CS1110 Spring 2010 Assignment A3 Monitoring Rhinos 2
Due (submitted on the CMS) by Monday, 22 February

This assignment continues with A1. Keep track of the time you spend on this assignment. When you submit it, tell us how many hours you spent on it.

You may work in groups of 2, but you must do all the work together, sitting at the computer together. For example, for one person to write functions and the other person to write test cases for them, with no interaction, is dishonest. If you want to form a group, *do it immediately*. *Both* partners must take action in the CMS before the group will be formed.

The deadline is Monday, 22 February, so that it is completed before the first prelim. Because you have a lot of work to do, including studying for the prelim, we have considerably reduced the work involved in this assignment by removing 5-6 functions from previous years.

In assignment A1, if you made a mistake, we asked you to fix it and resubmit. In this assignment, if you make a mistake, points will be deducted. Points may be deducted for lateness. Your new functions must have suitable javadoc comments, there must be appropriate test cases, and all methods should be correct. *You can achieve all this most easily by writing and testing one method as a time*.

A1 learning objectives. The purpose of A1 was for you 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 (use of "Constructor:" and "=" in specifications, indentation, short lines, etc.), which help make your programs readable and understandable and allow us to process your assignments and give you feedback more quickly.
- Learn and practice incremental coding, a sound programming methodology that interleaves method writing and testing.
- Learn about and practice thorough testing of a program.
- Learn about *preconditions* of a method, which need not be tested.

A3 learning objectives. The purpose of A3 is for you to:

- 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.

For this assignment, start wth your correct A1 --if you are still working on A1, you may modify it to include A3 stuff, but whatever you submit for A1 must still compile, and the A1 stuff should be correct. You will add fields and methods to your correct A1 solution, testing as you program. This may require you to change some of the testing procedures that you wrote for A1. We expect you to continue to use what you learned in doing A1, e.g. include the class invariant and javadoc specifications of all methods and program and test in an incremental fashion.

The steps to perform in doing this assignment

Step 1. Static variable. Add to class `Rhino` a **private static** variable that contains the number of `Rhino` objects that have been created so far. WHENEVER A RHINO OBJECT IS CREATED, THIS FIELD SHOULD BE INCREASED BY 1.

Add a **public static int** function `population`, with no parameters, that will return the value of the static variable. Change the constructors so that they properly maintain the value of the static variable.

Finally, add test cases to class `RhinoTester` to test whether the static variable is properly maintained. Because you may not know the order in which the testing procedures are called when you click button `Test`, test the difference in the static variable, as follows: (1) save its value in a local variable, (2) write a statement that you expect will change the static variable, say, by 1, and (3) write an `assertEquals` statement to test whether the change in the static variable was indeed 1.

Step 2. toString function. Write a toString function that produces a String representation of a rhino object. It is best to have a new testing procedure for it in RhinoTester. The representation it produces is illustrated by three examples:

"F rhino Sunny. Tag 13. Born 11/2002. Has 0 children."
"F rhino DeeDee. Born 8/1997. Has 1 child."
"M rhino A. Born 9/1998. Has 2 children."

Here are the rules.

0. You may not use an if-statement. This is easily done with a single return statement. You will use conditional expressions, and we give you one of them below.

1. Exactly one blank appears between words.

2. The first character is the gender ('F' or 'M').

3. The word "rhino", followed by the rhino's name, followed by a period '.' always appear.

4. The tag and following period appears only if the tag is >= 0. You can do this using this conditional expression: (tag == -1 ? "" : " Tag " + tag + ".")

5. The date followed be a period always appears, and in the format shown in the three examples.

6. The number of children and following period appear as shown in the examples; note that if the number of children is 1, "child" appears; otherwise, "children" appears. Use a conditional expression here.

7. Information about the parents does not appear.

Step 3. Comparison functions. Write the following functions, each of which produces a boolean value. **Do one at a time**: (1) write the function, then (2) test it thoroughly with test cases in class `RhinoTester`. **Then** proceed to the next function.

Method	Description
isMother(Rhino r)	= "r is not null, and r is this rhino's mother".
isFather(Rhino r)	= "r is not null, and r is this rhino's father".
isParent(Rhino r)	= "r is not null, and r is this rhino's parent".
isSister(Rhino r)	= "r is this rhino's sister". Precondition: `r` is not `null`. Note: r is Rhino A's sister if (1) r and A are two different objects, (2) r is female, and (3) r and A have at least one parent in common.
isYounger(Rhino r)	= "r is younger than this rhino". Precondition: `r` is not `null`.

Here are the ground rules for writing these functions:

1. The names of your methods much match those listed above **exactly**, including capitalization. The number of parameters and their order and types must also match. The best way to ensure this is to copy and paste. Our testing will expect those method names and parameters, so any mismatch will fail during our testing. Parameter names are never tested —you can change the parameter name if you want.

2. Each method **must** be preceded by an appropriate specification, as a Javadoc comment. The best way to ensure this is to copy and paste from this handout. After you have pasted, be sure to do any necessary editing.

3. Function `isParent` must be written to call `isFather` and `isMother`. We want you to learn to call methods already written instead of doing duplicate work.

4. You may not use if-statements, if-expressions, or arithmetic operations like addition, subtraction, multiplication, and division. The purpose is to practice using boolean expressions, with operators `&&` (AND), `||` (OR), and ! (NOT).

5. Note carefully the definition of sister in the specs.

6. In testing whether two rhino's are the same, do not use the field that contains their names. Instead, use the names on the tabs of their objects and operator ==.

Submitting the assignment

Check these points before you submit your assignment:

- Did you use an if statement or conditional expression in the new methods (except toString)? Get rid of them.
- Is each function tested enough? For example, if an argument can be **`null`**, is there at least one test case that has a call on the function with that argument being **`null`**? If not, points will be deducted.
- Did you check your javadoc? Click the javadoc button in the DrJava navigation bar. This will cause the specification of the classes and methods of the classes to be extracted from your program and html pages to be created that contain the specs. Look at those specs carefully and make sure that they are suitable. Can you understand precisely what a method does based on the extracted spec? If not, fix the spec, generate the javadoc, and look at it again.
- ADD A COMMENT AT THE BEGINNING OF FILE RHINO.JAVA THAT SAYS HOW MANY HOURS YOU SPENT ON THIS ASSIGNMENT. We will report the min, average, mean, and max.

Submit only files that end with the `.java`. Be careful about this, because in the same place as your `.java` files you may also have files that end with `.class` or `.java~` but otherwise have the same name.