Simplifying Generalization Relationships

With the generalization methods we want to introduce order and clarity in class hierarchies. There are methods that deal with moving class features in hierarchy to the subclass or to the super class (pull up/down attributes, pull up/down method), there is a method that takes care of standardizing the constructor and extracts it on the super class (pull up constructor body), and another very interesting method that helps us to implement the template method pattern of “Gang of Four” [GOF], which is useful when we have methods that perform the same steps but the steps are different.

We will see how to extract subclasses or super classes, useful when the single class tends to encapsulate different behaviors in different instances, or to help us understand how to collapse a hierarchy when useless.

Using inheritance in object-oriented programming is a good habit, but sometimes it can create confusion—for example, when some super class methods or parameters do not represent their subclasses, and it would be better to implement a delegation. In this chapter we will also see how we can replace inheritance with delegation or replace the delegation with inheritance.

The collection of refactoring techniques that we’ll present in this chapter was created by Martin Fowler [FOW01]. For each method, we’ll see the motivation and situation for using it, the mechanism that will explain how we can apply the method to our existing code, and some examples of how each method works in a real-world case.

Pull Up Field

**Problem:** “Two subclasses have the same field.”

**Solution:** “Move the field to the super class.”

**Motivation**

When we find two classes that extend the same super class having the same attributes, we can move these duplicated attributes in the super class.

It may happen that the same attributes have different names. In this case, check that the attributes are equal, and that they are used in the same way. If we are sure, we can rename the attribute in one of two classes, move it to the super class and remove them from their subclasses.
Mechanism

- Write unit tests to confirm that the behavior doesn't change after refactoring.
- Find similar attributes.
- Check that they are used in the same way.
- If attributes do not have the same name, rename one of the two with the same name of the other.
- Run tests and make sure everything still works properly.
- Create a new attribute in the super class with protected visibility so that the subclasses can access it.
- Delete the attributes in the subclasses.
- Run tests and make sure everything still works properly.

Example

After some refactoring steps, we have two classes—User and Admin—that extend the BaseUser class. These two classes were initially independent, and therefore both have the same attribute $username.

class User extends BaseUser
{
    public $username;

    public function getUsernameWithLabel()
    {
        return 'User: '.$this->username;
    }
}

class Admin extends BaseUser
{
    public $username;

    public function getUsernameWithLabel()
    {
        return 'Admin: '.$this->username;
    }
}

Before moving the common attribute in the super class, we write a unit test that preserves the proper class behavior.

class UserTest extends PHPUnit_Framework_TestCase
{
    public function testGetUsername()
    {
        $admin = new Admin();
        $admin->username = 'cphp';
        $this->assertEquals('Admin: cphp', $admin->getUsernameWithLabel());

        $user = new User();