# Liskov substitution principle

From Wikipedia, the free encyclopedia

**Substitutability** is a principle in object-oriented programming. It states that, in a computer program, if S is a subtype of T, then objects of type T may be replaced with objects of type S (i.e., objects of type S may *substitute* objects of type T) without altering any of the desirable properties of that program (correctness, task performed, etc.). More formally, the **Liskov substitution principle** (**LSP**) is a particular definition of a subtyping relation, called **(strong) behavioral subtyping**, that was initially introduced by Barbara Liskov in a 1987 conference keynote address entitled *Data abstraction and hierarchy*. It is a semantic rather than merely syntactic relation because it intends to guarantee semantic interoperability of types in a hierarchy, object types in particular. Barbara Liskov and Jeannette Wing formulated the principle succinctly in a 1994 paper as follows:

> *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 the same paper, Liskov and Wing detailed their notion of behavioral subtyping in an extension of Hoare logic, which bears a certain resemblance with Bertrand Meyer's Design by Contract in that it considers the interaction of subtyping with pre- and postconditions.

## Contents



## Principle

Liskov's notion of a behavioral subtype defines a notion of substitutability for mutable objects; that is, if *S* is a subtype of *T*, then objects of type *T* in a program may be replaced with objects of type *S* without altering any of the desirable properties of that program (e.g., correctness).

Behavioral subtyping is a stronger notion than typical subtyping of functions defined in type theory, which relies only on the contravariance of argument types and covariance of the return type. Behavioral subtyping is trivially undecidable in general: if *q* is the property "method for *x* always terminates", then it is impossible for a program (e.g. a compiler) to verify that it holds true for some subtype *S* of *T* even if *q* does hold for *T*. Nonetheless, the principle is useful in

reasoning about the design of class hierarchies.

Liskov's principle imposes some standard requirements on signatures that have been adopted in newer object-oriented programming languages (usually at the level of classes rather than types; see nominal vs. structural subtyping for the distinction):

- Contravariance of method arguments in the subtype.
- Covariance of return types in the subtype.
- No new exceptions should be thrown by methods of the subtype, except where those exceptions are themselves subtypes of exceptions thrown by the methods of the supertype.

In addition to these, there are a number of behavioral conditions that the subtype must meet. These are detailed in a terminology resembling that of design by contract methodology, leading to some restrictions on how contracts can interact with inheritance:

- Preconditions cannot be strengthened in a subtype.
- Postconditions cannot be weakened in a subtype.
- Invariants of the supertype must be preserved in a subtype.
- History constraint (the "history rule"). Objects are regarded as being modifiable only through their methods (encapsulation). Since subtypes may introduce methods that are not present in the supertype, the introduction of these methods may allow state changes in the subtype that are not permissible in the supertype. The history constraint prohibits this. It was the novel element introduced by Liskov and Wing. A violation of this constraint can be exemplified by defining a *mutable point* as a subtype of an *immutable point*. This is a violation of the history constraint, because in the history of the *immutable point*, the state is always the same after creation, so it cannot include the history of a *mutable point* in general. Fields added to the subtype may however be safely modified because they are not observable through the supertype methods. Thus, one may derive *a circle with fixed center but mutable radius* from *immutable point* without violating LSP.

# Origins

The rules on pre- and postconditions are identical to those introduced by Bertrand Meyer in his 1988 book. Both Meyer, and later Pierre America, who was the first to use the term *behavioral subtyping*, gave proof-theoretic definitions of some behavioral subtyping notions, but their definitions did not take into account aliasing that may occur in programming language that supports references or pointers. Taking aliasing into account was the major improvement made by Liskov and Wing (1994), and a key ingredient is the history constraint. Under the definitions of Meyer and America, a MutablePoint would be a behavioral subtype of ImmutablePoint, whereas LSP forbids this.

# A typical violation

A typical example that violates LSP is a Square class that derives from a Rectangle class, assuming getter and setter methods exist for both width and height. The Square class always assumes that the width is equal with the height. If a Square object is used in a context where a Rectangle is expected, unexpected behavior may occur because the dimensions of a Square cannot (or rather should not) be modified independently. This problem cannot be easily fixed: if we can modify the setter methods in the Square class so that they preserve the Square invariant (i.e., keep the dimensions equal), then these methods will weaken (violate) the postconditions for the Rectangle setters, which state that dimensions can be modified independently. Violations of LSP, like this one, may or may not be a problem in practice, depending on the postconditions or invariants that are actually expected by the code that uses classes violating LSP. Mutability is a key issue here. If Square and Rectangle had only getter methods (i.e., they were immutable objects), then no violation of LSP could occur.

# See also

- Refinement
- SOLID: the L in SOLID stands for Liskov substitution principle
- Type signature
- Composition over inheritance

# References

### General references


- Gary T. Leavens and Krishna K. Dhara, *Concepts of Behavioral Subtyping and a Sketch of Their Extension to Component-Bases Systems* in Gary T. Leavens, Murali Sitaraman, (ed.) *Foundations of component-based systems*, Cambridge University Press, 2000 ISBN 0-521-77164-1. This paper surveys various notions of behavioral subtyping, including Liskov and Wing's.
- Liskov, B. H.; Wing, J. M. (November 1994). "A behavioral notion of subtyping". *ACM Trans. Program. Lang. Syst.* **16** (6): 1811–1841. doi:10.1145/197320.197383 (http://dx.doi.org/10.1145%2F197320.197383). An updated version appeared as CMU technical report: Liskov, Barbara; Wing, Jeannette (July 1999). "Behavioral Subtyping Using Invariants and Constraints" (http://reports-archive.adm.cs.cmu.edu/anon/1999/CMU-CS-99-156.ps) (PS). Retrieved 2006-10-05. The formalization of the principle by its authors.
- Reinhold Plösch, *Contracts, scenarios and prototypes: an integrated approach to high quality software*, Springer, 2004, ISBN 3-540-43486-0. Contains a gentler introduction to behavioral subtyping in its various forms in chapter 2.
- Robert C. Martin, The Liskov Substitution Principle (http://www.objectmentor.com/resources/articles/lsp.pdf), C++ Report, March 1996. An article popular in the object-oriented programming

community that gives several examples of LSP violations.
- Kazimir Majorinc, Ellipse-Circle Dilemma and Inverse Inheritance, ITI 98, Proceedings of the 20th International Conference of Information Technology Interfaces, Pula, 1998, ISSN 1330-1012. This paper discusses LSP in the mentioned context.

### Specific references


- Liskov, B. (May 1988). "Keynote address - data abstraction and hierarchy". *ACM SIGPLAN Notices* **23** (5): 17–34. doi:10.1145/62139.62141 (http://dx.doi.org/10.1145%2F62139.62141). A keynote address in which Liskov first formulated the principle.
- Meyer B., *Object-oriented Software Construction*, Prentice Hall, New York, 1988, ISBN 0-13-629031-0


# External links

- The Liskov Substitution Principle (http://www.engr.mun.ca/~theo/Courses/sd/5895-downloads/sd-principles-3.ppt.pdf), T. S. Norvell, 2003



Categories: Object-oriented programming | Type theory | Programming principles | Formal methods | Programming language semantics

---