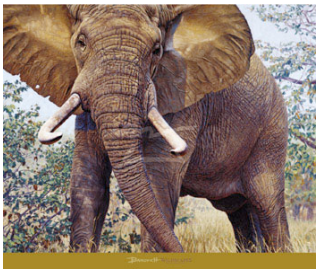
CS1110 Fall 2011 Assignment A1
Monitoring Elephants



This website given below tells you that elephants, the largest living land animals, are threatened by shrinking living space and poaching (for their tusks). This site says that elephants are key players in the forest. The water wells they dig are used by other animals, they create habitat for grazing animals, and the roadways they make act as fire breaks and drainage conduits:

worldwildlife.org/species/finder/elephants/elephants.html

The pygmy elephant in Borneo is also endangered. Much smaller than African elephants. This website talks about studying their habits: www.worldwildlife.org/species/finder/asianelephants/borneoelephant.html.

Web page www.redlist.org/ says that the number of endangered vertebrates (mammals, birds, reptiles, amphibians, and fishes) grew from 3,314 in 1996/98 to 6,714 in 2010, somewhere around 25% of species. For more info on endangered species, see www.worldwildlife.org/species/index.html.

Some animals are tagged to study their living habits. Here in Ithaca, one can see deer with tags on their ears wandering in the fields. The picture to the right is Gries's backyard, near Community Corners. The deer population is too big for the area; they get hit by cars (about 10 a year in Cayuga Heights) and they eat everyone's plants. Discussions are going on to figure out how to control them.

Your task in this assignment is to develop a Java class `Elephant`, which will maintain information about elephants, and a JUnit class `ElephantTester` to maintain a suite of test-cases for `Elephants`. This assignment is the first of two that, together, illustrate how Java's classes and objects can be used to maintain data about a collection of things —like entities in a large tracking system.

Learning objectives

- Gain familiarity with DrJava and the structure of a basic class within a record-keeping scenario (a common type of application).
- 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:" and "=" in specifications, indentation, short lines, etc.), which help make programs readable and understandable and allow us to process your assignments and give you feedback more quickly.
- 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.

The methods we ask you to write in this assignment are short and simple; the emphasis in assignment A1 is on "good practices", not complicated computations.

Iterative grading feedback (revise-and-resubmit cycle)

To help ensure that the learning objectives are met, we will engage in an iterative feedback process. If an objective has not been met, we give you feedback, you revise, and you resubmit; this continues until you are done. This process should be finished by 24-25 September. Finish this process and you get a perfect score of 10.

We will look at your code in several steps.

1. If the field specifications, javadoc specifications, or formatting are not appropriate, we will ask you to fix them and resubmit.

2. If the field and javadoc specs (specifications) are ok, we will look at your test cases. If they are inadequate, we will ask you to fix them, test your program again yourself with the new cases, and resubmit.

3. If the test cases are adequate, we will test your program with our own testing program. If there are errors, we will ask you to correct them and resubmit.

Until mastery is achieved, your "grade" on the CMS will be the number of revisions so far, so that we can keep track of your progress. Don't be alarmed if you see a "1" for the assignment at first! The assignment will be considered completed when it passes all three steps outlined above.

Reading carefully

Your chances of completing the assignment in 1 or 2 rounds will be increased by reading carefully and following all instructions. Many requests for resubmission are caused not by confusion about programming but simply by not following instructions. Please visit this webpage on the website of Fernando Pereira, research director for Google:

http://earningmyturns.blogspot.com/2010/12/reading-for-programmers.html

Did you read that webpage carefully? If not, read it again! The best thing you can do for yourself —and us— at this point is to read carefully. This handout contains many details. You *have* to read carefully to get this assignment right. Save yourself and us a lot of time by doing that as you do this assignment.

Collaboration policy

You may do this assignment with one other person. If you are going to work together, then, as soon as possible —and certainly before you submit the assignment— get on the CMS for the course and do what is required to form a group. Both people must do something before the group is formed: one proposes, the other accepts.

If you do this assignment with another person, you must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. Take turns "driving" —using the keyboard and mouse.

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

Getting help

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

Scan the whole assignment before starting. Then, develop class `Elephant` and test it using a class `Elephant-Tester` in the following incremental, methodologically sound way. This will help you complete this programming task (and others) quickly and efficiently. If we detect that you did not develop it this way, we may ask you to start from scratch on a different assignment.

