Programming Assignment 8: Critters

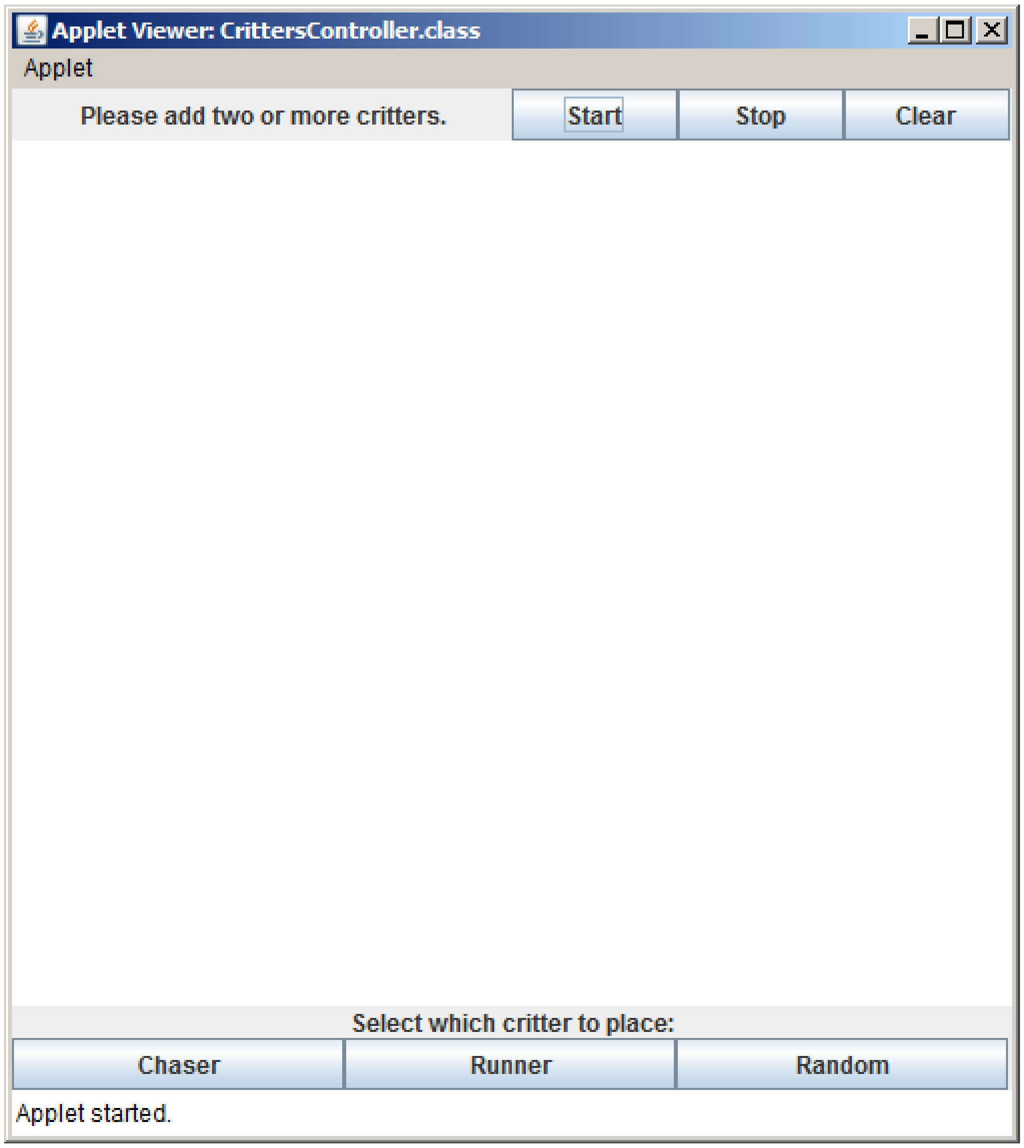
Due: 11:59pm Thursday March 7

Overview

You will create a simulation program that controls the movement of three types of Critters. The program will allow any number of Critters to be added to the world, which is the canvas of the applet. The Critter to be added is selected by pressing on the corresponding button for the Critter. The simulation can be started or stopped at any time, using the Start and Stop buttons. Additionally, all Critters can be removed from the simulation and from the canvas with the Clear button. Critters can be added at anytime, whether the simulation is running or not. If added in the running state, new Critters should start responding immediately. The simulation should only run if there are two or more Critters currently existing on the canvas.

GUI and Controller

Here is the screenshot of the how the program looks when it is first started up:


Applet Viewer: CrittersController.class
Applet
Please add two or more critters.
Start
Stop
Clear
Select which critter to place:
Chaser
Runner
Random
Applet started.

