CS2110 Fall 2014 Assignment A1
Monitoring Elephants

Introduction

The website www.worldwildlife.org/elephants tells you that elephants, the largest living land animals, are threatened by shrinking living space and poaching (for their tusks). The 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.

The pygmy elephant in Borneo is also endangered. Much smaller than African elephants, they don't get over 6.5 feet tall. The website www.worldwildlife.org/species/borneo-pygmy-elephant talks about tagging pygmy elephants in order to study their habits.

Elephants are not the only endangered species. 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. See www.worldwildlife.org/endangered for more info on endangered species.

Some animals are tagged to study their living habits. Some tags emit a signal, so that the animal can be tracked. Here in Ithaca, one can see deer with tags on their ears wandering in the fields. Gries and James see them around their houses. In fact, the deer population is too big for the area; they get hit by cars (about 10 a year in Cayuga Heights) and eat everyone's plants.

Your task in this assignment is to develop a Java class `Elephant` that will maintain information about elephants, and a JUnit class `ElephantTester` to maintain a suite of test-cases for `Elephant`s.

Learning objectives

- Gain familiarity with the structure of a 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 presentation conventions for this course (Javadoc specifications, indentation, short lines, etc.), which help make your programs readable and understandable.
- Learn and practice incremental coding, a sound programming methodology that interleaves coding and testing.
- Learn about and practice thorough testing of a program using JUnit testing.
- Learn to write *class invariants*.
- Learn about *preconditions* of a method (requirements of a call on the method that need not be tested) and the use of the Java assert statement for checking preconditions.

The methods to be written are short and simple; the emphasis in this assignment is on "good practices", not complicated computations.

Reading carefully

At the end of this document is a checklist of items for you to consider before submitting A1, showing how many points each item is worth. Check each one *carefully*. A low grade is almost always due to lack of attention to detail and to not following instructions —not to difficulty understanding OO. At this point, we ask you to visit the 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 now! The best thing you can do for yourself —and us— at this point is to read carefully. This handout contains many details. Save yourself and us a lot of anguish by reading carefully 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 need help with the CMS, visit www.cs.cornell.edu/Projects/CMS/userdoc/.

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. You should 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. You may not show or give your code to another person in the class.

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.

Using the Java assert statement to test preconditions

As you know, a *precondition* is a constraint on the parameters of a method, and it is up to the user to ensure that method calls satisfy the precondition. If a call does not, the method can do whatever it wants.

However, especially during testing and debugging, it is useful to use Java assert statements at the beginning of a method to check that the precondition is true. For example, if the precondition is “this elephant’s mother is unknown and e is female.”, one can use an assert statement like the following (using obvious names for fields):

```
assert mom == null  &&  e != null  &&  e.gender == 'F';
```

The additional test `e != null` is there to protect against a null-pointer exception, which will happen if the argument corresponding to `e` in the call is **`null`**. [This is important!]

In this assignment, all preconditions of methods must be checked using assert statements at the beginning of the method. Write the assert statements as the first step in writing the method body, so that they are always there during testing and debugging. Also, when you generate a new JUnit class, make sure the VM argument -ea is present in the Run Configuration. You will find these assert statements helpful in testing and debugging.

How to do this assignment

Scan the whole assignment before starting. Then, develop class `Elephant` and test it using class `ElephantTester` in the following incremental, sound way. This will help you complete this (and other) programming tasks quickly and efficiently. If we detect that you did not develop it this way, points will be deducted.

1. In Eclipse, create a new project, called a1Elephant. In a1Elephant, create a new class, called `Elephant`. It should not be in a package, and it does not need a method `main` (but you may add one if you want). As the first line of file `Elephant.java`, put this:

   ```
   /** An instance (i.e. object) maintains info about an elephant. */
   ```

   Remove the constructor with no parameters, since it will not be used and its use can leave an object in an inconsistent state (see below, the class invariant).