1. Set your DrJava preferences to indent 4 spaces (use menu item Preferences->Miscellaneous and change the entry "Indent Level" to 4). Make sure that the "Show All Line Numbers" option is turned on in DrJava (in the Preferences window, click "Display Options" in the Categories pane and check the appropriate box).

2. Start a new directory on your hard drive to contain the files for this project. (You should create a new directory for every assignment in this course). In DrJava, write a skeleton class definition for class `Elephant`, with no methods or fields in it, and save it in the new directory. Compile it.

3. In class `Elephant`, declare the following fields (you can choose the names), which are meant to hold information describing an elephant. Make all these fields private and properly comment them (see the "The class invariant" section below). Compile often as you proceed.

   Instruction: break any long lines in your code (including comments) into two or more lines so that we do not have to scroll right to read them. After compiling, make sure the program is properly indented by selecting (highlighting) all lines and hitting the tab key.

   - nickname (a String). Name given to this `Elephant`, a String of length > 0. Many `Elephants` can have the same name.
   - year of birth (an int). Year the `Elephant` was born —must be > 1900.
   - month of birth (an int). Month the `Elephant` was born —in range 1..12, with 1 meaning January.
   - gender of this `Elephant` (a character). Must be either 'B' (meaning boy) or 'G' (meaning girl).
   - this `Elephant`'s tag (an int). An integer >= 0. –1 means that this `Elephant` has no tag yet.
   - mother (an `Elephant`).
     (name of) the object of class `Elephant` that is the mother of this object —null if unknown.
   - father (an `Elephant`).
     (name of) the object of class `Elephant` that is the mother of this object —null if unknown.
   - number of children of this `Elephant` (based on the mother and father fields).

   The class invariant. Recall that comments should accompany the declarations of all fields to describe what each field means, what constraints hold on the fields, and what the legal values are for the fields. For example, for the year-of-birth field, state in a comment that the field contains the year of birth and that it must be > 1900. The collection of the comments on these fields is called the *class invariant*. Here is an example of a declaration with a suitable comment:

   char gender; // the gender of this elephant. 'B' for boy, 'G' for girl

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

   Important: Break long comments onto several lines so right-scrolling isn't necessary to read them, and indent lines properly to follow the structure of the code.

   Important: Do not include things like "(an int)" in a comment for a field. That phrase is placed above only so that you know what type the field should be. It is not needed in the comment because the type is obvious from the declaration.

4. Start a new JUnit class and call it `ElephantTester`.

5. Below are described three *groups* A, B, and C of methods. Work with *one* group at a time, performing steps (1)..(4). **Do not go on to the next group of methods until the group you are working on is thoroughly tested and correct.**

   (1) Write the Javadoc specifications for each method in that part. Make sure they are complete and correct — look at the specifications we give *you*. Copy and paste is best.

(2) Write the methods.

(3) Write *one* test procedure for this group in class `ElephantTester` and add test cases to it for all the methods in the group.

(4) Test the methods in the group thoroughly.

The descriptions below represent the level of completeness and precision we are looking for in your Javadoc comments. In fact, it is best to copy and paste these descriptions to create the first draft of your Javadoc comments. If you do not cut and paste, adhere to the conventions we use, such as using the prefixes "Constructor: …", or "= ..." (the equals sign), or double-quotes to enclose an English boolean assertion. Using a consistent set of good conventions in this class will help us all.

In our specifications, there are no references to specific field names, since the user may not know what the fields are, or even if there are fields. The fields are private. Consider class `JFrame`: you know what methods it has, but not what fields, and the method specifications do not mention fields. In the same way, a user of your class `Elephant` will know about the methods but not the fields.

The names of your methods must match those listed below exactly, including capitalization. The number of parameters and their order must also match: any mismatch will cause our testing programs to fail, meaning that you will have to resubmit. Parameter names will not be tested —change the parameter names if you want.

Important: **You may *not* use if-statements anywhere in this assignment**. They are not necessary. Submissions containing if-statements will be returned for you to revise.

Important: Do *not* write code that checks whether preconditions hold. It is the responsibility of the caller to ensure that the precondition is met. This means, for example, that you should *not* worry about (or write code that checks for) a display name that is null.

Group A: The first constructor and all the getter methods of class `Elephant`.

