# Objects Lab

This lab requires two files: Creature.java class file and a client code class file with the main method.

Suggested way(s) to write this lab:

When creating classes, you still want to be able to test your code. I would suggest writing each step of the creature class, then testing it in your client code.

Another method would be to attempt to do each step of the client code (Part 2) and only write the code in the creature class that will help make the client code to work. This way you are also testing the creature code as you go.

## Part 1: Create a Creature class

1. Create a class called Creature with the following private fields:
   1. name (string)
   2. description (string)
   3. endurance (int)
   4. filename (string)
   5. phrases (array of at least 3 strings) that the creature might say
   6. index of the last phrase used (int)
2. Create a constructor that takes no parameters. When called, this constructor should ask the user to input default values for all the fields. The last phrase used index field can be defaulted to 0.
3. Create a method that saves the creature to a data file (with a file name that is stored in the filename field). This method should not take any parameters and should write all the data for the creature to the file in a format that would allow it to be loaded by the first constructor.
4. Create a method that saves the creature to a data file, taking a string as a parameter that should be used as the new filename. The filename field should be changed and then the method from step 3 should be called (without parameters) to save the creature to the new file.
5. Create a second constructor that takes a filename as a parameter, stores it in the filename field, and uses that filename to load all the data for the creature from the file to initialize all the fields.
6. Create a getter method for the following fields:
   1. name
   2. description
   3. endurance
7. Create a setter method for name and description.
8. Create a setter method for endurance that makes sure the endurance can get to 0 but cannot go below it. If a value < 0 is passed, just set the endurance to 0. If endurance is ever set to 0, and the creature’s endurance is not already 0, have the creature print out a message to show that it is tired.
9. Create a method that returns false if the creature is tired (<= 0 endurance) and true otherwise.
10. Create a method that returns a phrase from the array of phrases the creature might say and updates the index of the last phrase used. We want it to be that the creature will not say the same thing twice in a row but will rotate through what they say.
11. Create a method that takes a string as a parameter and changes a phrase in the array to that new string. This method should also update the index of the last phrase used.
12. Create a toString() method for this class so that if you print the creature object, you can see all of the state of the creature.
13. Add one additional method of your choosing that returns a string saying how the creature reacts to that behavior.

## Part 2: Use the Creature class

In the client code, do the following:

1. Create a creature by calling the constructor.
2. Print the creature’s name, description, and endurance using the getter methods.
3. Print out three phrases from the creature’s phrases array.
4. Ask the user to provide three new phrases for the creature and send those to the method that saves them as the creature’s phrases.
5. Print out four phrases from the creature’s phrases array.
6. Use a loop to reduce the endurance of the creature by 1 until it is tired (endurance <= 0).
7. Set the creature’s endurance to 20.
8. Print out the response to you using your custom method (from step 1, item 13).
9. Set the creature’s name to a new name.
10. Save the creature to a new file called “creature.dat” using the method from step 1, item 4.
11. Create a new creature, loading from the “creature.dat” file using the second constructor.
12. Print the new creature using the toString() method.