Liskov Substitution Principle

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhán Drohan (sdrohan@wit.ie)

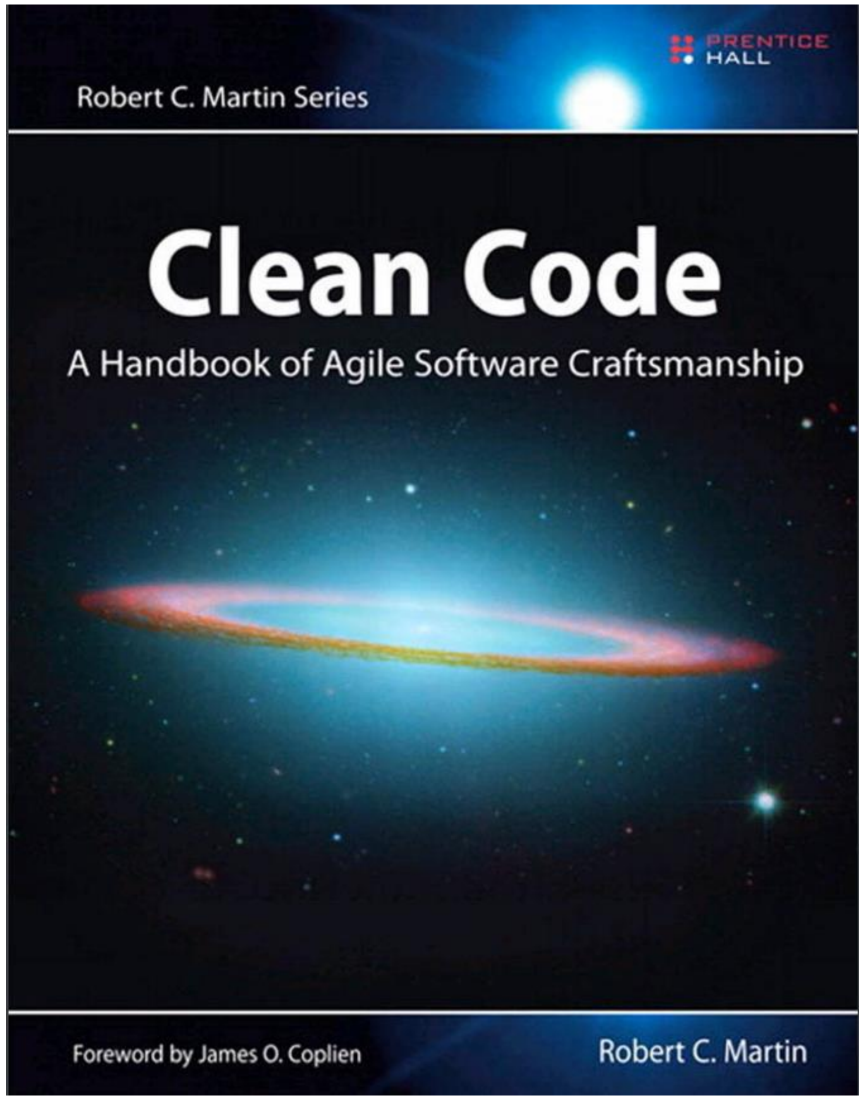
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

SOLID Class Design Principles

In this talk, we will refer to the SOLID principles examples in this book and also this website.

SOLID → five principles for object-oriented class design i.e. best guidelines for building a maintainable object-oriented system.


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

SOLID Class Design Principles

S	Single Responsibility Principle (SRP). Classes should have one, and only one, reason to change. Keep your classes small and single-purposed.
O	Open-Closed Principle (OCP). Design classes to be open for extension but closed for modification; you should be able to extend a class without modifying it. Minimize the need to make changes to existing classes.
L	**Liskov Substitution Principle (LSP). Subtypes should be substitutable for their base types. From a client's perspective, override methods shouldn't break functionality.**
I	Interface Segregation Principle (ISP). Clients should not be forced to depend on methods they don't use. Split a larger interface into a number of smaller interfaces.
D	Dependency Inversion Principle (DIP). High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.

