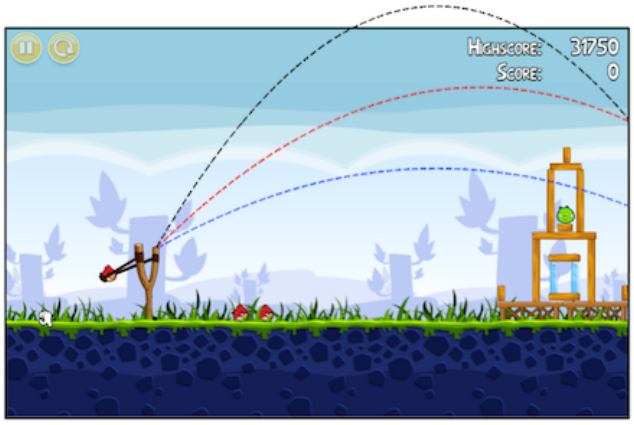
Tips from the experts: How to waste a lot of time on this assignment

1. Start the assignment right before it's due. That way, if you have questions, the TAs will be too busy to help you and you can blame the staff when you get a bad grade.
2. Don't bother reading the javadoc, or even the rest of this description, especially the "Getting started" section. Documentation is for wimps.
3. Don't test your code, and whatever you do, don't think about test cases or about how the code should work before you write the code.
4. The main guiding principle is: *Try to write all the code before you figure out what it's supposed to do.*

Overview

The purpose of this assignment is to give you some practice with the process of implementing a class from a specification and testing whether your implementation conforms to the specification.

For this miniassignment you will implement one class, called **`Projectile`**, that encapsulates the position and velocity of an object moving in two dimensions. One possible use for such a class would be to represent a moving object or "sprite" in a 2D video game. It is slightly more complex than the **`Atom`** or **`Basketball`** classes that you saw in Lab 2, so be sure you have done and understood Lab 2.



http://sites.davidson.edu/mathmovement/algebra-of-angry-birds/

Specification

The specification for this asssignment includes

- this pdf,
- any "official" clarifications posted on Piazza, AND

- the online javadoc, found below:

http://web.cs.iastate.edu/~cs227/homework/mini1/doc/mini1/Projectile.html

The javadoc contains precise descriptions of the required methods and their behavior as well as a brief overview.

Where's the main() method??

There isn't one. Like most Java classes, this isn't a complete program and you can't "run" it by itself. It's just a single class, that is, the definition for a type of object that might be part of a larger system. To try out your class, you can write a test class with a main method, analogous to **`AtomTest`** in Lab 2. For example:

```
import mini1.Projectile;

public class ProjectileTest
{
  public static void main(String[] args)
  {
    // Initial position (1, 2) and initial velocity (3, 4)
    Projectile p = new Projectile(1.0, 2.0, 3.0, 4.0, 10);

    System.out.println(p.getX());     // expected 1
    p.timeStep();
    System.out.println(p.getX());     // expected 4
    p.timeStep();
    System.out.println(p.getX());     // expected 7
  }
}
```

There is also a specchecker (see below) that will perform a lot of functional tests, but when you are developing and debugging your code at first you'll always want to have some simple test cases of your own as in the **`main`** method above.

We may also post some other sample applications that use the **`Projectile`** class.

Getting started

*Smart developers don't try to write all the code and then try to find dozens of errors all at once; they work **incrementally** and test every new feature as it's written. Here is a rough guide for how an experienced coder might go about creating a class such as this one:*

1. Create a new, empty project and add a package called **`mini1`**.

2. Create the **`Projectile`** class in the **`mini1`** package and put in stubs for all the required methods and the constructor, as shown in the javadoc. Remember that everything listed in the javadoc is declared **`public`**. For methods that are required to return a value, just put in a "dummy" return statement that returns zero or false.

3. Download the specchecker, import it into your project as you did in labs 1 and 2, and run it. There will be lots of error messages appearing in the console output, since you haven't actually implemented the methods yet. *Always start reading from the top.* All you really want to check at this point is whether you have a missing or extra public method, if the method declarations are incorrect, or if something is really wrong like the class having the incorrect name or package. **Any such errors will appear *first* in the output and will say "Class does not conform to specification."**

4. Look at each method. Mentally classify it as either an *accessor* (returns some information without modifying the object) or a *mutator* (modifies the object, usually returning **`void`**). The accessors will give you a lot of hints about what instance variables you need.

5. Before you write code for a method, always write a simple usage example or test case, similar to the main method shown above. This will make sure you understand what the code is really supposed to do, and it will give later you a way to check whether you did it correctly. Of course, if you are really *not* sure what a method is supposed to do, bring up your question for discussion on Piazza!

For example, as a simple starting point, look at **`isAlive()`** and **`kill()`**. Note that **`isAlive()`** is an accessor and **`kill()`** is a mutator. First, write down a quick example. If you create a new **`Projectile`** and call **`isAlive()`**, what value should you get? If you call **`kill()`** and then call **`isAlive()`** again, what value should you get?

```
Projectile p = new Projectile(1.0, 2.0, 3.0, 4.0, 10);
System.out.println(p.isAlive());
System.out.println("Expected true");
p.kill();
System.out.println(p.isAlive());
System.out.println("Expected false");
```

How can this be implemented? Well, in order for **`isAlive()`** to return the right answer, the information "am I alive or not?" needs to be stored inside the projectile object. That means you need a suitable instance variable to store this piece of information. Add one. Make sure it is declared **`private`** and make sure you initialize it in the constructor so that you get the right

answer for the first println. Then update it in the **`kill()`** method so you get the right answer for the second printly.

