The Liskov Substitution Principle

Prof. Dr. Eric Dubuis
Berner Fachhochschule, Engineering and Information Technology @ Biel

Course "UML and Design Patterns" of module "Software Engineering and Design", version March 2008
Session's Goals

You'll learn:

- “IS-A” inheritance relation is not always good enough
- It is the *client view* that determines whether a given class hierarchy is a good one
- The ultimate criterion is for a good class hierarchy is:

  The *Liskov Substitution Principle*¹

Readings


---

The Liskov Substitution Principle (LSP)

The LSP simplified:

Subtypes must be substitutable for their base types.

Barbara Liskov original wordings:¹

“What is wanted here is something like the following substitution property:

If

for each object \( o_1 \) of type \( S \) there is an object \( o_2 \) of type \( T \) such that for all programs \( P \) defined in terms \( T \), the behavior of \( P \) is unchanged when \( o_1 \) is substituted for \( o_2 \)

then

\( S \) is a subtype of \( T \).

Rectangles and Squares, a (Subtle) Violation of LSP

Assume a program that deals with Rectangle objects:

```java
1 public class Rectangle {
2   private int width, height;
3
4   public Rectangle(int width, int height) {
5       this.width = width;
6       this.height = height;
7   }
8   // getter and setters not shown...
9
10  public int getArea() {
11      return this.getHeight() * this.getWidth();
12  }
13}
```
A More Subtle Violation of LSP (cont'd)

Consider a piece of client code:

```java
// in some class, say Client:
public static void someMethodUsingGetArea(Rectangle r) {
  // ...
}
```

So far, so good.
Adding Square as an Extension of Rectangle

Let's introduce a Square class:

```java
1 public class Square extends Rectangle {
2
3
4
5}
```

Clearly, the above class breaks the following test:

```java
1 @Test
2 public void testSquareness() {
3     Square s = new Square(5);
4     s.setHeight(10);
5     assertTrue(s.getWidth() == 10);
6 }
```
Fixing the “Squareness” Problem

It's easy to fix the “squareness” problem:

```java
1 public class Square extends Rectangle {
2     // as above
3
4     public void setWidth(int w) {
5
6         }
7     }
8
9     public void setHeight(int h) {
10
11         }
12 }
13}
```
The Real Problem

... occurs when using method `someMethodUsingGetArea(Rectangle)` in conjunction with a `Square` object:

```
1 Rectangle r = new Square(...);
2
3 Client.someMethodUsingGetArea(r); // Allowed since square s IS-A rectangle!
```

However, method `someMethodUsingGetArea(Rectangle)` fails!
Discussion

- Where lies the problem?
  - in Rectangle?
  - in someMethodUsingGetArea()?
  - in Square?

- Validity is not intrinsic!

- “IS-A” relationship breaks in someMethodUsingGetArea(Rectangle) when providing a Square object as argument.
Post-Conditions for `setWidth()`

- of `Rectangle.setWidth(int newW)`:  
  - R1:
  - R2:

- of `Square.setWidth(int newW)`:  
  - S1 (= R1):
  - S2:
Design by Contract¹

When overwriting (redeclaring) methods in extensions, the rules for pre- and post-conditions can be stated by B. Meyer as follows:

A routine redeclaration [in a derivative] may only replace the original pre-condition by one equal or weaker, and the original post-condition by one equal or stronger.²

Specifying Contracts in Unit Tests

Unit tests can take the role of specifying contracts. Users of a class may review the unit tests in order to know what to reasonably assume about the class they are using.

---