Barbara Liskov

COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

07/2009 VOL.52 NO.07

Barbara Liskov
ACM's A.M. Turing Award Winner

Steps Toward Self-Aware Networks

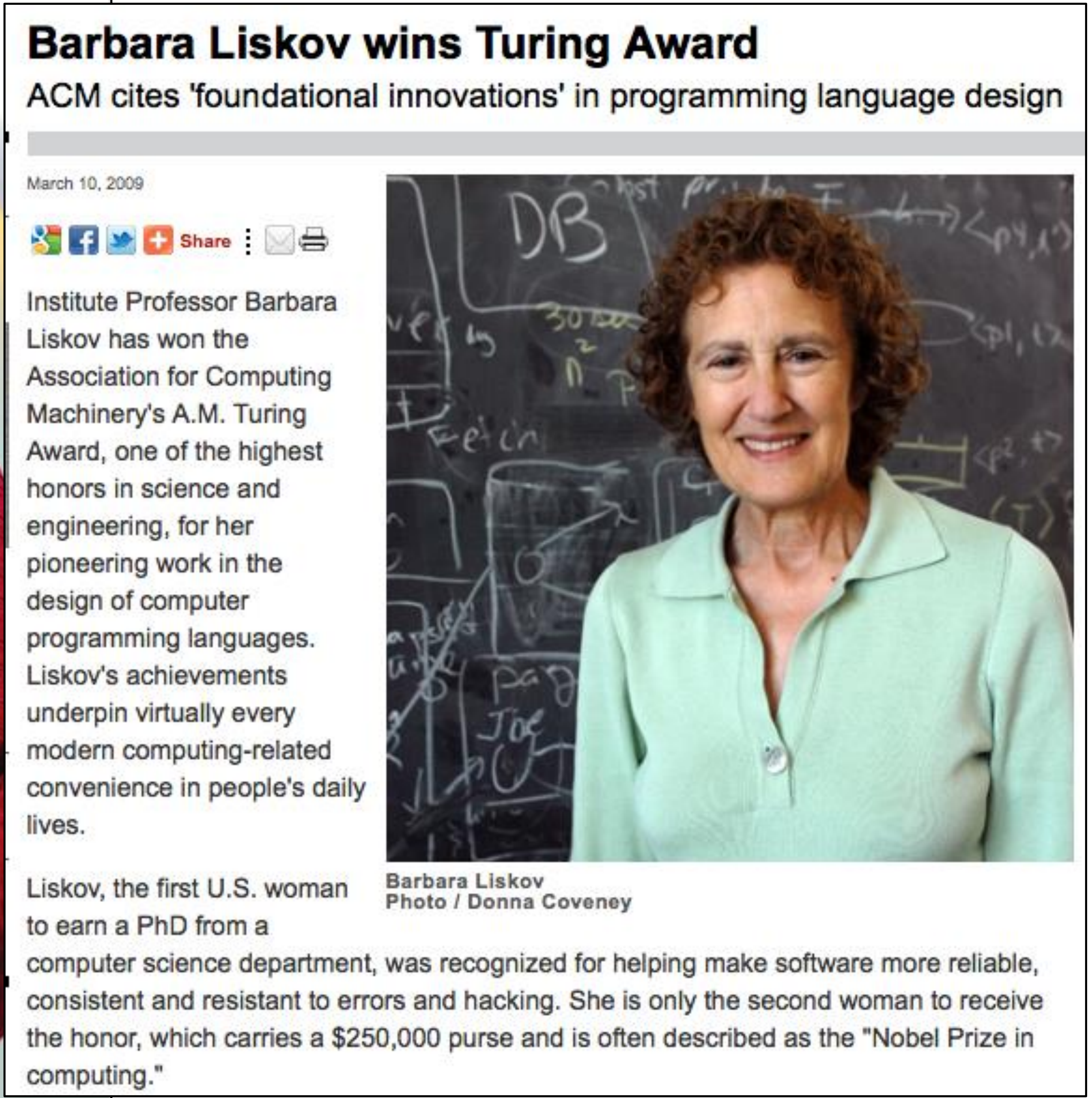
The Metropolis Model

Why Computer Science Doesn't Matter

Probabilistic Databases

The Five-Minute Rule 20 Years Later

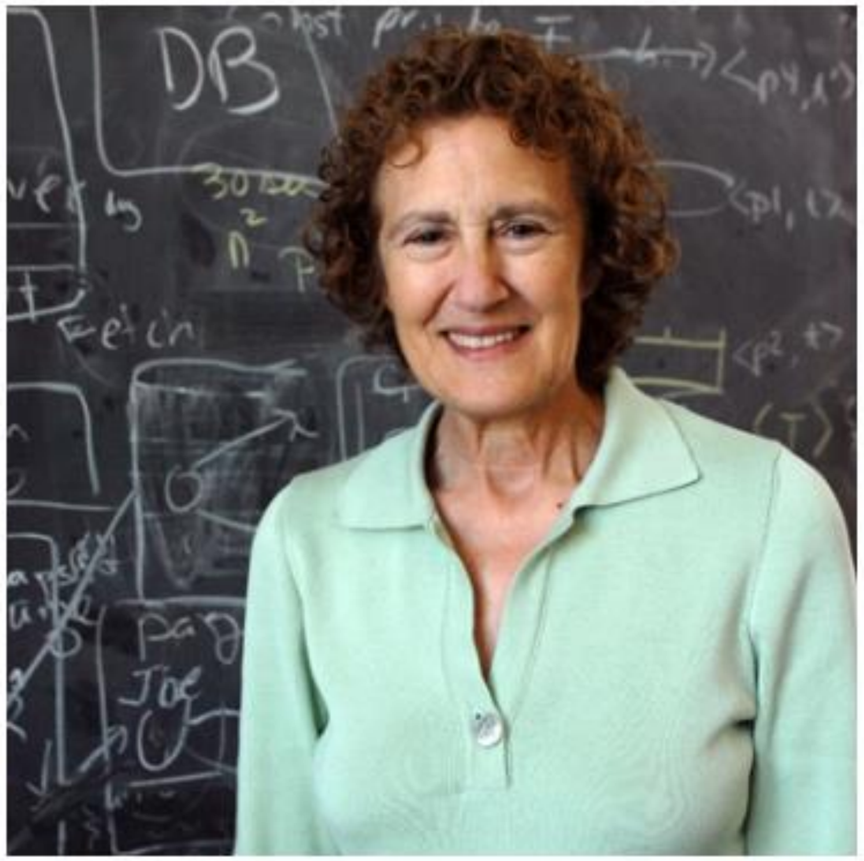
Association for Computing Machinery
acm


Barbara Liskov wins Turing Award

ACM cites 'foundational innovations' in programming language design

March 10, 2009

Institute Professor Barbara Liskov has won the Association for Computing Machinery's A.M. Turing Award, one of the highest honors in science and engineering, for her pioneering work in the design of computer programming languages. Liskov's achievements underpin virtually every modern computing-related convenience in people's daily lives.


