CSE 142 - Su 02
Homework 4
Assigned: Wednesday, July 17
Due: Wednesday, July 24, BEFORE MIDNIGHT

** General Comments about the Homework **

All homework is turned in electronically. Go to the class web site and use the link on the homework page to do the turnin. Don't be late! Late homeworks will not be accepted.

This homework assignment has three parts: Practice Problems, Graded Problems and a Programming Project.

** Homework 4 Practice Problems ** Do not turn in the Practice Problems.

1. Write a method that will copy the objects stored in an ArrayList into a new list and return the new list. There are several ways to accomplish this task. The method header in each case is the same, but the implementation of the body is different.

```
public ArrayList copyList(ArrayList source) {
      ... method body ...
}
```

a. Provide an implementation that creates a new ArrayList object, then copies each element from the source list to the new list using a loop, and then returns the new list to the caller.

b. Look at the documentation for ArrayList in the java.util package. There is a constructor that takes a Collection and returns a new ArrayList containing all the elements of the original Collection. Since an ArrayList is a Collection (it implements the Collection interface) you can use this constructor to build a new ArrayList that is a copy of original list. Provide an implementation that does this and returns the new ArrayList to the caller. (It can be done in one line).

2. Write a method that will copy the objects stored in an ArrayList into a new list in reverse order and then return the new list to the caller. There are several ways to accomplish this task. The method header is:

```
public ArrayList reverseList(ArrayList source) {
      ... method body ...
}
```

3. When you have finished the requirements for the programming problem below, feel free to extend it.

Take a look at class Polygon in the uwcse.graphics package. You can define arbitrarily complex outlines with that class instead of the simple Rectangles and Ovals.

In the example I showed in class, the Car makes a noise when it is clicked on and when it hits a window edge. These are not required features. There is example code in class ShapeSlider that shows how to load and play sound files, if you are interested in this. Note that there are several import statements at the start of the source file that tell Java where to find all the classes it needs to deal with sounds.

ShapeSlider also is an example of how to load and display images. There is an ImageShape class which you can use just like all the other Shapes (Oval, Line, etc). Images are not a required feature for this homework.

Describe your extensions in extensions.txt if you make any changes beyond those called out in the requirements below.

** Homework 4 Graded Problems ** Turn in your answers to the graded problems in the text file answers.txt.

Each of the following 5 questions is worth 2 points.

1. The midterm on Friday July 26 and the review session on Wednesday July 24th will NOT be in our regular classroom.

a. What building and what room will we use for the midterm and the review session?

b. Do you know where that building is located and how to find the room?

2. You have been working with classes that implement the Actor interface, which is defined in the uwcse animation package. There is another interface named Prop defined in that package too.

This is the code for the Actor interface:

```
package uwcse.animation;
import uwcse.graphics.*;
public interface Actor {
  /** Every Actor must implement some fundamental action.
      @param stage the stage on which the actor is performing. */
  public void doAction(Stage stage);

  /** Every Actor must be able to draw itself on a GWindow. */
  public void addTo(GWindow w);

  /** Every Actor must be able to remove itself from its GWindow. */
  public void removeFromWindow();
}
```

a. Refer to the documentation for Prop in the uwcse animation package. What are the methods that are defined in the Prop interface?

b. Notice that there is considerable overlap between the Prop interface methods and the Actor interface methods. Rewrite the Actor interface so that it extends the Prop interface and does not re-specify the methods that are in both interfaces. (Don't worry about the package statement and import statements, just include them in the rewritten code exactly the way they are shown above.)

3. Assume we have a Student class which has the following two methods:

```
public int getID() : Returns the student's ID number.
public String getName() : Returns the student's name.
```

We also have the following (rudimentary) QuizSection class definition.

```
public class QuizSection {
  /** Construct a new QuizSection object. */
  public QuizSection() {
     roster = new ArrayList();
  }
  /** store the currently registered students */
  ArrayList roster;
}
```

a. Write a new method for the QuizSection class called addStudent that takes a Student object as a parameter and adds it to the roster. Be sure to include the javadoc comments that describe the method and its parameter. addStudent does not return a value.

b. Write a new method called isEnrolled which takes a student's ID as a parameter and returns true or false, depending on whether the student is on the roster. Be sure to include the javadoc comments that describe the method, its parameter, and the return value.

4. We have talked about the ArrayList class. It is one of several classes that implements the List interface. Look at the documentation for the List interface (it is part of the Java package java.util). Notice that in addition to the add(Object o) method that I have talked about in class, there is another version of the add method defined that has different parameter variables: "void add(int index, Object element)".

Write a new method called addStudentAtFront for the QuizSection class in question 3 that takes a Student object as a parameter and adds it to the roster at the front of the list. Be sure to include the javadoc comments that describe the method and its parameter. addStudentAtFront does not return a value.

5. Use the Java library documentation to answer the following questions.

a. What methods are guaranteed to be implemented by any class that implements the Comparable interface defined in the java.lang package?

b. What methods are guaranteed to be implemented by any class that implements the FilenameFilter interface defined in the java.io package?

** Homework 4 Programming Project **

In this project, you will do two software upgrades:

1. Change the Car class to display a multi-shape Car instead of the simple Rectangle it uses now. For this, you will use ArrayLists to keep track of all the Car parts.
2. You will upgrade the existing Car class from its current implementation as a plain Actor to a ClickableActor that recognizes when it is being clicked and reverses direction as a result.

++ List of classes ++

As provided to you, homework 4 is a complete and running program. The overall concept of the program is that it builds a set of Actors which can draw images on the graphics window, and then it loops continuously and calls the doAction() method of each Actor every time the clock ticks.

