what you've made, that's good enough for us. Make sure your `NBodyExtreme` and `BodyExtreme` are pushed to GitHub. When you've posted your video, fill out this form.

Even though your youtube video will be public, you should not post your code publicly. In addition, **you should not display your code in the video**.

Frequently Asked Questions

I'm passing all the local tests, but failing even easy tests like testReadRadius in the autograder.

Make sure you're actually using the string argument that testReadRadius takes as input. Your code should work for ANY valid data file, not just planets.txt.

The test demands 133.5, and I'm giving 133.49, but it still fails!

Sorry, our sanity check tests have flaws. But you should ensure that your value for `G` is $6.67 \cdot 10^{-11} \frac{\text{Nm}^2}{\text{kg}^2}$ exactly, and not anything else (don't make it more accurate).

When I run the simulation, my planets start rotating, but then quickly accelerate and disappear off of the bottom left of the screen.

- Look at the way you're calculating the force exerted on a particular planet in one time step. Make sure that the force doesn't include forces that were exerted in past time steps.
- Make sure you did not use `Math.abs(...)` when calculating `calcForceExertedByX(...)` and `calcForceExertedByY(...)`. Also ensure that you are using a `double` to keep track of summed forces (not `int`)!

What is a constructor? How do I write one?

A constructor is a block of code that runs when a class is instantiated with

the `new` keyword. Constructors serve the purpose of initializing a new object's fields. Consider an example below:

```
public class Dog {
    String _name;
    String _breed;
    int _age;

    public Dog(String name, String breed, int age) {
        _name = name;
        _breed = breed;
        _age = age;
    }
}
```

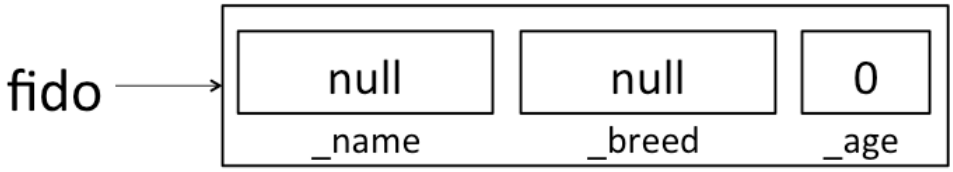
The `Dog` class has three non-static fields. Each instance of the `Dog` class can have a name, a breed, and an age. Our simple constructor, which takes three arguments, initializes these fields for all new `Dog` objects.

I'm having trouble with the second Body constructor, the one that takes in another Body as its only argument.

Let's walk through an example of how a constructor works. Suppose you use the `Dog` constructor above to create a new `Dog`:

```
Dog fido = new Dog("Fido", "Poodle", 1);
```

When this line of code gets executed, the JVM first creates a new `Dog` object that's empty. In essence, the JVM is creating a "box" for the `Dog`, and that box is big enough to hold a box for each of the `Dog`'s declared instance variables. This all happens before the constructor is executed. At this point, here's how you can think about what our new fluffy friend `fido` looks like (note that this is a simplification! We'll learn about a more correct view of this when we learn about Objects and pointers later this semester):



Java will put some default values in each instance variable. We'll learn more about where these defaults come from (and what `null` means)

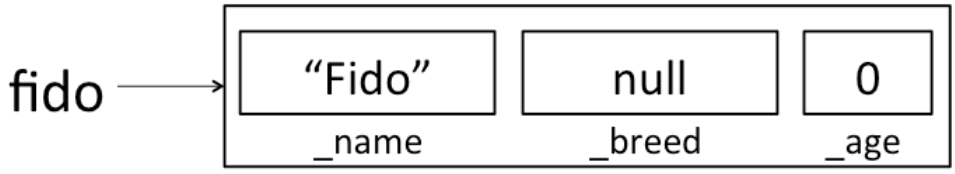
later this semester. For now, just remember that there's space for all of the instance variables, but those instance variables haven't been assigned meaningful values yet. If you ever want to see this in action, you can add some print statements to your constructor:

```
public Dog(String name, String breed, int age) {
    System.out.println("_name: " + _name + ", _breed: " + _breed +
    _name = name;
    _breed = breed;
    _age = age;
}
```

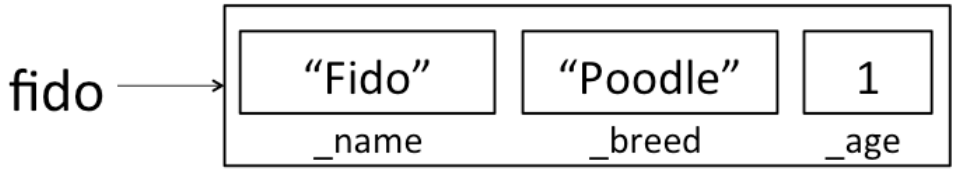
If this constructor had been used to create `fido` above, it would have printed:

```
_name: null, _breed: null, _age: 0
```