Barbara Liskov
Photo / Donna Coveney

Liskov, the first U.S. woman to earn a PhD from a computer science department, was recognized for helping make software more reliable, consistent and resistant to errors and hacking. She is only the second woman to receive the honor, which carries a $250,000 purse and is often described as the "Nobel Prize in computing."

✓ WEAR IT ON YOUR FACE
✓ BLOCKS THE SUN
✗ ~~PROTECTS~~ STINGS YOUR EYES
Suntan lotion
PROBABLY THE WRONG ABSTRACTION
LISKOV SUBSTITUTION PRINCIPLE

https://www.novoda.com/blog/designing-something-solid/

sunglasses interface would have fairly simple rules like:

- shields from the sun;
- attaches to a face.

Implementing the ***sunglasses*** interface with ***suntan lotion*** would seem to make sense:

- it shields from the sun and attaches to the face.

✓ WEAR IT ON YOUR FACE
✓ BLOCKS THE SUN
✗ ~~PROTECTS~~ STINGS YOUR EYES
Suntan Lotion
PROBABLY THE WRONG ABSTRACTION
LISKOV SUBSTITUTION PRINCIPLE

https://www.novoda.com/blog/designing-something-solid/

sunglasses interface would have fairly simple rules like:

- shields from the sun;
- attaches to a face.

Implementing the ***sunglasses*** interface with ***suntan lotion*** would seem to make sense:

- it shields from the sun and attaches to the face.

✓ WEAR IT ON YOUR FACE
✓ BLOCKS THE SUN
✗ ~~PROTECTS~~ STINGS YOUR EYES
Suntan Lotion
PROBABLY THE WRONG ABSTRACTION
LISKOV SUBSTITUTION PRINCIPLE

But **semantically** the expected behaviour is different enough to cause behavioural problems - in this case, by stinging your eyes!

https://www.novoda.com/blog/designing-something-solid/

“Don't implement interfaces in a way that breaks expected semantic behaviour.”

sunglasses interface would have fairly simple rules like:

- shields from the sun;
- attaches to a face.

Implementing the ***sunglasses*** interface with ***suntan lotion*** would seem to make sense:

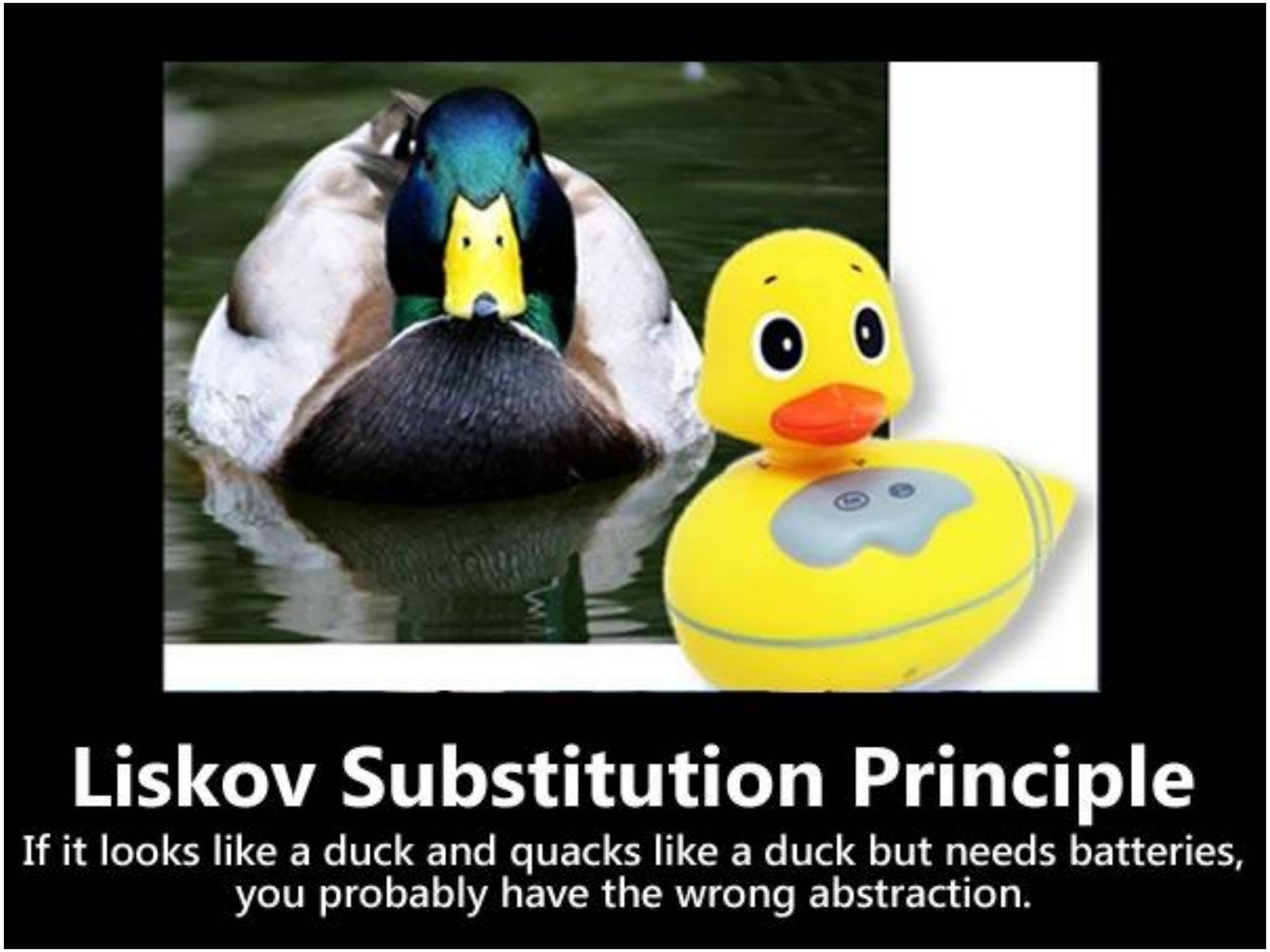
- it shields from the sun and attaches to the face.

