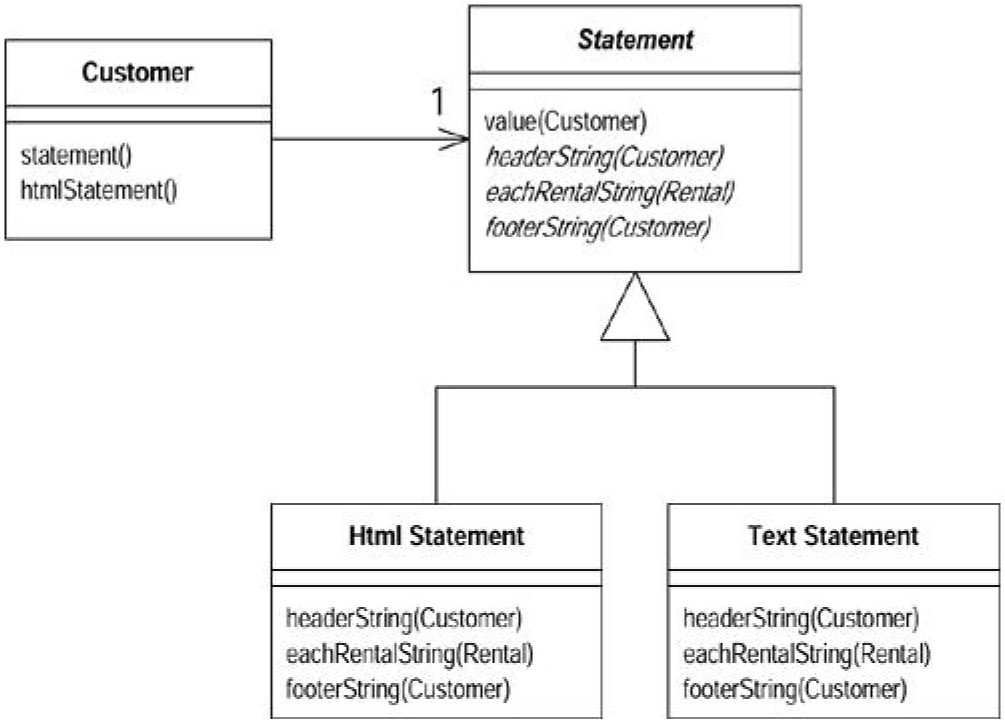
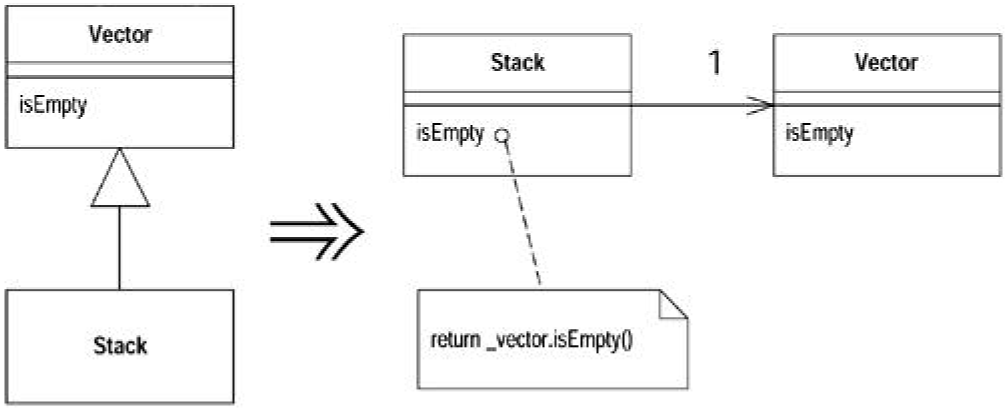


After this refactoring, it is easy to add new kinds of statements. All you have to do is create a subclass of statement that overrides the three abstract methods.

Replace Inheritance with Delegation

A subclass uses only part of a superclasses interface or does not want to inherit data.

Create a field for the superclass, adjust methods to delegate to the superclass, and remove the subclassing.



Motivation

Inheritance is a wonderful thing, but sometimes it isn't what you want. Often you start inheriting from a class but then find that many of the superclass operations aren't really true of the subclass. In this case you have an interface that's not a true reflection of what the class does. Or you may find that you are inheriting a whole load of data that is not appropriate for the subclass. Or you may find that there are protected superclass methods that don't make much sense with the subclass.

You can live with the situation and use convention to say that although it is a subclass, it's using only part of the superclass function. But that results in code that says one thing when your intention is something else—a confusion you should remove.

By using delegation instead, you make it clear that you are making only partial use of the delegated class. You control which aspects of the interface to take and which to ignore. The cost is extra delegating methods that are boring to write but are too simple to go wrong.

Mechanics

- Create a field in the subclass that refers to an instance of the superclass. Initialize it to `this.`
- Change each method defined in the subclass to use the delegate field. Compile and test after changing each method.

  ?rarr; *You won't be able to replace any methods that invoke a method on* `super` *that is defined on the subclass, or they may get into an infinite recurse. These methods can be replaced only after you have broken the inheritance.*

- Remove the subclass declaration and replace the delegate assignment with an assignment to a new object.
- For each superclass method used by a client, add a simple delegating method.
- Compile and test.

