# Design Pattern: Liskov's Substitution Principle (LSP)

As a java developper, I've never heard of the LSP pattern. It was only when I read some stuff about C++ that I encountered this pattern. It's very strange because this pattern is sometimes seen as **one of the 5 principles of object-oriented programming.**

This principle was first introduced by Barbara Liskov in 1987 and formulated in 1994 as:

> *"Let q(x) be a property provable about objects x of type T. Then q(y) should be provable for objects y of type S,where S is a subtype of T."*

In other words:

- if a class B is a subclass of a class A,
- if A has a method f(),
- if b is an instance of B and a an instance of A,
- then in all the parts of the code using "a.f()" should be able to use" b.f()" **without modifying the behavior of the code**

Let's have a look of an example of inheritance that doesn't respect the LSP:

The result of this code is:

> *The second element is :4*
> *The second element is :8*

**Why is this a problem?**

MyOrderedAndSortedCollection is ordered so derivate it from MyOrderedCollection seems a nice idea. But since both classes don't use the same order it could lead to big problems:

- MyOrderedCollection use the insertion ordering
- MyOrderedAndSortedCollection use a natural ordering

Suppose devA writes and uses MyOrderedCollection. Two years after, devB creates MyOrderedAndSortedCollection from MyOrderedCollection because he needs a sorted collection. Since it's an inheritance, the functions using MyOrderedCollection as parameter can also use MyOrderedAndSortedCollection. But what if some of those functions are using the specific ordering of MyOrderedCollection?
In order to avoid that, devB should look at the full legacy code that uses MyOrderedCollection and modify the legacy code to check if the reference is an instance of MyOrderedAndSortedCollection. It could takes weeks depending on the size/complexity of the legacy code and it migh not be a good idea to modifiy an existing (and working) code.

Here is a possible solution that respects LSP:

With this configuration MyOrderedCollection and MyOrderedAndSortedCollection are not alike (even if they share the same interface):

- A code that uses explicitly MyOrderedCollection uses its ordering and can't be changed with MyOrderedAndSortedCollection.
- A code using MyCollection doesn't care about the ordering so it can use whether MyOrderedCollection or MyOrderedAndSortedCollection.

This pattern can be seen as a **strong Behavioral subtyping**.