CPSC 233: Assignment 5 (*Due April 16 at 4 PM)*

New learning concepts: Inheritance, polymorphism/method overriding, casting within an Object-Oriented Hierarchy.

Problem Statement:

Create a racing simulation with two driving tracks: artic and desert. For this version of the program the artic track contains an SUV (Sport Utility Vehicle) and the desert track contains a sports car. Each car will try to reach the end of its respective track prior to the other car without running out of fuel (see Figure 1). If either or both cars reach the end of the track then the simulation ends: draw (if both reach it during the same turn), win (for the car that reached it first), and loss (for the car that didn't reach it first). If one car runs out of fuel then the simulation continues until: both cars run out fuel (tie) or the other car reaches the end. The simulation is also a 'draw' if the user quits the program early.

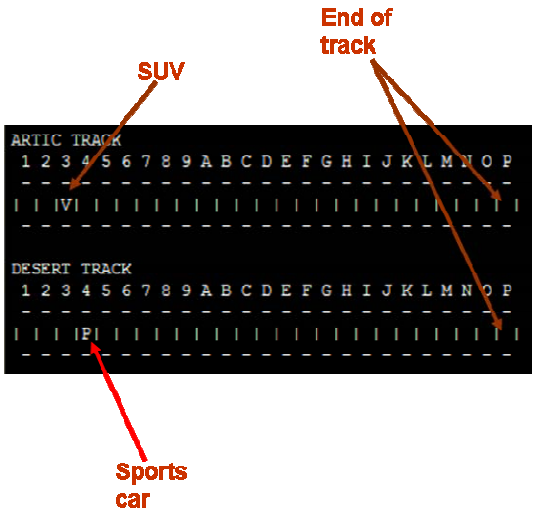

Figure 1: Race tracks with their respective vehicles

There is a random chance of a weather event occurring in each track. The artic track can have a blizzard blow in that prevents non-AWD (All Wheel Drive) movement (but doesn't stop the expenditure of fuel) while the desert track can be hit with a heat wave that can cause sports cars to overheat and double the fuel consumption rate (movement distance is unchanged). Finally the simulation allows for a 'cheat' option (for testing purposes): debugging messages can be toggled on/off, the fuel level of either car can be manually set, a car can be manually moved to any location within its current track, invoking a weather event in the other track (meant to hinder the other car so the cheat option shouldn't work on one's own track – look at the section "Ordering

of events during a turn" and you should see how trying to sabotage your own track will be canceled out by the random occurrence of weather events).

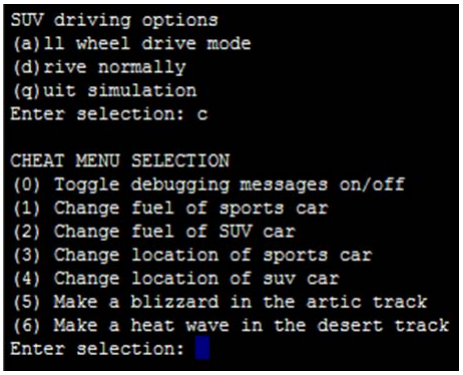

Figure 2: Cheat options available in the simulation

Overall program description:

Unlike the previous assignments this program will be interactive (allow for user input) and data will not be read from file. However, to make it easier to test (and mark) your program, there will be cases where the appropriate cheat feature MUST be implemented before you get functionality marks. (Refer to the section titled "Program functionality" for more detail.)

Pre-created classes to be used in your program (cannot be changed)	**Classes that you must create and implement yourself**
Car	ArticTrack (child of Track)
Debug	DesertTrack (child of Track)
Driver (starting execution point)	GameController
Track	Sports (child of Car)
	SUV (child of Car)

Ordering of events during a turn:

1. The two tracks are displayed
2. SUV movement occurs (display menu, get input, move car if applicable, selecting the quit option will exit directly out of the game with an appropriate status message).
3. The program determines if a blizzard occurs in the artic track (affects next turn).
4. Sports car movement occurs (display menu, get input, move car if applicable, selecting the quit option will exit directly out of the game with an appropriate status message).

5. The program determines if a heat wave occurs in the desert track (affects next turn).
6. Determine the outcome of the simulation (if a win/loss/tie occurred)

Note: invoking a 'cheat' option will use the respective user's turn e.g., the SUV user who uses the cheat to sabotage the sports car user will not have an option to move his/her car that turn.

Brief description of the pre-created classes

Car:

Each car is represented by a single character. The basic car is shown as a 'c' (the child classes have an appearance of 'V' for the SUV and 'P' for the Sports car). Individual cars have a fixed amount of fuel (40) that is consumed at a fixed consumption rate (2), allowing it to move a standard distance (2). If the fuel has been totally expended (zero) then the car can no longer move. (The movement menu for that car should no longer be displayed). All cars (including child classes) should display its current state during a turn: current fuel, consumption rate and distance traveled for that turn (see Figure 3).

