Using Patterns to Help Students See the Power of Polymorphism

Supplement: Using the Decorator Pattern

Eugene Wallingford
University of Northern Iowa
wallingf@cs.uni.edu

SIGCSE Technical Symposium
February 24, 2001
1. We begin with a simple Ball hierarchy.

```
   Ball
   |   MovableBall
   |   BoundedBall  CannonBall
   |   |   PinBall
```

2. Students implement a Decelerating-MovableBall class.

```
   Ball
   |   MovableBall
   |   BoundedBall  CannonBall  DeceleratingMovableBall
   |   |   PinBall
```
3 Students implement a `Decelerating-BoundedBall` class.

```
  Ball
   |    |
MovableBall  |
   |
BoundedBall  CannonBall  Decelerating MovableBall
    |          |
Decelerating BoundedBall  PinBall
```

4 Students recognize the duplication: the decelerating ball classes override their superclass in exactly the same way.

What happens if we need to have cannonballs and pinballs that decelerate, too?
5 The worst-case scenario (or is it?):

\[
\text{Ball} \\
\quad \text{MovableBall} \\
\quad \quad \text{BoundedBall} \quad \text{CannonBall} \\
\quad \quad \quad \quad \text{Decelerating BoundedBall} \quad \text{PinBall} \\
\quad \quad \quad \quad \quad \quad \quad \text{Decelerating CannonBall} \\
\quad \quad \quad \quad \quad \quad \quad \quad \text{Decelerating PinBall}
\]

6 How can we avoid this duplication?

BoundedBalls respond to the same messages as MovableBalls. So, they are substitutable for one another.

How can we use this to our advantage?
Create a class that holds a `MovableBall` as an instance variable. Instances of the new class respond to all the same messages as `MovableBall`s.

An instance of the new class delegates all its messages to its instance variable. The only method that is different is `move()`, which also tells its instance variable to slow down a bit.

We can create `DeceleratingBall`s that wrap `MovableBall`s and `BoundedBall`s:

```java
new DeceleratingBall(
    new MovableBall( ... )
);

new DeceleratingBall(
    new BoundedBall( ... )
);
```
How can clients use Decelerating-Balls in places where they expect to use MovableBalls?

![Class Hierarchy Diagram]

Now, a ball that decelerates can be used polymorphically in place of a Movable-Ball.
9 Later, we can consider...

- the general idea

```plaintext
text
```

- implementing other decorators

```plaintext
text
```
• how one decorator can wrap another

```java
new DeceleratingBall(
    new ExpandingBall(
        new MovableBall( ... )
    )
);
```

• how to implement the delegation methods only once

![Diagram of Ball inheritance hierarchy]

- Ball
  - MovableBall
    - BoundedBall
    - CannonBall
    - DecoratedBall
      - PinBall
        - ExpandingBall
        - DeceleratingBall
      holds an instance of MovableBall