Your controller class should be named `CrittersController.java`. As the very first step, create all the GUI elements and lay them out as shown in the screenshot above. You will need to make use of multiple JPanels to create the layout, and add the panels to CrittersController. The CrittersController should extend WindowController and implement ActionListener. As an ActionListener, it should add itself to the 6 buttons that it will create for the GUI. In addition to the 6 buttons, there are two JLabels: one at the top indicating the status of the simulation, and one at the bottom indicating which Critter is currently selected to be placed on the canvas.

The status label at the top will contain one of three status messages at any given time. Use the following rules to determine what status message to display:

- If the simulation is in the **stopped** state, the status should read **"Simulation is stopped."**
- If the simulation is in the **started** state, but there are less than 2 Critters on the canvas, the status should read **"Please add two or more critters."**
- If the simulation is in the **started** state, and there are 2 or more Critters on the canvas, the status should read **"Simulation is running."**

Note that it is possible for the status message to change by clicking ANY three of the control buttons on the top or by clicking on the canvas to place new Critters. Thus, you will probably want to create a method to which you delegate the task of figuring out and setting the status message, instead of having this logic spread out in different places. For example, hitting the Clear button in the started state should set the status to tell the user to add more critters. Adding two or more Critters in the started state to an initially empty canvas should change the status to "Simulation is running". And so on. Simply call your method any such time it is needed, which will set the status based on the three rules above.

Once the Chaser, Runner, or Random button is clicked, clicking anywhere on the canvas will place that selected Critter on the clicked location. A Critter's location is considered to be the point corresponding to its **center**. When placing Critters, you can switch between one Critter and another at any time. Also, when a Critter button is clicked, the creature selection label, which initially reads "Select which critter to place:", should be changed to say **"Click on canvas to place a [Critter]"**, where [Critter] is Chaser, Runner, or Random, depending on which Critter was clicked.

When starting up, the program should begin in the **start** state.

Critter appearance and movement

<u>Chaser</u>

The Chaser will be drawn as a **red** **circle** using a FilledOval object ●. The diameter of the circle should be 15 pixels. The Chaser's behavior is to move **towards** the Critter that is closest to it. However, a Chaser should NEVER chase after another Chaser. Thus, in determining which Critter is the closest to a Chaser, you should ignore all other Chasers.

<u>Runner</u>

The Runner will be drawn as a **green** **square** using a FilledRect object 🟩. The size of the square is 15x15 pixels. The Runner's behavior is to move **away from** the Critter that is closest to it. However, if a Runner ever touches any of the four borders of the canvas, it should be randomly moved to a different location inside the canvas.

<u>Random</u>

The Random will be drawn as a **blue** X using two individual Line objects ✕. The bounding box formed by the X should be 15x15 pixels. The Random's behavior is to move randomly from its current location to a nearby location. Thus, the movement of the Random is not influenced by any other Critters around it. However, Chasers and Runners can still be influenced by nearby Randoms.

No Critter should be allowed to move outside of the canvas boundaries. This means that you need to use the Critter's **perimeter** – NOT its center – to detect if a Critter is about to move outside of the view of the canvas. You do not need to worry about the applet resizing once it is started, so assume that the canvas size will not change.

The Chaser, Runner, and Random should each subclass a parent `Critter.java` ***abstract* class**. The Critter class should contain a constructor with the specified arguments, like so:

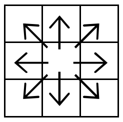
```
public Critter(Location loc, DrawingCanvas canvas)
```

You can add more constructor arguments to this constructor or have other constructors in addition to this one if you so need, but at a minimum you need to have the above constructor. Also, the Critter.java should have at least the following abstract method:

```
public abstract void reactTo(Critter other);
```

This method will be passed in a reference to a Critter that is determined to be the closest to this Critter on which reactTo() is called. This way, the current Critter can update its location based on the location of the other Critter that is closest to it. (Note that the Random will not need to use the "other" Critter argument, since a Random determines its next location randomly, without any influence from surrounding Critters). Refer to the objectdraw documentation for the distanceTo() method of Location, which you will very likely need to use to calculate distances between two Critters. Also, the Double.MAX_VALUE constant in Java will be useful.

Every time a Runner and Chaser need to react, they can move to one of 8 surrounding locations. For the Chaser, this will be whichever of the 8 locations brings it the closest to the "other" Critter. For the Runner, this will be whichever of the 8 locations puts it furthest away from the "other" Critter. To understand this better, imagine a 3x3 grid, with the center cell representing the Critter's current location, and the 8 surrounding cells are the possible new locations:



As a result, both the x coordinate and the y coordinate of the Critter will change by -1 or 1. You need to loop through each of the 8 locations and figure out which is best for the Critter, depending on whether it is a Chaser or a Runner. All of this has to do only with Chaser and Runner. The movement behavior of the Random will be very different. For Random, the x and y coordinates should each be changed by a random value between -10 and 10, inclusive. Make sure you generate random values in that range **separately** for the x coordinate and for the y coordinate, instead of using a single random value for both coordinates. **When figuring out the new location of ANY Critter, whether it is a Chaser, Runner, or Random, be very careful to ensure that the new location does not cause the Critter to move outside the canvas boundaries.** This will probably be more tricky for the Random.

