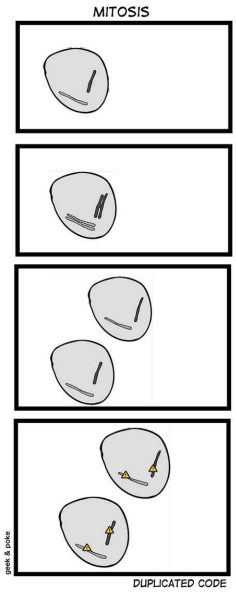


Duplication

www.cs.uoi.gr/~zarras/soft-devII.htm

Sources: M. Fowler Refactorings Catalog
W. Wake Refactoring Workbook

Duplication

Duplication

Symptoms

The easy version: Two fragments of code look nearly identical.

The hard version: Two fragments of code have nearly identical effects (at any conceptual level).

Causes

Some duplication occurs because programmers were working independently in different parts of the system, and they didn't realize that they were creating almost identical code.

A **worse case (but perhaps the most common)** occurs when the programmers intentionally duplicate code. They find some code that is "almost" right, so they copy-and-paste it into the new spot with some slight alterations.

Duplication

What to Do

If the duplication occurs because a **special number**, **string,** or other **value recurs**, use ***Replace Magic Number with Symbolic Constant***.

Replace Magic Number with Symbolic Constant

You have a literal number with a particular meaning.

Create a constant, name it after the meaning, and replace the number with it.

```
double potentialEnergy(double mass, double height) {
  return mass * height * 9.81;
}
```

