WIKIPEDIA
The Free Encyclopedia

Information hiding

In computer science, **information hiding** is the principle of segregation of the *design decisions* in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed. The protection involves providing a stable interface which protects the remainder of the program from the implementation (whose details are likely to change). Written in another way, information hiding is the ability to prevent certain aspects of a class or software component from being accessible to its clients, using either programming language features (like private variables) or an explicit exporting policy.

Overview

The term *encapsulation* is often used interchangeably with information hiding. Not all agree on the distinctions between the two, though; one may think of information hiding as being the principle and encapsulation being the technique. A software module hides information by encapsulating the information into a module or other construct which presents an interface.[1]

A common use of information hiding is to hide the physical storage layout for data so that if it is changed, the change is restricted to a small subset of the total program. For example, if a three-dimensional point (*x*, *y*, *z*) is represented in a program with three floating-point scalar variables and later, the representation is changed to a single array variable of size three, a module designed with information hiding in mind would protect the remainder of the program from such a change.

In object-oriented programming, information hiding (by way of nesting of types) reduces software development risk by shifting the code's dependency on an uncertain implementation (design decision) onto a well-defined interface. Clients of the interface perform operations purely through the interface, so, if the implementation changes, the clients do not have to change.

Encapsulation

In his book on object-oriented design, Grady Booch defined encapsulation as "the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation."[2]

The purpose is to achieve the potential for change: the internal mechanisms of the component can be improved without impact on other components, or the component can be replaced with a different one that supports the same public interface. Encapsulation also protects the integrity of the component, by preventing users from setting the internal data of the component into an invalid or inconsistent state. Another

benefit of encapsulation is that it reduces system complexity and thus increases robustness, by limiting the interdependencies between software components.[2]

In this sense, the idea of encapsulation is more general than how it is applied in object-oriented programming. For example, a relational database is encapsulated in the sense that its only public interface is a query language (such as SQL), which hides all the internal machinery and data structures of the database management system. As such, encapsulation is a core principle of good software architecture, at every level of granularity.

Encapsulating software behind an interface allows the construction of objects that mimic the behavior and interactions of objects in the real world. For example, a simple digital alarm clock is a real-world object that a layperson (nonexpert) can use and understand. They can understand what the alarm clock does, and how to use it through the provided interface (buttons and screen), without having to understand every part inside of the clock. Similarly, if the clock were replaced by a different model, the layperson could continue to use it in the same way, provided that the interface works the same.

In the more concrete setting of an object-oriented programming language, the notion is used to mean either an information hiding mechanism, a bundling mechanism, or the combination of the two. (See Encapsulation (object-oriented programming) for details.)

History

The concept of information hiding was first described by David Parnas in 1972.[3][4] Before then, modularity was discussed by Richard Gauthier and Stephen Pont in their 1970 book *Designing Systems Programs* although modular programming itself had been used at many commercial sites for many years previously – especially in I/O sub-systems and software libraries – without acquiring the 'information hiding' tag – but for similar reasons, as well as the more obvious code reuse reason.

Example

Information hiding serves as an effective criterion for dividing any piece of equipment, software, or hardware, into modules of functionality. For instance, a car is a complex piece of equipment. In order to make the design, manufacturing, and maintenance of a car reasonable, the complex piece of equipment is divided into modules with particular interfaces hiding design decisions. By designing a car in this fashion, a car manufacturer can also offer various options while still having a vehicle that is economical to manufacture.

For instance, a car manufacturer may have a luxury version of the car as well as a standard version. The luxury version comes with a more powerful engine than the standard version. The engineers designing the two different car engines, one for the luxury version and one for the standard version, provide the same interface for both engines. Both engines fit into the engine bay of the car which is the same between both

versions. Both engines fit the same transmission, the same engine mounts, and the same controls. The differences in the engines are that the more powerful luxury version has a larger displacement with a fuel injection system that is programmed to provide the fuel-air mixture that the larger displacement engine requires.

