Individual Project – Code Documentation

Watch Store

In my program there are 10 classes, each divided into a header (hpp) and an implementation (cpp) file. These classes are: “WristWatch,” “QuartzWatch,” “MechanicalWatch,” “AnalogWatch,” “DigitalWatch,” “HandWoundWatch,” “AutomaticWatch,” “Admin,” “User,” “InputValidation.”

**Abstract Classes:** “WristWatch,” “QuartzWatch,” “MechanicalWatch.”

**Instantiated Classes:** “AnalogWatch,” “DigitalWatch,” “HandWoundWatch,” “AutomaticWatch.”

**Utility Classes:** “Admin,” “User,” “InputValidation.”

**The Utility Classes** are used only for their methods. “Admin” and “User” have static methods that are used without objects calling them. “InputValidation” is a namespace that has three methods for validation of the user input (bool, int, double).

Class “Admin”: this class has three methods. One for logging into the admin account. One for adding watches to the JSON files and a method for removing watches from the files.

Class “User”: this class has four methods. A method for creating profiles that stores data into a JSON file only collecting user data, and one method for logging into the profile. The third method “buyWatch” is used for adding a watch to the user’s collection. This collection is stored in the user’s profile. The last method “displayCollection” is used to iterate through the users watch collection, and then display it on the screen. For all these methods to work, the username and password of the user are passed to them, so that the particular user can be found in the file containing all other profiles.

Namespace “InputValidation” contains three methods for validation of integers, Booleans, and double numbers. Every method checks if the user has entered the correct data type when asked for user input. If there is an invalid input the methods will catch it and an error message will be displayed. Once the user enters the correct data type the methods will return and the program will continue. This input validation is done, so that crashes are avoided. These three methods are used in various placed where user input is expected.

**Abstract Classes:** “WristWatch” is the base class in the hierarchy and “QuartzWatch” and “MechanicalWatch” are its children. They all contain the basis data members, constructors, and methods like getters and setters. The abstract methods in the base class “WristWatch” are printInfo and customizeWatch. The first method is just used for printing watch’s information to the user. In the second method the user can customize a chosen watch from the store. Every watch has a number of customizations as a data type, so that every one of them can be customized in a different way and different number of times. These methods do not have functionality in these three classes. They get implemented later in the derived classes on the bottom of the hierarchy.

**Instantiated Classes:** “AnalogWatch,” “DigitalWatch,” “HandWoundWatch,” and “AutomaticWatch” are the classes at the bottom of the class hierarchy. They contain the functionality of the two abstract methods, as well as their own methods. All the classes have data members, constructors, and methods like getters and setters. The printInfo method now has its printing structure declared, and in each class it differs slightly. The customizeWatch method contains the needed functionality like: loops and if-else statements. The user first enters a while loop and then depending on how many customize options their chosen watch has, this many times will the loop execute. There is also an option to exit the loop using a sentinel value. When the loop stops the users sees the information of the original watch and the newly customized one. Then one of them can be bought. These four classes also have two more static methods that are used for managing the data in the collections. The first method is createWatches, which is used for initializing objects from the JSON files. The method iterates through the files using a loop and in every iteration all the data from each element in the JSON is assigned to an object’s setter methods. Then when the object is ready it is passed to a vector which is later returned to the main method where the real vector is assigned. Because the method is using pointers to the objects and in every iteration the ‘new’ keyword is used for every new object, the program also needs a method for deleting this newly allocated memory. This is what the second method deleteWatches does. It iterates through the vectors and deletes the pointers. The method is again static and is called at the end of the program, after the user exits the program.

**The main file:** The main file contains several functions. The first one is printWatches, which is used in the UI, so that the user can see the brandName, modelName, and price of the watch. It iterates through the vectors and prints this info. The next two methods are insertionSort and quicksort, which are used for sorting the elements in the vectors by the number of custom options. The sorted vectors are only the vectors of the lowest derived classes, because their elements are all of one type and the sorting is consistent. If the other three bigger vectors are sorted, then their elements are sorted inconsistently because they are all different types. The fourth method is assignWatches, which is used for calling the static methods for creating objects from the JSON files. Then there are several for loops which iterate through the vectors of the derived classes and fill the other bigger vectors. The last function is the menu, which is the general UI that the user interacts with. The interface is in one big sentinel controlled while loop which contains a lot of if-else statements. The user can log in as admin or a user, buy different categories of watches, see their collection, and more. Everything is controlled by entering characters that are shown on the screen. The meu function contains all the calls for every other method that is needed. The main function only calls this menu function once.

**Additional Notes:**

This program uses Rapid\_JSON library that helps with the reading and writing data in the JSON files. The library is stored locally in the project file. The JSON files used in the program are also stored locally in their own folder “JSON\_Data.”

For the clean look of the user interface, I use a C++ library that cleans the screen after each user input, so that the information does not clutter over one another. The library used is “<cstdlib>” and the command for clearing the window is system(“cls”). This command is ONLY usable in Microsoft Windows, so Linux operating systems will not recognize it. I tested it on a virtual machine running Ubuntu. The code compiles and everything runs as intended. The only problem is that after each call of this command additional error message will pop up together with the rest of the information on the screen.

**Known Bugs:**

I have tried to minimize the bugs and crashes related to user input. The input validation methods and the other error messages included are informative enough, so that no accidental crash can happen.

One Rapid\_JSON related problem that I found is that, if in one file there are more than 10 elements, the code will compile but nothing will show up on the screen. I could not find any information about the specific reasons why this happens. The only reasonable thing that I read is that some configurations of the different IDEs can sometimes cause this issue. My solution to this problem is to create 4 different files for the different watch categories, with each file containing at maximum of 9 elements, so that when the user adds a watch to some of the files, the program will execute with no problems.

Sometimes when the program is run through the integrated terminal of an IDE, the system(“cls”) does not completely clear out the screen and if you scroll up there is a chance to see some residue printed information. This does not happen when you directly run the executable file that is also included in the project file.