Program Language and Design Assignment 3

Divij Singh

17/11/18

The paper addresses a principle in Object Oriented Design, known as the Liskov Substitution Principle.

Put simply, if you have an object in a program, say F, and make an object S that is a subtype of F, then you should be able to swap any instances of F in the program with instances of S without any issues in terms of correctness.

The idea behind the principle is to reduce a program into simple, interchangeable blocks; if you need a feature of one object, you should be able to access it without having to create a unique copy of the object.

While the author uses the example of squares and rectangles, an alternative example could be that of addition and subtraction. (This would also help those unfamiliar with C++ code understand the concept.)

Say we have a function F_1, which accepts two integers, I_1 and I_2, and returns their sum. We want the result of $2 + 3$, so we pass the values $+2$ and $+3$ to a function F_2, which performs the same function for positive integers (and is thus a subtype of F_1). In return, we will receive a value of $+5$.

Now, we want the result of 105. This can of course be re-written as $+10 + (-5)$, which is an addition calculation. However, upon passing the values to F_2, we receive an error! After all, I_1 and I_2 must both be positive integers, much as we would like to pretend that $+(-5)$ qualifies as one.

Now, one possible solution could be to only allow the user to give positive integers. However, that would require informing each and every user of this constraint, which is, to be honest, a rather silly constraint. So instead, we realise we must make our F_2 more robust.

So, we change the code to accept any two integers, I_1 and I_2, and return their sum. This allows us to complete our previous calculation, giving us the answer of $10 - 5 = 5$. Thus the requirement for the principle is met.

In addition to this signature characteristic, F_2 must also behave in a similar

manner to F_1.

If we were to pass three values to F_2, it should throw an error, as F_1 is unable to handle three values as well. This is known as trying to strengthen a precondition.

If we pass two numbers to F_2, we should only receive one number in return, not more nor less. This is known as trying to weaken a postcondition.

In order to satisfy this principle, any object and its subtypes must meet the above conditions. It is essentially the expected behaviour of the object that is under consideration. If any F_1 is replaced by a subtype F_2, the replacement must behave in the same manner as F_1 in order for the principle to hold.