OK, back to making `fido`. Now that the JVM has made some "boxes" for `fido`, it calls the `Dog` constructor function that we wrote. At this point, the constructor executes just like any other function would. In the first line of the constructor, `_name` is assigned the value `name`, so that `fido` looks like:



When the constructor completes, `fido` looks like:



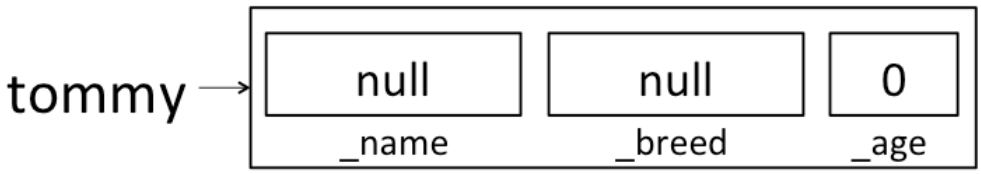
Now, suppose you want to create a new `Dog` constructor that handles cross-breeding. You want the new constructor to accept a name, an age, and two breeds, and create a new `Dog` that is a mixture of the two breeds. Your first guess for how to make this constructor might look something like this:

```
public Dog(String name, String breed1, String breed2, int age) {
    Dog dog = new Dog(name, breed1 + breed2, age);
}
```

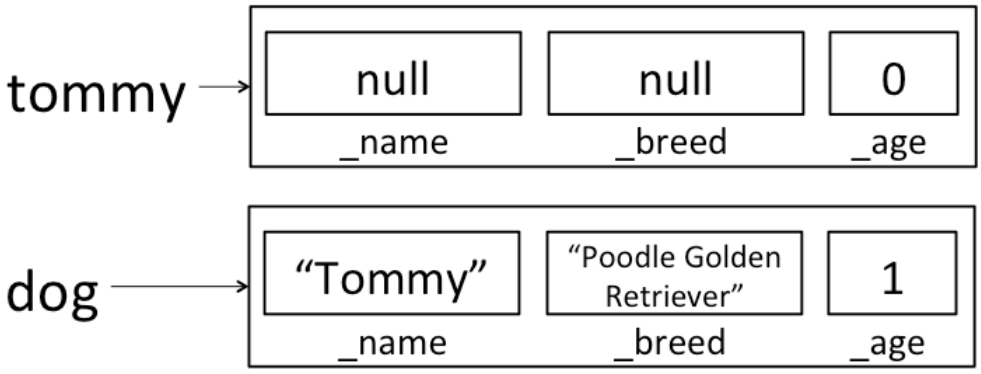
However, if you try to create a new `Dog` using this constructor:

```
Dog tommy = new Dog("Tommy", "Poodle", "Golden Retriever", 1);
```

This won't do what you want! As above, the first thing that happens is that the JVM creates empty "boxes" for each of `tommy`'s instance variables:



But then when the 4-argument constructor got called, it created a second `Dog` and assigned it to the variable `dog`. It didn't change any of `tommy`'s instance variables. Here's how the world looks after the line in our new constructor finishes:

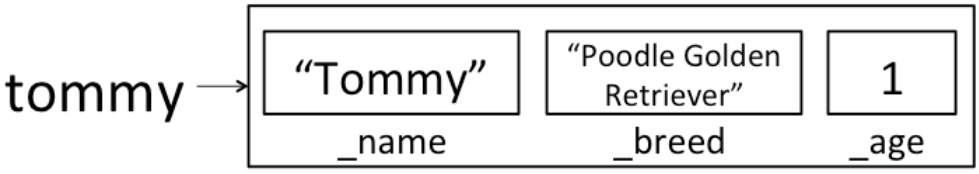


`dog` isn't visible outside of the constructor method, so when the constructor completes, `dog` will be destroyed by the garbage collector (more on this later!) and all we'll have is the still un-initialized `tommy` variable.

Here's a cross-breed constructor that works in the way we'd like:

```
public Dog(String name, String breed1, String breed2, int age) {
    this(name, breed1 + breed2, age);
}
```

Here, we're calling the old 3-argument constructor on `this`; rather than creating a new `Dog`, we're using the 3-argument constructor to fill in all of the instance variables on this dog. After calling this new constructor to create `tommy`, `tommy` will correctly be initialized to:



We could have also written a new constructor that assigned each instance variable directly, rather than calling the existing constructor:

```
public Dog(String name, String breed1, String breed2, int age) {
    _name = name;
    _breed = breed1 + breed2;
    _age = age;
}
```

Acknowledgements

This assignment is a major revision by Josh Hug, Matthew Chow, and Daniel Nguyen of an assignment created by Robert Sedgewick and Kevin Wayne from Princeton University.