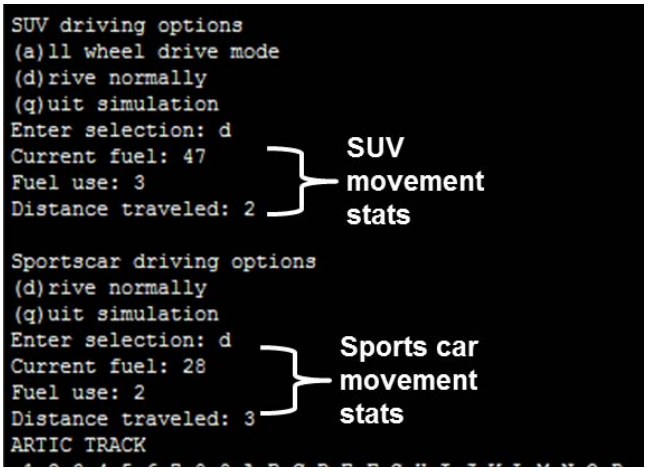

Figure 3: Menu options, fuel consumption and distance traveled.

Debug:

The sole purpose of this class is to determine if the program is operating in debugging mode (it is the sole exception on the prohibition on using static variables). Debugging messages should inform the user about the state of the program. The exact content of the debugging messages is left to your discretion. If the program is debug mode (the flag is set to 'true') the messages will appear, otherwise they will not. Example:

```
main (String args[]) {
    if (Debug.on == true)
        System.out.println("<<< Driver.main() >>> ");
}
```

Driver:

The starting execution point of the program: it gets the game controller to start the simulation.

GameController:

This class will act as the user interface for the program (responsible for input and output: displaying the appropriate menu, getting user input, validating input etc.). To a large extent it will then be directly or indirectly responsible for carrying out those actions. (Directly when the required code is included in a method of this class; indirectly when it calls the method of another class). Be careful that you don't break encapsulation by putting code in this class that should belong to another class! (The same applies to the other classes but the problem is most likely to occur with this class). Most likely this class will be the largest one for this program. There will be numerous variable and constant attributes for this class. At a minimum it should contain two fields: one for the artic track and one for the desert track. There were two examples of user-interface classes providing during the term that you can refer to for this assignment: one was a dice simulation, the other was for a book manager program (employing an array of books). You can find them in the assignment directory in UNIX (see the "supporting resources" section) and linked in from James Tam's course web page.

Track:

The parent race track class. All tracks will contain a fixed size array of car objects. The array will be empty (null) except for the one element that refers to a car. During the simulation the car will move along the array to simulate the movement of a car along a track. Responsibilities of the car include: displaying the state of the track, putting a car at a particular location on the track (can be used in cheat mode or by the child classes to place the sports car and SUV at the start of their respective tracks); determining if a car has reached the end of the track.

Brief description of the classes you must create and implement

ArticTrack:

A child of class Track that has the additional ability: generate a blizzard. A car traveling during a blizzard will be stuck (not move forward and just "spin its wheels") but still expend fuel appropriate to the type of car and driving mode (if applicable). The exception is a SUV traveling in AWD mode (see Figure 4). Each turn there's a 10% chance of a blizzard occurring, otherwise the driving conditions will be normal.

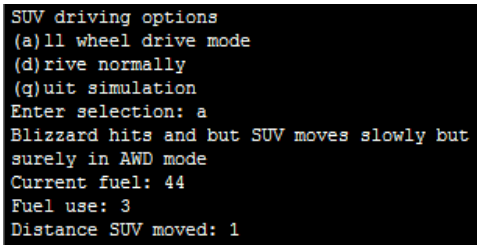

Figure 4A: AWD mode still consumes fuel but allows for some movement during a blizzard.

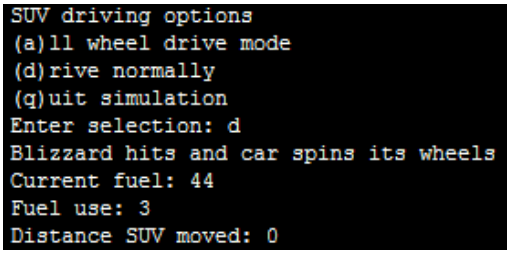

Figure 4B: Normal driving is not successful when a blizzard hits the arctic track.

DesertTrack:

A child of class Track that has the additional ability: generate a heat wave. A sports car traveling during a heat wave will overheat (see the description of the sports car class for the effect of overheating). Each turn there's a 10% chance of a heat wave occurring, otherwise temperature will be normal.

