More on Abstract classes, part 1

Example of using abstract classes: the NinjaTracker class

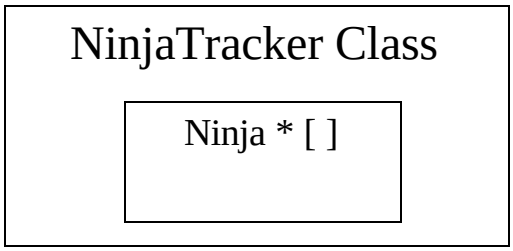
We're going to design a program that keeps track of CSE 143 Section AB's ninja forces. To do this, we're going to have a NinjaTracker class, which will hold some kind of data structure that conmtains all of our ninja.

However, there's a problem: there are many types of Ninja in our army!

Luckily for us, we know about class inheritance. Using this, let's create our class.

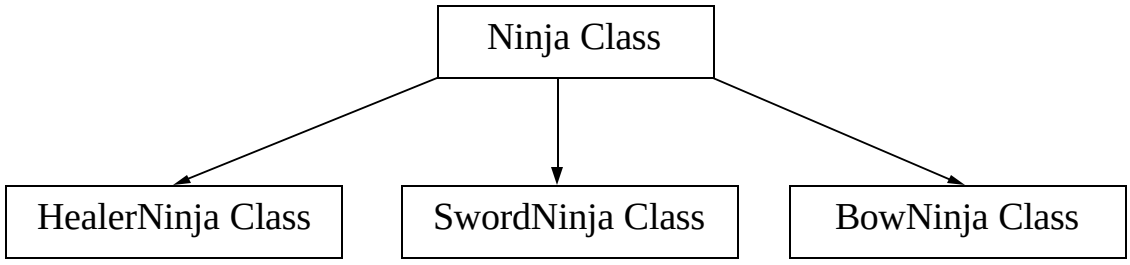
Step 1: DESIGN

Lets design a class hierarchy that will fit what we need to do. We should have one class of type NinjaTracker, that HAS A data structure (we'll use an array) of Ninjas. The array should be an array of pointers so we can have each element point to a separate Ninja allocated dynamically.



However, there are several different types of Ninja! So, there are going to have to be classes derived from the Ninja Class.

MORE IMPORTANTLY, since every Ninja is either a Healer Ninja, a Sword Ninja, or a Bow Ninja, **we make the Ninja class abstract**.



Now, every element in our array of Ninja * class can point to anything of type HealerNinja, SwordNinja, or BowNinja! (Remember how this works?)

More on Abstract classes, part 2

Step 2: Implement!

```
// The NinjaTracker class
#include "Ninja.h"
#include "BowNinja.h"
#include "SwordNinja.h"
#include "HealerNinja.h"

class NinjaTracker {
public:
        NinjaTracker() { data[0] = new SwordNinja(); data[1] = new BowNinja();
                                                        data[2] = new HealerNinja(); }
        void outputNinja() { data[0]->outputType(); data[1]->outputType();
                                                        data[2]->outputType(); }
        ~NinjaTracker() { delete data[0]; delete data[1]; delete data[2]; }
private:
        Ninja * data[3];
};
```

```
// Abstract Ninja class
#include <iostream>
using namespace std;

class Ninja {
public:
        virtual void outputType()=0;
};
```

```
// BowNinja class inherits from Ninja
#include "Ninja.h"

class BowNinja : public Ninja {
public:
        virtual void outputType() {
         cout << "I am a Bow NINJA!" << endl; }
};
```

```
// HealerNinja class inherits from Ninja
#include "Ninja.h"

class HealerNinja : public Ninja {
public:
        virtual void outputType() {
        cout << "I am a Healer NINJA!" << endl; }
};
```

```
// SwordNinja class inherits from Ninja
#include "Ninja.h"

class SwordNinja : public Ninja {
public:
        virtual void outputType() {
        cout << "I am a Sword NINJA!" << endl; }
};
```

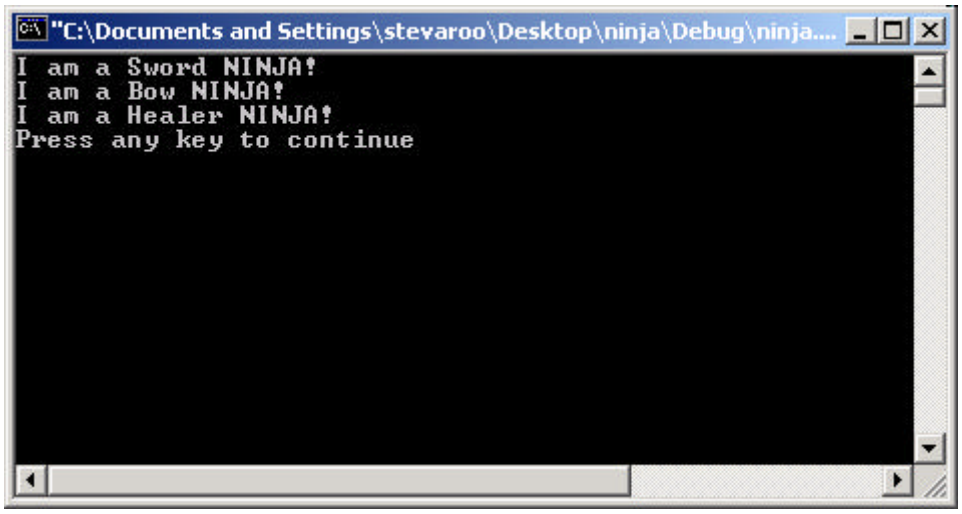
More on Abstract classes, part 3

Step 3: Test out our implementation. . .

Look at the preceding code carefully. What will the following main program output? Any errors?

```
// main

#include "NinjaTracker.h"

int main () {
        NinjaTracker SectionAB_ARMY;

        SectionAB_ARMY.outputNinja();

        return 0;
}
```

No errors. Here's what is output to the screen—the complete record of our Army of Ninja!



This is a quick and easy program using Abstract Classes! Note how we have an array of pointers to a base class, initialize them to hold instances of derived classes, and then use **virtual** to call methods in the derived class.

Can you think of some uses for this? (HW4, perhaps. . .?)