

## Object-Oriented Design Exercise #2

This exercise will further our explorations of object-oriented design.

The goal of this exercise is to design a series of related classes that model some real-world situation. An example would be a simulation of a zoo, which might use Zoo, Habitat, and Animal parent classes, with some subclassing of the latter two.

Your design should have 3-5 classes. Choosing a design problem that is of some interest to you personally will make it easier to take advantage of your intuitions and come up with a reasonable design.

For this exercise, you will be turning in three items:

- 1) A sketch of the classes and their hierarchies (1 page)
- 2) An entity-relationship diagram (1 page; back of #1 is fine)
- 3) Printouts of Java classes that implement your design. These classes should compile and be commented, but most of the methods will not be implemented.

***Your class hierarchy and entity-relationship diagram should be neat!!! Pencil is fine, but neatness counts! Anything that looks like a draft will be returned ungraded.***

For your Java code, please see the guidelines on pages 80-81 of [Be Prepared for the AP Computer Science Exam in Java.](#)

We will be discussing entity-relationship diagrams (ERD's) in class. ERD's are one of a number of ways to think about object-oriented design.

### Suggested Methodology

- 1) Start by jotting down your classes and doing an initial draft of the class hierarchy. Think about properties and methods your classes might have, but don't go too far into that.
- 2) Do a draft of an ERD for your classes. This will help clarify the properties and methods your classes might need that control/implement how the classes interact.
- 3) Return to your hierarchy and refine it in tandem with your ERD.
- 4) In an Xcode project, draft your classes. Your classes should have the following:
  - a) Major properties

b) One or more constructors that make sense for how the classes will be used

c) Accessors and modifiers (getters and setters) for the major properties. Put in comments explaining any special processing or error handling that you might want to add to these.

d) Methods that show the major operations and relationships between the classes. You may want your classes to have methods that return lists of other classes (e.g., a Habitat might return a list of all the Animals inside it). For all such lists use the built-in ArrayList class to represent the type of the list.

e) Have a main() that constructs some of your classes for testing purposes. (They won't do much, if anything...)

### Extra Credit

For extra credit, you can actually implement some of the relationships between your classes using properties that are references to other objects, ArrayLists, or other techniques. You could then have test code that not only creates some objects but also has them print out some information about their relationships.

### Example

Example of some test code (probably the main() method of the Zoo class) that includes a little extra credit (style is mediocre but it gets the point across)...

```
Zoo theZoo = new Zoo("San Diego Zoo");
Habitat h1, h2;
Animal a1, a2, a3;

h1 = new EnclosureHabitat("Big Cat Enclosure");
h2 = new AviaryHabitat("Tropical Forest");

a1 = new Lion("Simba");
a2 = new Lion("Elsie");
a3 = new Toucan("Fruitloop");

h1.putOccupant(a1);
h1.putOccupant(a2);
h2.putOccupant(a3);

h1.displayOccupants(); // prints list of all animals in habitat h1
a3.displayHabitat();   // prints habitat that a3 is in
```

