Assignment V: Introduction to Object-Oriented Programming

|  |  |  |
| --- | --- | --- |
|  |  | COP3330.R01  Spring 13’  Online Section  Andrew Eisler, Lecturer |

A Fill-in-the-Framework “Comets/Asteroids” game,  
used to teach OOP by example.

# Introduction

The software company you're working for is developing a game named Comets that is essentially a clone of an old popular arcade game. The player controls a spaceship that floats around in two-dimensional space propelled by its engines and carried by its inertia. Comets are also floating around in the same space. Anything that goes off one side of the screen re-emerges on the opposite side. The player's goal is to destroy all of the comets without colliding with any of them. The situation is complicated somewhat by the fact that shooting larger comets causes them to break into several smaller comets, thus making them more difficult to avoid.

# Game Architecture

A basic game interface is already written, so you don't need to worry about drawing anything on the screen or responding to the player pressing buttons. All you need to do is write classes to represent the various objects in the game. All classes are to be located in the package "comets". JavaDoc specifications for these classes are provided on WebCourses (along with a few other important files), but here's a general outline:

* **SpaceObject** is an abstract class representing all of the objects in game, including the comets, bullets fired by the player, and even the player's ship. This should keep track of the position, velocity, and size of the object. It provides methods for updating the object's position based on its velocity, determining whether the object is overlapping with another object, and accessors for the object's position and size. SpaceObject has static fields for the playfield width and height that are set up by the main class and should be used to determine the size of the play area for wrap-around purposes.
* **Shot** objects represent shots fired by the player. Shots should only stay on the screen for a certain length of time, so they have an age counter that increments every time the shot is told to move. The main class will take care of removing the shot from the game based on this age.
* **Comet** is an abstract class representing comets. **LargeComet**, **MediumComet**, and **SmallComet** all extend Comet. Large comets break into two medium comets when shot, and medium comets break into three small comets. Small comets are destroyed outright by being shot. Every comet has an explode() method that returns a Vector containing newly created comets that are produced by their destruction, although SmallComet should return an empty vector since it doesn't spawn additional comets.
* **Ship** represents the player's spaceship. The ship has a direction that it is facing that it uses to figure out the change in velocity when it accelerates, as well as the trajectory of the shots it fires. A ship has methods to turn left, turn right, accelerate, and fire shots. The fire() method returns a Shot object that originates from the center of the ship and travels in the direction that the ship is facing, adjusted, of course, for the ship's own velocity. An important note about acceleration: Even though in real space objects can basically travel arbitrarily fast, it wouldn't be much fun if the ship travels so fast that you can't see it. Limit the maximum speed of the ship to 10 pixels per frame (i.e. per move()). That's fast enough for the game to be exciting, but not so fast that it's impossible to follow the ship.
* **CometsMain** is already provided as part of this assignment via Canvas. CometsMain isn't very good, unfortunately. Up to 10 points of extra credit are available on this assignment for making substantial improvements to CometsMain**.** However, the classes you write *MUST* be compatible with CometsMain in its original form. Examples of substancial improvements are: sounds (.wav files), a scoreboard, and inserting a background image. The highest score you will earn without the improvements is 90 percent (of 8 points toward the entire 3330 grade).
* A configuration file named "**comets.cfg**" should be placed in the **root** directory of the project. This gives the initial layout of the comets. Each line in the file describes a comet. The string at the start of the line specifies the size of the comet ("Large", "Medium", or "Small"), the two numbers after that are the position of the comet (x and y coordinate), and the last two numbers are its velocity.