Keeping track of Critters and running the simulation

In CrittersController, you will want to use a Java ArrayList or Vector to hold all the Critters that are added to the canvas. For example,

```
ArrayList<Critter> critters = new ArrayList<Critter>();
```

To actually run the simulation, you will need something that extends ActiveObject. This is where you will create a `CrittersSimulator.java` class, which will extend ActiveObject and define the corresponding **run()** method. CrittersController should create an instance of CrittersSimulator, passing in a reference to the critters ArrayList. Thus, both the Controller and the Simulator will act on the same reference to the critters ArrayList. You will likely need to use a boolean flag in CrittersSimulator, along with a setter and getter for the flag, to allow the simulation to be stopped and started. CrittersSimulator should run the simulation only in the started state, and only when there are 2 or more Critters on the canvas. The bulk of the work that CrittersSimulator's run() method has to do is to calculate the distance from every Critter to every ***other*** Critter (with the Chaser-Chaser exception mentioned earlier), and then call the reactTo() method for each Critter after having determined the closest Critter to it.

For the run method, please **stick to the following structure**:

```
public void run(){
   while(true){
       pause(DELAY);

       /* All the rest of your work should go here */

   }
}
```

Note the pause(DELAY). Please leave this pause() at the top level of the infinite while-loop as you see here, and nowhere else. Regardless of whether the simulation is currently running or stopped, the pause() should always be executed inside the infinite while-loop for the run() method. You can play around with different values of DELAY when you are doing the assignment, but please make sure to set DELAY to somewhere around 40-50 before you turn in the assignment.

Extra Credit

1. <u>(2 points)</u>

Add a fourth Critter called Imitator. The Imitator will be drawn as a **black triangle** using three Line objects △. The bounding box created by the triangle should be 15x15 pixels. The Imitator's behavior is to mimic the movement of the Critter that is closest to it. While imitating that Critter, the Imitator should change its color to the color of the Critter that it is imitating (but the shape should always stay a triangle). So, the Imitator should make its next move be exactly what the closest Critter would do if the closest Critter were at the same position as the Imitator. Like the Chaser-Chaser exception mentioned earlier in the assignment, all other Imitators should be ignored when finding the closest Critter to an Imitator. Remember to add a corresponding Imitator button and select creature text to the GUI controller.

2. <u>(2 points)</u>

For Extra Credit 1, all other Imitators were ignored when finding the closest Critter to an Imitator. For this Extra Credit, change this so that if the closest Critter to an Imitator is another Imitator, then the current Imitator starts spinning around in confusion. To simulate the spinning of the triangle, you will need to change the orientation of the triangle to one of four possibilities, cycling from one orientation to the next:

△ ▷ ▽ ◁

3. <u>(1 point)</u>

If an Imitator is imitating a Chaser, prevent actual Chasers from chasing this Imitator and prevent this Imitator from chasing other Chasers. As you might have realized, this is to keep our Chaser-Chaser interaction the same for an Imitator for the times that it impersonates a Chaser.

README
You need to provide a README file with this assignment. Make sure that the filename is "README". Note that this is in all caps and has no extension. In your README, you should

1. Provide a high level description of what your program does and how you can interact with it. Make this explanation such that your grandmother or uncle or someone you know who has no programming experience can understand what this program does and how to use it.
2. Answer the following questions:
    1. A high school student is trying to write a Java program that will draw different shapes and in different possible colors. To do this, she has written just one Java class called ShapeDrawer, which contains all the necessary methods like drawRedCircle(), drawBlueCircle(), drawYellowSquare(), drawGreenSqare(), and so on. Using object-oriented terminology, describe how you can help the student improve her design.
    2. List at least two ways in which Java Interfaces and Java abstract classes are **different**.
    3. List at least two ways in which Java Interfaces and Java abstract classes are **alike**.
    4. How can you run gvim through the command line to open all Java source code files in the current directory, each file in its own tab?
    5. Suppose you are currently inside a directory and in there you want to make a new directory called fooDir. And inside fooDir, you want another directory called barDir. Using only a **single** mkdir command, how can you create a directory called fooDir with a directory called barDir inside it?

Files to turn in

- CrittersController.html
- CrittersController.java
- CrittersSimulator.java
- Critter.java
- Chaser.java
- Runner.java
- Random.java
- README

- Extra Credit: Imitator.java

Grading Rubric

- README – 10 points
- Style – 20 points
- Correctness – 70 points
- Extra Credit – 5 points

<u>Due: 11:59pm Thursday, March 7</u>

START EARLY!!!