This homework is a little different from earlier homeworks in that the program will loop forever until you click the green box in the upper left corner.

The files involved in this program are the following.

* Property files in subdirectory "cast" *

As in homework 3, this directory holds one file for each element in your picture. See the homework 3 writeup for details on how the files are used, including directions on how to open them with a text editor.

* Car.java *

This is the class that you will modify. As delivered to you, there is one constructor and three methods. You should have a copy of Car.java in front of you while you read this description.

The constructor is called when an object of this class is first created. The constructor gets values from the parameter variable pMap exactly the same way as OvalSlider did in homework 3. Refer to homework 3 for details, or read the existing code.

Right now, the constructor just builds a Car shape that is a single Rectangle. Part of your job, described below, is to implement a fancier, multi-shape Car and use an ArrayList to keep track of all the Shapes that are part of the image.

The doAction(Stage stage) method is called by the animation framework over and over to implement the behavior of your Actors. Each time it is called, you can move the Car a little bit and do whatever else you have defined for this vehicle.

Right now, the doAction method moves the Shape a little bit each clock tick, and bounces it off the walls of the window frame when it reaches an edge. There are several changes to be made to the doAction method. They are described below.

The addTo(GWindow w) method is called by the animation framework early on when the Actor is first created. You need to modify this method to add all the new Shapes.

The removeFromWindow() method is called by the animation framework to remove an Actor from the stage. You need to modify this method to remove all the new Shapes.

* Director.java *

This class contains the main method that is used to start the program running. In BlueJ, you can right-click on the class definition square for Director, and then select "void main(arg)" It will ask you for an argument to pass to main. This argument is the name of the cast directory and it defaults to "cast". You can just click okay, and the program will start running. The main method creates a new Cast, Stage, and AssistantDirector, then asks the AssistantDirector to run the performance. You don't need to make any changes to the Director class.

* the skel directory *

This directory contains all the utility classes for this homework. You can look at this code for interest and information, but you do not need to change it for this assignment.

++ Project Requirements ++

1. Download the cse142-hw4.zip file and unzip it. The project skeleton is in the directory hw142\hw4. Your task is to open the project using BlueJ and modify Car to meet the requirements below.

2. Start BlueJ and open the hw4 project. Open the Car source file and read it. You can see that it will compile and run, but it doesn't implement all the capabilities that are needed.

3. I have given you examples of the way to implement each of the required features. The use of ArrayLists was demonstrated in the upgraded PetSet that I gave you in class on Monday July 15. An implementation of a ClickableActor is provided in class ShapeSlider, included in homework 4.

4. The function of an Actor class like Car is to create and then control one or more Shapes that are shown in the graphics window. A basic Car is already implemented; your job is to implement more features as described below.

4a. Change the Car design so that the image is built from multiple Shapes. In the code provided to you, the Car is just one plain Rectangle. You should add details to provide a more exciting driving experience. Two wheels at minimum should be added to the Car's image.

You should define an instance variable that is of type ArrayList. This will be the container for the Shapes that you create to be the car's image on screen, and will replace the variable "Shape blob" that is used to hold the simple Rectangle in the provided code.

In the constructor, you should create new Shapes representing the parts of the car. At minimum, the car should have a body (represented by a Rectangle or other suitable shape) and two wheels (represented by Ovals). You should add all of the car part Shapes to the ArrayList you created earlier.

You should pay attention to how you calculate the dimensions of the various parts of the car. The prop files can specify width and height values, and your

constructor should use these values to proportion the parts so that a car can be constructed in different sizes and still look okay.

In doAction, you need to change the code so that instead of just moving a single Rectangle Shape, it loops through the entire ArrayList and does a moveBy call for each of the Shapes in your Car.

In addTo, you need to modify the code to add all the Shapes in the fancy car, rather than just the single Rectangle that it has now.

In removeFromWindow, you need to modify the code to remove all the Shapes in the fancy car, rather than just the single Rectangle that it has now.

When this is working correctly, each of the Cars will bounce off the sides of the window frame and all the parts will move together.

Remember, you must use an ArrayList to store all the various Car parts. Although it would be possible to just define more variables like you did when you added a third pet to PetSet, we now have a better way and we need to use it. So define and use an ArrayList.

4b. The second major change is to upgrade the Car so that it is a ClickableActor, not just an Actor. A ClickableActor can take special action when the user clicks on it. The ClickableActor interface extends the Actor interface, and is defined in the file skel\ClickableActor.java.

In order to implement this, you need to make several changes.

The line "public class Car implements Actor" should be changed to "implements ClickableActor".

In the constructor, no changes are needed for the clickability.

In doAction, no changes are needed for the clickability.

You need to add two new methods in order to implement the new behavior called for by the ClickableActor interface.

The method "boolean intersects(Shape other)" is called to decide if some Shape intersects any of the Shapes that this object is displaying. Since your car object is the only object that knows what Shapes it is displaying, it is the only object that can answer this question. So you must implement this method so that it loops through all of the Shapes that are being used to represent the Car, and uses the intersects(other) method defined for all Shapes to decide if there is an intersection. If there is, this method returns "true", if not, it returns "false".

Somewhere in this method you will have a loop containing a statement like this:

```
if (((Shape)myShapes.get(i)).intersects(other)) {...
```

The method "void doClickAction(Stage stage)" is called whenever it is decided that your Car is the object that was clicked on. This method should implement a direction reversal immediately. The car goes the opposite direction from its current direction, just as though it had run into the edge of the window at the moment you clicked on it. Note that you don't actually move anything in this method, you just change the deltaX and deltaY values so that the next time doAction is called, the Car moves in the opposite direction.