CS1110 Fall 2011 Assignment A3: Elephants 2

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

Collaboration policy You may work in groups of 2 (form your group *immediately*; *both* partners must take action on the CMS to form the group). 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.

With the exception of your CMS-registered partner, you may not look at anyone else's code, from this semester or previous ones, in any form, or show your code to anyone else, in any form.

Grading. 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. Once graded, you cannot resubmit. 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 at 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.
- Learn about and practice reading carefully.
- Work with examples of good Javadoc specifications to serve as models for your later code.
- Learn the code format conventions for this course (use of "Constructor:" "=" in specifications, etc.).
- Learn to write *class invariants*.
- Learn and practice incremental coding, a sound programming methodology that interleaves coding 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 static variables.
- Learn to use functions you have already written (and tested) to save time and promote modularity in your code.
- Learn to write boolean expressions that use *short-circuit evaluation* (look up that phrase in the text).
- Learn about the use of **null** —and testing for it.

For this assignment, start with 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. 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.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., 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.

The steps to perform in doing this assignment

Step 1. Static variable. Add to class `Elephant` a **private static** field that contains the number of `Elephant` objects that have been created so far. WHENEVER AN `Elephant` OBJECT IS CREATED, THIS FIELD SHOULD BE INCREASED BY 1. So you should change constructors accordingly, to keep the class invariant true.

Add a **public static int** function `population`, with no parameters, that will return the value of the static variable.

Finally, add test cases to class `ElephantTester` 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 change in the static variable's value as follows:

(1) Write a statement to save its value in a local variable `pop` (say);

(2) Write a statement that you expect will change the static variable, say, by 1, and

(3) Write an `assertEquals` call to test whether the change in the static variable was indeed 1 —using `pop`.

Remember, do *not* declare fields in class `ElephantTester`. The only variables declared in the class should be local variables of the procedures.

Step 2. toString. Override function `toString` (which is declared in superclass `Object`): Write a `toString` function with no parameters that produces a `String` representation of the `Elephant` object. Write a separate testing procedure for it in `ElephantTester`. The representation it produces is illustrated by three examples:

```
"Male elephant R1Fatso. Born 5/1993. 1 child."
"Female elephant R2JustRight. Born 6/1995. 1 child."
"Female elephant R3  JustRight. Tag 19. Born 6/2006. Mother R2JustRight. Father
 R1Fatso. 0 children."
```

Here are the rules.

0. You may not use an if-statement. This function is best written with a single return statement that consists of the catenation of several parts. For readability, put each part on a separate line. In developing the return statement, it is best to work with one part at a time, making sure it is right before proceeding to the next part. You will use conditional expressions, and we give you one of them below.
1. Exactly one blank appears between words (with the exception of the nickname, which can have any spacing within it, depending on what is in the nickname field). Periods '.' are as indicated in the examples.
2. Get the gender with a blank after it using this conditional expression: (isMale() ? "Male " : "Female ")
3. The word "elephant" is followed by a blank, the elephant's nickname, a period '.', and a blank.
4. The tag is given in the form " `Tag` <integer>`.` ". It appears *only* if the elephant has a tag.
5. The birth month and year always appear, as in the examples.
6. Suppose the mother is known and its nickname is "`xxx` ". Then "`Mother xxx.` " appears, as in the examples. Similarly for the father. If the mother (father) is not known, nothing appears about it. The mother appears first.
7. The number of children appears as shown in the examples, with a period after it —when it is 1 child, use "child"; otherwise, use "children".

In testing `toString`, you need enough test cases to ensure that each different way of evaluating a conditional expression is tested. For example, to test whether the gender appears correctly, you need at least two test cases, for a female `Elephant` and for a male `Elephant`.

When making up a test case, construct it by hand, based on what you read above. Then, write an `assertEquals` statement that compares this expected value to the computed one. This is what we did.

Step 3. Comparison functions. Write the functions in the table below. Each produces a boolean value. **Do one at a time**: (1) write the function and (2) test it thoroughly with test cases in class `ElephantTester`. **Then** proceed to the next function. Put all the test cases for these methods in one test procedure.

Here are the ground rules for writing these comparison functions:

1. The names of your methods much match those listed above **exactly**, including capitalization. The number of pa-

rameters 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 cause our testing program to fail. Parameter names are never tested, so you can change the parameter names 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. You may not use if-statements, conditional expressions, or arithmetic operations (e.g. addition, multiplication, division). The purpose is to practice using boolean expressions, with operators && (AND), || (OR), and ! (NOT).
4. Function `isParent` must be written in terms of calls on `isMother` and `isFather`. We want you to learn to call methods already written instead of doing duplicate work, saving you time and avoiding redundant code. Similarly, the second `isSister` function should be written in terms of a call on the `isSister`.

Method	Suggested javadoc specifications
isMother(Elephant e)	= "e is not null and e is this elephant's mother". Note: *The notation above means precisely this: the value of the function is true if e is not null and e is this elephant's mother —and is false otherwise. So, e may be null (in which case, the function call's value should be false).* The same comment holds for the remaining specifications.
isFather(Elephant e)	= "e is not null and e is this elephant's father".
isParent(Elephant e)	= "e is not null and e is this elephant's parent".
isSister(Elephant e)	= "e is this elephant's sister". Precondition: e is not null. Note: *B is A's sister if* (1) *B and A are different elephants,* (2) *B is female, and* (3) *B and A have at least one parent in common.*
isSister(Elephant f, Elephant e) *Make this function static*!!	= "e is f's sister". Precondition: e and f are not null. *Write this function in terms of the previous isSister function.*
isYounger(Elephant e, Elephant f) *Make this function static*!!	= "e is younger than f". Precondition: e and f are not null.

Submitting the assignment

Check these points before you submit your assignment:

1. Did you use an if-statement? Did you use a conditional expression outside of `toString`? Get rid of them.
2. 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.
3. Did you check your javadoc? Click the javadoc button in DrJava. As you know, this generates webpages, which contain specification of the classes and methods of the classes. 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.
4. Review the learning objectives and re-read this document to make sure your code conforms to our instructions.
5. ADD A COMMENT AT THE BEGINNING OF FILE Elephant.java THAT SAYS HOW MANY HOURS YOU SPENT ON THIS ASSIGNMENT. We will report the min, mean, median, and max.
6. Did you set the preferences in your operating system so that extensions always appear, as requested earlier? For you own sake, do so.
7. Upload files Elephant.java and ElephantTester.java to A3 on the CMS by the due date, Wednesday, 28 September, 11:59PM. Be careful; the same directory as your .java files may also have files that end with .class or .java~ but otherwise have the same name.