Sports: ("light and fast")

A sports car is a child of class **`Car`** that has a different appearance ('`P`') with a standard fuel consumption rate of 2 units of fuel but the ability to move 3 distance units. To keep the car lighter (and faster) the fuel tank capacity is only 30 units. Normally a sports car has an efficient cooling system but if there is a heat wave, a sports car will overheat and consume fuel at double the normal rate (move distance is unchanged) (refer to Figure 5). The car will remain overheated only so long as the heat wave lasts e.g., if there is a heat wave during the current turn but not the next then the car will only remain overheated for only one turn.

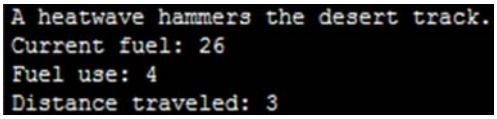

Figure 5: A sports car overheating during a desert heat wave.

SUV: ("big and heavy but with good off road handling")

A child of class **`Car`** that has a different appearance ('`V`') and different movement options: under normal driving mode the distance traveled is still the default rate of 2 but fuel consumption is 3 units, under AWD mode the distance traveled is 1 unit with a fuel consumption of 3 units. The advantage of AWD mode is the car still moves 1 unit (and consumes 3 units of fuel) during a blizzard whereas under normal mode the SUV would consume 3 units of fuel but won't move (refer back to Figure 4). The fuel tank is larger with a capacity of 50 units.

Program functionality:

- Displays an introduction to the program that describes the rules. It only displays these instructions right after the program is run. The exact content of the instructions are left to your discretion but it should be sufficient to instruct the user how to run the program.
- Displays an appropriate status message when the program ends e.g., "SUV reached the end first", "quitting before the simulation ended" etc.
- Displays the appropriate menu for the SUV and sports car movement.
- Gets user input for the SUV and sports car menu and repeats prompts until valid input is entered.
- Program runs until the user quits.
- Cheat menu can be invoked from either car menu.
- Gets user input for the cheat menu and repeats prompts until valid input entered.
- Each track is properly initialized and displayed at the appropriate time.
- SUV can move and consume fuel as specified in the class description (non-AWD mode).
- Sports car can move and consume fuel as specified in the class description.
- Program checks and properly handles when cars run out of fuel: *Getting credit for this feature requires that the cheat menu allows the user to manually set the fuel levels of both car* (see Figure 6).
- SUV AWD mode working (fuel consumption and movement).
- Artic track can randomly generate and properly handle the effects of a blizzard on the SUV (regular mode): *Getting credit for this feature requires that the cheat menu allows the user to invoke a blizzard in the artic track.*
- AWD mode working properly during a blizzard (requires the other two features (AWD driving and blizzard) to be implemented first.
- Desert track can randomly generate and properly handle the effects of a heat wave on the sports car: *Getting credit for this feature requires that the cheat menu allows the user to invoke a heat wave on the desert track.*
- Program can determine when one or both cars have reached the end: *Getting credit for this feature requires that the cheat menu allows the user to manually move either car to any location in the respective track.*

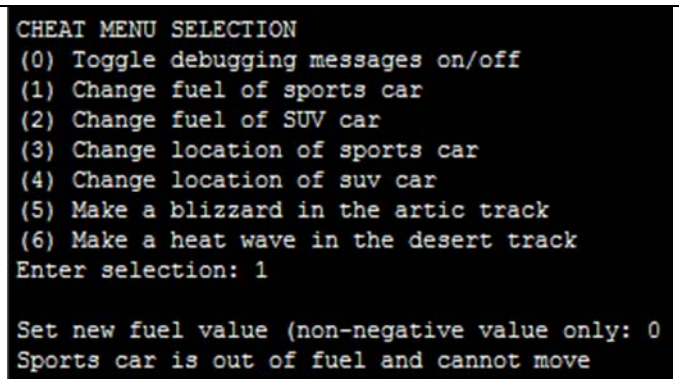

Figure 6: Cheat option used to make the opposing car run out of fuel (it can also be inflicted on your own car as well).

Non-functional requirement:

Draw a UML class diagram for the following classes: Track, Artic Track, DesertTrack and SUV. Make sure that all the attributes and method (plus permission levels and parameter information) are represented along with all class relationships for these 4 classes.

Supporting resources

To help see how your program should execute you can find sample output text files in UNIX under the directory: /home/233/assignments/assignment5 (available via a web link from James Tam's course web page).

External libraries/methods that can be used (you don't write yourself)

You can use pre-created code for console input and output (e.g., 'print', class 'Scanner'). Wrapper static classes can also be used to convert types (e.g., class Integer's parseInt() method) as needed. Also the random number generation capability of class Math or class Random can be used.

Important reminders

- **Assignment Guidelines:** When working on your assignments, please adhere strictly to the generic assignment submission guidelines (which apply to all assignments) as well as the ones listed in this assignment.
- **Handing in your assignment**: Use the UNIX 'submit' program. Make sure you submit the source code files (*.java) for both classes.
- **Collaboration**: Assignments must reflect individual work, group work is not allowed in this class nor can you copy the work of others.