✓ WEAR IT ON YOUR FACE
✓ BLOCKS THE SUN
✗ ~~PROTECTS~~ STINGS YOUR EYES
Suntan Lotion
PROBABLY THE WRONG ABSTRACTION
LISKOV SUBSTITUTION PRINCIPLE

But **semantically** the expected behaviour is different enough to cause behavioural problems - in this case, by stinging your eyes!

https://www.novoda.com/blog/designing-something-solid/

“Don't implement interfaces in a way that breaks expected semantic behaviour.”

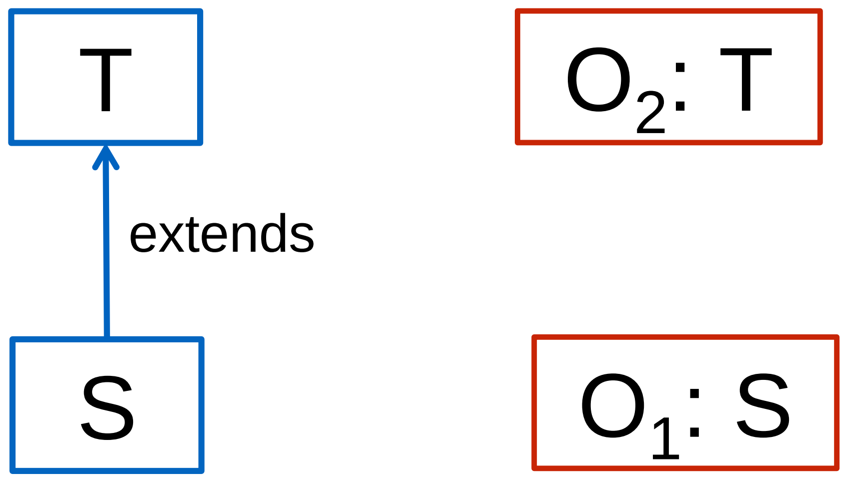

Liskov Substitution Principle
If it looks like a duck and quacks like a duck but needs batteries, you probably have the wrong abstraction.

LSP – Formal Definition

- Methods that refer to base classes must be able to use objects of derived types without knowing it.

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 of T, the behaviour of P is unchanged when o_1 is substituted for o_2 then S is a subtype of T.

Barbara Liskov, “Data Abstraction and Hierarchy,” *SIGPLAN Notices,* 23,5 (May, 1988).



LSP: Simple Violation (and fix)

Simple Violation of LSP

references a base type Shape

violates LSP because it must know of every derived type of Shape.

```
void drawShape (Shape shape)
{
  if (shape instanceof Square)
  {
    drawSquare ((Square)shape);
  }
  else if (shape instance of Circle)
  {
    drawCircle ((Circle) shape);
  }
}
```

- drawShapes must be <u>modified</u> whenever new derivatives of Shape are presented.   What other SOLID principle does it violate?

Adhering to LSP

```
class Shape
{
  void draw()
  {//…}
}
```

```
class Circle extends Shape
{
  private double itsRadius;
  private Point  itsCenter;
  public void draw()
  { //… }
}
```

```
class Square extends Shape
{
  private double itsSide;
  private Point itsTopLeft;
  public void draw()
  { //… }
}
```

```
  void drawShape (Shape s)
  {
    s.draw();
  }
```

- drawShape now adheres to LSP

LSP: Semantic Violation

LSP

An object inheriting from

a base class, interface,

or other abstraction

must be ***semantically*** *substitutable*

for the original abstraction.

http://www.codemag.com/article/1001061

Rectangle

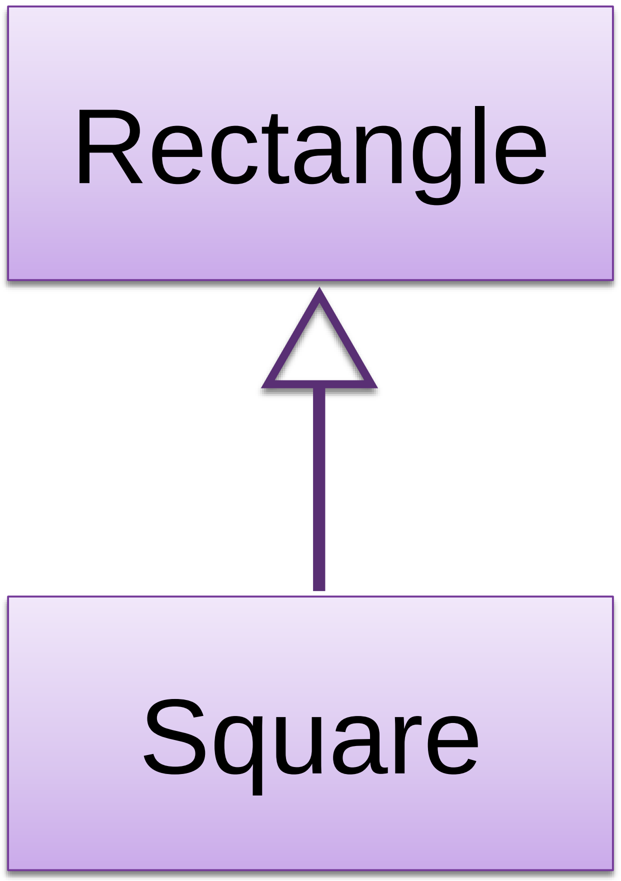
```
class Rectangle
{
  private int width;
  private int height;

  public void setWidth (int width)
  {...}
  public void setHeight (int height)
  {...}
  public int getWidth ()
  {...}
  public int getHeight ()
  {...}
}
```

