Lab 9: Polymorphism

1.1 Information

Topics: Polymorphism

Turn in: All source files (.cpp and .hpp).

Starter files: Download on GitHub or D2L.

```
Lab 09 - Polymorphism/
├── lab9_main.cpp ...  Contains main()
├── lab9_game.hpp ...  Contains game code
├── lab9_Character.hpp ...  Character family of
│                           classes
├── lab9_Character.cpp ...  Functions for Character
│                           family functions
├── CodeBlocks Project/ ...  Code::Blocks project here
├── Makefile/ ...  Makefile here
└── VS2015 Project/ ...  Visual Studio 2015 project here
```

This program does not have any tests associated with it; you will have to manually test it by playing the game, or write your own tests.

Contents

1.1 Information 1
1.2 Class declarations 3
1.2.1 ICharacter interface class 3
1.2.2 NPC class 4
1.2.3 Player class 4
1.3 Function definitions 5
1.3.1 ICharacter::GetAttack 5
1.3.2 ICharacter::GetHealing 5
1.3.3 ICharacter::GetHP 5
1.3.4 ICharacter::GetName 5
1.3.5 ICharacter::Setup 6
1.3.6 ICharacter::SubtractDamage 6
1.3.7 NPC::GetChoice 6
1.3.8 Player::GetChoice 6
1.4 The game 7
1.4.1 Example output 8
1.5 Appendix A: Starter code 9

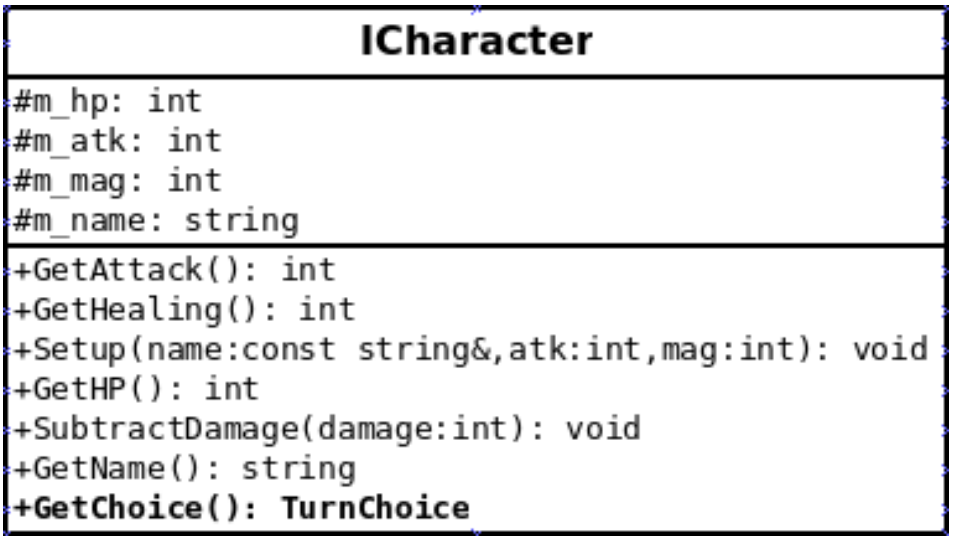
1.2 Class declarations

First, you will add class declarations in the `lab9_Character.hpp` file. This file will contain all members of the Character family of classes.

1.2.1 ICharacter interface class

The ICharacter class is the interface class, which will contain the¡ common functionality between all Character classes. This class will also contain one *pure virtual function*: GetChoice(). This function is pure virtual because each subclass will be *required* to implement their own version of this function.

Build the ICharacter class based on this UML diagram:



Key

The top section contains the member variables. The # symbol means that these are **protected**.

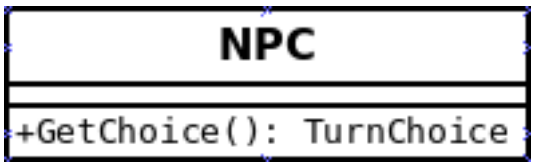
The bottom section are the member functions. The + symbol means that these are **public**.

