Assignment 4
COMP 86

For this assignment, we will create a Model with data structures to support a collection of spaceships or other "vehicles" that make up the simulation and provide a canvas and paint callback routine that can draw the vehicles on the screen from the stored data structures. Your program will then simply draw the Model data onto the screen, including its Vehicles. You will also provide some interactive GUI widgets to control some aspects of the display.

Vehicle Class

First, create classes for the vehicles in the simulation. There should be one abstract base class for all vehicles, so you can later keep them in a single array or list and exploit polymorphism. Then, create different subclasses for different kinds of vehicles that might behave differently or have different drawing procedures. Each vehicle will be an instance of one of your classes, for example, if they were spaceships, you might have robots, landing vehicles, and satellites.

Each vehicle object should have instance variables that store the data needed for drawing the object and for executing commands on it (and, later, for making it move). For a start, it might contain current position, direction, speed, altitude or depth (if applicable), color, a text label, and perhaps parameters about its appearance to be used by your draw routine.

Vehicle should have a constructor that specifies its initial position, orientation, label, or whatever other information you need.

Vehicle will also have a draw method, which is your own subroutine, and which would draw just that vehicle in its correct location onto the canvas. It can use regular Java Graphics or Graphics2D drawing methods to draw the vehicle onto the canvas. You can draw each Vehicle any way you want, as a simple 2D drawing made with lines, shapes, or polygons. Draw its ID or name next to it in text. Your different subclasses of Vehicles should look different (by overriding the draw method).

Model Class

You should also provide an object that holds any overall data that goes beyond a single vehicle. Together, the Model and the Vehicles hold the back-end data for your system. Model will contain data that applies to the simulation as a whole, rather than to any individual vehicle. It will create and maintain the drawing canvas, provide the main redrawing routine for it, create the vehicles for the simulation, and maintain a list of them. That list will be used so that the Model's draw() method can call each Vehicle to draw itself. The list should be declared as holding instances of the abstract base class for Vehicle, so that you can plug any subclass of Vehicle into any slot on the list.

All drawing is done through the paintComponent() callback, which might be called by the system at any time. It will be called if the window is exposed or resized; and you will also need to trigger it manually when the state of the simulation changes due to a user command. Your paintComponent will likely call other routines (like the draw methods you have written in Model and Vehicle) to complete the job. Your paintComponent should always be prepared to draw the canvas from scratch, using data you have stored inside your objects. To redraw the canvas, first draw your background map, and then call your draw method for each of the Vehicles in the simulation.

Controls

Include some more GUI controls like those in Assignments 2 and 3, but now they will allow the user to input more commands that change the display (for example, the background color or map overlay or perhaps the colors of all the vehicles). As before, they do this by modifying some data that you maintain in Model or Vehicle and that your paintComponent routine uses when it draws. Then, after you modify such data, call repaint() on the canvas to cause your changes be displayed. Use some of the new GUI widgets you have learned about, and try to choose widgets that are appropriate to their commands (for example, pulldown lists for choosing from a small number of choices, scrollbars for entering continuous values).

Main Program

Your main program (the public static method in your top level class) should instantiate an object of that class. Your constructor for it will then set up your window and instantiate the objects you need, such as the Model object, which in turn instantiates the Vehicles.

Program Design and Practices

Your program design should exploit the features of object-oriented programming (encapsulation of code and data, support for abstract data types, polymorphism/overloading, inheritance).In particular, object-oriented programming provides us a good way to handle the various data needed in callback routines. You should use objects to encapsulate each interactive widget with the routines and data you need to use it or pointers to other objects containing such.

You should provide an object for each interactive widget or small group of widgets you create. That object should hold anything you need to remember about the widget from one callback to another, all the data pertinent only to the command for that widget or that you need to operate this control, (including, in most cases, a pointer to the model or other outside object needed to perform the actual action the user requested), and the widget's own listener callback routines.

If you have several widgets that share some behavior or properties, you should organize your objects into an appropriate inheritance hierarchy.

You will have other data that must be accessed by several widgets, particularly shared information about the state of the program or global information about the state of the user interface. Provide additional classes and objects for holding this kind of information.

Remember to trigger your drawing to repaint itself explicitly whenever one of your commands causes a change that should be reflected on the screen. And remember that the way to change the screen is first to change the data stored your classes and then to trigger the repaint.

You should follow these general Java programming practices:

- Make all instance variables of your classes **protected** or **private**. If you need access outside of the class, provide it with *set* and/or *get* methods; don't access protected the variables from outside the class (even though Java -- unfortunately -- allows us to).
- Avoid most global variables or widely-accessible public variables; pass the data you need explicitly.
- Put each class in a separate .java file.

Design Documentation

In addition to your program, submit documentation about the design of your system in these forms:

- An outline showing the inheritance hierarchy
- An outline showing the aggregation hierarchy (which objects contain or "own" which other objects)

Submit this documentation electronically in text form, along with brief instructions for how to compile and run your program. Include it as part of the `readme` file that you submit with your assignment.