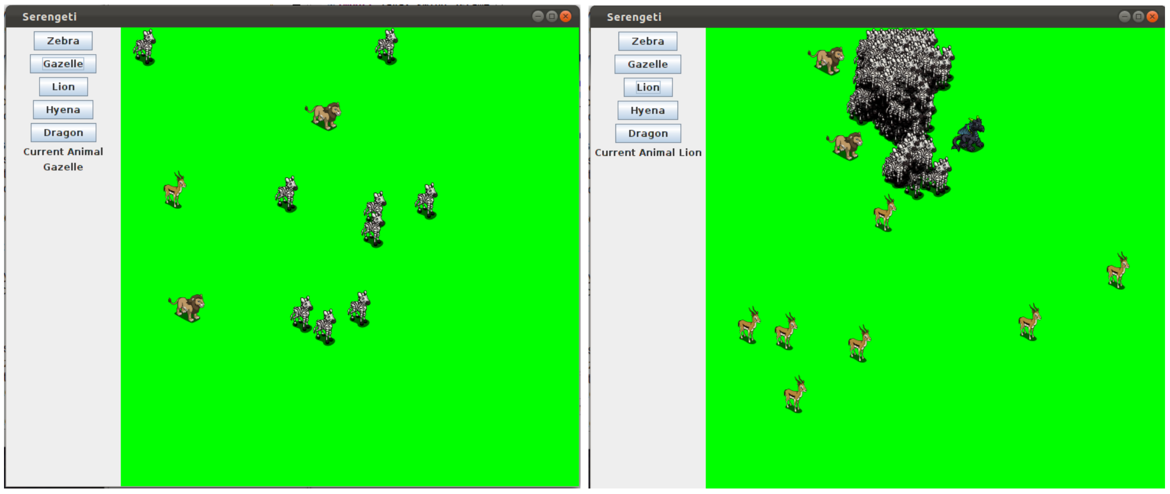
CS 1331 Homework 11

Due April 5, 2012 at 8:00 pm

In this assignment you'll be implementing an Animal hierarchy to fit into the provided GUI components. This will demonstrate how inheritance, polymorphism, and general good OOP style can make your code easily extensible.

When doing the homework, you should not change the provided GUI code except where noted (ie to add additional Animals). **The main objective of this assignment is to get the GUI provided to work as described while conforming to the guidelines of the hierarchy. Exact implementation may be variable.**



Important

Do not forget about javadocing. All methods in all .java files should be javadoc'd. Please follow the javadoc guidelines outlined in Homework 4.

To recap:
  You must have class javadocs:
    description, @author, and @version tags
  You must have method javadocs:
    description, @param, and @return tags

- `@author` For indicating who wrote the class
- `@version` Shows the version number for the class, and possibly the date
- `@param paramName` For explaining what a parameter means in a method or constructor. Note the parameter name is repeated in the javadoc.
- `@return` For explaining what a method returns

1 Serengeti

We are providing you with three files, `Serengeti.java`, `SerengetiPanel.java`, and `ControlPanel.java`. You should refer to these files when constructing your hierarchy. The basic requirement is that your hierarchy work with the provided GUI and your overall program run within the constraints provided (ie population control, etc).

To see where objects are instantiated, refer to `ClickListener` inside the Panel. As you can see, instantiated Animals are passed an x location, a y location, and a Rectangle object. The Rectangle object, `bounds`, is used to designate the area in which the Animal is confined (in this case, the size of the Serengeti plains). This means that the Animal should never be allowed to move outside that area.

Note that all the Animal references are stored in a single ArrayList, `ArrayList<Animals> animals`. You should not be modifying the SerengetiPanel code in any way (ie keeping track of separate Carnivore and Herbivore lists). The only modifications you may add are to the mouseListener to support more Animals (and also in Control Panel).

2 Animal Hierarchy

The main part of this assignment will be implementing the Animal hierarchy. There are a few components to this hierarchy, and this document will give an overview as to each of those, and it is up to you to fill in the details. Remember to use good OOP design when creating this hierarchy.

If you find yourself copy-pasting code or writing redundant code, this is a warning that your design might not be as good as it could be. In general, any similar code should be placed as high up in the hierarchy as possible.

2.1 `Animal` class

The Animal class going to be the top-most class in the hierarchy. You can see it being used in the provided GUI code (see `SerengetiPanel.java`).

Remember that Java allows you to create special abstract classes that do not implement all of their methods, but simply declare some of them (like interfaces do) and requires that children of that class implement them. Although this will not be explicitly stated for the following classes (you must use your own discretion), Animal should be `abstract`.

