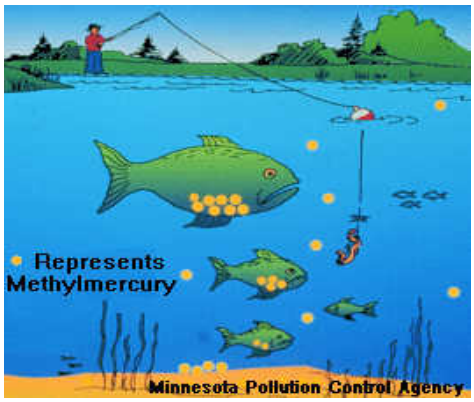
CS1110 Fall 2009 Assignment A3
More on mercury in the food chain

Submit on the CMS by midnight on Wednesday, 30 September

Introduction



For assignment A1, you wrote a Java class, `Organism` and a JUnit test class `OrganismTester`. The purpose was to:

- Gain familiarity with DrJava and the structure of a basic class within a record-keeping scenario (a common type of application)
- Work with examples of good Javadoc specifications to serve as models for your later code
- Learn to write class invariants
- Learn the code format conventions for this course
- Learn and practice incremental coding and testing
- Learn about *preconditions* of a method, which need not be tested

In this assignment, you will add fields and methods to your correct solution to A1, testing as you program. We expect you to continue to use what you learned in doing A1. We expect you to continue to include the class invariant and javadoc specifications of all methods. We expect you to program and test in an incremental fashion. In addition, this assignment has the following new learning objectives:

Learning objectives

- Learn how keeping the class invariant true may require you to change fields in more than one object
- Learn to use functions you have already written (and tested) to save time and promote modularity
- Learn to write boolean expressions that use short-circuit evaluation (look it up in the text)
- Learn about the use of null --and testing for it
- Learn to use static variables

NO iterative grading feedback (no revise-and-resubmit cycle)

But there will be no iterative feedback on this assignment. You now know our guidelines and how we grade, so there should be no need for iterative feedback. We will deduct points where our guidelines are not followed and for errors in your program. Your test cases will also be graded.

Groups

You may do this assignment with one other person. If you do this, both of you *must* visit the CMS and do what has to be done in order to group yourselves. One person will invite the other using CMS, and the other will then respond using CMS. *Do this well before you submit the assignment*. You create a mess for yourselves and for us when you submit and then realize that you have not yet grouped.

Getting help

If you need help, please SEE SOMEONE IMMEDIATELY —a course instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders. See the Staff page on the course homepage, www.cs.cornell.edu/courses/cs1110, for contact information.

How to do this assignment

For the purpose of this assignment, you may *not* use if-statements or loops, and you can use conditional expressions only in function toString.

Read the whole assignment before starting. Start with your answer to assignment a1, which should be correct. Add more components to `Organism` and test methods by adding more tests to `OrganismTester` as indicated below. This incremental development/testing will help you complete this programming task quickly and efficiently. If we detect that you did not develop it this way, we may ask you to start from scratch and do a different assignment.

Whenever you write a method (see below), look through the class invariant and convince yourself that class invariant still holds when the method terminates. This habit will help you catch or prevent bugs

Step 1: Add another constructor. Add a constructor with the specification given below. After adding it, start a new testing procedure in `OrganismTester` and add tests to it that ensure that all fields are properly set by the new constructor. Add tests to ensure that parameter eater's number of victims and amount of mercury are properly maintained. Hint: Rely on previously written methods; don't reprogram something you already programmed.

Constructor	Description (and suggested javadoc specification)
Organism(int lev, int m, String nn, Organism eater)	Constructor: a new Organism at food-chain level lev containing m units of methylmercury; its nickname is nn. This new Organism has not eaten any Organism and is immediately eaten by eater. Precondition: nn has at least 1 character, lev is non-negative, m is non-negative, and eater is not null.

Step 2: Add a toString function. As you know, a parameterless toString function produces a String representation of the object in which it appears. Write function toString in class Organism. The String that it returns must obey the following requirements exactly, down to the number of spaces and punctuation.

