

C++ Week 6 OOP Revision

Introduction

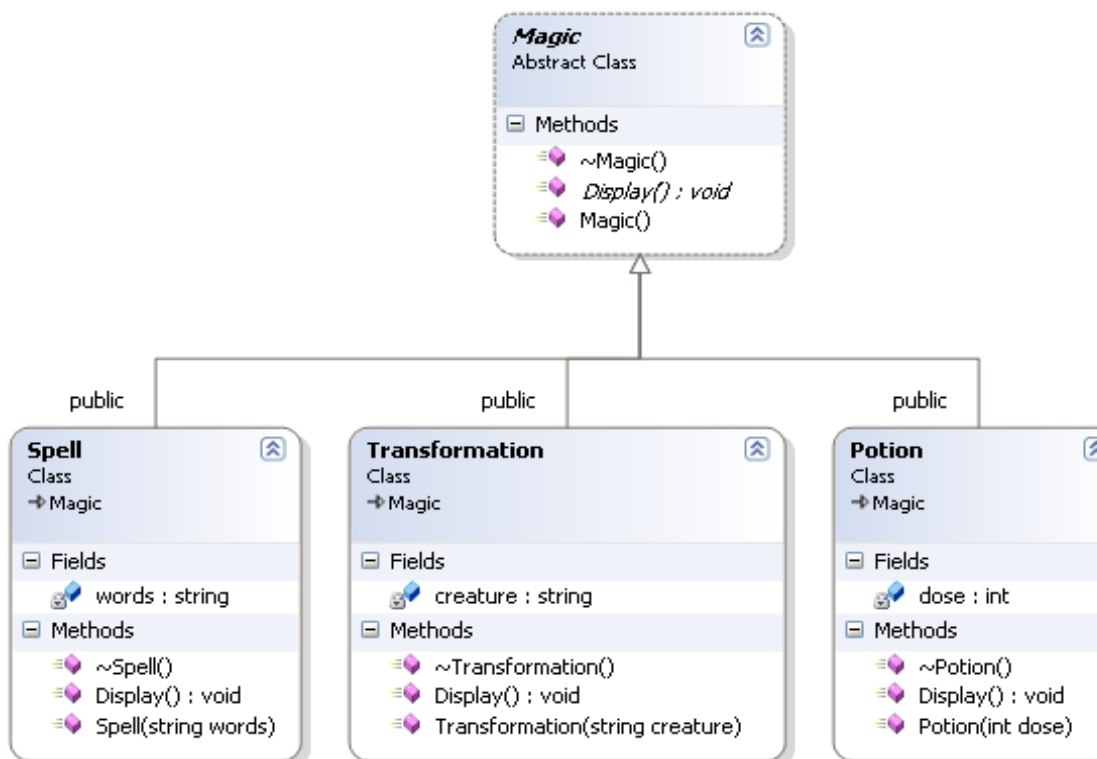
This practical will step you through the development of an Object-Oriented System that manages Wizards and the different Magic they can perform.

It covers the following subject areas

- Inheritance
- Polymorphism
- Abstract classes
- Dynamic arrays of pointers
- Passing pointers as arguments
- Invoking base member functions
- Enumerate types