The function that is bold, **GetChoice()**, is a pure virtual function. To make it virtual, put the keyword **virtual** at the beginning of the function signature, and to make it pure-virtual, put **= 0** at the end of the signature.

```
virtual TurnChoice GetChoice() = 0;
```

1.2.2 NPC class

Declare the NPC class as per the diagram. The GetChoice function should also be marked as **virtual**, but it won't be pure-virtual, so don't include the **= 0** at the end.



1.2.3 Player class

Declare the Player class as per the diagram. The GetChoice function should also be marked as **virtual**, but it won't be pure-virtual, so don't include the **= 0** at the end.

1.3 Function definitions

Now in the .cpp file you will implement the functions.

1.3.1 ICharacter::GetAttack

```
1  int ICharacter::GetAttack()
2  {
3      return rand() % m_atk;
4  }
```

1.3.2 ICharacter::GetHealing

```
1  int ICharacter::GetHealing()
2  {
3      int healing = rand() % m_mag;
4      m_hp += healing;
5
6      if ( m_hp > 100 )
7      {
8          m_hp = 100;
9      }
10
11     return healing;
12 }
```

1.3.3 ICharacter::GetHP

```
1  int ICharacter::GetHP()
2  {
3      return m_hp;
4  }
```

1.3.4 ICharacter::GetName

```
1  string ICharacter::GetName()
2  {
3      return m_name;
4  }
```

1.3.5 ICharacter::Setup

```
1 void ICharacter::Setup( const string& name, int atk, int
      mag )
2 {
3     m_name = name;
4     m_atk = atk;
5     m_mag = mag;
6     m_hp = 100;
7 }
```

1.3.6 ICharacter::SubtractDamage

```
1 void ICharacter::SubtractDamage( int damage )
2 {
3     m_hp -= damage;
4 }
```

1.3.7 NPC::GetChoice

```
1 TurnChoice NPC::GetChoice()
2 {
3     return TurnChoice( rand() % 2 );
4 }
```

1.3.8 Player::GetChoice

```
1 TurnChoice Player::GetChoice()
2 {
3     int choice;
4     cout << "Select your choice: ";
5     cin >> choice;
6     cout << endl;
7     return TurnChoice( choice );
8 }
```

1.4 The game

Back in `lab9_game.hpp`, uncomment out the code in the `Game()` function. Afterward, take note of the following parts of the code:

```
1 Player player;
2 player.Setup( "Player", 10, 10 );
3
4 NPC npc;
5 npc.Setup( "Enemy", 5, 5 );
6
7 ICharacter* ptrPlayers[2];
8 ptrPlayers[0] = &player;
9 ptrPlayers[1] = &npc;
```

Here, Iv'e declared a Player object and an NPC object, as well as an array of ICharacter pointers. Both the Player and the NPC can be treated as ICharacter objects to use their common interface. The game itself doesn't care which class it is, just that the functions of ICharacter are used.

Later on in the game, it reduces duplicate code by using a for-loop to execute the same code for each player:

```
1 for ( int i = 0; i < 2; i++ )
2 {
3     TurnChoice choice = ptrPlayers[i]->GetChoice();
```

This would be much more managable if we were going to keep building on this program and adding more characters. Conversely, if we had almost-duplicate code with only slight differences back-to-back, the more characters we add, the more cluttered our code would get.

This is why we write generic **interface** classes, and then write more specialized child classes to handle specific functionality. In the implementation, we cut down on duplicate code, and in the design, we spend more time thinking about how to keep the interface and the specialized classes organized and clean.

