Coping with Requirement Change

One thing is constant which is Requirements are bound to change.

* Let’s say farmer told you that can you give me apples of Green color. That is easy. Loop through the List<Apple> and search for the green color. Below is the code.

**public** **static** List<Apple> filterApples(**final** List<Apple> apples) {

List<Apple> greenApples = **new** ArrayList<Apple>();

**for** (Apple apple : apples) {

**if** (apple.getColor().equalsIgnoreCase("Green")) {

greenApples.add(apple);

}

}

**return** greenApples;

}

* Next day farmer comes to you and tells you that can you find me red apples? You know that previously you hard-coded the values. So there is small change. You will just set one parameter as color and act upon it.

**public** **static** List<Apple> filterApples(**final** List<Apple> apples, **final** String color) {

List<Apple> result = **new** ArrayList<Apple>();

**for** (Apple apple : apples) {

**if** (apple.getColor().equalsIgnoreCase(color)) {

result.add(apple);

}

}

**return** result;

}

* Everything is good. Farmer is happy & hopefully you are happy too. Now farmer comes again and asks you can you provide me apples whose weights is greater than 50 grams (I really don’t know how much apple weights, just putting some number ☺). You know exactly what you have to do but this time you will not hard code the values as again you have to change the values. So you come up with a solution.

**public** **static** List<Apple> filterApplesByWeight(**final** List<Apple> apples, **final** **int** weight) {

List<Apple> result = **new** ArrayList<Apple>();

**for** (Apple apple : apples) {

**if** (apple.getWeight() > 50) {

result.add(apple);

}

}

**return** result;

}

* So far so good. Everything works out. But now real challenge comes. You designed 3 methods and see how code is duplicated. The for-each loop if condition and adding apples in result list. As a good programmer you think of DRY principle i.e. don’t repeat yourself. So you think of a solution.

/\*\*

\* flag denotes presence of color.

\* flag = true means color comparison

\* flag = false means weight comparison

\* \*/

**public** **static** List<Apple> filterApplesByColorAndWeight(**final** List<Apple> apples, **final** String color, **final** **int** weight, **boolean** flag){

List<Apple> result = **new** ArrayList<Apple>();

**for** (Apple apple : apples) {

**if** ( (flag && apple.getColor().equalsIgnoreCase(color))

|| (!flag && apple.getWeight() > weight) ) {

result.add(apple);

}

}

**return** result;

}

You are proud that you solved a problem. But you unnecessary made this problem complex. The flag is a clutter and it is hard to debug in case it fails.

So, how to solve this problem? Well we will see that in our next post called Behavior Parameterization.