- Assume the **Rectangle** class is released for general use in the company.

Square

- Introduce Square as a subclass of Rectangle.

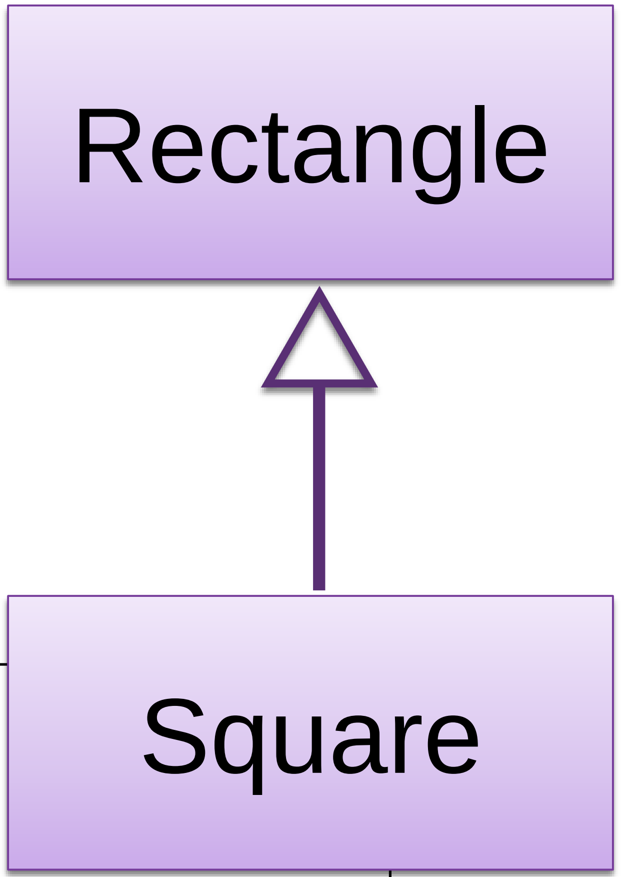
- Inheritance "is a" relationship:
  - A Square is a rectangle.
  - However, there is a subtle difference…it's width and height are equal:
    - Square only needs one dimension but both are inherited.



Square

- For a Square, both setWidth() and setHeight() should not vary independently.

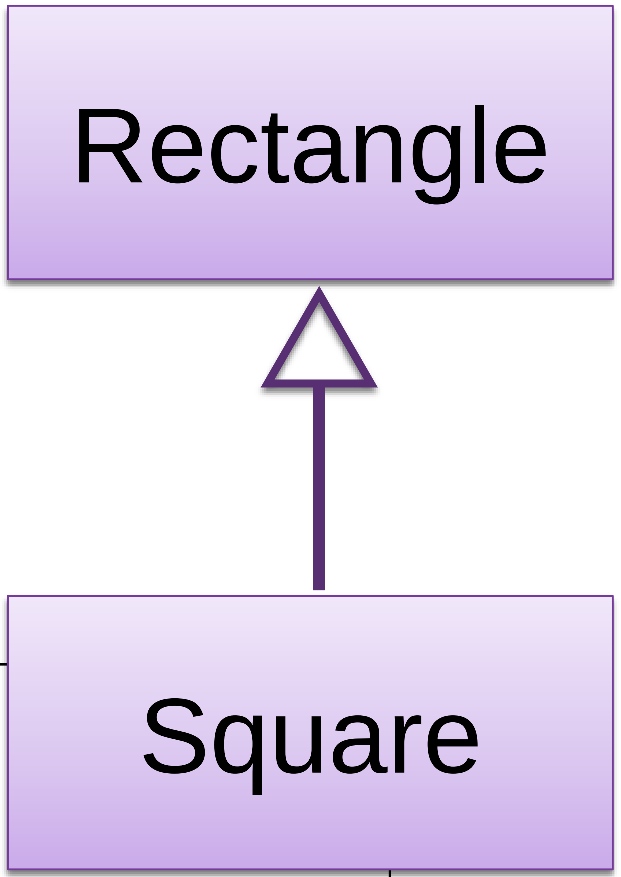
- Client could easily call one and not the other – thus compromising the Square.



```
class Rectangle
{
  private int width;
  private int height;

  public void setWidth (int width)
  {...}
  public void setHeight (int height)
  {...}
  public int getWidth ()
  {...}
  public int getHeight ()
  {...}
}
```

Square

- Potential solution:
  - implement setWidth() and setHeight() in Square class.
- Each of these methods should then make sure both width & height are adjusted.