In addition to the more powerful engine, the luxury version may also offer other options such as a better radio with CD player, more comfortable seats, a better suspension system with wider tires, and different paint colors. With all of these changes, most of the car is the same between the standard version and the luxury version. The radio with CD player is a module that replaces the standard radio, also a module, in the luxury model. The more comfortable seats are installed into the same seat mounts as the standard types of seats. Whether the seats are leather or plastic, or offer lumbar support or not, does not matter.

The engineers design the car by dividing the task up into pieces of work that are assigned to teams. Each team then designs their component to a particular standard or interface which allows the team flexibility in the design of the component while at the same time ensuring that all of the components will fit together.

Motor vehicle manufacturers frequently use the same core structure for several different models, in part as a cost-control measure. Such a "platform" also provides an example of information hiding, since the floorplan can be built without knowing whether it is to be used in a sedan or a hatchback.

As can be seen by this example, information hiding provides flexibility. This flexibility allows a programmer to modify the functionality of a computer program during normal evolution as the computer program is changed to better fit the needs of users. When a computer program is well designed, decomposing the source code solution into modules using the principle of information hiding, evolutionary changes are much easier because the changes typically are local rather than global changes.

Cars provide another example of this in how they interface with drivers. They present a standard interface (pedals, wheel, shifter, signals, gauges, etc.) on which people are trained and licensed. Thus, people only have to learn to drive a car; they don't need to learn a completely different way of driving every time they drive a new model. (Granted, there are manual and automatic transmissions and other such differences, but on the whole, cars maintain a unified interface.)

See also

- Implementation inheritance
- Inheritance semantics
- Modularity (programming)
- Opaque data type
- Virtual inheritance
- Transparency (human–computer interaction)

- Scope (programming)
- Compartmentalization (information security)
- Law of Demeter

Notes

1. Rogers, Wm. Paul (18 May 2001). "Encapsulation is not information hiding" (https://www.infoworld.com/article/2075271/encapsulation-is-not-information-hiding.html). *JavaWorld*. Retrieved 2020-07-20.
2. Booch, Grady (2007). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley. pp. 51–52. ISBN 978-0-201-89551-3.
3. Parnas, David L. (1972). "On the Criteria To Be Used in Decomposing Systems into Modules" (https://doi.org/10.1145%2F361598.361623). *Communications of the ACM*. **15** (12): 1053–58. doi:10.1145/361598.361623 (https://doi.org/10.1145%2F361598.361623). S2CID 53856438 (https://api.semanticscholar.org/CorpusID:53856438).
4. Scott, Michael L. (2009) [2000]. Broy, Manfred; Denert, Ernst (eds.). *Programming Language Pragmatics* (https://www.cs.rochester.edu/~scott/pragmatics/) (Third ed.). Morgan Kaufmann Publishers. p. 173. doi:10.1007/978-3-642-59412-0 (https://doi.org/10.1007%2F978-3-642-59412-0). ISBN 978-3-540-43081-0. S2CID 2698265 (https://api.semanticscholar.org/CorpusID:2698265).

References

- Parnas, David L. (1971). "Information Distribution Aspects of Design Methodology" (https://cseweb.ucsd.edu/~wgg/CSE218/Parnas-IFIP71-information-distribution.PDF) (PDF). In Charles V. Freiman and John E. Griffith and Jack L. Rosenfeld (ed.). *Information Processing, Proceedings of IFIP Congress 1971, Volume 1 - Foundations and Systems, Ljubljana, Yugoslavia, August 23–28, 1971*. IFIP Congress 1971. Vol. 1. North-Holland. pp. 339–344. doi:10.1184/R1/6606470.V1 (https://doi.org/10.1184%2FR1%2F6606470.V1).
- Parnas, David L. (2002). "The Secret History of Information Hiding". In Manfred Broy and Ernst Denert (ed.). *Software Pioneers* (https://www.springer.com/gp/book/9783540430810). Springer-Verlag Berlin Heidelberg. ISBN 978-0-12-374514-9.