

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

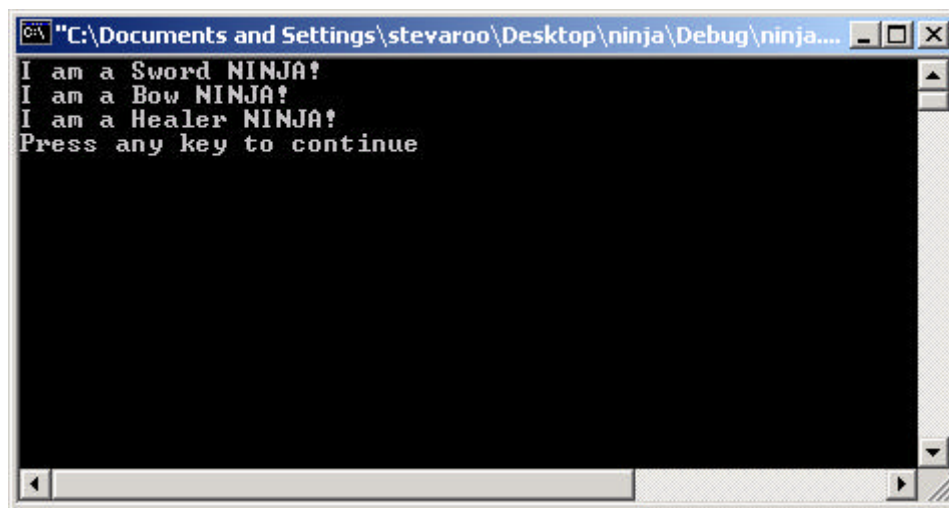
#include "NinjaTracker.h"

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!

A screenshot of a Windows command prompt window. The title bar shows the file path: "C:\Documents and Settings\stevaroo\Desktop\ninja\Debug\ninja....". The window contains the following text:

```
I am a Sword NINJA!
I am a Bow NINJA!
I am a Healer NINJA!
Press any key to continue
```

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. . . ?)

Recursion

Recursion is a powerful tool that can be used to solve just about any problem. A recursive function works by checking to see if the end case has been reached, doing a part of a problem, and then calling itself again to do the rest of the problem.

The hardest part of grasping recursion is thinking recursively! Do the following problems recursively:

$$5 + 3 + 1 + 0 = ?$$

$$10! = ?$$

Just about anything you can do iteratively (with loops) you can do recursively! As an example, convert the following familiar *while* loop into a recursive function.

```
// inside "readfile.cpp"
// get data from a file into an array

int data[100];
int i = 0;
ifstream in("test.txt")

while(!in.eof()) {
    in >> data[i];
    i++;
}
```

The answer:

```
// recursive function readfile
int* readfile(int* data, ifstream& in, int i) {
    if (in.eof()) return data;
    else {
        in >> data[i];
        i++;
        return readfile(data, in, i);
    }
}
```

Recursion, Example 2

As a final example, let's implement the following recursive function:

We want a function that will continually take in integers and *average* them. This is a simple standalone function that we can call and then get a final value from.

As a hint, here's how the function is called in main, along with its prototype:

```
double findAverage (int input, int total, int numterms);

int main () {
    double Average;

    Average = findAverage(0, 0, 0);

    cout << "The Average is: " << Average << endl;

    return 0;
}
```

The answer:

```
double findAverage (int input, int total, int numterms) {
    cout << "Please enter the next number: ";
    cin >> input;
    cout << endl;
    if (cin.good()) {
        total = total + input;
        numterms++;
        return findAverage(0, total, numterms);
    }
    else return ((double)total / (double)numterms);
}
```