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

   Note: you must break long lines (including comments) into two or more lines so that the reader does not have to scroll right to read them. This makes your code much easier for us and *you* to read. A good guideline: No line is more than 80 characters long.

   - nickname (a `String`). Name given to this `Elephant`, a String of length > 0.
   - year of birth (an **`int`**). Must be >= 2000.
   - month of birth (an **`int`**). In range 1..12 with 1 being January, etc.
   - gender of this `Elephant` (a **`char`**). 'M' means male and 'F' means female.
   - mother (an `Elephant`). The object of class `Elephant` that is the mother of this object—null if unknown
   - father (an `Elephant`). The object of class `Elephant` that is the father of this object—null if unknown.
   - number of children of this `Elephant`.

   Note the following about the field that contains the number of children. The user *never* gives a value for this field; it is completely under control of the program. For example, if an `Elephant e` is given a mother `m`, then `m`'s number of children must be increased by 1. It is up to the program, not the user, to increase the field.

   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 >= 2000. The collection of the comments on these fields is called the *class invariant*. Here is an example of a declaration with a suitable comment. Note that the comment does *not* give the type (since it is obvious from the declaration), it does not use noise phrases like "this field contains …", and it *does* contain constraints on the field.

   ```
   char birthYear; // year of birth of this Elephant. >= 2000
   ```

   Note again that we did not put "(a char)" in the comment. That information is already known from the declaration. Don't put such unnecessary things in the comments.

   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 prevent or catch bugs later on.

3. In Eclipse, start a new JUnit test class and call it `ElephantTester`. You can do this using menu item **File —> New —> JUnit Test Case**.

4. Below, four *groups* A, B, C, and D of methods are described. Work with *one* group at a time, performing steps (1)..(4). **Do not go on to the next group 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 below. Copy-and-paste from this handout makes this easy.

   (2) Write the method bodies, starting with assert statements for the preconditions.

   (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.

Discussion of the groups of methods. The descriptions below represent the level of completeness and precision we are looking for in your Javadoc specification-comments. In fact, you may (should) 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 prefix "Constructor: …" and double-quotes to enclose an English boolean assertion. Using a consistent set of good conventions in this class will help us all.

Method specifications do not mention fields because 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 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 and will result in loss of points for correctness. Parameter names will not be tested —change the parameter names if you want.

In this assignment, you may *not* use if-statements, conditional expressions, or loops.

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

Constructor	**Description (and suggested javadoc specification)**	
Elephant(String n, char g, int y, int m)	Constructor: an instance with nickname n, gender g, birth year y, and birth month m. Its parents are unknown, and it has no children. Precondition: n has at least 1 character, y >= 2000, m is in 1..12, and g is 'M' for male or 'F' for female.	
Getter Method	**Description (and suggested javadoc specification)**	**Return Type**
getName()	Return this elephant's nickname	String
getYear()	Return the year this elephant was born	**int**
getMonth()	Return the month this elephant was born	**int**
isFemale()	Return the value of "this elephant is a female"	**boolean**
getMom()	Return this elephant's mother (null if unknown)	Elephant (not String!)
getDad()	Return this elephant's father (null if unknown)	Elephant (not String!)
getChildren()	Return the number of children of this elephant	**int**

Consider testing the constructor. Based on its specification, figure out what value it should place in each of the 7 fields to make the class invariant true. Then, write a procedure named `testConstructor1` in `ElephantTester` to make sure that the constructor fills in ALL 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 7 fields, there should be 7 `assertEquals` statements. As a by-product, all getter methods are also tested.

We advise creating a second `Elephant` (in `testConstructor1`) of the other sex than the one first created and testing —using function `isFemale()` — that its sex was properly stored.

Group B: the setter/mutator methods. Note that methods `addMom` and `addDad` may have to change fields of both this `Elephant` and its parent in order to maintain the class invariant —the parent's number of children changes!

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 these setter methods may change more than one field; your testing procedure should check that *all* fields that may be changed are changed correctly.

