**Chapter 7 Lab**

**Classes and Members**

**Lab Objectives**

* Be able to declare a new class
* Be able to write a constructor
* Be able to write member functions that return a value
* Be able to write member functions with parameters
* Be able to instantiate an object
* Use calls to member functions to access and change the state of an object
* Be able to write assert statements to test a class
* Use default values to overload a constructor

**Introduction**

Everyone is familiar with a television. It is the object we are going to create in this lab. First, we need a blueprint. All manufacturers have the same basic elements in the televisions they produce as well as many options. We are going to work with a few basic elements that are common to all televisions. Think about a television in general. It has a brand name (i.e. it is made by a specific manufacturer). The television screen has a specific size. It has some basic controls. There is a control to turn the power on and off. There is a control to change the channel. There is also a control for the volume. At any point in time, the television’s state can be described by how these controls are set.

We will write the television class. Each object that is created from the television class must be able to hold information about that instance of a television in fields. So a television object will have the following attributes:

* **manufacturer**. The manufacturer attribute will hold the brand name. This cannot change once the television is created, so will be a named constant.
* **screenSize**. The screenSize attribute will hold the size of the television screen. This cannot change once the television has been created so will be a named constant.
* **powerOn**. The powerOn attribute will hold the value true if the power is on, and false if the power is off.
* **channel**. The channel attribute will hold the value of the station that the television is showing.
* **volume**. The volume attribute will hold a number value representing the loudness (0 being no sound).

These attributes become **member variables** in our class.

The television object will also be able to control the state of its attributes. These controls become **member functions** in our class.

* **SetChannel**. The setChannel function will store the desired station in the channel field.
* **Power**. The power function will toggle the power between on and off, changing the value stored in the powerOn field from true to false or from false to true.
* **IncreaseVolume**. The increaseVolume function will increase the value stored in the volume field by 1.
* **DecreaseVolume**. The decreaseVolume function will decrease the value stored in the volume field by 1.
* **GetChannel**. The getChannel function will return the value stored in the channel field.
* **GetVolume**. The getVolume function will return the value stored in the volume field.
* **GetManufacturer**. The getManufacturer function will return the constant value stored in the MANUFACTURER field.
* **GetScreenSize**. The getScreenSize function will return the constant value stored in the SCREEN\_SIZE field.

We will also need a constructor that will be used to create an instance of a Television.

These ideas can be brought together to form a UML (Unified Modeling Language) diagram for this class as shown below.

🡸**Class Name**

Television

-MANUFACTURER: string

-SCREEN\_SIZE: int

-powerOn: bool

-channel: int

-volume: int

🡸**Attributes or fields**

🡸**Functions**

+Television(brand: string, size: int):

+SetChannel (station: int): void

+Power( ): void

+IncreaseVolume( ): void

+DecreaseVolume( ): void

+GetChannel( ): int

+GetVolume( ): int

+GetManufacturer( ): string

+GetScreenSize( ): int

+ public

- private

Data type returned

**Task #1 Creating a New Class**

1. Create a new project in Visual Studio. Add a new Class called Television. This should create a .cpp file and a .h file.
2. In the .h file, add “using namespace std;” just before the class name.
3. Be sure to only add the preprocessor directives needed. Adding files that are not needed take up extra memory and processing power.
4. Declare the private member variables in the header file.
   1. Declare the 2 constant fields listed in the UML diagram.
   2. Declare the 3 remaining fields listed in the UML diagram.
5. Write a comment for each field indicating what it represents.
6. Declare the public member functions in the header file.
   1. Declare the constructor that takes 2 parameters
   2. Declare the mutator (set) function for the channel.
   3. Declare the function to toggle the power.
   4. Declare the 2 functions to change the volume.
   5. Declare the 4 accessor (get) functions corresponding to the remaining member variables. Be sure to add the keyword **const** after the ( ) to specify that the accessor functions will not change any data stored in the calling object.

**Task #2 Writing a Constructor**

1. In the .cpp file, note that Visual Studio already added the preprocessor directive for the header file of this class.
2. For all functions in this file, be sure to add the class name and scope resolution operator immediately preceding the function name in the function heading.
3. Create a constructor definition that has two parameters, a manufacturer’s brand and a screen size. These parameters will bring in information.
4. Inside the constructor, assign the values taken in from the parameters to the corresponding fields.
5. Initialize the powerOn field to false (power is off), the volume to 20, and the channel to 2.
6. Write comments describing the post-conditions of the constructor above the function header.

**Task #3 Functions**

1. Define accessor functions called GetVolume, GetChannel, GetManufacturer, and GetScreenSize that return the value of the corresponding field. Be sure to add the keyword **const** after the ( ) in the function heading to specify that the accessor functions will not change any data stored in the calling object.
2. Define a mutator function called SetChannel accepts a value to be stored in the channel field.
3. Define a mutator function called Power that changes the state from true to false or from false to true. This can be accomplished by using the NOT operator (!). If the boolean variable powerOn is true, then !powerOn is false and vice versa. Use the assignment statement

powerOn = !powerOn;

to change the state of powerOn and then store it back into powerOn (remember assignment statements evaluate the right-hand side first, then assign the result to the left-hand side variable.

1. Define two mutator functions to change the volume. One function should be called IncreaseVolume and will increase the volume by 1. The other function should be called DecreaseVolume and will decrease the volume by 1.
2. Write comments above each function header describing the pre- and post-condition of the function.

**Task #4 Running the application**

1. You can only execute (run) a program that has a main function, so there is a driver program that is already written, TelevisionTest.cpp, to test out your Television class. Make sure to put the source file in the same directory as Television.cpp and Television.h. prior to adding it to your project.
2. Compile and run. You will find that it aborts. This is because in real life, the television controls only work if the power is on. You will need to go back to the function implementations and put in a condition that checks the power before it allows changes to the volume or channel.
3. Compile and run. It should exit normally.

**Task #5 Creating other instances of a Television**

1. Edit the TelevisionTest.cpp file.
2. Declare another Television object called portable.
3. Instantiate portable to be a Sharp 32-inch television.
4. Write assert statements to check the manufacturer and size.
5. Compile and run. Debug and fix any issues.
6. Declare another Television object called flatScreen but use the default constructor. Why is there a problem with this? Fix the constructor declaration that is currently written in Television.h so that the default will be a LG 54-inch television.
7. Write assert statements to check the changes you made.
8. Compile and run. Debug and fix any issues.