# **CS 1331**: Introduction to Object Oriented Programming

## Homework 8

**Due:** March 28, 2014 @ 8:00 PM

# Vehicles: An intro to OO design!

This homework is different from other homeworks you have done in this class. The emphasis of this homework is not on the final product but rather your object oriented design. You will be making a very simple GUI that displays a vehicle. This vehicle can be changed by clicking one of four buttons which will create either a truck, ferrari, sail boat, or yacht. The user can then use the W, A, S, and D keys to move the vehicles around.

Read the prompt below. Then reread it carefully and think about the relationships between the different vehicles, what properties and functions they have in common or do differently. We **highly** recommend that you draw out some sort of class hierarchy diagram and really plan your classes out before you even start typing code.

When asking TAs questions about how pieces connect and interact with each other, we will be expecting you to show us your diagram of how *you* think they will interact. By this point in the class, you should be able to come up with some idea of how multiple objects can work together to solve a problem.

Again, the main point of this homework is the class design. Having a functioning GUI is not guaranteed to get you a good grade. If you find yourself copying and pasting code, repeating lots of things, etc you probably need to rethink your design.

# Prompt

In our imaginary world, we have **Vehicles**. Every vehicle has an x,y position on our GUI and some speed with which it moves. Every vehicle also has some sort of way it can be drawn and some way it can move in response to a character (WASD) being pressed.

Two types of vehicles exist, **Cars** and **Boats**. All cars start at position 150,150 on our GUI. All cars are represented by an Image that is drawn to the GUI at the car's x,y location. All cars move left when 'a' is clicked, right when 'd' is clicked, up when 'w' is clicked, and down when 's' is clicked. The amount by which they move is denoted by their speed.

Two types of cars exist, **Trucks** and **Ferraris**. A truck has a speed of 1 and is represented by 'truck.jpg'. A truck moves like any other car except for one extra issue. The truck is old and pretty run down. Anytime it moves, there is a 1% chance it will break down. After the truck is broken down it can no longer be moved at all. A Ferrari has a speed of 3 and is represented by 'ferrari.jpg'. The Ferrari moves like any other car expect for one extra feature. Anytime a Ferrari moves, there is a 5% change it will increase its speed by 1 (it doesn't matter if it moves then increases its speed or increases its speed then moves).

All boats start at position 150,300 on the GUI. Every boat is represented by a black rectangle of 50 width and 30 length. When 'a' is clicked, the boat moves left by speed and when 'd' is clicked, the boat moves right by speed. The user cannot manually move the boat in the vertical direction. When 'w' or 's' is clicked, the boat does not move at all. Boats have sails which when they pick

up wind cause the boat to move in the vertical direction. Everytime a boat moves there is a 10% chance it will move in the vertical direction by 1. There is a 50% chance this vertical direction will be up and a 50% chance it will be down.

Two types of boats exist, **SailBoats** and **Yachts**. A yacht has speed 2. A yacht has some sort of flair on its black rectangle to denote it is a yacht (ex, a white Y). A yacht has fancy sails. It moves like any other boat but has an additional feature. Anytime it moves, there is a 15% chance it will 'skip' an extra 10 units in what ever direction it was moving.

A sailboat has a speed of 1. It has some sort of flair on its black rectangle to denote it is a sailboat(ex, a red triangle).

## `Vehicle` Abstract Class

We are giving you your first class, Vehicle. This is an abstract class. You will learn more about these later but feel free to do some research on them now. An abstract class cannot itself be instantiated. However, it can still have constructors that its subclasses can call. An abstract class can also have abstract methods. These are the same type of deal you have with interfaces. On HW6, preview was an abstract method in Previewable. These abstract methods (like with interfaces) must be implemented in Vehicle's subclasses.

Vehicle has everything all vehicles have. Recall from the prompt that this is an x,y position and a speed. Vehicle has a constructor that takes in and sets these values.

Vehicle has two abstract methods related to the two functions all Vehicles can do. Recall from the prompt that all vehicles have some sort of way they can be drawn and some sort of way they can move in response to WASD. Hence, we have two abstract methods: draw and move. Whatever extends Vehicle will have to decide how to actually implement these two methods.

# STOP

Do not read ahead of this section yet. We are NOT going to tell you which classes to write. We are also NOT going to tell you what should extend what. Go back through the previous sections and write down all of the classes that you will need to write code for. Seriously. Go do it.

After you've done that, write down what methods you think you will need to write in those classes.

Now try and figure out if any of those methods can be inherited from other classes, or if they will need help from other objects to get their job done. Draw some lines or arrows to make it clear to yourself exactly how it should be laid out.