The general format of the output is indicated by the following two examples, but read a-d below for exceptions:

> Organism (nickname tasty), level 1, mercury content 4, 0 victims, eaten by jaws.
>
> Organism level 2, mercury content 2, 1 victim.

where

a. Exactly one blank separates each piece of information, and commas and periods appear exactly as written.
b. In the first example above, the organism has nickname "tasty". If the organism has no nickname, then " (nickname tasty)," should be omitted entirely, as shown in the second example.
c. If the organism has exactly 1 variation, then the word "victim" should be used instead of "victims", as shown in the second example.
d. If the organism is still alive, omit the phrase ", eaten by jaws", as shown in the second example. If the organism has been eaten but the eater has no nickname, the phrase should be ", eaten by". (This looks a little strange but, for the purposes of this assignment, leads to simpler solutions).

In function toString, (but nowhere else), you may use conditional expressions; you may *not* use if-statements. The general form of a conditional expression is

> (boolean-expression ? expression1 : expression2)

(the parentheses are not needed but are generally included). It is evaluated as follows: if boolean-expression evaluates to true, the value of the conditional epression is the value of expression1; otherwise, it is the value of expression2. An example usage is this: (x > 0 ? x : -x), which evaluates to the absolute value of x.

Your test case(s) for toString must test *every part* of toString. Your function toString is probably more complex than the previous methods you have written, so your testing will probably need to be more elaborate. You need to test that everything is correct for important different combinations of inputs —think about what kinds of situations would call for significantly different outputs of toString, and test whether your toString indeed produces such outputs.

It is a good practice to write function toString before writing elaborate methods, because toString can be quite helpful for debugging.

Step 3: Add comparison functions. Start a procedure in OrganismTester that will test the three functions described in the table below. Then deal with the three functions *one at a time*: write test cases for it, write the function, and test it. *The second and third functions must be* ***static***. In writing these, you may not use the if-statement or the conditional expression; you must rely on boolean expressions.

function	Description (and suggested javadoc specification)	return type
isHigherThan(Organism org)	= "org is not null and this organism is at a higher level than org."	boolean
isHigherThan(Organism org1, Organism org2)	= "org1 is not null and org2 is not null and org1 is at a higher level than org2."	boolean
eatenBySame(Organism org1, Organism org2)	= "org1 and org2 are not null and were eaten by the same Organism."	boolean

Note that whether org is null or not makes a difference in the output; org == null is *not* a precondition. The specification of the first function is equivalent to: if org is not null and this organism is at a higher level than org, return true; otherwise, return false.

Here is a principle to keep in mind: If an argument of a call can be null, then one should have a test case for it. Keep this in mind when constructing your test cases.

The body of the second isHigherThan function is best written in terms of a call on the first one; write it that way.

Step 4. Add a static variable and test its use. We want your program to keep track of the number of organisms that are created. To do this, add a private static int variable that will contain the number of organisms created and add a static getter method getNumberOfOrganisms that will return the number of organisms created. Then, change your program so that whenever an Organism is created, the static variable is incremented by 1, thus helping to ensure that your program is maintaining the class invariant.

Finally, add test cases in OrganismTester to make sure that the static variable is properly maintained. You will encounter problems if you assume that the field has a particular value at any point during testing, because you don't know the order in which testing methods are called. Therefore, to test it, you should test the *change* in the static field, as follows: (1) save the value of the field in a local variable, (2) do something to create a new Organism, and (3) test whether the static field increased by 1 --the saved value will help.

Submitting assignment A3

Do the following:

1. Click the Javadoc button in DrJava and examine the output. You should see your method and class specifications. Read through them from the perspective of someone who has not read your code, and fix them if necessary so that they are appropriate to that perspective. You *must* be able to understand everything there is to know about making a call on each method from the specification that you see by clicking the Javadoc button —without knowing anything about the private fields. Then and only then, add a comment to the top of your code saying that you checked the Javadoc output and it was OK.

2. Review the learning objectives and re-read this document to make sure your code conforms to our instructions. Make sure the class invariant is OK, the method specs are OK, no line is too long, there is a blank line between each pair of methods, the lines are properly indented, and so on.

3. Upload files Organism.java and OrganismTester.java on the CMS. Do not submit any files with the extension/suffix .java~ (with the tilde) or .class. It will help to set the preferences in your operating system so that extensions always appear.