WIKIPEDIA

Liskov substitution principle

Substitutability is a principle in object-oriented programming stating 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. an object of type T may be *substituted* with any object of a subtype S) without altering any of the desirable properties of the 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 titled *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 described the principle succinctly in a 1994 paper as follows:

> *Subtype Requirement*: Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true 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 to Bertrand Meyer's design by contract in that it considers the interaction of subtyping with preconditions, postconditions and invariants.

Contents

Principle

Liskov's notion of a behavioural subtype defines a notion of substitutability for 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).

Behavioural 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. Behavioural subtyping is 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 the signature requirements, the subtype must meet a number of behavioural conditions. 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). Because 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 can 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 *Object-Oriented Software Construction*. 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 languages that support 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.

See also

- Composition over inheritance
- Program refinement
- Referential transparency

- Type signature
- SOLID – the "L" in "SOLID" stands for Liskov substitution principle

References

Bibliography

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). pp. 1811–1841. doi:10.1145/197320.197383 (https://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 (https://web.archive.org/web/20151128004108/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 (https://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.
- Liskov Substitution Principle Explained (https://medium.com/@wrong.about/liskov-substitution-principle-a982551d584a)
- SOLID Class Design: The Liskov Substitution Principle (https://www.tomdalling.com/blog/software-design/solid-class-design-the-liskov-substitution-principle/)