Finn Lidbetter and William Fiset
Comp 3721
Dr. Ricker
November 25, 2016

Droles de Petites Betes Design Document

Design Decisions

To design the software requested, we came up with the structure for a program that will notify beasts of trees in bloom. This structure was designed with Object Oriented principles in mind. To meet the requirements of the software we wanted to implement the following basic functionalities:

- The ability to represent multiple types of trees
- The ability to represent multiple trees of the same type that can each bloom at different times
- Multiple different types and instances of beasts that are each interested in possibly different types of trees
- Some subscription system that allows beasts to subscribe/unsubscribe to certain types of trees
- Some notification system that lets all subscribed beasts know when a tree is and is not in bloom

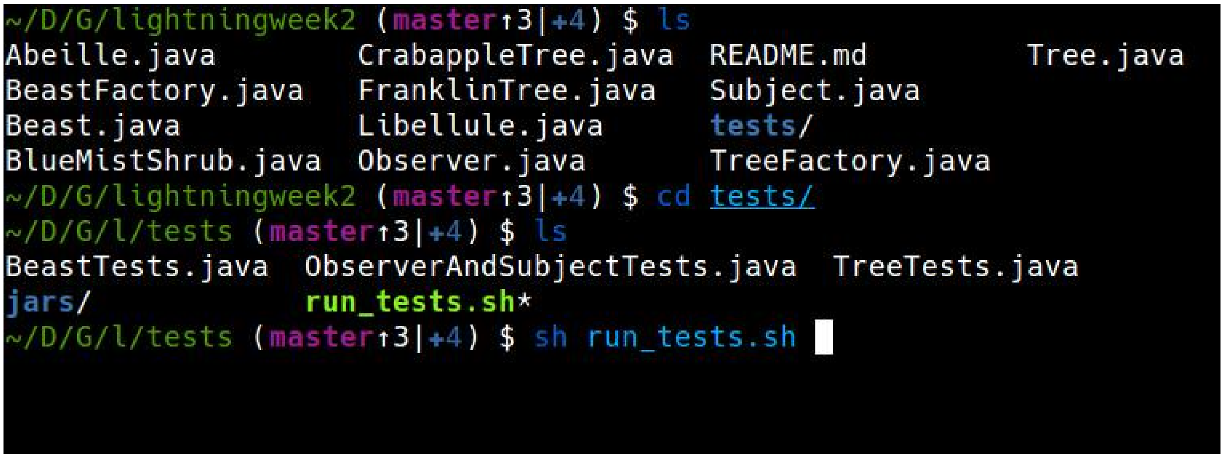
To meet these requirements we found that the most natural solution was to use an Observer pattern. This is because we have a number of different entities that are interested in receiving information from a set of, possibly different, information providers. In this case the information receivers are the beasts, and the information providers are the trees. We chose to have a tree notify all of its 'subscribed' beasts any time that it changes from being not in bloom to in bloom, or in bloom to not in bloom. The beasts then each maintain a set of all of the trees that are in bloom at a given time such that they can harvest pollen from those trees.

We also decided to make use of the Factory design pattern to create each of the Trees and each of the Beasts. This provides a clean and easy way to construct different types of Trees and Beasts.