Before you even *think* about the GUI portion of this assignment, you should code up all of these classes for vehicles and test them to the best of your ability without the GUI. We won't be grading you on this stuff, but *we will be grading you on the strength of your design* so put some serious time and effort into it before diving into the code.

You shouldn't be writing the *exact* same code in two different classes. You should be able to add more vehicle types easily if necessary. If something changes in one of your classes, it shouldn't effect others that are unrelated. Objects should be working together to get the job done. These are all signs of a good Object-Oriented Design.

Ready to keep reading? Good :)

## `ImageReader` Utility Class

Much like we gave you FileScanner in the HW6, for this homework we are giving you a class called ImageReader. This class has one static method that takes in a string name of a file and returns an Image from that file. If no file is found it returns null. Use this to get your Images from the jpg files.

# The GUI

You GUI should consistent of the following:

- A panel of size 500x500 that you create with four buttons
- A frame that holds and displays that panel.

You should not need multiple panels. Your panel will need a `KeyListener` to listen for keyboard clicks. Initially, there will be no vehicle displayed. After a button is clicked, that vehicle will be created and displayed. There will only be one vehicle displayed on your panel at a time. When a button is clicked, that vehicle is made and replaces the vehicle currently on the panel.

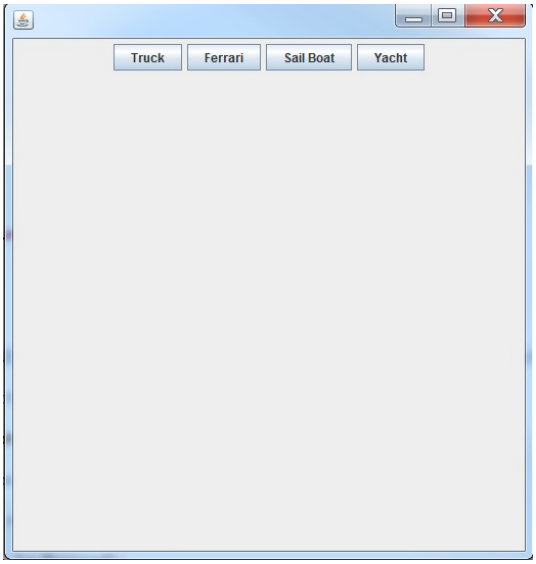
You should not pass any sort of buttons or labels or panels into your vehicle classes. The only GUI aspect in your vehicle classes will be the `Graphics g` object that is passed into `draw`.

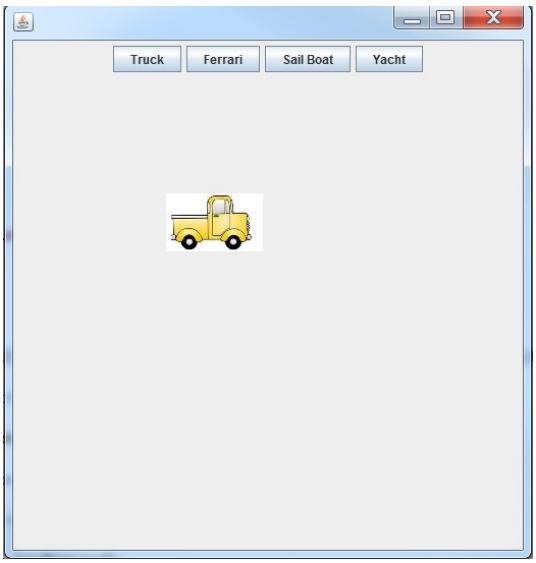
Focusability can be an issue with keyboard listeners. Think about your web browser. If I click on the url box and start typing, text will appear in the url. If I click on a Facebook message and start typing, it will type there. If I click on some whitespace on the page and then type, perhaps my typing won't do anything. This is because of focus. Initially, you will be focused on your panel but when you click on a button your focus shifts to that button. You should use `setFocusable` to manually enable and disable what can and cannot be focused. Read the Java API to learn what this method does and what to pass into it.

