Assign-03

Upgrading Your Car Radio

Retrieve and use the ***CarRadio-StartingPoint.zip*** file from eConestoga as your starting point. The CarRadio example provided above is an okay … but is significantly flawed in a lot of ways. Your task in this assignment is to take this "okay" class definition and make it better and more general by improving it in the following ways:

Generalizations:

- Change the names of the two class-related files to `AmFmRadio.h` and `AmFmRadio.cpp`.
- Change the name of the class to *`AmFmRadio`* (this class no longer models a radio within a car)
- Rename the *`struct Button`* to *`struct Freqs`*, since we're going to use the struct for more than buttons.
  - Change *`AMButton`* to *`AMFreq`* and *`FMButton`* to *`FMFreq`.*
  - You can keep the same default values for the AM/FM frequencies as found in CarRadio

Additions & Replacements:

- Replace the existing constructor with two constructors that initialize all private data and that use the following parameters to initialize the appropriate data:
  - one that takes a single bool parameter that indicates whether the radio should be on or not when instantiated. Give this parameter a default of *false* (as discussed in Module 1 (default parameters)).
  - one that takes a single bool parameter that indicates whether the radio should be on or not when instantiated and an array of 5 *struct Freqs* that contains the initial radio preset values (i.e. the station's frequency).
- Add a destructor that simply displays "Destroying AmFmRadio".
- Make sure that the instance of *AmFmRadio* created in the `driver.cpp` mainline is instantiated powered on.
- Add a method called *`ScanDown()`* that behaves similarly to *`ScanUp()`*, except that the scanning is down.
  - When the AM and FM band reach the minimum frequency, they roll-over to the maximum frequency in the band
  - Make menu entry #8 correspond to "Scan Down". Make menu entry #9 correspond to "Quit the program".
    - Do not otherwise change the user interface.
      - **I'm serious**.
- Create a mutator for the *`current_station`* data member.
  - Actually while you're at it, please make sure that all variables / elements holding an FM Frequency elsewhere in the code match the data type of current_station … I don't think that they did in the starting code
- When switching from AM to FM (and vice versa), go immediately to the previous frequency tuned on that band.
  - e.g. if you're listening to 700 on AM and switch to FM and then back to AM, make sure that you go to 700, not 530.
  - Hint: it would be a good idea to use a new data member of type *struct Freqs.*
- When turning the radio on after being off, go immediately to the previous band (AM or FM) and last frequency tuned on that band.
  - e.g. if you're listening to 92.9 on FM and turn the radio off, turning the radio on again should tune to 92.9 on FM.
  - Hint: the previous hint works here too.
- When turning the radio off, turn the volume down to 0. When turning the radio back on, restore the previous volume level.
  - Turn the volume level to 0 when the radio is turned on for the first time.
- Create a second version of *`SetVolume()`*. The second one takes a volume as a parameter. The reason for this addition is so that someone using *`SetVolume()`* can get the value in their own way and pass it to the method.
  - You will not be using this second version in this assignment. You will be using it in a later assignment.

  - There is already a method called *SetVolume()* and you are adding another – what does this mean? What are the best practices in coding something like this?
- Change *ScanUp()* so that it can be called without any output being displayed. If you have any other methods (except *ShowCurrentSettings()*) that display output, make it possible for them to be called without any output being displayed.
  - Note that this does **not** mean that you should totally eliminate output being displayed but should make output display optional.
  - This does **not** apply to *ShowCurrentSettings()*, as its entire purpose is to display output. This also doesn't apply to the prompt in the existing *SetVolume()*
  - Suggestion: Create a private bool data member (e.g. "displayOutput") that keeps track of whether or not output should be displayed (except *ShowCurrentSettings()* and prompting *SetVolume()*).
  - Reasoning: You will be inheriting from this class in a later assignment and you don't want to display output from the parent class when you'll be doing it from the child class instead. The child class will contain the user interface to be used for that assignment.
  - Please ensure that by default this private bool data member is set so that no output is coming from any methods (except for *ShowCurrentSettings()*and prompting *SetVolume()*)
  - Also create a mutator for this "displayOutput" data member
- Make sure to create **accessors** for the private data members representing the current station, the current volume, the set of radio presets, the current band (AM or FM), whether the radio if on or off as well as the "displayOutput".

Style and Convention Requirements

- Make any new data members that you create private.
- As discussed in class, all of the class' source code belongs in the `AmFmRadio.cpp` – there should be no code in the `AmFmRadio.h` (class definition)
- Your code must not unnecessarily duplicate similar code segments. Make use of your methods (e.g. the mutator mentioned above) appropriately.
- Using the what was called the **classic commenting** style in the lessons,
  - Add full file header / class header comments in each of your three source files (yes, even the .h file). For the descriptions of each, describe what is done (generally) by the contents of the file being commented.
  - Add full method header comments on each method you develop and don't forget your inline comments as well
  - Make sure that all comments that already exist in the example are correct. Yes – even when you take over someone else's code – you become responsible for their comments!

Other Stuff

- If you want to create additional methods or data members, you can.
- Watch the datatypes being used – ensure that all code within the class is consistent with the data member datatypes.
- Please realize that not all of the changes that you are making will be used in this assignment. We are trying to design the class to be more useful in general. We will be using many of the changes in subsequent assignments.
- Also realize that some of the requirements above have already been implemented.

ZIP up the source files : `AmFmRadio.h, AmFmRadio.cpp`, and `driver.cpp` and submit them to the eConestoga dropbox by the deadline.