ICS 141

Lab #5

Assume the same two classes we looked at during lecture.

**public** **class** Yorkie {

**private** String name;

**private** **int** age;

**public** Yorkie(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** String toString() {

String temp = "Yorkie Data\n";

temp += "Name: " + name + "\n";

temp += "Age: " + age + "\n";

**return** temp;

}

}

**public** **class** Husky {

**private** String name;

**private** **int** age;

**public** Husky(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** String toString() {

String temp = "Husky Data\n";

temp += "Name: " + name + "\n";

temp += "Age: " + age + "\n";

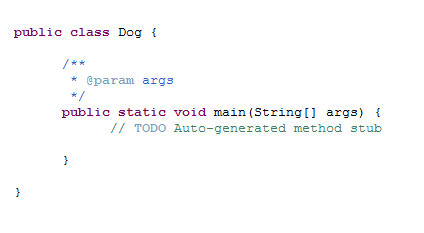
**return** temp;

}

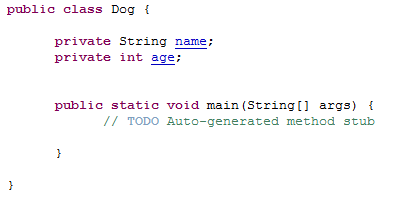
}

As we discussed, these classes are very similar. In fact, this would be a perfect time to utilize inheritance in order to achieve some code reuse. This is the goal of this lab. We’ll create a superclass, called Dog, from which both of these classes will inherit.

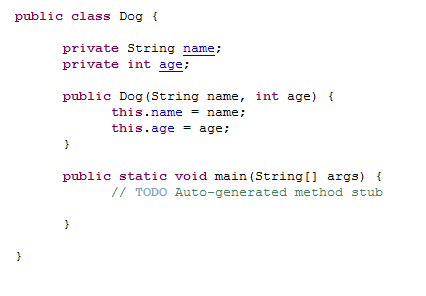
First, let’s create the superclass, Dog. Create a new project. Then, create a new class, Dog.



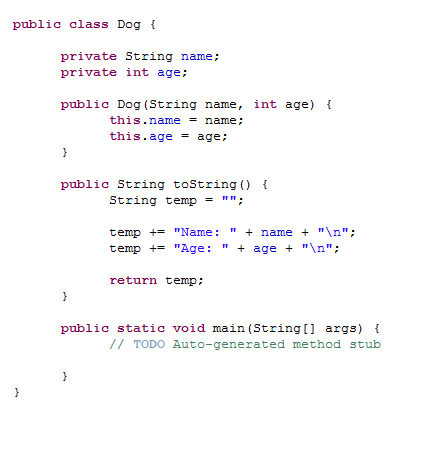
Leave the main method there for now. We’ll use it to test our classes later on. For now, we need to begin by declaring some data fields. Looking at the Husky/Yorkie classes, we can see they’ll have two fields in common: name and age. We’ll keep these as private in order to keep a proper encapsulation from other classes.



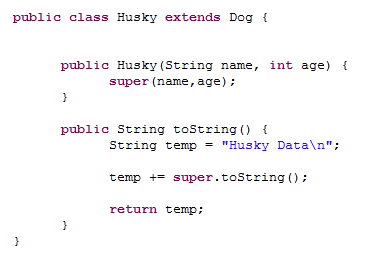
Next, let’s tackle the constructor. We simply want one to set the two fields accordingly. Note that we’ll actually make this public; however, as we’ll see later, the protected modifier would be better. We’ll also use the **this** reference here to distinguish between local parameter and data field.



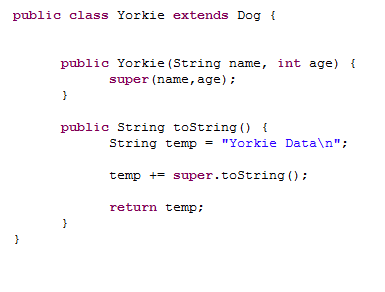
Up next, we must write out a toString method as it would also be useful seeing that both the subclasses would use it in some way.



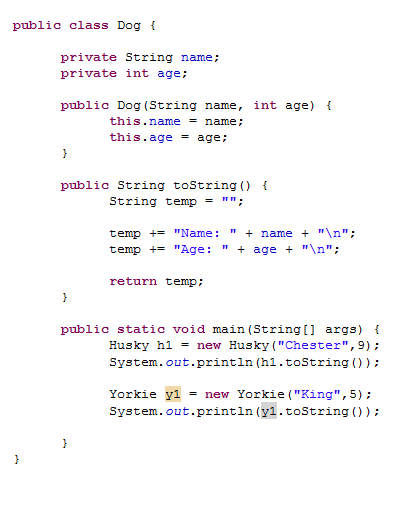
That should do it for the Dog class for now. We have taken care of all of the shared content between the original Husky/Yorkie classes. Now, let’s rewrite the Husky class. We can now get rid of the data fields as we’ll have access through the constructor of the Dog class. In rewriting the constructor, we’ll simply use the superclass constructor to set both of the fields. Finally, the toString will call the superclass’ toString in order to build the String.



See how much less there is in this class? Now, since this was a small example, this may not seem to matter too much; however, this code reuse builds up over time. Now, do the same for the Yorkie class.



Now, in order to test this, let’s go to the main method of the Dog class.



In testing this, you’ll see that both types of dog print out properly.

**Deliverables:**

The .java file(s) for this lab. These are the files that contain your source code. I do not need the .class files. Submit the java file(s) to the lab #5 drop box.