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

Setter Method	Description (and suggested javadoc specification)
addMom(Elephant e)	Add e as this elephant's mother. Precondition: this elephant's mother is unknown and e is female.
addDad(Elephant e)	Add e as this elephant's father. Precondition: This elephant's father is unknown and e is male.

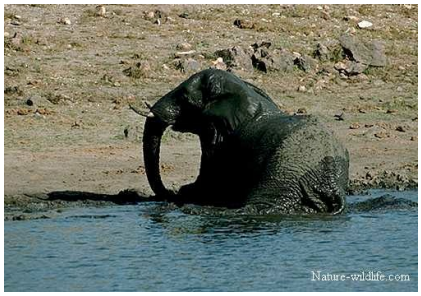
Group C: Another constructor. The test procedure for group C has to create an `Elephant` using the constructor given below. This will require first creating two `Elephants` using the first constructor and then checking that the new constructor set *all* 7 fields properly.

Constructors	Description (and suggested javadoc specification)
Elephant(String n, char g, int y, int m, Elephant ma, Elephant pa)	Constructor: an elephant with nickname n, gender g, birth year y, birth month m, mother ma, and father pa. Precondition: n has at least 1 character, y >= 2000, g is 'F' for female or 'M' for male, m is in 1..12, ma is a female, and pa is a male.

Group D: Write two comparison methods —to see which of two elephants is older and to see whether two elephant are siblings (i.e. they are not the same object and they have a non-null mother or a non-null father in common). Write these using only boolean expressions (with !, &&, and || and relations <, <=, ==, etc.). *Do not use if-statements, switches, addition, multiplication, etc*. Each can be written as a single return statement.

Comparison Method	Description (and suggested javadoc specification)	Return type
isOlder(Elephant e)	Return value of "this elephant is older than e." Precondition: e is not null.	**boolean**
isSibling(Elephant e)	Return value of "this Elephant and e are siblings." (note: if e is null they can't be siblings, so false should be returned).	**boolean**

5. In Eclipse, click men item **Project -> Generate Javadoc**. In the window that opens, make sure you are generating Javadoc for project, a1Elephant, using visibility **`public`**, and storing it in a1Elephant/doc. Then open doc/index.html. You should see your method and class specifications. Read through them from the perspective of someone who has not read your code. Fix the comments in class `Elephant`, 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 at the top of file Elephant.java saying that you checked the Javadoc output and it was OK.

6. Check carefully that a method that adds a mother or father to a `Elephant` updates the mother's or father's number of children. Four methods do this.

7. Review the learning objectives and reread this document to make sure your code conforms to our instructions. Check each of the following, one by one, carefully. Note that 50 points are given for the items below and 50 points are given for actual correctness of methods.
    - 5 Points. 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 it?
    - 10 Points. Is your class invariant correct —are all fields defined and all field constraints given?
    - 5 Points. Is the name of each method and the types of its parameters exactly as stated in step 4 above (the simplest way to do this was to copy and paste). (More points may be deducted if we have difficulty fixing your submission so that it compiles with our grading program.)
    - 10 Points. Are all specifications complete, with any necessary preconditions? Remember, we specified every method carefully and suggested copying our specs and pasting them into your code. Are they in Javadoc comments?
    - 5 points. Do you have assert statements at the beginning of each method that has a precondition to check that precondition?
    - 5 Points. Did you check the Javadoc output and then put a comment at the top of class `Elephant`?
    - 10 Points. Did you write *one* (and only one) testing method for each of the groups A, B, C, and D of step 4? Thus, do you have four (4) test procedures? Did you properly test? For example, in testing each constructor, did you make sure to test that all 7 fields have the correct value? For example, testing whether one date comes before another date, when each is given by a month and a year, probably requires at least 5 test cases.
8. Upload files `Elephant.java` and `ElephantTester.java` on the CMS by the due date. 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.