⇓

```
double potentialEnergy(double mass, double height) {
  return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Duplication

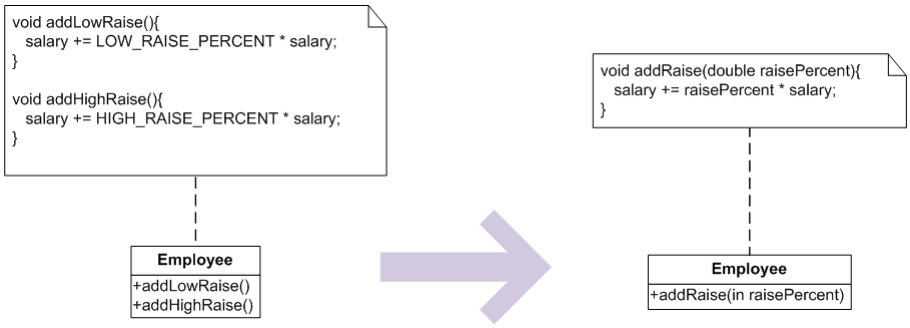
What to Do

If the duplication is within a **method** or in two **different methods** in the same class: use ***Extract Method*** *and* ***Parameterize Method*** *to pull the common/different part out into separate methods*.

Parameterize Method

Several methods do similar things but with different values contained in the method body.

Create one method that uses a parameter for the different values.



Duplication

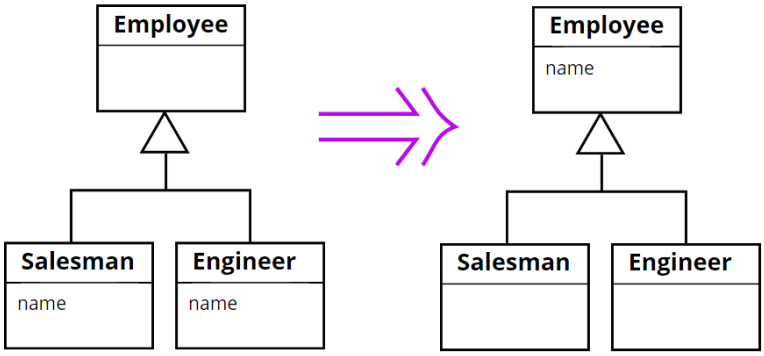
What to Do

If the duplication is within **two sibling classes**: use *Extract Method* and *Parameterize Method* to create a single routine, then ***Pull Up Field and/or Pull Up Method*** to bring the common parts together.

Pull Up Field

Two subclasses have the same field.

Move the field to the superclass.



Pull Up Field

If subclasses are developed independently, or combined through refactoring, you often find that they duplicate features.

*** In particular, certain fields can be duplicates. Such fields sometimes have **similar names but not always**. The **only way** to determine what is going on is to look at the fields and see how they are used by other methods. If they are being used in a similar way, you can generalize them.

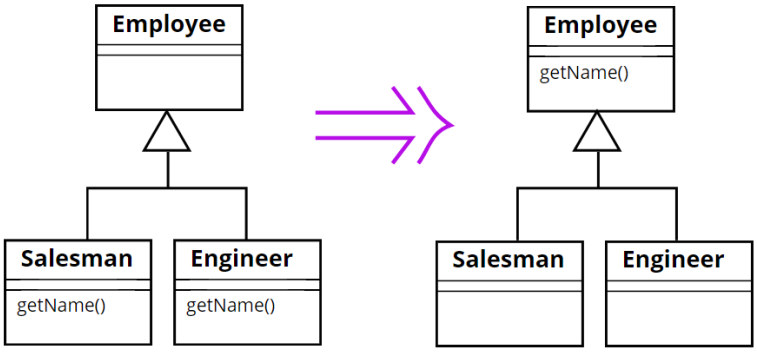
Doing this **reduces duplication in two ways**. It removes the **duplicate data** declaration and allows you to move from the subclasses to the superclass **methods or parts of them that use the field**.

*If the fields are **private**, you will need to **protect the superclass** field so that the subclasses can refer to it.*

Pull Up Method

You have methods with identical results on subclasses.

Move them to the superclass.



Pull Up Method

Eliminating duplicate behavior is important. Whenever there is duplication, you face the risk that an alteration to one will not be made to the other.

The easiest case of using *Pull Up Method occurs when* ***the methods have the same body***, implying there's been a copy and paste. You may need to change a method's signature to get this to work.

Of course it's not always as obvious as that. Often *Pull Up Method comes after other steps. You see two methods in different classes that* ***can be parameterized*** in such a way that they **end up as essentially the same method**. In that case the smallest step is to parameterize each method separately and then generalize them.

*** The most awkward element of *Pull Up Method is that the body of the methods may* ***refer to features that are on the subclass but not on the superclass***. If the feature is a method, you can either generalize the other method or create an abstract method in the superclass.

Duplication

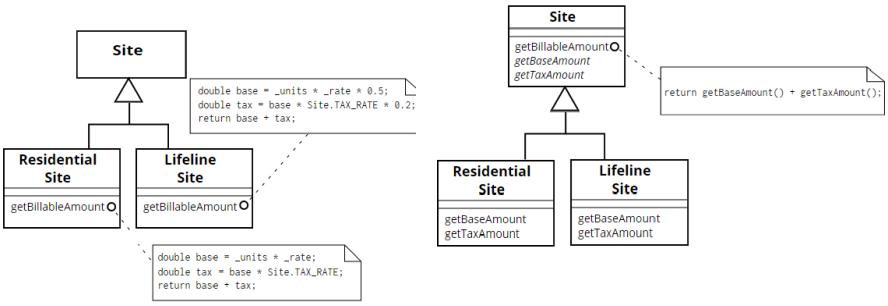
What to Do

Then you may be able to use ***Form Template Method*** *to create a* common algorithm in the parent, and unique steps in the children.

Form Template Method

You have two methods in subclasses that perform similar steps in the same order, yet the steps are different.

Get the steps into methods with the same signature, so that the original metho become the same. Then you can pull them up.



Form Template Method

Inheritance is a powerful tool for eliminating duplicate behavior. Whenever we see two similar methods in subclasses, we want to bring them together in a superclass.

A common case is **two methods that seem to carry out broadly similar steps in the same sequence, but the steps are not the same**. In this case we can **move the sequence to the superclass and allow polymorphism to play its role in ensuring the different steps do their things differently**. This kind of method is called a *template method*

Duplication

What to Do

If the duplication is in **two unrelated classes**:

1. either **extract the common part** into a new class via ***Extract Class***, *or*
2. *Check if* ***the*** **common code really belongs on only one class or the other**.

In any of these cases, you may find that the two places aren't literally identical but have the same effect. Then you may do a ***Substitute Algorithm*** *so that only one* copy is involved.

Payoff

Reduces duplication, lowers size. Can lead to better abstractions and more flexible code.

Substitute Algorithm

You want to replace an algorithm with one that is clearer.

Replace the body of the method with the new algorithm.

```
String foundPerson(String[] people){
  for (int i = 0; i < people.length; i++) {
    if (people[i].equals ("Don")){
      return "Don";
    }
    if (people[i].equals ("John")){
      return "John";
    }
    if (people[i].equals ("Kent")){
      return "Kent";
    }
  }
  return "";
}
```

```
String foundPerson(String[] people){
  List candidates = Arrays.asList(new String[] {"Don", "John", "Kent"});
  for (int i=0; i<people.length; i++)
    if (candidates.contains(people[i]))
      return people[i];
  return "";
}
```