Tutorial 5

Objectives

Practice using basic inheritance and polymorphism with abstract classes and methods. Introduction to Javadocs.

Attendance Quiz

Please log on to cuLearn using on of the computer in the tutorial room and complete the attendance quiz. You can only access the quiz if you log in using one of the computers in the lab. You cannot use a laptop for this. This is a time limited quiz. Be sure to do this as soon as you arrive.

At the end of the tutorial a TA will assign you a grade, call it G, which is 0, 1 or 2, depending on the progress you make during the tutorial. If you spend your time reading mail and chatting with your friends you will receive 0. If you have completed the attendance quiz on time then G will be your tutorial grade. If you have not completed the attendance quiz on time, then your tutorial grade will be max(0, G - 1/2). Each tutorial grade will therefore be one of 0, 0.5, 1.5 or 2.

Animal

Consider the abstract *Animal* class provided. Read the class and see what it provides. Override Object's *toString()* method in the *Animal* class so that it prints out the animals name and current age. For the current age, just use the age in years that the animal will be this year. For example, if a dog was born in 2000 and it is currently 2017, the dog's current age is 17 years. The program should not hard-code the current year (see the Note below for help on this).

Note: assuming your computer's clock is correct, we can get the current year (as an int) using two classes of Java's *time* package as follows:

```
ZonedDateTime.now( ZoneId.of("America/Montreal")).getYear()
```

Note: *Animal* is abstract. You cannot create an *Animal* object using the *new Animal(...)*. In order to test your *Animal* class, you need to create a **concrete** child class that overrides all abstract methods in *Animal*.

Cat and Dog

The *Cat* and *Dog* classes should **extend** the animal class. Add any needed constructors and methods to make these work. The classes should be **concrete** classes.

Note: The noise that a cat makes should be meow or prrr (randomly chosen each time it's noise method is called), and the noise a dog makes should be woof or grrrr (randomly chosen each time it's nose method is called).

AnimalApp

The *AnimalApp* program generates an array of animals and calls the noise method of each. You can use this to test your *Animal*, *Cat* and *Dog* classes.

Owl

Create an *Owl* class that extends the *Animal* class. The class should have a three-argument constructor that takes a string (name), an integer (birth year) and a boolean (that determines if it is a **wise** owl or not). The string representation of an owl (i.e., the output of *toString()*) should include its name, age and whether or not it is wise or not.

An owl's noise is hoooo.

Modify the *AnimalApp* program so that some owl objects are also created.

JavaDocs

Look at the code in the *Animal* class. You'll notice that the comments look slightly different. In DrJava, with the *Animal* class open, click the **Javadoc** button (just to the right of the compile and run buttons).

After answering yes to the next two questions, you should be directed to a web browser with a new tab Showing the **API** of all the classes you have open in DrJava right now. From the left-side menu, click the *Animal* class. Look at the API that was generated for this class. Look at all of the API for this class (click the links for example). Now go look at the comments in the *Animal.java* file. You should see some patterns/connections here.

The API you are looking at was generated by the Javadoc tool, using the special comments in the *Animal* class. What is the Javadoc tool? From wikipedia, we find

Javadoc is a documentation generator from Oracle Corporation for generating API documentation in HTML format from Java source code. The HTML format is used to add the convenience of being able to hyperlink related documents together.

In your code, you can add a special comment block just **before** a class, attribute, method or constructor declaration. This block of comments will be used in the generated API to describe whatever it is that the comment blocks comes before. There are some special tags that the Javadoc tool will read and use in this comment block. Here some simple rules

- The comment block must start with */** (two stars instead of one)
- the comment block ends with **/*
- The *@author* tag will list the author of the class or method
- For methods, each input argument should have an associated *@param* tag describing that input. (We can add pre-conditions on the argument here.)
- For methods, the *@returns* tag is used to describe the output (and any post-conditions).
- Do not include a *@returns* for constructors or for methods that return *void*.
- You can use basic HTML tags to help format the text. For example, *<code>main</code>* will format *main* in code format. Use *<p>* to start a new paragraph (with a blank line).

Read the Javadoc Comments and Javadoc Tags sections of the following short tutorial

http://students.cs.byu.edu/~cs240ta/fall2012/tutorials/javadoctutorial.html

Skim through the Oracle documentation for writing Javadoc comments
http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Add some Javadoc tags to your various animal (sub)classes.

More Programming

Create two new subclasses for each of the *Cat*, *Dog* and *Owl* classes. For each pair of new classes, have one introduce a new noise that is specific to that specific kind of cat, dog or owl.