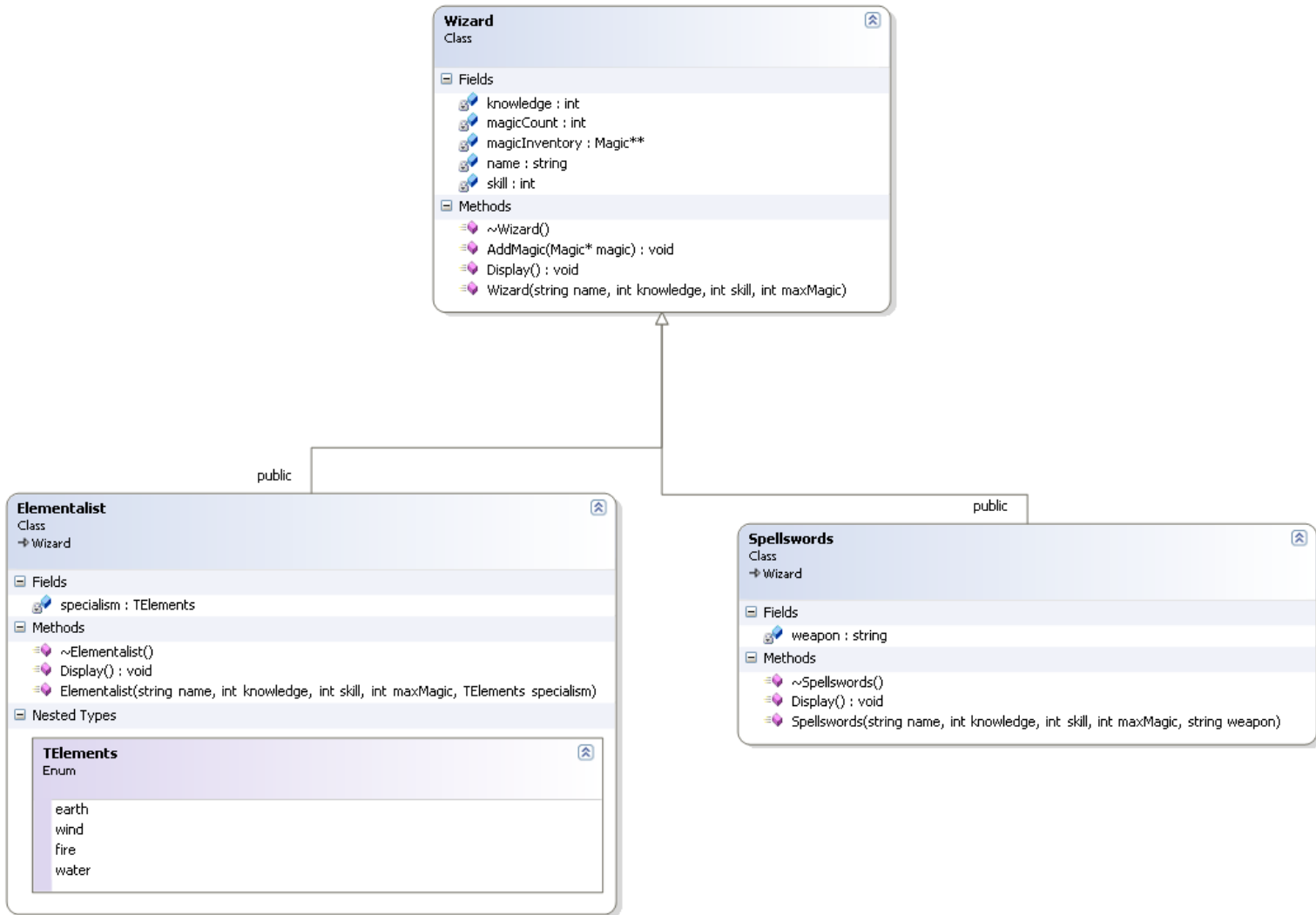
The Design

The design includes two class hierarchies one for the different type of Magic and a second for different types of Wizards. Each of the wizard types maintains an array of Magic objects, the magic they can perform.



C++ Week 6 OOP Revision

The functionality of the application is limited. It will allow you to create instances of different wizards and assign them the magic they can perform. By invoking the Wizard's Display member function, the details of the Wizard along with the Magic they can perform will be output to the console.



C++ Week 6 OOP Revision

Below is a textual description of each class.

Magic

Abstract class.

Spell

words: The words spoken to cast the spell

Constructor: Assigns argument to words data member.

Display: Outputs the text "Spell Words" followed by the words and new line.

Potion

dose: The amount of potion required

Constructor: Assigns argument to dose data member.

Display: Outputs the text "Potion dose" followed by the dose and new line.

Transformation

creature: The creature to be transformed into

Constructor: Assigns argument to creature data member.

