Cars w/ Object Oriented Programming

Description

Back to the garage for our third and final Car assignment. This program will be very similar to the previous iterations of the Car assignment, however, we'll now be instantiating Car objects. The Car class should have an ignition, a color, and a position (given by an x coordinate and a y coordinate) as well as various methods to generate and change these instance variables.

The ignition state and the position state may change during the course of the program's execution, but the color will stay the same. Before the user is given control of the car via the menu, the car's attributes should already be assigned (i.e., it will have its ignition set to "off", it will be given a color, and it will have random position coordinates). The car's current location will be represented by a **char** that stands for its color (e.g., "R" for red). Available colors are: Red, Green, Blue, White, Silver (R, G, B, W, S). The ignition can be set to either on (True) or off (False).

The user should be prompted to choose a car and then, once a car is selected, the user can (i) turn the ignition on/off; (ii) move the car around the 20x20 grid; (iii) change cars; or (iv) quit the program.

Specific error checking will be expected. The car should be prevented from going out of bounds and, if the ignition is off, the car should not be able to move anywhere.

Lastly, the most current grid and the status of the car should be printed after each user action (i.e., turning the ignition on/off, legal movement of the car, and when the user chooses to quit).

Example Output

```
Enter a number between 1-10 to select a car:
Input: 3
```

Print initial car information; prompt user for an action:

```
Car #3 Information
Color: Red
Ignition: Off
Location: (5, 7)
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - R - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -

What would you like to do?
1: Turn the ignition on/off
2: Change the position of the car
3: Change cars
Q: Quit program

Input: 1
```

Print state of car after every action:

```
Car #3
Information
Color: Red
Ignition: On
Location: (5, 7)
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - R - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - -
```

Design and Implementation

This will be very similar to the previous Car assignments. You should initially create an array of 10 Car instances. Each car should be instantiated with (i) a random color; (ii) a random x/y position on the grid; and (iii) the ignition set to off.

You'll need to implement several methods to set these initial values and ultimately manipulate them. You should create instance methods for all of the following, but you're not limited to just these. You may create additional helper methods as you deem necessary (Remember best practices discussed in lecture with respect to getter/setter methods).

assignColor: Randomly picks one of the five colors (represented by a char) and returns the corresponding char. This method should be called once from the Car's class constructor. The color will not change during the execution of the program.

moveHorizontal: This method will need the number of spaces you would like to move the car (a negative value will move the car left; positive value will move the car right) and will adjust the xPosition of the car.
If the car's ignition is not on, this method tells the user that s/he must turn the ignition on first and leaves the xPosition unchanged.

If the user tries to move the car beyond the border of the 20x20 grid, this method reports an informative error message and leaves xPosition unchanged.

moveVertical: This method will need the number of spaces you would like to move the car (a negative value will move the car up; positive value will move the car down) and will adjust the yPosition of the car.
If the car's ignition is not on, this method tells the user that s/he must turn the ignition on first and leaves the yPosition unchanged.

If the user tries to move the car beyond the border of the 20x20 grid, this method reports an informative error message and leaves yPosition unchanged.

ignitionSwitch: This method will turn the car off when it is on or on when it is off. (i.e. change the value of the instance variable).

getColor: This method will return the color of the car as string.

getIgnition: This method will return the ignition status of the car as a boolean variable

getX: This method will return the x position of the car as an int.

getY: This method will return the y position of the car as an int.

toString: After each action, you should print the "state" of the currently modified Car. This method will call the "get" methods and report the status of each (including a copy of the Car's position in the grid). See "Car Information" above in the Example Output.

Method signatures are not provided, but take note of the descriptions to determine what is required. If a method is said to ***return*** a value, how will that affect your method signature? How many parameters are required, if any, to properly implement?

As demonstrated above in the example output, main() should have clear and simple instructions when requesting user input and you should always be checking for invalid input at each occurrence. Be mindful of which error checking you're doing and where as some of it needs to occur within certain methods. Read the method descriptions carefully.

Formatting and Style

Your Car Information and accompanying grid should print neatly to the screen as demonstrated in the Example Output above. User prompts as well as error messages should be short and concise. You're not beholden to the phrasing or structure of the menus referenced above, but yours should be similarly clear and easy to use. Be mindful of your formatting when using nested loops and complicated control flow statements. Comment as necessary to clarify certain things as you see fit, but remember that properly formatted and well designed code often speaks for itself.

Grading Rubric

Points	Criteria
3	Car instances properly instantiated with default attributes via appropriate methods
3	All instance methods properly implemented; error handling in place for illegal operations
2	Car attributes are properly maintained; status and grid print correctly to the screen
1	Catches invalid input at each menu
1	Good style demonstrated in the code; sensible formatting