CS1110 Spring 2010 Assignment A1
Beautiful Butterflies

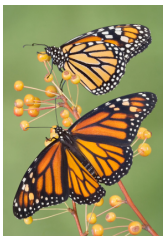
Introduction

There are 24,000 different kinds of butterflies. The one on the right is a Blue Morpho. We don't know the name of the one the left. The website www.butterflywebsite.com will tell you all about butterflies, including telling you which butterflies live where in the US, and giving you links to websites for butterflies in other countries. Some species are endangered. The National Geographic has an article on bringing back the Schaus swallowtail, which was put on the endangered list in 1984.

Imagine being able to monitor different butterflies to see how they live, eat, travel, mate, etc., the way we do elephants, deer, and other animals. Here in Ithaca, one can see deer with tags on their ears wandering in the fields. Gries sees them often in his back yard near Community Corners. 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. There are endless discussions on how to control them.

Monitoring butterflies would be difficult —would we do it by attaching RFID tags to them (what an awful idea)? But from www.nature-gifts.com/live-butterfly-kits.html you can buy butterfly kits, which contain a net-cage, and a few caterpillars, so you can watch them change from caterpillar, to pupa or chrysalis —in a cocoon— and finally to butterfly. The images below were taken from www.thebutterflysite.com/life-cycle.shtml.



In this assignment, we assume we can monitor butterflies. Your task will be to develop a Java class `Butterfly` to maintain information about them and a testing class `ButterflyTester` to maintain a suite of test-cases for `Butterfly`. This assignment illustrates 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 Java and Eclipse 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 (Javadoc specifications, indentation, short lines, etc.), which help make your programs readable and understandable
- 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 using JUnit testing
- Learn about *preconditions* of a method: requirements of a call on the method that need not be tested

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

Reading carefully

To help ensure that the learning objectives are met, at the end of this document is a checklist of points for you to consider before submitting A1, showing how many points each is worth. Check each point *carefully*. A low grade is almost always the lack attention to detail, to not following instructions, and not to difficulty understanding OO.

At this point, we ask you to visit 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. You *have* to read carefully to get this assignment right. Save yourself and us a lot of anguish 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 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. Or, ask a question and look for answers on the Piazza for the course. Do not wait. A little in-person help can do wonders. See the course homepage for contact information.

How to do this assignment

Scan the whole assignment before starting. Then, develop class `Butterfly` and test it using a class `ButterflyTester` 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 a1. In a1, create a new class, called `Butterfly`. It should not be in a package, and it does not need a method main. As the first line of Butterfly, put this: /

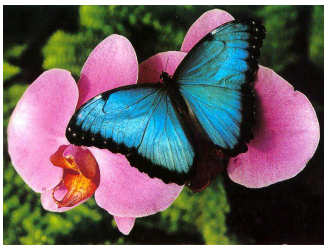
   /** An instance (i.e. object) maintains information about a butterfly. */

   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 `Butterfly`, declare the following fields (you can choose the names), which are meant to hold information describing a butterfly. Make these fields private and properly comment them (see the "The class invariant" section below).

   Note: 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.

   - nickname (a String)



name given to this `Butterfly`, a String of length > 0. Many butterflies can have the same nickname.

- year of hatching (an int). Must be >= 2000.
- month of hatching (an int), in range 1..12 with 1 being January.
- gender of this `Butterfly` (a character)
  'M' means male and 'F' means female.
- mother (a `Butterfly`) . (Name of) the object of class `Butterfly` that is the mother of this object—null if unknown
- father (a `Butterfly`). (Name of) the object of class `Butterfly` that is the father of this object—null if unknown
- number of children of this `Butterfly`

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-hatching field, state in a comment that the field contains the year of hatching 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), and it *does* contain restraints on the field.

char hatchYear; // year of hatching of this `Butterfly`. >= 2000

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.

Remember to 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.

3. In Eclipse, start a new JUnit test class and call it `ButterflyTester`. You can do this using men 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 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 below. Copy-and-paste makes this easy.

   (2) Write the methods.

   (3) Write *one* test procedure for this group in class `ButterflyTester` 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 comments. In fact, you may 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 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 `Butterfly` 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, meaning that you will have to resubmit. Parameter names will not be tested —change the parameter names if you want.

