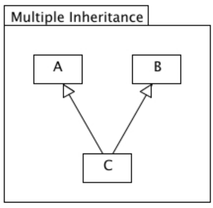
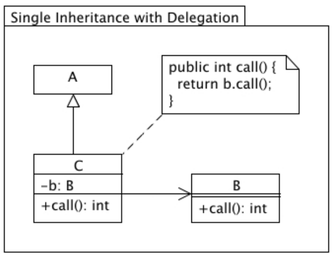
Delegation Pattern

"Delegation is like inheritance done manually through object composition." *[Lecture slides of course 601: "Object-Oriented Software Development" at the University of San Francisco]*





It is a technique where an object expresses certain behavior to the outside but in reality delegates responsibility for implementing that behaviour to an associated object. This sounds at first very similar to the proxy pattern, but it serves a much different purpose. Delegation is an abstraction mechanism which centralizes object (method) behaviour.

The diagram above shows how multiple-inheritance behavior can be accomplished in Java for actual classes. Of course delegation is not limited to scenarios where multiple-inheritance has to be avoided (as Java does not support this), but can be seen in general as an alternative to inheritance. The same functionality can also be accomplished with interfaces, however sometimes the relationship you want between two classes is actually one of containment rather than inheritance.

Applicability / Uses

Use the delegation pattern when:

- you need to reduce the coupling of methods to their class
- you have components that behave identically, but realize that this situation can change in the future.

Generally spoken: use delegation as alternative to inheritance. Inheritance is a good strategy, when a close relationship exist in between parent and child object, however, inheritance couples objects very closely. Often, delegation is the more flexible way to express a relationship between classes.

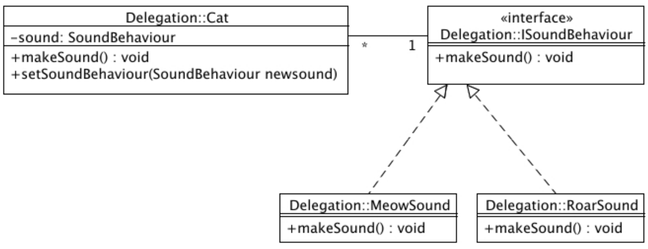
This pattern is also known as "proxy chains". Several other design patterns use delegation - the State, Strategy and Visitor Patterns depend on it.

Related Patterns

- Decorator: is similar to delegation and uses it heavily to accomplish its task.
- Delegation is very common, it is used by many other pattern like Visitor, Observer, Strategy, and Event Listener.

Structure

This example is actually more advanced than just plain delegation, it shows how delegation makes it easy to compose behaviors at run-time.



Sample

First lets create an Interface for our Delegates, since Delegates can be interchanged they all must implement the same interface:

```
public interface ISoundBehaviour {

        public void makeSound();
}

public class MeowSound implements ISoundBehaviour {

        public void makeSound() {
                System.out.println("Meow");
        }
}

public class RoarSound implements ISoundBehaviour {

        public void makeSound() {
                System.out.println("Roar!");
        }
}
```

Now lets create a cat class which uses the MeowSound per default:

```
public class Cat {
  private ISoundBehaviour sound = new MeowSound();

  public void makeSound() {
    this.sound.makeSound();
  }

  public void setSoundBehaviour(ISoundBehaviour newsound) {
        this.sound = newsound;
  }
}
```

Finally a Main class to test our delegation pattern:

```
public class Main {
        public static void main(String args[]) {
                Cat c = new Cat();
                // Delegation
                c.makeSound();          // Output: Meow
                // now to change the sound it makes
                ISoundBehaviour newsound = new RoarSound();
                c.setSoundBehaviour(newsound);
                // Delegation
                c.makeSound();          // Output: Roar!
        }
}
```

References / More Info

Wikipedia

Software Patterns, lecture slides of course 601: "Object-Oriented Software Development" - University of San Francisco

Proxy chains (also known as: Delegator pattern)

Patterns for Java Events

Software Patterns