Constructor	**Description (and suggested javadoc specification)**	
Elephant(String n, int y, int m, char g)	Constructor: a new Elephant with nickname n, birth year y, birth month m, and gender g. Its parents are unknown, and it has no children. Precondition: n's length is > 0. y > 1900. g is either 'B' (for boy) or 'G' (for girl)	
Getter Method	**Description (and suggested javadoc specification)**	**Return Type**
getName()	= this Elephant's nickname	String
getYear()	= year this Elephant was born	int
getMonth()	= month this Elephant was born --1 means jan, 2 feb, etc.	int
isMale()	= "this Elephant is a boy"	boolean
getTag()	= this Elephant's tag (>= 0; -1 if one)	int
getMother()	= this Elephant's mother (null if unknown)	Elephant (not String!)
getFather()	= this Elephant's father (null if unknown)	Elephant (not String!)
getNumChildren()	= the number of children of this Elephant	int

Consider testing the constructor. Based on the specification of the constructor, figure out what value it should place in *each* field to make the class invariant true. Then, write a procedure named `testConstructor1` in `ElephantTester` to make sure that the constructor filled in ALL the fields correctly. The procedure

should: create one `Elephant` object using the constructor and then check, using the getter methods, that all fields have the correct values, Since there are 8 fields, there should be 8 `assertEquals` statements. As a by-product, all getter methods are also tested.

Group B: the setter methods. When testing the setter methods, you will have to create one or more `Elephant` objects, call the setter methods, and then use the getter methods to test whether the setter methods set the fields correctly. Good thing you already tested the getters! Note that two of the setter methods may change more than one field; your testing procedure should check that *all* fields that may be changed are changed correctly.

Important: We are *not* asking you to write methods that change an existing father or mother to a different `Elephant`. Doing so would require if-statements, which are not allowed in this assignment. Read preconditions of methods carefully.

Setter Method	Description (and suggested javadoc specification)
setTag(int n)	Set this Elephant's tag to n. Precondition: n >= 0.
addMother(Elephant e)	Add e as this Elephant's mother. Precondition: e is not null, e is a girl, and this Elephant does not have a mother.
addFather(Elephant e)	Add e as this Elephant's father. Precondition: e is not null, e is boy, and this Elephant does not have a father.

Important: methods `addMother` and `addFather` may have to change fields of both this `Elephant` and the parent, in order to maintain the class invariant. But do not write code to check whether preconditions hold —for example, method `addMother` should not check that `e` is a girl.

Group C: the second constructor. The test procedure for group C has to create an `Elephant` using the second constructor (this will require first creating two `Elephants`, using the first constructor) and then check that the second constructor sets *all* seven fields properly.

Constructor	Description (and suggested javadoc specification)
Elephant(String n, int y, int m, char g, int t, Elephant ma, Elephant pa)	Constructor: a new `Elephant` with nickname n, birth year y, birth month m, gender g, tag t, mother ma, and father pa. Precondition: n's length is > 0, y > 1900, m is the month number, in 1..12, g is 'B' for boy or 'G' for girl, tag t >= 0 (or –1 if not tagged), and ma and pa may not be null.

6. 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 writing a call on each method from the specification that you see by clicking the Javadoc button —that is, without knowing anything about the private fields. Thus, the fields should not be mentioned.

   Then, and only then, add a comment to the top of your code saying that you checked the Javadoc output and it was OK.

7. Review the learning objectives and re-read this document to make sure your code conforms to our instructions; this may help reduce the number of revisions that will be needed. Check the following:

   - Are all lines short enough that horizontal scrolling is not necessary (about 80 chars is long enough).

- Do you have a blank line before the specification of each method and no blank line after the specification?
- Is your class invariant appropriate —are constraints for all fields listed?
- Did you check to make sure that all specifications are complete? Did you check the Javadoc version and then put a comment at the top of the class?

8. Upload files `Elephant.java` and `ElephantTester.java` on the CMS by the due date, Saturday, 17 September, 11:59PM. 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.
9. **Check the CMS daily until you get feedback from a grader**. Make sure your CMS notifications for CS1110 are set so that you are sent an email when one of your grades is changed.

   Within 24 hours, do **RRRRR**: **R**ead the feedback, **R**evise your program accordingly, **R**esubmit, and **Request** a **Regrade** using the CMS. *If you do not request a regrade, we have no simple way of knowing that you have resubmitted*. The whole process should be finished within one week.