Display: Outputs the text "Transformation creature" followed by the creature and new line.

Wizard

name: First and last name of the wizard

knowledge: Value between 1 and 100

skill: Value between 1 and 100

magicInventory:Dynamic array of pointers to Magic objects

magicCount: Number of Magic objects saved in the array

Constructor: Assigns the three arguments to their corresponding data members. Creates an array with maxMagic elements and sets the magicCount to 0.

AddMagic: Adds the argument to the array at the index indicated to by the value in magicCount and increments the value in magicCount by one.

Display: Outputs the data members and iterates through the magicInventory array invoking each magic's Display member function.

Elementalist

specialism: Assigned the element the wizard specialises in.

Constructor: Assign the specialism to the corresponding data member and passed the three remaining arguments to the Wizard's constructor.

Display: Invokes the Wizard's Display member function and then using a switch statement outputs one of the following "Earth", "Wind", "Fire" or "Water" depending on the value within the specialism data member.

C++ Week 6 OOP Revision

Spellswords

weapon: The weapon used to cast spells.

Constructor: Assigns the weapon argument to the corresponding data member and passed the remaining arguments to the Wizard constructor.

Display: Invokes the Wizard's Display member function and outputs the text "Weapon" followed by the weapon.

The Implementation

Step 1:

Create the skeletons of each class using separate header(.h) and source (.cpp) files (Project->Add Class).

If you haven't done so already when creating the classes, create the class inheritance hierarchy, by including the parent classes in the child class's definition. All inheritance in this application will be public, so include the word public before the Parent Class.

e.g.

```
#pragma once
#include "magic.h"

class Spell :public Magic {
public:
    Spell(void);
    ~Spell(void);
};
```

When inheriting a class ensure you include the header for the parent class in the child class header file.

Step 2:

Within Elementalist add the enum type (with public access). Place the line below above any reference to the TElements type within the header file.

```
enum TElements {earth,wind,fire,water};
```

C++ Week 6 OOP Revision

Step 3:

Add the data members to the class definitions. The class diagram includes the data member type and access modifiers (key – protected, padlock- private otherwise public.

```
#pragma once
#include "magic.h"
#include <string>

using namespace std;

class Spell :public Magic {
private:
    string words;
public:
    Spell(void);
    ~Spell(void);
};
```

Note: For classes using the string type include the string header and std namespace as illustrated above

The Wizard contains an array of Magic objects this is defined as

```
Magic **magicInventory;
```

The array name contain a pointer to the first element in the array. In this case the first element of the array is a pointer hence **.

You will need to include the magic.h file within wizard.h in order for it to recognise the Magic class.

The specialism data member declared within Elementalist is private but must be declared under the enum type which is public. So you definition may look like that below.

```
class Elementalist : public Wizard {
public:
    enum TElements {earth,wind,fire,water};
private:
    TElements specialism;
public:
    Elementalist(void);
    ~Elementalist(void);
};
```

C++ Week 6 OOP Revision

Step 4:

Visual Studio will have created the default constructors and destructors for you. We do however need to change the constructors to mirror those in the design. Add the formal parameters to **each** constructor and where appropriate assign these to the data members.

In the header file

```
class Spell : public Magic {  
private:  
    string words;  
public:  
    Spell(string words);  
    ~Spell(void);  
    void Display();  
};
```

In the cpp file we assign the parameter to the data member using the initialise list.

```
Spell::Spell(string words):words(words){ }
```

Both the Elementalist and Spellwords class constructors will need to invoke the parent class constructor in order to pass the parameter values up to the parents constructor so they can be assigned to the relevant data members,

e.g

```
Elementalist::Elementalist(string name,int knowledge,int skill,int maxMagic,TElements  
specialism):Wizard(name,knowledge,skill,maxMagic),specialism(specialism){
```

Step 5:

Only the Wizard constructor contains statements. These should initialise the magicCount variable to 0 and instantiate the magicInventory, the code for which is included below.

```
magicInventory = new Magic*[maxMagic];
```

C++ Week 6 OOP Revision

Step 6:

Add the Display member functions and include the code necessary to implement the behaviour outlined in the class descriptions. Where you are required to invoke the parent class's Display member function do so by prefixing the function name with the Class name and scope operator.

