# CSE142—Computer Programming I
# Programming Assignment #8
# due: Tuesday, 3/7/2006, 2 pm
# (Final late deadline: Friday, 3/10/2006, 5 pm)

## Introduction.

This assignment will give you practice with defining classes. You are to write a set of classes that define the behavior of certain animals. You will be given a program that runs a simulation of a world with many animals moving around in it. You get to implement three aspects of the behavior of each kind of animal:

- what character the simulation uses to graphically represent an animal; different animals will use different characters. For example, a turtle uses the character T.
- how the animal moves around the world; different kinds of animals will move in different ways.
- when two animals meet each other in the simulation, they play a simple game called "The Prisoner's Dilemma." You get to implement the strategy that each kind of animal uses when playing the game. This will be very simple unless you are interested in making complicated critters.

Each of your classes will implement the Critter interface, shown below.

```
// Stuart Reges, Carl Ebeling, Steve Gribble
// Last modified: March 1st, 2006
//
// The Critter interface defines the methods necessary for an animal to
// participate in the critter simulation.
//
// getChar():
//   It must return a character when getChar is called that
//   is used for displaying it on the screen.  For example, a Bird
//   should return a 'B' character.
//
//   ... etc.  (comments truncated; the rest are in Critter.java)

public interface Critter {

    public char getChar();

    public int getMove(int x, int y);

    public boolean decide(int lastgame, Class animal);

    public static final int NORTH = -1;
    public static final int SOUTH = 3;
    public static final int EAST = 8;
    public static final int WEST = 11;

}
```

Interfaces are discussed in detail in chapter 9 of the textbook, but to do this assignment you just need to know a few simple rules about interfaces. Your class headers should indicate that they implement this interface, as in:

```
public class Bird implements Critter {
    <class definition>
}
```

Because you are implementing the Critter interface, you will be able to refer to the class constants directly (e.g., NORTH). The other important detail about interfaces is that you must include in each class a definition for the three methods in the interface (the getChar method, the getMove method and the decide method).

## Critter characters.

You will be implementing five different kinds of animal; accordingly, you are to implement five separate Java classes, one per animal type. Here is the list of classes you should implement, and what each of these classes should return from the getChar method:

| Class | getChar() |
|---|---|
| Bird | B |
| Frog | F |
| Mouse | M |
| Turtle | T |
| Wolf | W |

## Moving around the world.

Critters move around in a world of finite size, but the word is toroidal (going off the end to the right brings you back to the left and vice versa; going off the end to the top brings you back to the bottom and vice versa). Whenever the simulation decides to move an critter, it will invoke the getMove method on the object instance associated with that critter. As arguments to the getMove method, the simulation will pass in the current position of the critter in the simulated world. You get to implement the getMove method for each kind of critter; here's how each critter should move:

| Class | getMove(int x, int y) |
|---|---|
| Bird | Randomly selects one of the four directions to move each time. |
| Frog | Picks a random direction, moves 3 in that direction, repeat. (Same as bird, but staying in a single direction longer) |
| Mouse | Moves west 1, north 1, repeat (zig zag to the NW) |
| Turtle | South 5, west 5, north 5, east 5, repeat (clockwise box) |
| Wolf | You get to pick whatever you like for the Wolf! However, Wolves get to move twice as often as all other animals. |

For the random moves, each possible choice must be equally likely. Use the Math.random() method to obtain pseudorandom values.

The first four of these classes won't use the (x, y) coordinates that are passed to getMove. This information is provided in case you want to do something with this information in defining your Wolf class. The critter world is divided into cells that have integer coordinates, much like the pixels in a DrawingPanel. There are 100 cells across and 50 cells up and down. As with the DrawingPanel, the upper-left cell has coordinates (0, 0), increasing x values move you right and *increasing y values move you down*.

Notice that the Critter class defines four constants for the various directions. You can refer to these using the name of the interface (Critter.NORTH, Critter.SOUTH, etc) or you can refer to them directly (NORTH, SOUTH, etc). Your code should not depend upon the specific values assigned to these constants, although you may assume they will always be of type int. You will lose style points if you fail to use the named constants when appropriate.

## What happens when critters meet?

The simulator has two different varieties -- a “simple” simulator, and a “complex” simulator. The difference between these two varieties lies in what happens if two critters happen to move into the same location.

In the “simple” variety, the simulator flips a coin to decide which of the two critters lives, and which dies. Each critter has an equal probability of living and dying. If a critter dies, it is removed from the simulation. So, over time, the simulation will have fewer and fewer critters running around the world.

In the “complex” variety, when a critter moves to a location where there is already another critter, the two critters play a game with each other called “The Prisoner’s Dilemma.” By playing the game, a critter has the opportunity to earn money. Over time, the simulation keeps track of how much money a critter has earned.