6. Another easy one would be **`getAge()`**. This is an accessor method that is supposed to tell us the number of times that **`timeStep`** (either version) has been called. According to the documentation, a newly constructed projectile should have "age" 0. Try a simple test case:

```
p = new Projectile(1.0, 2.0, 3.0, 4.0, 10);
System.out.println(p.getAge());
System.out.println("Expected 0");
p.timeStep();
p.timeStep();
System.out.println(p.getAge());
System.out.println("Expected 2");
```

As in the previous step, in order to provide the correct result, the information "how many times has timestep been called?" has to be stored in the projectile object. Clearly we need another instance variable. Add one and make sure it is correctly initialized in the constructor. Then, in order to get the correct answer for the second println in the sample code above, we need to update that variable in the **`timeStep`** method.

> But there are two **`timeStep`** methods, a simple one with no parameters, and a more general one that has a **`double`** parameter called **`gravity`**. Here is a tip: methods in a class can call each other, so the simpler **`timeStep`** method could be implemented by just calling the more general **`timeStep`** method with an argument of zero:
>
> ```
> timeStep(0);
> ```

7. Now it's time to start thinking about position and velocity. The x velocity is an easy one, because it never changes:

```
p = new Projectile(1.0, 2.0, 3.0, 4.0, 10);
System.out.println(p.getVelocityX());
System.out.println("Expected 3.0");
```

What kind of instance variable is needed? Where does the initial value come from? Get that working, then think about the x position. The x position is initialized in a similar way, but changes as **`timeStep`** is called. Again, start with a simple usage example:

```
p = new Projectile(1.0, 2.0, 3.0, 4.0, 10);
System.out.println(p.getX());
System.out.println("Expected 1.0");
p.timeStep();
System.out.println(p.getX());
```

```
System.out.println("Expected 4.0");
p.timeStep();
System.out.println(p.getX());
System.out.println("Expected 7.0");
```

Does your code also work using the other **`timeStep`** method?

8. The y velocity and y position are similar, except that the y velocity can change with gravity. So start with the simpler case in which gravity is zero, which will be very similar to those for the x position. Then account for the effect of gravity, which is to add a value to the y velocity each time step. Write a test case first and post your questions on Piazza if you are not sure what the results should be.

9. You also have an accessor method **`getBoundingBox`** that returns a new object of type **`java.awt.Rectangle`**. You have already seen the **`Rectangle`** class in lecture examples, e.g.,

http://web.cs.iastate.edu/~smkautz/cs227f15/examples/week2/RectangleTest.java
http://web.cs.iastate.edu/~gsong/ComS227/inClassExamples/week02/RectangleTest.java

Think about a test case. One helpful thing for testing is that you can just pass a **`Rectangle`** object to **`System.out.println`** and it will convert it to a readable String description that you can compare to the correct answer. For example,

```
p = new Projectile(1.0, 2.0, 3.0, 4.0, 10);
Rectangle box = p.getBoundingBox();
System.out.println(box);
System.out.println("Expected Rectangle at (1, 2), width 10, height 10");
```

Make sure you have an appropriate **`import`** statement, as in the examples. Your method will need to

a) determine the correct x, y, width, and height values
b) construct a new **`Rectangle`** object using an appropriate constructor from the **`Rectangle`** class, and
c) return that object

The x and y coordinates are **`double`**s, so they have to be rounded to integers to construct the bounding box; see "Additional notes" #3 below. How will you check whether your rounding is being done correctly? Come up with some test cases in which the values are not whole numbers. For example,

```
p = new Projectile(1.3, 2.5, 3.0, 4.0, 10); // what is the bounding box?
```

9. Once you have bounding boxes, the **`collides`** method is easy: just use the **`Rectangle`** method **`intersects`** to determine whether the bounding box intersects the bounding box for the other **`Projectile`**. See the **`Rectangle`** API documentation for details.

Additional notes

1. You do NOT need conditional statements ("**`if`**" statements) or loops for this assignment. We will start covering conditional statements later next week, and you won't be penalized if you use them, but you would just be making things more complicated.

2. Do not create any additional Java classes.

3. To round a **`double`** to the nearest **`int`**, you can use the method **`Math.round()`**. However, the **`round()`** method actually returns a value of type long, that needs to be "cast" to **`int`**. (A Java **`long`** is an integer type, but occupies 64 bits instead of 32.) Sample usage:

```
double d = 2.5;
int x = (int) Math.round(d);  // rounds to 3
```

4. Accessor methods should not modify instance variables.

The SpecChecker

You can find the SpecChecker online at
http://www.cs.iastate.edu/~cs227/homework/mini1/speccheck_mini1.jar .

Import and run the SpecChecker just as you practiced in Labs 1 and 2. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the console output. There are many test cases so there may be an overwhelming number of error messages. *Always start reading the errors at the top and make incremental corrections in the code to fix them.*

When you are happy with your results, click "Yes" at the dialog to create the zip file. See the document "SpecChecker HOWTO", which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links," if you are not sure what to do.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, *documentation is not required*. However, we recommend that you

document each method as you create it. It is usually easier to write a method correctly after you have written down what it is supposed to do!

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`miniassignment1`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`miniassignment1`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Blackboard, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_mini1.zip`**. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`mini1`**, which in turn contains one file, **`Projectile.java`**. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Blackboard using the Miniassignment 1 submission link and verify that your submission was successful by checking your submission history page. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links."

*We strongly recommend that you just submit the zip file created by the specchecker. If you mess something up and we have to run your code manually, you will receive **at most half the points**.*

We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **mini1**, which in turn should contain the file **Projectile.java**. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and NOT a third-party installation of WinRAR, 7-zip, or Winzip.