Note: You will need to include iostream and using namespace std; in any file that uses cout to output text to the console.

e.g.

```
#include "StdAfx.h"
#include "Spellwords.h"
#include <iostream>

using namespace std;

void Spellwords::Display(){
    Wizard::Display();
    cout << "Weapon " << weapon << endl;
}
```

Note: The Magic's Display member function is virtual and abstract, it has no implementation in the cpp file and should be declared as follows

```
virtual void Display() = 0;
```

The = 0 indicates that there is no definition and as a consequence of this objects can't be instantiated from this class. The Wizard's Display member function while not being abstract is virtual and should display each of the magic objects contained within the array using the code below.

```
for(int n = 0; n < magicCount; n++)
    magicInventory[n]->Display();
```

The Elementalist's Display member function should contain a switch statement that outputs the element corresponding to the value held within the specialism data member.

C++ Week 6 OOP Revision

Step 7:

Declare the Wizard's addMagic member function within the header file

```
void AddMagic(Magic *magic);
```

and define it within the cpp file.

```
void Wizard::AddMagic(Magic *magic){  
    magicInventory[magicCount++] = magic;  
}
```

Step 8:

At this stage we can create a couple of instances of wizard types and store them in an array. First above the main function include the header files for all the classes. Now add the statements below to the main function.

```
Wizard **wizards = new Wizard*[2];  
wizards[0] = new Elementalist("Fred",80,20,5,Elementalist::fire);  
wizards[1] = new Spellswords("John",60,50,3,"Wand");
```

Now let's assign some magic to each wizard

```
wizards[0]->AddMagic(new Potion(50));  
wizards[0]->AddMagic(new Transformation("Cat"));  
wizards[1]->AddMagic(new Potion(30));  
wizards[1]->AddMagic(new Spell("Abra cadabra"));  
wizards[1]->AddMagic(new Transformation("Bird"));
```

Here we assign two magic types to the first wizard and three to the second. Note how I've used anonymous objects here, I could have written this

```
Potion *potion = new Potion(50);  
wizards[0]->AddMagic(potion);
```

rather than

```
wizards[0]->AddMagic(new Potion(50));
```

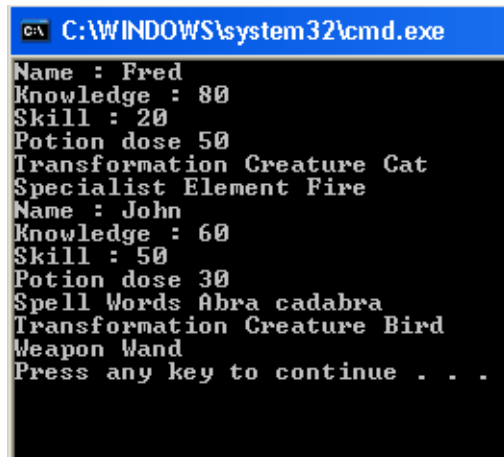
However, it is rather long winded as I don't need a pointer to the Magic within main.

Finally, we can invoke the Display member functions of each Wizard.

```
for(int n=0;n < 2;n++)  
    wizards[n]->Display();
```


C++ Week 6 OOP Revision

When you run the program, your output should be similar to that below:

A screenshot of a Windows command prompt window. The title bar is blue and contains the text 'C:\WINDOWS\system32\cmd.exe'. The command prompt itself has a black background with white text. The output of a program is displayed as follows:

```
Name : Fred
Knowledge : 80
Skill : 20
Potion dose 50
Transformation Creature Cat
Specialist Element Fire
Name : John
Knowledge : 60
Skill : 50
Potion dose 30
Spell Words Abra cadabra
Transformation Creature Bird
Weapon Wand
Press any key to continue . . .
```