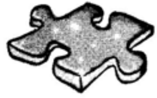


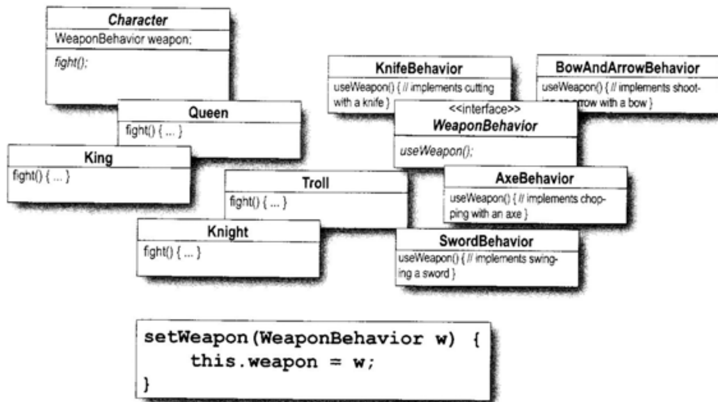
Design Puzzle

Below you'll find a mess of classes and interfaces for an action adventure game. You'll find classes for game characters along with classes for weapon behaviors the characters can use in the game. Each character can make use of one weapon at a time, but can change weapons at any time during the game. Your job is to sort it all out...

(Answers are at the end of the chapter.)

Your task:

1. Arrange the classes.
2. Identify one abstract class, one interface and eight classes.
3. Draw arrows between classes.
   - a. Draw this kind of arrow for inheritance ("extends").
   - b. Draw this kind of arrow for interface ("implements").
   - c. Draw this kind of arrow for "HAS-A".
4. Put the method setWeapon() into the right class.



Strategy Pattern
Pre-Test

Name ______________________________

Signature __________________________

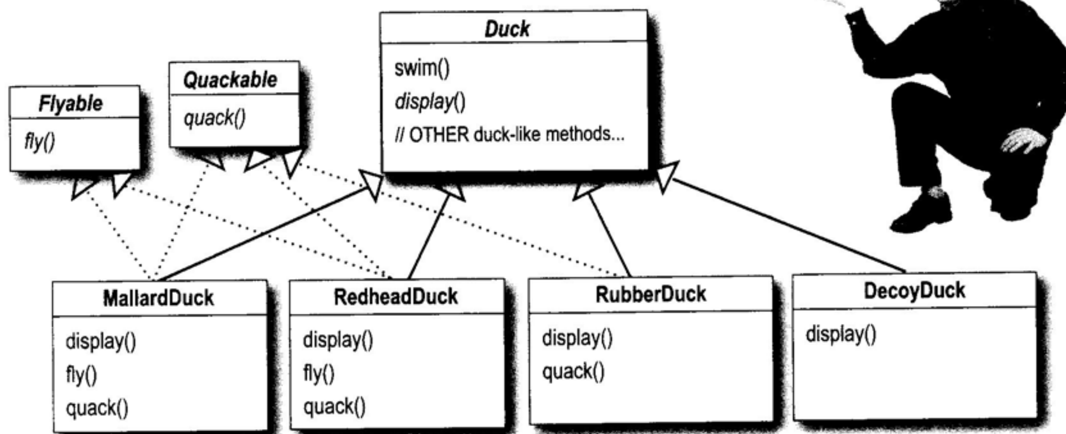
Student ID _________________________

How about an interface?

Joe realized that inheritance probably wasn't the answer, because he just got a memo that says that the executives now want to update the product every six months (in ways they haven't yet decided on). Joe knows the spec will keep changing and he'll be forced to look at and possibly override fly() and quack() for every new Duck subclass that's ever added to the program... *forever.*

So, he needs a cleaner way to have only *some* (but not *all*) of the duck types fly or quack.

I could take the fly() out of the Duck superclass, and make a ***Flyable() interface*** with a fly() method. That way, only the ducks that are *supposed* to fly will implement that interface and have a fly() method... and I might as well make a Quackable, too, since not all ducks can quack.



What do YOU think about this design?

What are some of the main problems with this design?

How would you use the Strategy Pattern to solve these problems?