Example

One of the classic examples of inappropriate inheritance is making a stack a subclass of vector. Java 1.1 does this in its utilities (naughty boys!), but in this case I use a simplified form of stack:

```
class MyStack extends Vector {

   public void push(Object element) {
       insertElementAt(element,0);
   }

   public Object pop() {
       Object result = firstElement();
       removeElementAt(0);
       return result;
   }
}
```

Looking at the users of the class, I realize that clients do only four things with stack: push, pop, size, and `isEmpty.` The latter two are inherited from Vector.

I begin the delegation by creating a field for the delegated vector. I link this field to `this` so that I can mix delegation and inheritance while I carry out the refactoring:

```
  private Vector _vector = this;
```

Now I start replacing methods to get them to use the delegation. I begin with push:

```
      public void push(Object element) {
        _vector.insertElementAt(element,0);
    }
```

I can compile and test here, and everything will still work. Now pop:

```
      public Object pop() {
        Object result = _vector.firstElement();
        _vector.removeElementAt(0);
        return result;
    }
```

Once I've completed these subclass methods, I need to break the link to the superclass:

```
 class MyStack extends Vector

   private Vector _vector = new Vector();
```

I then add simple delegating methods for superclass methods used by clients:

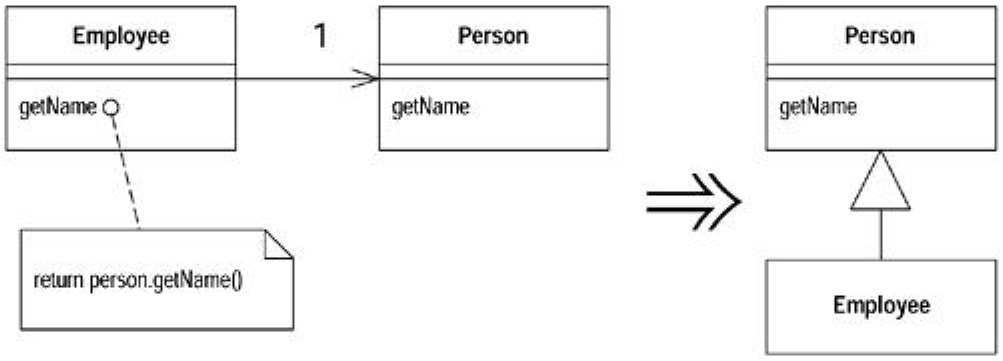
```
   public int size() {
       return _vector.size();
   }
   public boolean isEmpty() {
       return _vector.isEmpty();
   }
```

Now I can compile and test. If I forgot to add a delegating method, the compilation will tell me.

Replace Delegation with Inheritance

You're using delegation and are often writing many simple delegations for the entire interface.

Make the delegating class a subclass of the delegate.



Motivation

This is the flip side of Replace Delegation with Inheritance. If you find yourself using all the methods of the delegate and are sick of writing all those simple delegating methods, you can switch back to inheritance pretty easily.

There are a couple of caveats to bear in mind. If you aren't using all the methods of the class to which you are delegating, you shouldn't use *Replace Delegation with Inheritance,* because a subclass should always follow the interface of the superclass. If the delegating methods are tiresome, you have other options. You can let the clients call the delegate themselves with Remove Middle Man. You can use Extract Superclass to separate the common interface and then inherit from the new class. You can use Extract Interface in a similar way.

Another situation to beware of is that in which the delegate is shared by more than one object and is mutable. In this case you can't replace the delegate with inheritance because you'll no longer share the data. Data sharing is a responsibility that cannot be transferred back to inheritance. When the object is immutable, data sharing is not a problem, because you can just copy and nobody can tell.

Mechanics

- Make the delegating object a subclass of the delegate.
- Compile.

> ?rarr; *You may get some method clashes at this point; methods may have the same name but vary in return type, exceptions, or visibility. Use* Rename Method *to fix these.*

- Set the delegate field to be the object itself.
- Remove the simple delegation methods.
- Compile and test.
- Replace all other delegations with calls to the object itself.
- Remove the delegate field.

Example

A simple employee delegates to a simple person:

```
class Employee {
  Person _person = new Person();

  public String getName() {
      return _person.getName();
  }
  public void setName(String arg) {
       _person.setName(arg);
  }
  public String toString () {
      return "Emp: " + _person.getLastName();
  }
}

class Person {
  String _name;

  public String getName() {
      return _name;
  }
  public void setName(String arg) {
      _name = arg;
  }
  public String getLastName() {
      return _name.substring(_name.lastIndexOf(' ')+1);
  }
}
```

The first step is just to declare the subclass:

```
class Employee extends Person
```

Compiling at this point alerts me to any method clashes. These occur if methods with the name have different return types or throw different exceptions. Any such problems need to be fixed with Rename Method. This simple example is free of such encumbrances.

The next step is to make the delegate field refer to the object itself. I must remove all simple delegation methods such as `getName` and `setName.` If I leave any in, I will get a stack overflow error caused by infinite recursion. In this case this means removing `getName` and `setName` from `Employee.`

Once I've got the class working, I can change the methods that use the delegate methods. I switch them to use calls directly:

```
  public String toString () {
       return "Emp: " + getLastName();
```

```
    }
```

Once I've got rid of all methods that use delegate methods, I can get rid of the `_person` field.