hw07: Inheritance

Focus

- Inheritance

Problem:

We want to again extend our game of Warriors and Nobles

It turns out that there is more than one type of Noble. And in fact Warriors aren't the only people they can hire to do their fighting. There is magic in the land! Life (and death) are otherwise fairly similar. Nobles are still the only ones who go around declaring war upon each other.

Nobles

Nobles come in two varieties with rather fancy sounding titles:

- Those With Strength to Fight
- Those Who are Lords of the Land

Those With Strength to Fight

This type of Noble is rather different from those we have encountered before. They actually do their own fighting!!!

Those With Strength to Fight will fight using *only their own*strength. They do not hire anyone to fight for them and therefore do not have an army. They are born with a certain strength and they have no hope, neither through magic nor excessive exercise, to ever again increase their strength. Alas, our poor fighters will eventually have no strength left with which to fight and thus they shall meet their final demise.

Those Who are Lords of the Land

Those Who are Lords of the Land have no strength of their own but are able to fight with a certain strength by delegating their fighting to Protectors of the Nobles who have been hired by Those Who are Lords of the Land. A Lord's strength is the combined strength of his defenders. (Thus these are the people whom we knew as Nobles in the past.)

Protectors

Who are these Protectors that defend Lords to the death?

- they are not Nobles!
- they are entities for hire with strength to defend. The amount thereof set at birth.
- they are entities for hire that have names handed down from times of yore such as "QuessTar" and "VerTraahn", sacred names given at birth.

[**Clarification:** Hm, there's nothing you have to do about making sure that the names are spelled weirdly or any such. That's just there to make the story line sound more exciting.]

Lords approach Protectors to attempt to engage the service of the Protector. A Lord asks of the Protector if they are at present hired to serve another Lord and if the Protector states that he is, no transaction can take place. However if the transaction can be made, it is - and the Protector is, from that moment onward, in the service of the Lord as defender.

[**Clarification:** All this so-called dialog simply comes down to is the Lord trying to hire the Protector and succeeding if it's possible.]

In this land there are two kinds of Protectors:

- Wizards
- Warriors

They differ in their ways of defending: Wizards state "POOF". It is such a hard job to control the strength expended with magic!

There are, further, two kinds of Warriors whose strength is spent in much more known ways:

- Archers
  - who defend by stating "TWANG! <*archer's name*> says: Take that in the name of my lord, _____" (whence he shouts the name of the lord he is sworn to defend)
- Swordsmen
  - who defend by stating "CLANG! <*swordsman's name*>says: Take that in the name of my lord, _____" (whence he shouts the name of the lord he is sworn to defend)

Again, coders beware that your code do rightly enforce all these things about a Protector, be he Wizard, Archer or Swordsman.

Loss of Strength

Each entity with strength loses it in the same manor as described in prior assignments.

Death

It's a sad topic, but one we do have to address.

- People die when they lose a battle, whether they are a Noble or a Protector.
- Lords who are dead are in no position to hire anyone. Any attempt by a dead Lord to hire someone will simple fail and the Protector will remain unhired.
- Similarly dead Protectors cannot be hired. Any attempt to hire the dead simple fails.
- However curiously, as has been seen before, Nobles can declare battle even though they are dead.
- A Protector who is dead, however, cannot fight and so will not have anything to say, even if his Lord does go into battle again.

A sample test file

```
/*  Your classes go here */

int main() {
    Lord sam("Sam");
    Archer samantha("Samantha", 200);
    sam.hires(samantha);
    Lord joe("Joe");
    PersonWithStrengthToFight randy("Randolf the Elder",
250);
    joe.battle(randy);
    joe.battle(sam);
    Lord janet("Janet");
    Swordsman hardy("TuckTuckTheHardy", 100);
    Swordsman stout("TuckTuckTheStout", 80);
    janet.hires(hardy);
    janet.hires(stout);
    PersonWithStrengthToFight barclay("Barclay the Bold",
300);
    janet.battle(barclay);
    janet.hires(samantha);
```

```
    Archer pethora("Pethora", 50);
    Archer thora("Thorapleth", 60);
    Wizard merlin("Merlin", 150);
    janet.hires(pethora);
    janet.hires(thora);
    sam.hires(merlin);
    janet.battle(barclay);
    sam.battle(barclay);
    joe.battle(barclay);
}
```

My output for the above test file is below.

```
Joe battles Randolf the Elder
Randolf the Elder defeats Joe
Joe battles Sam
He's dead, Sam
Janet battles Barclay the Bold
CLANG!  TuckTuckTheHardy says: Take that in the name of my
lord, Janet
CLANG!  TuckTuckTheStout says: Take that in the name of my
lord, Janet
Barclay the Bold defeats Janet
Janet battles Barclay the Bold
He's dead, Barclay the Bold
Sam battles Barclay the Bold
TWANG!  Samantha says: Take that in the name of my lord,
Sam
POOF!
Sam defeats Barclay the Bold
Joe battles Barclay the Bold
Oh, NO!  They're both dead!  Yuck!
```

The Focus of the Assignment

The *main* focus of the assignment is to do the inheritance correctly! Place any members as high in the heirarchy as possible, but only as high as makes sense. The Noble battle method should only be defined once, not once for each possible combination of Nobles. Make use of abstract methods / classes where appropriate.

Cyclic association: While not discussed above or demonstrated in the test code, all of the Protectors can quit, just as our Warriors were able to do previously.

Separate compilation and namespaces: Again, use separate compilation. You may, if you like put all of the Protector types in one header / implementation pair, and the various Noble types in another pair. Or create a header / implementation pair for every class, as you like. And put all classes in the WarriorCraft namespace.