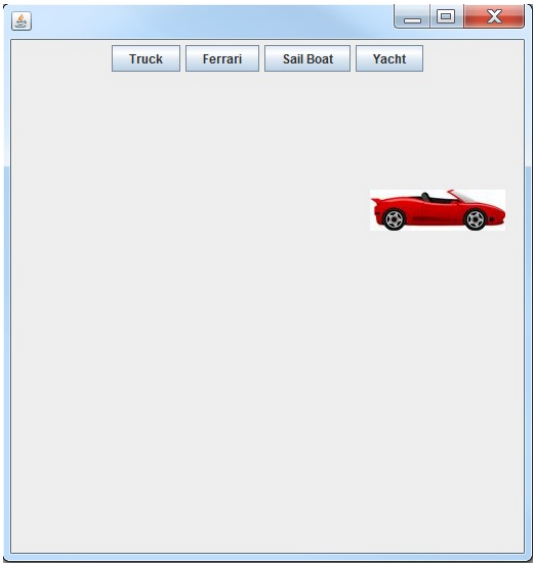
**General Tips**

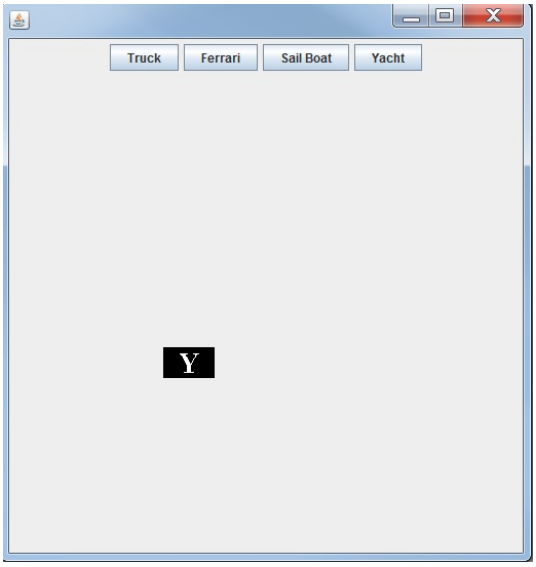
- This is a long read but it is very important that you read the prompt carefully and digest how our world works and how different entities relate to one another.
- Draw and plan out your classes before you start coding.
- When asking TAs for help on this assignment, you should reference your outlines and drawings of how you want to lay out your classes and how you expect them to interact with each other
- Copy-pasted, redundant code will not earn you a good grade on this assignment. Use inheritance, overloading/overriding, and method chaining as needed to make your code clean and object-oriented.
- If needed, refresh yourself on how we draw directly to panels in Swing (ie paintComponent).
- Work on your homework incrementally, checking that bits and pieces work in succession. Slowly add more and more features until you have everything. Do not write a dozen class files and then try to debug your way backwards.
- Write with good style as you go. By now you are well aware of common Checkstyle errors you run into. There is no reason to only fix this at the very end.

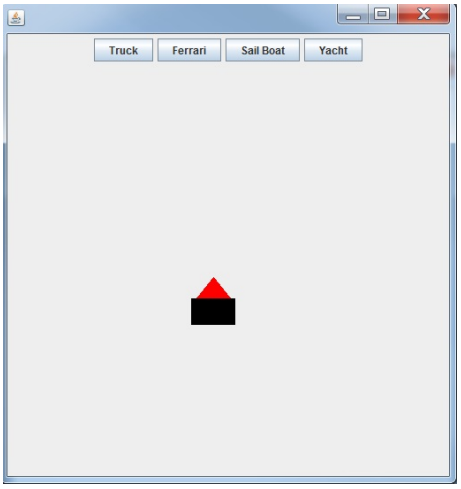
Some examples of what your GUI could look like. Feel free to go above and beyond decorating the landscape and boats :D



# Javadocs

We are going to have you do Javadocs again for this assignment (and for all assignments here on out). The samples from the last HW writeup are below for your reference.

```
import java.util.Scanner;

/**
* This class represents a Dog object.
* @author Ethan Shernan
* @version 1.0
*/

public class Dog {
....
}

/**
* This method takes in two ints and returns their sum
* @param a, b
* @return their sum
*/

public int add(int a, int b)){
...
}
```

# Checkstyle

Just in case you forget how to do Checkstyle, here are the instructions again!

You have been provided with a Checkstyle Jar file and the CS 1331 Checkstyle configuration file (`cs1331-checkstyle.xml`). We do this to make sure you are following the proper style guidelines as posted in the Style Guide under T-Square resources. Please refer to these guidelines for any help as you are writing your programs.

We will run checkstyle on all the Java source files you submit. You can run Checkstyle with the command below (in the directory containing all your Java source files):

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by using the command below and subtracting 2 from the number printed (which is how we will deduct points). For example:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | wc -l
2
```

Also means no errors.

If you have any questions about a Checkstyle error, please let a TA know! They should be pretty self-explanatory and you should be able to fix them easily. We will be taking points off for checkstyle on this assignment, so please make sure you run it!

# Turn-in Procedure

Submit all of the Java source files you created to T-Square. Do not submit any compiled bytecode (`.class` files). When you're ready, double-check that you have submitted and not just saved a draft.

**Please remember to run your code with Checkstyle!**

**Verify the Success of Your Submission to T-Square**
Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is 8PM Friday. Do not wait until the last minute!