Below is a list of the methods that is used from the Animal class in the GUI code. If you feel that more will be needed, feel free to add them; but remember to always use good design.

- `move()` will move the Animal in a random yet effective manner (it doesn't move in circles). Each time an animal moves, it's health should decrease and its age should increase.
- `draw(Graphics)` should draw the Animal at its location.
- `collidesWithAnimal(Animal)` returns whether or not this Animal is colliding with a given Animal. This should be determined by the Animal's location and the dimensions of its image.
- `canReproduceWithAnimal(Animal)` returns whether or not the two animals can reproduce, which is only if they're of the same class.

- `reproduceWithAnimal(Animal)` if the two Animals are able to reproduce (as determined by `canReproduceWithAnimal(`
  this method returns a new Animal of the same type and in the same location. If for some reason, reproduction does not happen, `null` should be returned.
  Reproduction rates should be **controlled** in some manner so that infinite population growth does not occur. This can be acheived in a variety of ways: reproduction probability, limiting total offspring per animal, etc.
- `isOld()` returns whether or not an Animal has surpassed some determined maximum Age.
- `die()` called when an Animal dies.
- `isDead()` returns whether or not the animal is dead (ie health ≤ 0).

2.2 `Herbivore` class

`Herbivore`s extend the functionality of the `Animal` class, but change some if it slightly.

Since all Animals lose health as they move, Herbivores have the ability to "graze" once their health has fallen below some set level. This will replenish their health.

One of the main goals of this hierarchy is to not be copy and pasting code. Code should be placed up as high in the hierarchy as possible, and should be reused by calling a superclass's method where appropriate.

2.3 `Carnivore` class

`Carnivore`s also extend the functionality of `Animal` class, but add in their own methods.
The methods that `SerengetiPanel` requires from `Carnivore` are:

- `canEatAnimal(Animal)` returns whether or not this Carnivore can eat a given Animal. Typically `Carnivore`s can eat anything that are `Herbivore`s.
- `eatAnimal(Animal)` will eat the given Animal by taking all of its health away and adding it to its own health. This should not do anything if it cannot eat the animal (as described by `canEatAnimal(Animal)`).

2.4 `Zebra` class

`Zebras` are `Herbivores`. You can choose specific parameters for them, such as how fast they move, how much health they lose per movement, how often they have to eat, etc.

2.5 `Gazelle` class

`Gazelles` are `Herbivores`. Gazelles should be able to move faster than any of the other Animals we have listed.

2.6 `Lion` class

`Lions` are `Carnivores`.

2.7 `Dragon` class

`Dragonss` are `Carnivores`. However, they are different from all other carnivores in that they can eat any type of `Animal` that is not a `Dragon`, including other Carnivores. Dragons also never die of old age, but do not reproduce frequently.

2.8 Additional classes

In addition to having Zebras, Gazelles, Lions, and Dragons, you should create two custom Animal classes. In addition to having basic stats, each should have some special aspect or action that radically differs from other Animals (eg, `Dragon`'s "eat everyone" ability).
You are allowed to edit SerengetiPanel and ControlPanel to incorporate these additional classes.

There are nice animal pictures at `http://farmville.wikia.com/wiki/Zoo#Zoo_Animals`. Also searching for "`ANIMALNAME 50px farmville`" on google images will give you the smaller sizes.

3 General notes

Follow good OOP principles and programming practices.

- Classes that should be logically abstract should be indicated as such in the code.
- Proper visibility modifiers should be used.
- Understand fully how your code works, in the event that you are asked to demo your program
- Remember to use relative image paths, or else your code will not compile
- Since no one reads the last page apparently, if the code you download from T-Square does not compile or run, you will get a 0.
- Use proper hierarchy construction. Make sure you are reusing constructors and other methods by calling super().
- Alter various parameters such as reproduction rate, movement speed, movement patterns, etc to get a decent looking simulation. We will be grading on functionality.

Turn-in Procedure

Turn in the following files on T-Square. When you're ready, double-check that you have submitted and not just saved as draft.

- Serengeti.java
- SerengetiPanel.java
- ControlPanel.java
- All the classes in your Animal hierarchy (including custom ones)
- All the images required to run your program
- Any other files needed to compile and run

All .java files should have a descriptive javadoc comment.

Verify the Success of Your HW Turn-In

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.** (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

**Note: Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework (past the grace period of 2 am) will not be accepted regardless of excuse. Treat the due date with respect. The real due date and time is 8 pm Thursday.