CSC 143 Programming Project 1A[i]
Inheritance

This project involves building a number of classes to become more comfortable with inheritance and the associated concepts.

To the Zoo!

Open a new Eclipse (or using your favorite Java editor) Java Project Chapter9Exercise and create a class Animal. Give the Animal a single private int attribute hunger to hold how hungry the Animal is, a constructor that takes no arguments and sets this attribute to zero, and a method getHunger that returns it. Then write an abstract method talk that takes no arguments and returns nothing. (Recall that to make a method abstract, you put abstract at the beginning of the method signature (before public) and put a semicolon at the end of the signature instead of an opening brace.)

Try to compile this. You will get an error message about not having overridden the abstract method talk. This error comes because the class Animal has an abstract method, but it is not declared to be an abstract class. Add abstract between public and class at the top of the file. This tells the computer that you intend Animal to be an abstract class. (Eclipse tries to help you with errors as you write code; pay attention to these hints and messages; don't just simply correct code without understanding what these errors are pointing to!)

Now check that Animal compiles. It does, but because Animal is an abstract class, you are not able to actually create Animal objects. For that, we need to create a subclass.

Create a class Zebra. Make it a subclass of Animal by adding extends Animal to the end of the line giving the class name. Write a constructor for Zebra that takes no arguments. The entire body of this method should be super(); This calls the constructor for Animal. That constructor then sets its attribute hunger to 0. (Actually, Java will automatically call the superclass constructor for us, but it's a good habit to explicitly include a call to a superclass constructor in each subclass constructor.)

If you try to compile at this point, you again get the error message about not having overridden the talk method. Oops! By inheriting from Animal, we promised to implement a talk method. Add a method talk that takes no arguments and returns nothing, (as promised in the Animal class method declaration). Make this method print the string "The Zebra quietly chews.".

After writing talk, the project should compile. Create a Zebra object (you can do this by writing a main method either in the Zebra class or in a client class; the latter is preferred as it is not a good practice to add main method to a pure Java class). Even though we did not declare any attributes within the code for Zebra, you see that it has the int attribute inherited from Animal. Note the methods are able to invoke on the Zebra object – talk and getHunger. Call both methods to verify that talk prints out the message about chewing and that getHunger returns 0.

There wouldn't be any reason to write the abstract class Animal if we planned to write only one subclass. Write another subclass of Animal called Lion. As we did with Zebra, give it a

constructor that takes no arguments and calls the Animal constructor. Then write a talk method that just prints the message "Roar!". Verify that it compiles and that both getHunger and talk work as intended in the Lion.

So far, the hunger attribute does not do very much because it never changes from 0. Add a method timePasses to Animal that increases the hunger attribute by one. The idea is that this is called after each unit of time, so that the animals gradually get hungrier over time. Lions are not content to quietly get hungrier, though. Override the timePasses method in Lion with a method that increases hunger by 1 as above, but if the new hunger value is at least three, also prints the message, "The Lion paces hungrily." Note that you will not be able to access hunger directly because it is a private attribute of Animal. One solution is to change the access restrictions on hunger (e.g. make it public), but better is to access the attribute indirectly. To increase hunger, call the the timePasses method of Animal (using super.timePasses();). Then, use the getHunger method to read the value of hunger.

Now that the animals can get hungry, we should have a way to feed them. Add a method feed to Animal that sets hunger back to 0. Compile and create some animals to make sure they get hungrier and that hungry lions start pacing.

As a last step for the animals, let's write a toString method. This is the method that is called automatically when an object is printed, or when a String representation is needed for some other reason. It must take no arguments and return a String. The classes already inherit such a method from the Object class (which you can verify by looking in "inherited from Object" in the menu of methods). Since this toString method doesn't give a very useful string, let's override this method in Zebra and Lion with methods that return the class of the animal, so the method in Zebra returns "Zebra" and the one in Lion returns "Lion". Because the signature of your toString method needs to exactly match the signature of the one you're overriding, you will need to explicitly make it public:

```
public String toString()
```

Once you complete this method, again compile, create some animals, and ensure that this works before proceeding.

Now let's write some code to use our animal classes. Create a new class called Zoo. This one should not inherit from Animal since it doesn't have an "is a" relationship with Animal. To start with, give your Zoo a single attribute called cage which stores an Animal. Give it a constructor that takes an Animal object and stores it in this attribute. Then write a method print that prints the message "The zoo contains a " followed the animal's type. Since we've written a toString method, you can print an Animal object as if it were a String. Printing the Zoo should produce a message such as the following:

```
The zoo contains a Lion
```

A zoo with only one animal isn't going to attract many visitors. Therefore, we want to expand the Zoo class so that it can accommodate multiple Animal objects. Rename cage to cage1, and add cage2 & cage3. Remove the argument to the constructor and just have it set all these variables to null. (That means that the variable doesn't reference any object.) Then create

methods to set each of these (call them putInCage1, putInCage2, and putInCage3); the methods take an Animal object and set the appropriate variable. Then modify print to print any of these that are not null in a format such as

```
The zoo contains the following:
    Lion
    Zebra
```

Since you don't want to print a null reference, you'll need to check each of them before printing it:

```
if(cage1 != null) {
   //print first animal
} if(cage2 != null) {
   //print 2nd animal
} ...
```

Write Zoo methods timePasses, allTalk, and feedAll that call the corresponding method for each (non-null) animal in the zoo. Now add another subclass of Animal (your choice), writing corresponding constructor, talk, and toString methods.

To submit this exercise, gather all *.java source-code files that have been created for the above classes (in Eclipse, you can find these files in src folder under the folder that has been created for this Java project), place them in a folder with name format, YourLastNameZoo, zip/compress the folder and submit thru course Canvas page.

---

[i] Adopted from: https://www.engage-csedu.org/find-resources/lab-4-zoo and https://www.engage-csedu.org/