1.4.1 Example output

When running the game, it should look like this:

```
ROUND 3
Your HP: 98 Enemy HP: 92

0. Attack
1. Heal
Select your choice: 0

* Player attacks Enemy for 6 damage!
* Enemy attacks Player for 4 damage!

Press a key to continue...
```

1.5 Appendix A: Starter code

lab9_main.cpp

```
#include <iostream>
#include <cstdlib>
using namespace std;

#include "lab9_game.hpp"

int main()
{
    Game();

    return 0;
}
```

lab9_game.cpp

```
#ifndef _GAME
#define _GAME

#include "lab9_Character.hpp"

void ClearScreen()
{
    #if defined(WIN32) || defined(_WIN32) || defined(
   __WIN32) && !defined(__CYGWIN__)
        system( "cls" );
    #else
        system( "clear" );
    #endif
}

void DrawSeparator( int length )
{
    cout << endl;
    for ( int i = 0; i < length; i++ )
    {
        cout << "-";
    }
    cout << endl;
```

```
23 }
24
25 void DisplayMenu( int round, int playerHP, int enemyHP )
26 {
27     ClearScreen();
28     cout << "ROUND " << round << endl;
29     cout << "Your HP: " << playerHP << "\tEnemy HP: " <<
       enemyHP << endl << endl;
30     cout << "0. Attack" << endl;
31     cout << "1. Heal" << endl;
32 }
33
34 void Game()
35 {
36 //     Player player;
37 //     player.Setup( "Player", 10, 10 );
38 //
39 //     NPC npc;
40 //     npc.Setup( "Enemy", 5, 5 );
41 //
42 //     ICharacter* ptrPlayers[2];
43 //     ptrPlayers[0] = &player;
44 //     ptrPlayers[1] = &npc;
45 //
46 //     int round = 1;
47 //
48 //     while ( ptrPlayers[0]->GetHP() > 0 && ptrPlayers
       [1]->GetHP() > 0 )
49 //     {
50 //         DisplayMenu( round, ptrPlayers[0]->GetHP(),
       ptrPlayers[1]->GetHP() );
51 //
52 //         for ( int i = 0; i < 2; i++ )
53 //         {
54 //             TurnChoice choice = ptrPlayers[i]->
       GetChoice();
55 //
56 //             int otherPlayer;
57 //             if ( i == 0 )   { otherPlayer = 1; }
58 //             else            { otherPlayer = 0; }
59 //
60 //             if ( choice == ATTACK )
61 //             {
```

```
62 //                int damage = ptrPlayers[i]->GetAttack
   ();
63 //
64 //                cout    << "* " << ptrPlayers[i]->
   GetName() << " attacks "
65 //                        << ptrPlayers[ otherPlayer ]->
   GetName() << " for "
66 //                        << damage << " damage!" <<
   endl;
67 //
68 //                ptrPlayers[ otherPlayer ]->
   SubtractDamage( damage );
69 //            }
70 //            else if ( choice == HEAL )
71 //            {
72 //                int health = ptrPlayers[i]->GetHealing
   ();
73 //
74 //                cout    << "* " << ptrPlayers[i]->
   GetName() << " heals themself for "
75 //                        << health << " HP!" << endl;
76 //            }
77 //        }
78 //
79 //        cout << endl;
80 //
81 //        round++;
82 //
83 //        cout << "Press a key to continue..." << endl;
84 //        cin.ignore();
85 //        cin.get();
86 //    }
87 //
88 //    for ( int i = 0; i < 2; i++ )
89 //    {
90 //        if ( ptrPlayers[i]->GetHP() <= 0 )
91 //        {
92 //            cout << ptrPlayers[i]->GetName() << " has
   been defeated!" << endl;
93 //        }
94 //    }
95 //
96 //    cout << endl << "GAME OVER" << endl;
```

```
97 }
98
99 #endif
```

lab9_Character.hpp

```
1  #ifndef _CHARACTER
2  #define _CHARACTER
3
4  #include <string>
5  using namespace std;
6
7  enum TurnChoice { ATTACK = 0, HEAL = 1 };
8
9
10
11 #endif
```

lab9_Character.cpp

```
1 #include "Character.hpp"
2
3 #include <cstdlib>
4 #include <iostream>
5 using namespace std;
```