The rules of the Prisoner’s Dilemma are easy to understand. The game proceeds through a fixed number of rounds. (The number of rounds is randomly selected at the beginning of the game.) For each round, the simulator calls the decide() method on the two critters playing the game. Each critter has to return one of two values: **true** if the critter wants to “cooperate” this round, and **false** if the critter wants to “defect” this round. The simulator then looks at the responses from the two critters, and based on the values returned, the simulation decides how much money to give each critter as follows:

| | | **Critter #2** | |
|---|---|---|---|
| | | Cooperate | Defect |
| **Critter #1** | Cooperate | Both gain 2 dollars | Critter #1 gains 0 dollars<br>Critter #2 gains 3 dollars |
| | Defect | Critter #1 gains 3 dollars<br>Critter #2 gains 0 dollars | Both gain 1 dollar |

(The Prisoner’s Dilemma is a very interesting game with lots of strategic theory behind it. In case you’re curious, you can find more information about it at the following Web page: http://plato.stanford.edu/entries/prisoner-dilemma/ Or, look on Google! )

Here's the strategy that you should implement for each kind of critter:

| Class | decide(int lastgame, Class animal) |
|---|---|
| Bird | Always cooperates (i.e., always returns true). |
| Frog | Flips a coin (i.e., returns true at random half the time, returns false at random half the time.) |
| Mouse | Always defects (i.e., always returns false). |
| Turtle | Alternates between cooperating and defecting. |
| Wolf | You get to pick whatever strategy you choose! |

The decide method has the following interface:

```
public boolean decide(int lastgame, Class animal);
```

Although you must declare your decide method with these parameters, you do not actually have to use them to implement the actions in the table above. Since using these parameters is optional, they are described in the Strictly Optional section on the next page.

## Additional Considerations

Style points will be awarded on the basis of your use of good comments and variable names and your ability to express these operations simply and clearly. Any data fields (state variables) of your classes should be declared using the "private" keyword.

Your classes should be stored in files called Bird.java, Frog.java, Mouse.java, Turtle.java and Wolf.java. You will need to include the files Critter.java, CritterModel.java, CritterFrame.java, CritterGameParameters.java, CritterPanel.java and CritterMain.java in the same folder as your program to run the GUI. You should open and compile CritterMain to run the program. The files that you need will be included in a zip folder called ass8.zip available from the class web page. You should download and unzip this folder, then write your five classes, compile CritterMain and run.

You are allowed to include constructor methods for your classes if you want, although any constructor that you define has to be a zero-argument constructor (one that takes no arguments).

We recommend that you should design your solution in two stages. In the first stage, assume that we will be playing the simple variety of simulation. As a reminder, in the simple variety, the decide method is never invoked, since the critters do not play the Prisoner's Dilemma game. In the second stage, the world becomes more complicated, and critters play Prisoner's Dilemma and accumulate points.

When you run the simulation (by running CritterMain), you will be asked whether you want the world to be of the simple or complex variety.

Notice that you are asked to define the Wolf class yourself. Four of the style points for this assignment will be awarded on the basis of how much energy and creativity you put into defining an interesting class. These four points will be much harder to earn than the other 16 points for the assignment, so it really is a way for us to reward the students who decide to spend time figuring out an interesting critter definition.

## Strictly Optional

Unless you are interested in designing a very clever Wolf, you do not have to worry about how this game is played, and need only implement the critters as described in the tables above. However, if you do want to design a better wolf, there two ways to get extra useful information.

First, the decide method provides two very useful parameters:

```
public boolean decide(int lastgame, Class animal);
```

The first parameter, lastgame, tells you what happened during the previous round of the game. If this is the very first round of the game, lastgame will have the value -1. If this is a subsequent round, then lastgame will be 0 if your opponent defected last time, and it will be 1 if your opponent cooperated last time. So, the simulation gives you a little bit of history about the game, and you can use that to define complex strategies, if you so choose.

The second parameter, animal, tells you what kind of critter you are playing against. If you decide to make use of this parameter, you can write code that looks something like:

```
if (animal.getSimpleName().equals("Bird")) {

   // do something against the bird

} else if (animal.getSimpleName().equals(“Frog”)) {

  // do something against the frog

} …etc.
```

Second, we have made available a method which the Wolf can use to look for critters on neighboring cells. The static method CritterModel.neighbor(int dx, int dy) returns the Class of the critter on the neighboring square. dx and dy gives the location relative to the Wolf’s position. For example, to look at the cell to the EAST, a Wolf would invoke CritterModel.neighbor(1, 0) and for the cell to the southwest, CritterModel.neighbor(-1, 1). dx and dy must be -1, 0, or 1. If there is no critter at the specified location, null is returned. Here is an example of using the neighbor method:

```
// look NORTH

Class x = CritterModel.neighbor(x, y, 0, -1);

if (x != null) {

   if (x.getSimpleName().equals(“Bird”)) {
      return true;
   } else {
      return false;
   }

}
```

For those of you who are interested, we will run a tournament at the end of class where your wolves can take on the wolves designed by other students. The details of this tournament will be published later.