# Game Constants

* Shot radius – 3 pixels  
  Shot speed – 3 pixels/frame in the direction of the shot + velocity of the ship when it was fired
* Ship radius – 10 pixels Maximum ship speed – 10 pixels/frame  
  Ship acceleration – 0.1 pixels/frame2 Ship turning rate – 0.1 radians/frame
* Large comet radius – 40 pixels Medium comet radius – 30 pixels  
  Small comet radius – 20 pixels

# Useful Formulas

## Determining if two objects overlap:

Let objects one and two be at positions (*x*1, *y*1) and (*x*2, *y*2) and have radii *r*1 and r2, respectively. Objects one and two overlap if:

## Updating the position of an object:

Let an object at position (*x*0, *y*0) be traveling with velocity (*vx*, *vy*). Its position (*x*, *y*) after being moved is determined by: and

## Accelerating the ship:

Let the ship have direction θ in radians. The change in x and y velocity when accelerating is given by: and

The ship's speed, s, is:

if(angle>360)angle=0;

If this speed is greater than 10, scale the velocity down by multiplying both the *x* and *y* velocity by 10/*s*.

## Firing a shot:

If a shot is fired from a ship traveling at velocity (*vship.x*, *vship.y*) and facing at angle *θ* in radians, the resulting shot velocity should be: and

## Generating a Random Velocity:

When a large or medium-sized comet is destroyed, it creates smaller comets that fire off in different directions. The direction and speed of these comets (ie. velocity) is random; however, the speed of the comet must not exceed 10. To prevent from generating random x and y velocities such that the speed exceeds 10, you are advised to generate a random angle and speed, and convert this angle and speed to corresponding x and y velocities.

# Comments on the ‘Vector’ Class

As per the Java 6 API Reference: The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

You will need to use the Vector class in this assignment. The API reference for the Vector class is located here: <http://download.oracle.com/javase/6/docs/api/java/util/Vector.html>

# dELIVERABLES

You must submit the source code (The .java files, not the .class files) for your programs over Canvas by 11:55 PM on the day that it is due. You must send your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will not be accepted.

You must also submit a short report, at minimum ~1-2 pages in length, about your programs using the “Add Attachments” button. The purpose of the report is to highlight the manner in which you applied yourself in developing the assignment, and to allow you to evaluate your own work. It also serves as evidence that you did your own work. Base your report on the Programming Project Report Template provided to you, and focus on the areas being emphasized on the per-assignment point distribution as specified below.

# Restrictions

Your program must compile using Java 6.0 or later. You may develop your program using the IDE of your choice, although Eclipse is recommended. Your program should include a header comment with the following information: your name, date, course number, assignment title, and assignment description. You are advised to use the Java Source File Template provided to you.

Your report must be submitted in either .doc, .docx, or .pdf format. If you submit a Word Document, it must be compatible with Word 2010; if you are using Word 2013, save the file with the “ensure backward compatibility” box enabled.

# Grading Details

Your programs will be graded holistically. The following rubric is an approximation of the point distribution of your program.  
Note that the point distribution will vary from program to program.

|  |  |
| --- | --- |
| Program Implementation | 35% |
| Quality of Construction (Programming Style, White Space, Code Comments, etc.) | 20% |
| Specifications Conformance | 15% |
| Program Evaluation | 30% |
| Usability | 5% |
| Error Handling | 5% |
| Correctness | 20% |
| Project Report (~1-2 pages) Refer to the Project Report Template provided to you. | 25% |
| Program Testing | 2% |
| Solution Evaluation | 6% |
| Overall Quality (Formatting, Spelling & Grammar, Word Choice, Effectiveness of Technical Communication Delivery, etc.) | 2% |
| Respond to the following prompts:   * Why is SpaceObject abstract? How is this useful? * In this game, there are three comet classes corresponding to three comet sizes, but in the real world comets can be of any size. How could this program be modified to allow for comets of varying sizes? Is it possible to use a single comet class instead of four? You can still assume that a larger comet breaks up into average-sized comets, which break up into smaller-sized comets. * Draw a UML diagram outlining the relationships between all of the classes used in the game. | 15% (5% each) |
| IMPROVEMENT Credit Since the program is being graded holistically, you have the opportunity to make up for lost points. Apply yourself. Exceed expectations. Innovate something new. Incorporate what you’ve learned outside the classroom. Be creative. Be sure to meet program requirements and conform to specifications, though! | 10% |