AP Computer Science
Curriculum Module: Flocking Birds

Christian Day
Emma Willard School
Troy, New York
Flocking Birds

In this lesson, we will focus on the Actor class in the GridWorld Case Study. As you know, the case study is a large body of code designed to simulate the movement of creatures, known as actors, in a grid environment. The world containing the grid allows for many ways of interacting with the actors in the grid. Actor is very loosely defined, but provides the core methods that will be needed in order to provide a rich and interesting set of behaviors to the various actors in the world.

Specifically, the Actor class defines the following methods:

```java
public Actor()
public Color getColor()
public void setColor(Color newColor)
public int getDirection()
public void setDirection(int newDirection)
public Grid<Actor> getGrid()
public Location getLocation()
public void putSelfInGrid(Grid<Actor> gr, Location loc)
public void removeSelfFromGrid()
public void moveTo(Location newLocation)
public void act()
public String toString()
```

The case study is designed so that any class that you wish to build (e.g., Bug, Flower, Critter) will extend the behavior of an Actor. In the case study narrative, students are introduced right away to the Bug class, which is a class that inherits from Actor. For this exercise we will concentrate on creating a Bird class which inherits from Actor, but exhibits behavior typical of birds. Specifically, we would like to get our Bird objects to flock together.

The Bird class

We will introduce the Bird class by inheriting from class Actor and adding the following methods:

```java
public Bird()
public void move()
public void turn()
public boolean canMove()
```
We will also override the `act` method inherited from the `Actor` class:

```java
public void act() {
    if (Math.random() > TURNPCT) {
        turn();
    }
    if (canMove()) {
        move();
    }
}
```

This makes for an interesting-looking world. Rather than copying the `Bug` behavior by turning whenever it encounters an obstacle, the bird will randomly turn fairly regularly based on the value we assign to the constant `TURNPCT`. If a bird cannot move, it will simply stop. This is like having the bird alight on a branch for a brief rest.

**The Flockable interface**

This is interesting behavior, but we would like to enhance the behavior of the bird by giving it the ability to flock to another object. For this we will create the `Flockable` interface:

```java
public interface Flockable {
    public void setLeader(Actor flockLeader);
    public void stopFollowing();
    public boolean isFollowing();
}
```

The `Flockable` interface gives an object the ability to acquire and follow any `Actor` given as its leader. The interface is quite simple, but provides for the implementation of some powerful behavior.
**Exercise A:**

Option 1: Modify the *Bird* class so that it implements the methods in the *Flockable* interface. Observe that you will need to create a private data member of class *Actor* in order to store state relevant to the flocking behavior of the *Birds*.

Option 2: Consider adding the following private data member to class *Bird*:

```java
private Actor myLeader = null;
```

Use `myLeader` to implement the methods required to get class *Bird* to implement the *Flockable* interface. In implementing these methods, write your code in a way that takes advantage of the initialization of `myLeader` to `null`:

```java
/**
 * Provide this Flockable object with a leader that it can follow.
 * Subsequent actions by this Flockable object should cause it to
 * behave in a manner that causes it to follow flockLeader
 * @param  flockLeader The Flockable that this Flockable will follow
 **/ public void follow(Actor leader) {

}

/**
 * Instruct this Flockable to permanently stop following its leader
 **/ public void stopFollowing() {

}

/**
 * Return true if this Flockable has a leader that it is following
 * @return true if this Flockable has a leader, false otherwise
 **/ public boolean isFollowing() {

}
```

Now that the *Bird* class implements the *Flockable* interface, we would like to modify its behavior so that it is capable of following its leader. The following method is essential in making the changes to *Bird* behavior:
private void turnToFace(Location loc) {
    if (loc.equals(this.getLocation())) return;
    int xDist = this.getLocation().getCol() - loc.getCol();
    int yDist = this.getLocation().getRow() - loc.getRow();
    double turn = Math.atan2(yDist, xDist);
    setDirection((int)Math.toDegrees(turn) - 90);
}

Do not be scared off by the use of Math.atan2 in this method. This simply gives us a
direction to face, in radians. This direction is then converted to degrees and transposed to
fit the setDirection method.

Exercise B:
Modify the act method so that if a Bird is following a leader, it will turn to face the
leader before moving. If the Bird is not following a leader, allow it to continue acting
like a normal Bird. In both cases, the move method should be called.

public void act() {
    // Before turning, see if you have a leader. If so, turn to face that
    // leader
    if (Math.random() > turnPct) {
        turn();
    }
    if (canMove()) {
        move();
    }
}

The Flock class

A Flock is a group of Flockable objects stored together in a common class. The
underlying data structure of this class is a List of Flockable objects:

List <Flockable> theFlock = new ArrayList<Flockable>();

The List is defined as holding objects of class Flockable. However, you will observe
that Flockable is an interface, not a class. This restricts the List such that it can only
hold objects of a class that implement the Flockable interface. Since the Flock class
requires behaviors supported by the Flockable interface in order to create a Flock, we
put in this restriction.
The definition of the Flock, above, guarantees that our Flock will only contain objects that implement the Flockable interface. The leader of this Flock, however, can be any Actor object. The addLeader method accepts a leader of class Actor, and stores it in a private data member theLeader. All of the Flockable objects in the Flock are instructed to set that Actor as their leader.

```java
public void addLeader(Actor leader) {
    theLeader = leader;
    Iterator <Flockable> flockIter = iterator();
    while (flockIter.hasNext()) {
        Flockable nextFlockObject = flockIter.next();
        nextFlockObject.setLeader(leader);
    }
}
```

Observe that the Iterator flockIter is used to instruct each Flockable object in the List to follow the leader given. We do not care if the leader is a Bird, a Fish, a Bug, or any other object, so long as it is an Actor.

