Reference

Student's Name: Tara Aida **Course Name: Computer Science 01100101**

Teacher's Name: Joe Puccio

Create a program that tracks the motion of a ball accelerating toward the ground due to a gravitational force. Allow the user to specify a horizontal (perpendicular to the gravitational force) velocity. Have the program print out the x (horizontal) and y (vertical) position of the ball, as well as the y velocity (this should be originally negative if you use the convential reference frame of the ground being the origin).

Example of using constructors to create objects from a class:

We start by creating a class that will act as a blueprint for creating objects (we will create the objects in the main method (the method with public static void)). Remember to import the scanner! Also, please don't copy and paste this code when creating your class for this assignment and refer to the bottom of this document to find out more on constructing classes like these.

```
public class prettyGirl {

	String name = "Tara Aida";
	int prettiness = 10;
	String looksBestIn;
	Scanner scan = new Scanner(System.in);

	public void whatDoesSheLookBestIn(){
	 System.out.println("Tell me, what does "+this.name+" look best in?");
	 this.looksBestIn=scan.next();
	}
}
```

Within the main method, we create an instance of an object using the blueprint of the other class.

```
prettyGirl physicsGirl = new prettyGirl();
```

Now that we've created the object, we can access properties of the object that were defined and set to a value in the blueprint (highlighted).

```
System.out.println("We're looking at pretty girl "+physicsGirl.name+" today.");
```

Reference

We can even use the object to execute methods that we've defined in the object's parent class.

```
physicsGirl.whatDoesSheLookBestIn();
```

Additional information on class and method construction as well as property access:

There's a lot to be said on these things but I'm going to try to give you the basics without ~~overwhelming~~ killing you.

Within a class, you've got essentially two things: properties and methods. Properties are the attributes of the object that you will create from the class, which means that they tell you a little something about the object. So in the example I gave, properties are `name` and `prettiness`, and they tell you about the object: `physicsGirl`.

Methods are doers. They are used to change the values of the properties of their parent object (the object they belong to) or change the values of properties of other objects. When we use one object's methods to manipulate the values of the properties of another object, we can say that these objects "interact". For example, if we were to instantiate (create) two "prettyGirl"s, calling one "physicsGirl" and the other "econGirl" and then in addition to having a `prettiness` property, we had a `relativePrettiness` property, we could have the int `relativePrettiness` for any given girl be calculated by going to each of the "prettyGirl"s that had been created, comparing the given girl's prettiness to every other girl's prettiness using some algorithm, and then setting the `relativePrettiness` to the result of that algorithm.

So object-oriented programming is the result of applying the object-oriented style (representing every component of an actual system with a class, and then having properties of the class in the program represent the attributes of that component in the actual system, and then having methods of the class in the program represent the functions of that component in the actual system) to an actual system. Then you create objects from these classes (blueprints), and have them interact the same way they do in the actual system. The nice thing about classes is that if you have many of the same component in a system, you can just create an object for each component that you have from the same class without having to rewrite the same thing 10 times. So for our example, I could make an "econGirl" from the "prettyGirl" class on top of the already created "physicsGirl", all I'd have to do is change the econGirl.name property and econGirl.prettiness property to the right values. Very soon I'll teach you more about better ways to do this.

Now for syntax (which is why I made this section in the first place):

```
public class prettyGirl {

	String name = "Tara Aida";
	int prettiness = 10;
	String looksBestIn;
	Scanner scan = new Scanner(System.in);

	public void whatDoesSheLookBestIn(){
	 System.out.println("Tell me, what does "+this.name+" look best in?");
	 this.looksBestIn=scan.next();
	}
}
```

Reference

Eclipse will make any parameter turn blue so that you can differentiate between parameters and variables that you may have defined locally (which are usually temporary variables used in any calculations you may have to alter parameters). Some parameters from this example are: `name,prettiness,looksBestIn,scan.` Each of these parameters, as well as their local/temporary counterparts, have what are called variable types. Variable types let the IDE determine what variables can be used for what purpose (for instance, you can't multiply two strings). Some variable types from this example are `String, int,Scanner.`

Now that we've talked about parameters, we need to talk about the other major component of classes: methods. There's a certain syntax to creating methods that, if not explained, can be very confusing. The most important part is the method's header. In our example, the method's header is :

```
public void whatDoesSheLookBestIn(){
```

The word `public` is there to allow the method to be used by any instance of the class. Let's look at an example to understand what I mean by this: let's say that we brought back the `relativePrettiness` paremeter to the "prettyGirl" class, and we had some algorithm that we used to determine the value of the `relativePrettiness` paremeter, but we didn't want any of the "prettyGirl"s (such as physicsGirl or econGirl) to be able to access the method that we used to calculate the `relativePrettiness` parameter, then instead of putting `public` as the access type, we'd put `private`. Then if we were to type the name of a "prettyGirl" object and then put a dot ("."), we wouldn't be able to access this method, but we could allow access to the results, which would be the `relativePrettiness` parameter value. Also note that you can make parameters `public` or `private` if you want.

The next important thing to note about the header is what is called the return type. In our example, the return type is `void`, which means that the method does not return anything. You'll usually use `void` return types when you're performing actions on your own parameters, or taking input from the user and setting a parameter to that input. Other options for the return type are `int,String,Array.`

The last important thing you need to know is how to refer to the object that you're currently calling a method on. That's simply using the word `this`. So, if you were to define a method that needed the value of a paremeter of the object that the method was being called on, you would say `this.parameter`.

Next time, we'll talk about constructor methods.

Send your source code to the following email address for grading:

Joe@pooch.us

Extra credit: once the ball meets the ground, allow for it to bounce. Choose some constant for the elasticity of collision/how much energy is lost each time the ball hits the ground.

Notes on the grading method: I will be inputting the printout of your program into grapher to check for errors.