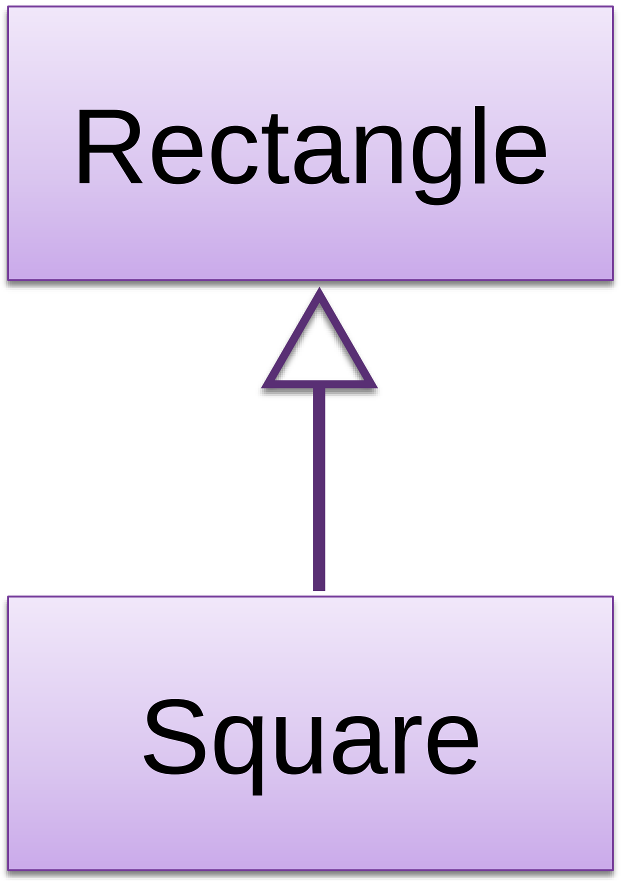


```
class Rectangle
{
  private int width;
  private int height;

  public void setWidth (int width)
  {...}
  public void setHeight (int height)
  {...}
  public int getWidth ()
  {...}
  public int getHeight ()
  {...}
}
```

Square

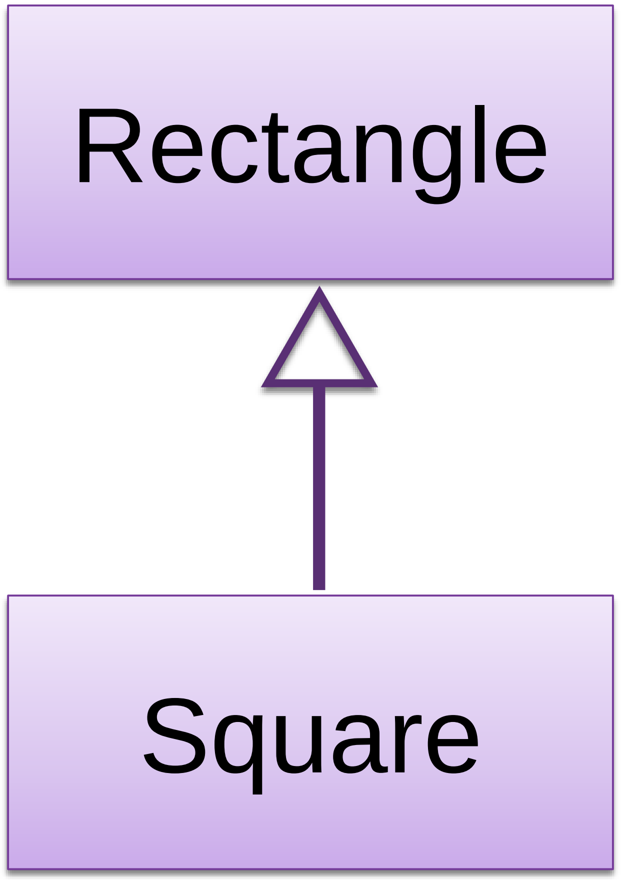
Potential solution implementation:

```
class Square extends Rectangle
{
  public void setWidth (int width)
  {
    super.setWidth(width);
    super.setHeight(width);
  }
  public void setHeight (int height)
  {
    super.setWidth(height);
    super.setHeight(height);
  }
}
```



Polymorphism

```
void f (Rectangle r)
{
   r.setWidth(5);
}
```



- Polymorphism ensures, if the f() method:
  - is passed a Rectangle, then its width will be adjusted.
  - is passed a Square, then both height and width will be changed

- Assume model is consistent & correct.
- However….