In this assignment, **you may *not* use if-statements or loops anywhere**. They are not necessary.

Do *not* write if-statements to check whether preconditions hold. It is the responsibility of the caller to ensure that each precondition is met.

However, you may use an assert statement

assert <boolean expression> ;

to check a method's precondition. This can be useful in finding errors not only in your code but in your testing procedures. For example, in procedure `addMother(Butterfly e)` in group B, based on its specification, you can use an assert statement like this:

```
assert e != null  &&  !e.isMale()  &&  getMother() == null;
```

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

Constructor	**Description (and suggested javadoc specification)**	
Butterfly(String n, int y, int m, char g)	Constructor: a new Butterfly with nickname n, hatching year y, hatching month m (in 1..12 for Jan..Dec), and gender g. Its parents are unknown, and it has no children. Precondition: n's length is > 0 and y >= 2000. g is either 'M' or 'F'. m in 1..12.	
Getter Method	**Description (and suggested javadoc specification)**	**Return Type**
getName()	Return this Butterfly's nickname	String
getYear()	Return the year this Butterfly hatched from its egg	int
getMonth()	Return the month this Butterfly hatched from its egg	int
getGender()	Return this Butterfly's gender ('M' or 'F')	char
isMale()	Return "this Butterfly is male"	boolean
getMother()	Return this Butterfly's mother (null if unknown)	Butterfly (not String!)
getFather()	Return this Butterfly's father (null if unknown)	Butterfly (not String!)
getNumChildren()	Return the number of children of this Butterfly	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 `ButterflyTester` to make sure that the constructor filled in ALL fields correctly. The procedure should: create one `Butterfly` 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.

Group B: the setter or mutator methods. Note that methods `addMother` and `addFather` may have to change fields of both this `Butterfly` and the parent, in order to maintain the class invariant —the parent's number of children changes! Do not write code to check whether preconditions hold —for example, method `addMother` should not check that `e` is female.

When testing the setter methods, you will have to create one or more `Butterfly` 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 `Butterfly`. This would require if-statements, which are not allowed here. Read preconditions of methods carefully.

Setter Method	Description (and suggested javadoc specification)
addMother(Butterfly e)	Add e as this Butterfly's mother. Precondition: e is not null, e is female, and this Butterfly does not have a mother.
addFather(Butterfly e)	Add e as this Butterfly's father. Precondition: e is not null, e is male, and this Butterfly does not have a father.

Group C: the second constructor. The test procedure for group C has to create a `Butterfly` using the second constructor (this will require first creating two `Butterfly`s, using the first constructor) and then check that the second constructor sets *all* 7 fields properly.

Constructor	Description (and suggested javadoc specification)
Butterfly(String n, int y, int m, char g, Butterfly ma, Butterfly pa)	Constructor: a new `Butterfly` with nickname n, hatch year y, hatch month m (in 1..12 for Jan..Dec), gender g, mother ma, and father pa. Precondition: n's length is > 0; y >= 2000, m is in 1..12, g is `'M'` for male or `'F'` for female, and ma and pa are not null.

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

Comparison Method	Description (and suggested javadoc specification)	Return type
isOlder(Butterfly e)	Return value of `"this butterfly is older than e"`. Precondition: e is not null.	boolean
isSibling(Butterfly e)	Return value of `"this butterfly and e are siblings"`. Precondition: e is not null.	boolean

5. In Eclipse, click men item Project -> Generate Javadoc. In the window that opens, make sure you are generating Javadoc for project a1, using visibility **public**, and storing it in a1/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 Butterfly, 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.

6. Check carefully that a method that adds a mother or father to a butterfly updates the mother's or father's number of children. Three 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).
- 10 Points. Is your class invariant correct —are all fields defined and are constraints for all fields 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).
- 5 points. Do you have a blank line before the specification of each method and no blank line after it? Is the specification in a Javadoc comment?
- 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.
- 5 Points. Did you check the Javadoc version and then put a comment at the top of class `Butterfly`?
- 10 Points. Did you write *one* (and only one) testing method for each of the groups A, B, C, and D of step 4? So, there are four (4) test procedures? Did you properly test? For example, in testing each constructor, did you make sure to test that all seven fields have the correct value?

8. Upload files `Butterfly.java` and `ButterflyTester.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.