This ability of the Java programming language (and all object-oriented programming languages, for that matter) to handle objects of different classes through an interface is a common and powerful feature. We use the interface to define the behaviors we want, and then we use a class to implement those behaviors. This ability to deal with objects of many different classes through a common interface is known as polymorphism.

Exercise C:
Observe that after running the FlockRunner sample for some time, the leader Bird inevitably becomes stuck in some position. This generally happens when the leader goes into a corner, and its flock traps it there. Come up with a solution to ensure that the leader will not become stuck in this manner. You may use one of these suggested methods, or come up with one of your own:

1. Tag! After a certain number of steps, the Flock picks a new leader.
2. Survival: After a certain number of steps without moving, the whole Flock becomes hungry and stops following its leader.
3. Darwin: After a certain number of steps without moving, the Flock picks a new leader within its Flock.
4. AI: The leader learns how to lead and avoids getting trapped in corners.
Appendix A: The complete Flockable interface

```java
/*
 * Flockable.java
 */

import info.gridworld.actor.Actor;

/**
 * @author Christian Day
 * @version June 5, 2007
 */
public interface Flockable {
    public void setLeader(Actor flockLeader);
    public void stopFollowing();
    public boolean isFollowing();
}
```
Appendix B: the complete Bird class

```java
/*
 * Bird.java
 */
import info.gridworld.actor.Actor;
import info.gridworld.grid.Grid;
import info.gridworld.grid.Location;

/**
 * @author Christian Day
 * @version June 4, 2007
 */
public class Bird extends Actor implements Flockable {

    private final double TURNPCT = 0.75;
    private Actor myLeader = null;

    public void move() {
        Grid<Actor> gr = getGrid();
        if (gr == null)
            return;
        Location loc = getLocation();
        Location next = loc.getAdjacentLocation(getDirection());
        if (gr.isValid(next))
            moveTo(next);
        else
            removeSelfFromGrid();
    }

    public void act() {
        if (isFollowing()) {
            turnToFace(myLeader.getLocation());
        } else if (Math.random() > TURNPCT) {
            turn();
        }
        if (canMove()) {
            move();
        }
    }

    public void turn() {
        setDirection(getDirection() + Location.HALF_LEFT);
    }

    /** Given a Location, turn to face that Location
     * @param loc The location this Bird should face
     **/
    private void turnToFace(Location loc) {
        if (loc.equals(this.getLocation())) return;
        int xDist = this.getLocation().getCol() - loc.getCol();
        int yDist = this.getLocation().getRow() - loc.getRow();
        double turn = Math.atan2(yDist, xDist);
        setDirection((int)Math.toDegrees(turn) - 90);
    }
}
```
public boolean canMove() {
    Grid<Actor> gr = getGrid();
    if (gr == null)
        return false;
    Location loc = getLocation();
    Location next = loc.getAdjacentLocation(getDirection());
    if (!gr.isValid(next))
        return false;
    Actor neighbor = gr.get(next);
    return (neighbor == null);
}

public void stopFollowing() {
    myLeader = null;
}

public boolean isFollowing() {
    return myLeader != null;
}

public void setLeader(Actor leader) {
    myLeader = leader;
}
Appendix C: The Flock class

```java
public class Flock {
    List<Flockable> theFlock = new ArrayList<Flockable>();
    Actor theLeader;

    public void add(Flockable f) {
        theFlock.add(f);
    }

    public Iterator<Flockable> iterator() {
        return theFlock.iterator();
    }

    public void addLeader(Actor leader) {
        theLeader = leader;
        Iterator<Flockable> flockIter = iterator();
        while (flockIter.hasNext()) {
            Flockable nextFlockObject = flockIter.next();
            nextFlockObject.setLeader(leader);
        }
    }

    public void removeLeader() {
        theLeader = null;
        Iterator<Flockable> flockIter = iterator();
        while (flockIter.hasNext()) {
            flockIter.next().stopFollowing();
        }
    }
}
```
Appendix D: A main class for creating a running two Flocks

```java
/*
 * FlockRunner.java
 */
import info.gridworld.actor.Actor;
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.*;
import java.util.Iterator;

/**
 * @author Christian Day
 * @version June 5, 2007
 **/
public class FlockRunner {
    public static void main (String[] args) {
        Flock robins = new Flock();
        Flock jays = new Flock();

        ActorWorld world = new ActorWorld();

        for (int r = 0 ; r < 5 ; r++) {
            Bird robin = new Bird();
            robin.setColor(java.awt.Color.ORANGE);
            robins.add(robin);
        }
        Bird robinLeader = new Bird();
        robinLeader.setColor(java.awt.Color.ORANGE);
        robins.addLeader(robinLeader);

        for (int j = 0 ; j < 6 ; j++) {
            jays.add(new Bird());
        }
        Bird jayLeader = new Bird();
        jays.addLeader(jayLeader);

        Iterator flockIter = robins.iterator();
        while (flockIter.hasNext()) {
            Bird b = (Bird)flockIter.next();
            world.add(world.getRandomEmptyLocation(), b);
        }
        world.add(world.getRandomEmptyLocation(), robinLeader);

        flockIter = jays.iterator();
        while (flockIter.hasNext()) {
            Bird b = (Bird)flockIter.next();
            world.add(world.getRandomEmptyLocation(), b);
        }
        world.add(world.getRandomEmptyLocation(), jayLeader);

        world.show();
    }
}
```
See FlockingBirds_ProgramFiles_Day.zip file containing java files and images.

2 Bird Images by Ji Mi Lee, Emma Willard School class of 2008