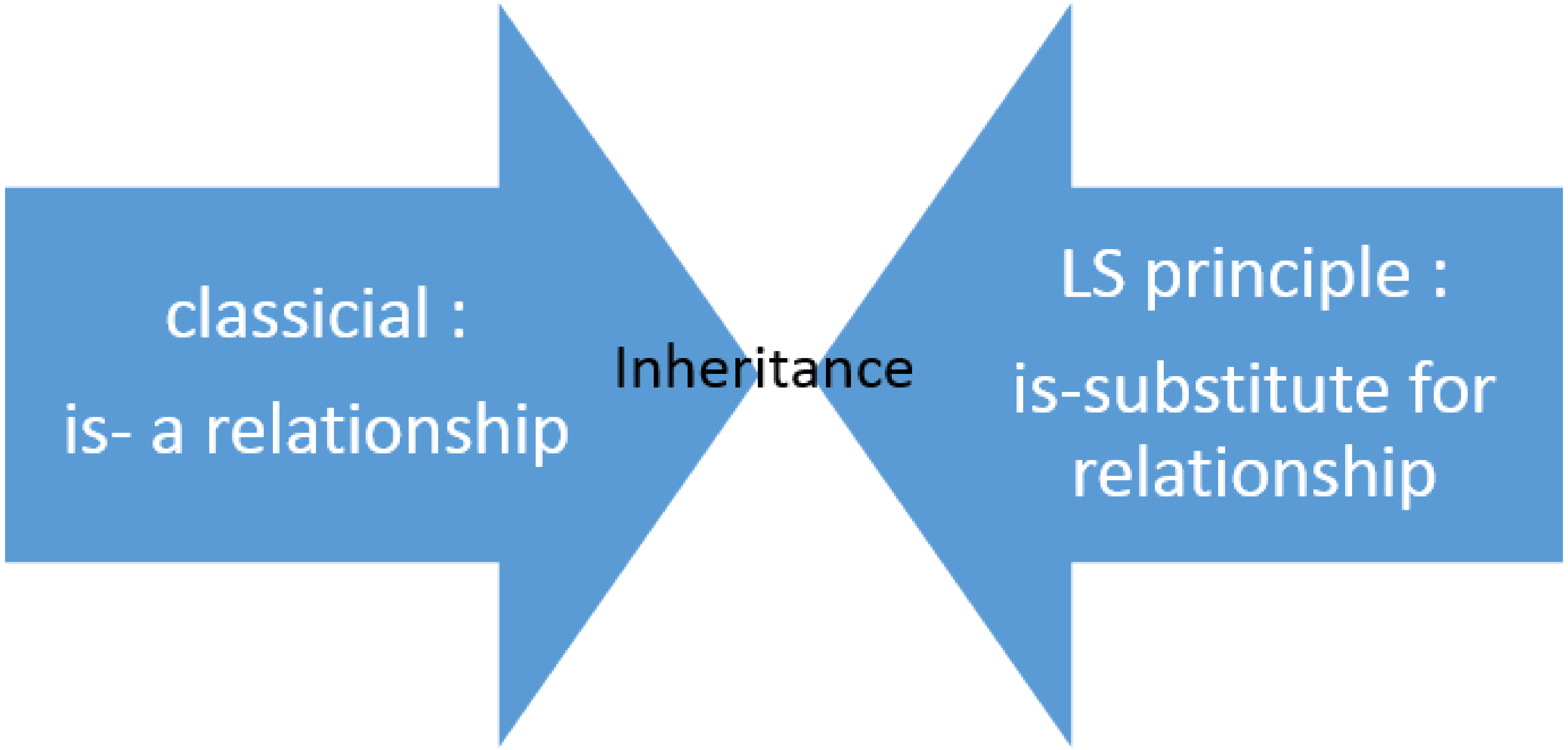
More Subtle Problem

```
void g (Rectangle r)
{
  r.setWidth(5);
  r.setHeight(4);
  assert (r.getWidth() * r.getHeight()) == 20;
}
```

- If r is a Rectangle instance…
  - g() methods works as expected
- If r is a Square instance…
  - g() assertion fails
- g() assumes that width and height of a Rectangle can be varied independently.
- Substitution of a Square violates this semantic assumption.
- Square violates LSP.

LSP: Semantic Violation

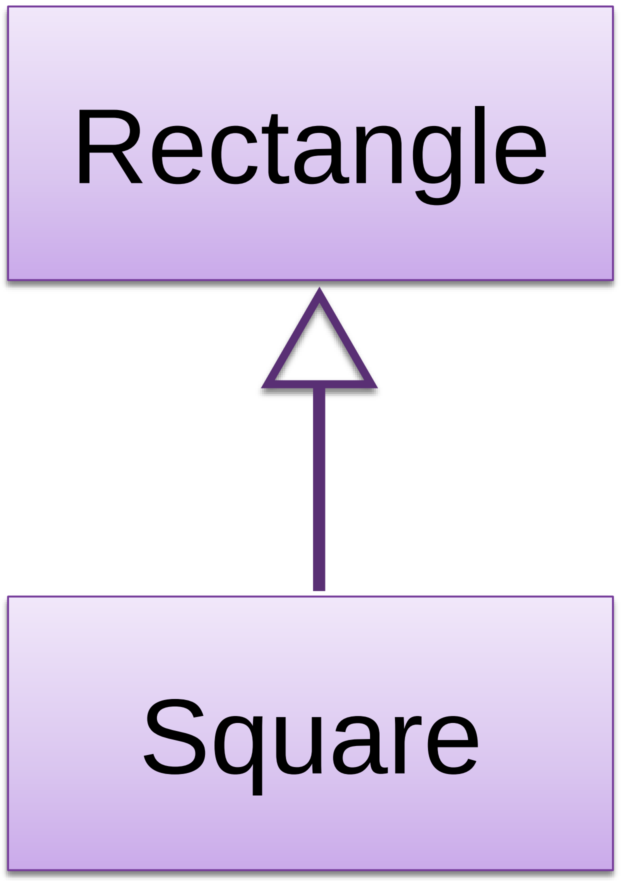
Square != Rectangle



LSP: Subtypes should be substitutable for their base types.