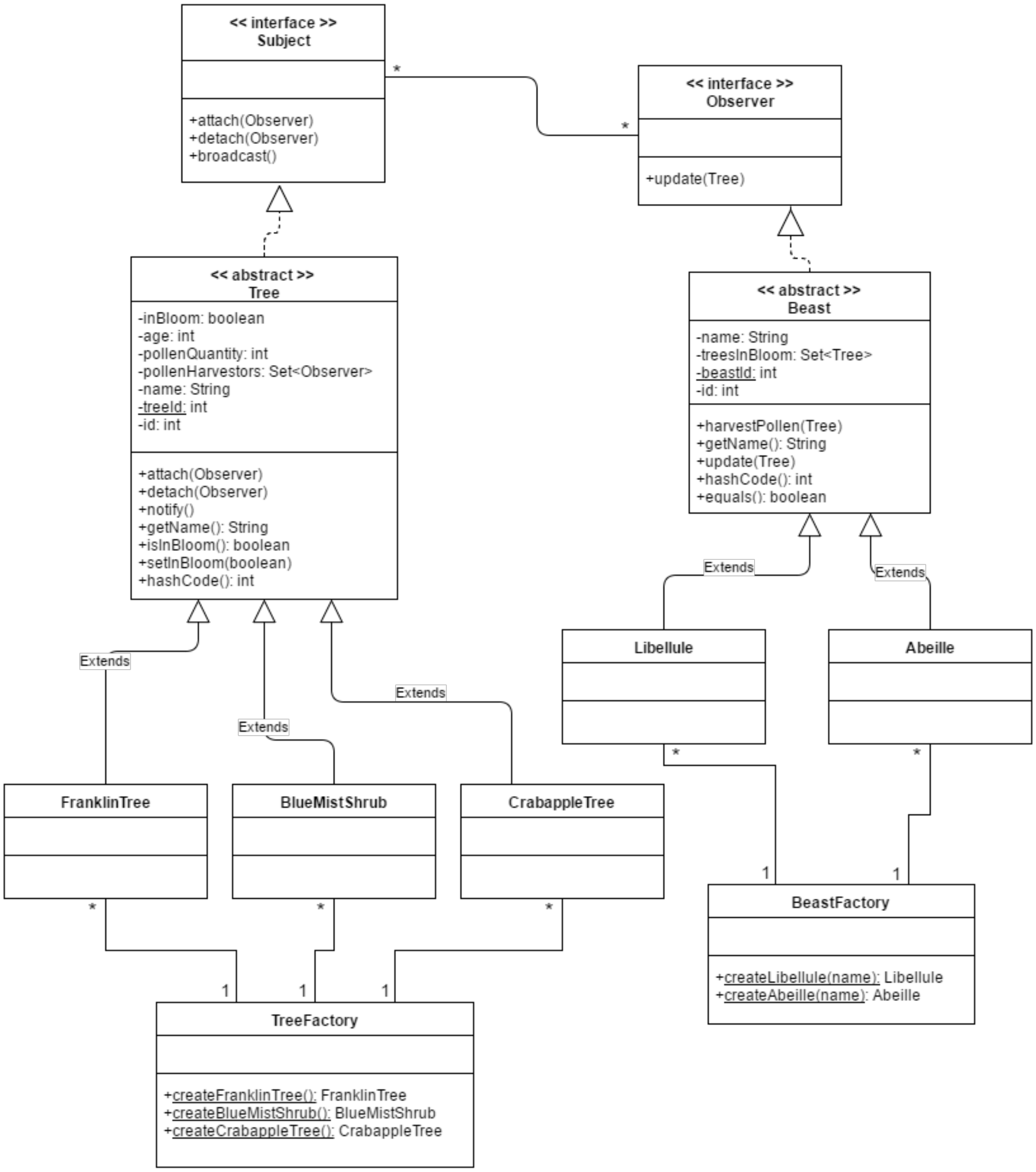
Finally, we decided to use abstract classes for Beasts and Trees. This is because all Beasts share some of the same qualities and all Trees share some of the same qualities. This removes the need for code repetition for the similar implementations that they each have.

How to run tests:

You can run the tests for this program by navigating to the tests/ directory at the root of the submission folder and then executing the shell script named "run_tests.sh" (see image below).



```
~/D/G/lightningweek2 (master↑3|+4) $ ls
Abeille.java        CrabappleTree.java  README.md         Tree.java
BeastFactory.java   FranklinTree.java   Subject.java
Beast.java          Libellule.java      tests/
BlueMistShrub.java  Observer.java       TreeFactory.java
~/D/G/lightningweek2 (master↑3|+4) $ cd tests/
~/D/G/l/tests (master↑3|+4) $ ls
BeastTests.java  ObserverAndSubjectTests.java  TreeTests.java
jars/            run_tests.sh*
~/D/G/l/tests (master↑3|+4) $ sh run_tests.sh
```



<< interface >>
Subject
+attach(Observer)
+detach(Observer)
+broadcast()
*
*
<< interface >>
Observer
+update(Tree)
<< abstract >>
Tree
-inBloom: boolean
-age: int
-pollenQuantity: int
-pollenHarvestors: Set<Observer>
-name: String
-treeId: int
-id: int
+attach(Observer)
+detach(Observer)
+notify()
+getName(): String
+isInBloom(): boolean
+setInBloom(boolean)
+hashCode(): int
<< abstract >>
Beast
-name: String
-treesInBloom: Set<Tree>
-beastId: int
-id: int
+harvestPollen(Tree)
+getName(): String
+update(Tree)
+hashCode(): int
+equals(): boolean
Extends
Extends
Libellule
Abeille
Extends
Extends
Extends
FranklinTree
BlueMistShrub
CrabappleTree
*
*
1
1
BeastFactory
+createLibellule(name): Libellule
+createAbeille(name): Abeille
*
*
*
1
1
1
TreeFactory
+createFranklinTree(): FranklinTree
+createBlueMistShrub(): BlueMistShrub
+createCrabappleTree(): CrabappleTree