# Practical 2.1 - Classes and File Structure

In this practical, you will build two simple classes representing Monsters and Witches. These two classes (mine are called Monster and Witch) have **no data fields**. They each have a constructor, and a single class method called Speak(). When a Monster speaks, it pops up a MessageBox containing “I am a Monster! Roar!”. When a Witch speaks, it pops up a MessageBox containing “I am a Witch! Hehehe!”.

On the I: drive you will find two resources for this practical. One is an executable called MonsterDemo2016.exe. This shows you the application you are to build. The other is a code skeleton for you to work with. This is the folder MonsterDemoSkeleton. This is a partially completed VS2013 project with the Form already created.

1. Copy the folder MonsterDemoSkeleton to your desktop. Please DO NOT try to work with the remote copy.
2. Open the folder, and double click on the solution file MonsterDemoSkeleton.sln to open it in VS2013.
3. Create the two new classes:
   1. Create each class via Project->Add Class. Create each as a managed class.
   2. Remember to copy the **using** statements from Form1.h into each new .h file.
   3. Delete anything you want from the .h and .cpp files VS creates for you **except** the compiler directives. These are the statements that begin with #.
   4. Remember: Place the class definitions in the .h files and place the actual code in the .cpp files (please **do not** define the Speak() methods inline).
   5. Remember to preface each method in the .cpp with the name of the class. For example, I have the following line in my Monster.cpp

void Monster::Speak()

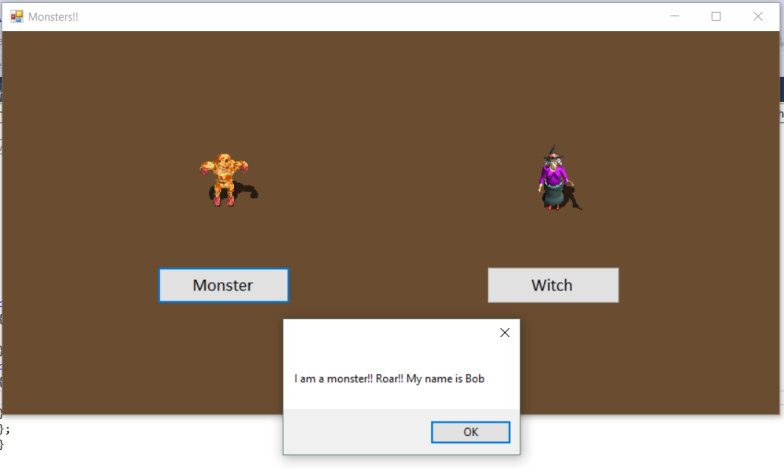
* 1. Since these classes have no data members and need no initialisation, their constructors will be empty.

1. Add a Witch and a Monster object to the Form class as data members. Remember to use handles (managed pointers). For example, my Form1.h file contains this line:

Monster^ monster;

1. Instantiate your Witch and Monster objects in the Form1\_Load event.
2. Write the handlers for each of the two buttons so that your application behaves like the demo.

**Optional Extension if you finish early:** Add a String data member to the Monster class to hold its name. Pass the value into the constructor, and add it to the output of Speak(), as shown in the following image:



**Think about:** An application’s “class architecture” is the collection of classes it contains. Can you think of a better architecture for this program? (Hint: When you have two classes that have nearly identical code, there is almost always a better architecture.) If you finish early, research the necessary C++/CLI syntax, and implement the correct architecture.