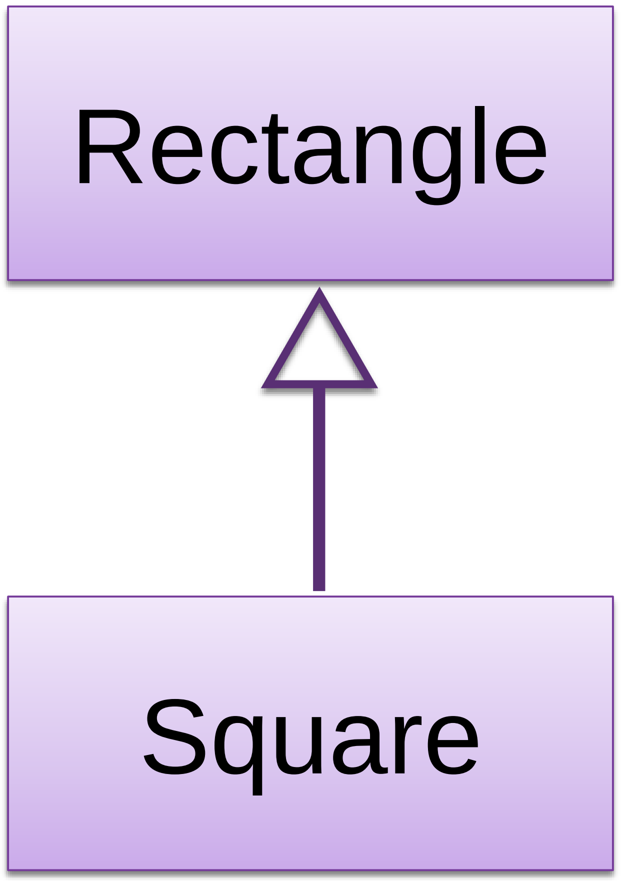
Validating the Model

- A model, viewed in isolation, cannot be meaningfully validated.
- The validity of a model can only be expressed in terms of its clients:
  - Examining the final version of the Square and Rectangle classes in isolation, we found that they were self consistent and valid.
  - When we examined from the viewpoint of g() (which made reasonable assumptions) the model broke down.



Validating the Model

- When considering whether a design is appropriate or not, it must be examined in terms of the ***reasonable assumptions*** that will be made by the users of that design.

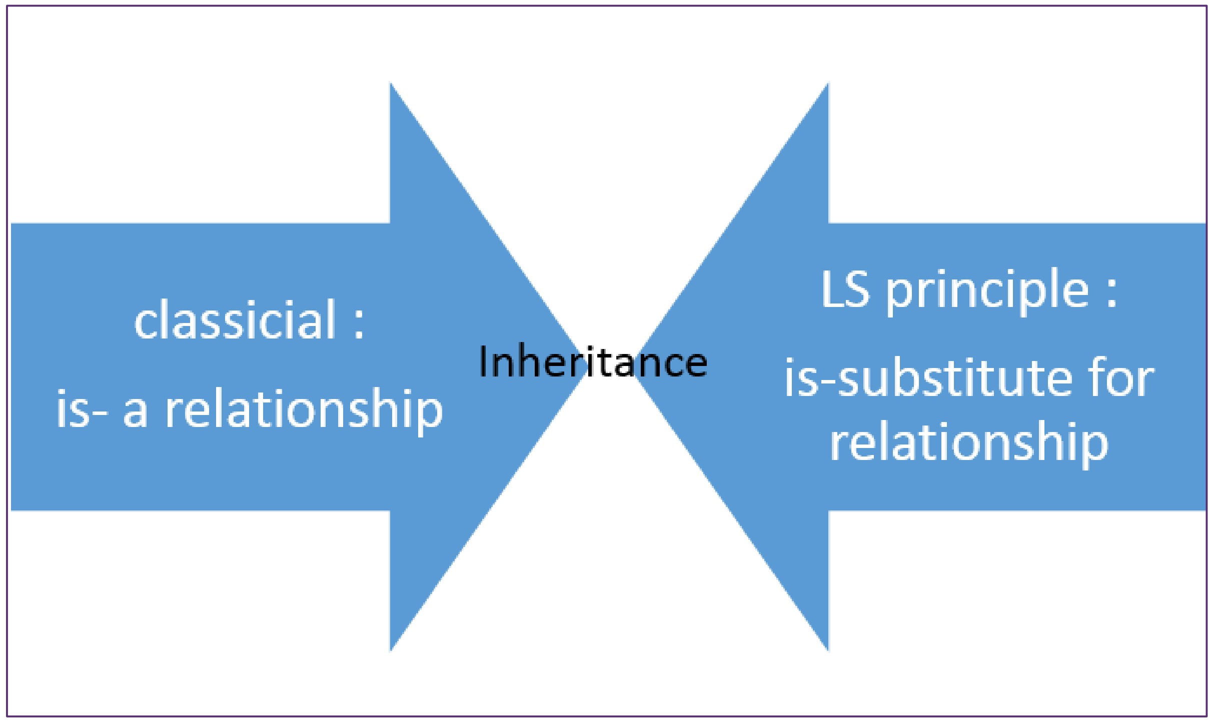


Behavioural Problems

- A square might be a rectangle, but a ***Square object*** is *not* a ***Rectangle object***.

  - the *behaviour* of a Square object is <u>not</u> consistent with the *behaviour* of a Rectangle object.

- The LSP makes clear that the inheritance relationship pertains to *behaviour* that clients depend upon.



Square

!=

Rectangle

With LSP…

“is-a” really means

“behaves exactly like”

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit