[**SOLID 개념]**

[컴퓨터 프로그래밍](https://ko.wikipedia.org/wiki/%EC%BB%B4%ED%93%A8%ED%84%B0_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D)에서 **SOLID**란 [로버트 마틴](https://ko.wikipedia.org/w/index.php?title=%EB%A1%9C%EB%B2%84%ED%8A%B8_%EB%A7%88%ED%8B%B4&action=edit&redlink=1)[[1]](https://ko.wikipedia.org/wiki/SOLID#cite_note-ub-old-web-solid-1)[[2]](https://ko.wikipedia.org/wiki/SOLID#cite_note-ub-solid-2)이 2000년대 초반[[3]](https://ko.wikipedia.org/wiki/SOLID#cite_note-metz-presentation-2009-3)에 명명한 [객체 지향 프로그래밍](https://ko.wikipedia.org/wiki/%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D) 및 [설계](https://ko.wikipedia.org/w/index.php?title=%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%EC%84%A4%EA%B3%84&action=edit&redlink=1)의 다섯 가지 기본 원칙을 마이클 페더스가 [두문자어](https://ko.wikipedia.org/wiki/%EB%91%90%EB%AC%B8%EC%9E%90%EC%96%B4) [기억술](https://ko.wikipedia.org/wiki/%EA%B8%B0%EC%96%B5%EC%88%A0)로 소개한 것이다. 프로그래머가 시간이 지나도 [유지 보수](https://ko.wikipedia.org/w/index.php?title=%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EC%9C%A0%EC%A7%80_%EB%B3%B4%EC%88%98&action=edit&redlink=1)와 확장이 쉬운 시스템을 만들고자 할 때 이 원칙들을 함께 적용할 수 있다.[[3]](https://ko.wikipedia.org/wiki/SOLID#cite_note-metz-presentation-2009-3) SOLID 원칙들은 소프트웨어 작업에서 프로그래머가 [소스 코드](https://ko.wikipedia.org/wiki/%EC%86%8C%EC%8A%A4_%EC%BD%94%EB%93%9C)가 읽기 쉽고 확장하기 쉽게 될 때까지 소프트웨어 소스 코드를 [리팩토링](https://ko.wikipedia.org/wiki/%EB%A6%AC%ED%8C%A9%ED%86%A0%EB%A7%81)하여 [코드 냄새](https://ko.wikipedia.org/w/index.php?title=%EC%BD%94%EB%93%9C_%EB%83%84%EC%83%88&action=edit&redlink=1)를 제거하기 위해 적용할 수 있는 지침이다. 이 원칙들은 [애자일 소프트웨어 개발](https://ko.wikipedia.org/wiki/%EC%95%A0%EC%9E%90%EC%9D%BC_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EA%B0%9C%EB%B0%9C)과 [적응적 소프트웨어 개발](https://ko.wikipedia.org/w/index.php?title=%EC%A0%81%EC%9D%91%EC%A0%81_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EA%B0%9C%EB%B0%9C&action=edit&redlink=1)의 전반적 전략의 일부다.[[3]](https://ko.wikipedia.org/wiki/SOLID#cite_note-metz-presentation-2009-3)

**1.; [[단일 책임 원칙|단일 책임 원칙 (Single responsibility principle)]]**

[객체 지향 프로그래밍](https://ko.wikipedia.org/wiki/%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D)에서 **단일 책임 원칙**이란 모든 클래스는 하나의 책임만 가지며, 클래스는 그 책임을 완전히 [캡슐화](https://ko.wikipedia.org/wiki/%EC%BA%A1%EC%8A%90%ED%99%94)해야 함을 일컫는다. 클래스가 제공하는 모든 기능은 이 책임과 주의 깊게 부합해야 한다.

이 용어는 [로버트 마틴](https://ko.wikipedia.org/w/index.php?title=%EB%A1%9C%EB%B2%84%ED%8A%B8_%EB%A7%88%ED%8B%B4&action=edit&redlink=1)이 그의 저서 *기민한 소프트웨어 개발과 원칙, 패턴, 실례*[[1]](https://ko.wikipedia.org/wiki/%EB%8B%A8%EC%9D%BC_%EC%B1%85%EC%9E%84_%EC%9B%90%EC%B9%99#cite_note-1)으로 유명해진 *객체 지향 설계 원칙*[[2]](https://ko.wikipedia.org/wiki/%EB%8B%A8%EC%9D%BC_%EC%B1%85%EC%9E%84_%EC%9B%90%EC%B9%99#cite_note-2)이란 문서의 같은 이름을 가진 단락에서 소개되었다. 로버트 마틴은 이를 [톰 디마르코](https://ko.wikipedia.org/w/index.php?title=%ED%86%B0_%EB%94%94%EB%A7%88%EB%A5%B4%EC%BD%94&action=edit&redlink=1)의 책 *구조적 분석과 시스템 명세*[[3]](https://ko.wikipedia.org/wiki/%EB%8B%A8%EC%9D%BC_%EC%B1%85%EC%9E%84_%EC%9B%90%EC%B9%99#cite_note-3)에서 설명한 [응집성](https://ko.wikipedia.org/w/index.php?title=%EC%9D%91%EC%A7%91%EC%84%B1&action=edit&redlink=1) 원칙에 근거하여 설명하였다.

로버트 마틴은 책임을 *변경하려는 이유*로 정의하고, 어떤 클래스나 모듈은 변경하려는 단 하나 이유만을 가져야 한다고 결론 짓는다. 예를 들어서 보고서를 편집하고 출력하는 모듈을 생각해 보자. 이 모듈은 두 가지 이유로 변경될 수 있다. 첫 번째로 보고서의 내용 때문에 변경될 수 있다. 두 번째로 보고서의 형식 때문에 변경될 수 있다. 이 두 가지 변경은 하나는 실질적이고 다른 하나는 꾸미기 위한 매우 다른 원인에 기인한다. 단일 책임 원칙에 의하면 이 문제의 두 측면이 실제로 분리된 두 책임 때문이며, 따라서 분리된 클래스나 모듈로 나누어야 한다. 다른 시기에 다른 이유로 변경되어야 하는 두 가지를 묶는 것은 나쁜 설계일 수 있다.

한 클래스를 한 관심사에 집중하도록 유지하는 것이 중요한 이유는, 이것이 클래스를 더욱 튼튼하게 만들기 때문이다. 앞서 든 예를 계속 살펴보면, 편집 과정에 변경이 일어나면, 같은 클래스의 일부로 있는 출력 코드가 망가질 위험이 대단히 높다.

**2. [[개방-폐쇄 원칙|OCP]]**

**; [[개방-폐쇄 원칙|개방-폐쇄 원칙 (Open/closed principle)]]**

소프트웨어 개발 작업에 이용된 많은 모듈 중에 하나에 수정을 가할 때 그 모듈을 이용하는 다른 모듈을 줄줄이 고쳐야 한다면, 이와 같은 프로그램은 수정하기가 어렵다. 개방-폐쇄 원칙은 시스템의 구조를 올바르게 재조직(리팩토링)하여 나중에 이와 같은 유형의 변경이 더 이상의 수정을 유발하지 않도록 하는 것이다. 개방-폐쇄 원칙이 잘 적용되면, 기능을 추가하거나 변경해야 할 때 이미 제대로 동작하고 있던 원래 코드를 변경하지 않아도, 기존의 코드에 새로운 코드를 추가함으로써 기능의 추가나 변경이 가능하다.

**3. [[리스코프 치환 원칙|LSP]]**

**; [[리스코프 치환 원칙|리스코프 치환 원칙 (Liskov substitution principle)]]**

**치환성**([영어](https://ko.wikipedia.org/wiki/%EC%98%81%EC%96%B4): substitutability)은 [객체 지향 프로그래밍](https://ko.wikipedia.org/wiki/%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D) 원칙이다. [컴퓨터 프로그램](https://ko.wikipedia.org/wiki/%EC%BB%B4%ED%93%A8%ED%84%B0_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8)에서 [자료형](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%A3%8C%ED%98%95) S가 자료형 T의 [하위형](https://ko.wikipedia.org/w/index.php?title=%ED%95%98%EC%9C%84%ED%98%95&action=edit&redlink=1)이라면 필요한 프로그램의 속성(정확성, 수행하는 업무 등)의 변경 없이 자료형 T의 객체를 자료형 S의 객체로 교체(치환)할 수 있어야 한다는 원칙이다. **리스코프 치환 원칙**([영어](https://ko.wikipedia.org/wiki/%EC%98%81%EC%96%B4): Liskov substitution principle, LSP)은 [바바라 리스코프](https://ko.wikipedia.org/w/index.php?title=%EB%B0%94%EB%B0%94%EB%9D%BC_%EB%A6%AC%EC%8A%A4%EC%BD%94%ED%94%84&action=edit&redlink=1)가 *자료 추상화와 계층 (Data abstraction and hierarchy)*이라는 제목으로 기조연설을 한 1987년 컨퍼런스에서 처음 소개한 내용으로, 이 원칙을 엄밀한 용어로 말하자면 **(강한) 행동적 하위형화**라 부르는 하위형화 관계의 특정한 사례이다. 이 정의는 1994년 논문에서 다음 원칙을 만들어낸 [자료형](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%A3%8C%ED%98%95)의 의미론적 상호처리를 보장하기 때문에 단순한 문법적 관계일 뿐만 아니라 [의미론](https://ko.wikipedia.org/w/index.php?title=%EC%9D%98%EB%AF%B8%EB%A1%A0_(%EC%BB%B4%ED%93%A8%ED%84%B0_%EA%B3%BC%ED%95%99)&action=edit&redlink=1)적 관계다.

q(x)를 자료형 T의 객체 x에 대해 증명할 수 있는 속성이라 하자. 그렇다면 S가 T의 하위형이라면 q(y)는 자료형 S의 객체 y에 대해 증명할 수 있어야 한다.

같은 논문에서 리스코프와 윙은 사전 및 사후 조건으로 하위형화의 상호 작용을 고려함으로써 행동 하위형화 개념을 [베르트랑 메이어](https://ko.wikipedia.org/w/index.php?title=%EB%B2%A0%EB%A5%B4%ED%8A%B8%EB%9E%91_%EB%A9%94%EC%9D%B4%EC%96%B4&action=edit&redlink=1)의 [계약에 의한 설계](https://ko.wikipedia.org/w/index.php?title=%EA%B3%84%EC%95%BD%EC%97%90_%EC%9D%98%ED%95%9C_%EC%84%A4%EA%B3%84&action=edit&redlink=1)와 분명한 유사성을 지닌 [호어 논리](https://ko.wikipedia.org/w/index.php?title=%ED%98%B8%EC%96%B4_%EB%85%BC%EB%A6%AC&action=edit&redlink=1)의 확장으로 자세히 설명한다.

**4. [[인터페이스 분리 원칙|ISP]]**

**; [[인터페이스 분리 원칙|인터페이스 분리 원칙 (Interface segregation principle)]]**

**인터페이스 분리 원칙**은 클라이언트가 자신이 이용하지 않는 메서드에 의존하지 않아야 한다는 원칙이다.[[1]](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4_%EB%B6%84%EB%A6%AC_%EC%9B%90%EC%B9%99#cite_note-ASD-1) 인터페이스 분리 원칙은 큰 덩어리의 인터페이스들을 구체적이고 작은 단위들로 분리시킴으로써 클라이언트들이 꼭 필요한 메서드들만 이용할 수 있게 한다. 이와 같은 작은 단위들을 *역할 인터페이스*라고도 부른다.[[2]](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4_%EB%B6%84%EB%A6%AC_%EC%9B%90%EC%B9%99#cite_note-RoleInterface-2) 인터페이스 분리 원칙을 통해 시스템의 내부 의존성을 약화시켜 리팩토링, 수정, 재배포를 쉽게 할 수 있다. 인터페이스 분리 원칙은 SOLID 5원칙의 하나이며, GRASP의 밀착 원칙과 비슷하다. [[3]](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4_%EB%B6%84%EB%A6%AC_%EC%9B%90%EC%B9%99#cite_note-CB-3)

**5. [[의존관계 역전 원칙|DIP]]**

**;[[의존관계 역전 원칙|의존관계 역전 원칙 (Dependency inversion principle)]]**

객체 지향 프로그래밍에서 **의존 관계 역전 원칙**은 소프트웨어 모듈들을 분리하는 특정 형식을 지칭한다. 이 원칙을 따르면, 상위 계층(정책 결정)이 하위 계층(세부 사항)에 의존하는 전통적인 의존 관계를 반전(역전)시킴으로써 상위 계층이 하위 계층의 구현으로부터 독립되게 할 수 있다. 이 원칙은 다음과 같은 내용을 담고 있다. [[1]](https://ko.wikipedia.org/wiki/%EC%9D%98%EC%A1%B4%EA%B4%80%EA%B3%84_%EC%97%AD%EC%A0%84_%EC%9B%90%EC%B9%99#cite_note-1)

첫째, 상위 모듈은 하위 모듈에 의존해서는 안된다. 상위 모듈과 하위 모듈 모두 추상화에 의존해야 한다.

둘째, 추상화는 세부 사항에 의존해서는 안된다. 세부사항이 추상화에 의존해야 한다.

이 원칙은 '상위와 하위 객체 모두가 동일한 추상화에 의존해야 한다'는 객체 지향적 설계의 대원칙을 제공한다. [[2]](https://ko.wikipedia.org/wiki/%EC%9D%98%EC%A1%B4%EA%B4%80%EA%B3%84_%EC%97%AD%EC%A0%84_%EC%9B%90%EC%B9%99#cite_note-2)

컴퓨터 프로그래밍에서 **SOLID**란 **로버트 마틴**이 2000년대 초반에 명명한 **객체 지향 프로그래밍 및 설계의 다섯 가지 기본 원칙**을 **마이클 페더스**가 두문자어(낱말의 머리글자를 모아서 만든 준말) 기억술로 소개한 것이다. 프로그래머가 시간이 지나도 **유지 보수와 확장이 쉬운**시스템을 만들고자 할 때 이 원칙들을 함께 적용할 수 있다. SOLID 원칙들은 소프트웨어 작업에서 프로그래머가**소스 코드가 읽기 쉽고 확장하기 쉽게 될 때까지**소프트웨어 소스 코드를**리팩토링**하여 **코드 스멜를 제거**하기 위해 적용할 수 있는 지침이다. 이 원칙들은 애자일 소프트웨어 개발과 적응적 소프트웨어 개발의 전반적 전략의 일부다.

**1. 단일 책임 원칙 (Single responsibility principle)**

두문자 : S 약어 : SRP

모든**클래스**는 **하나의 책임**만 가지며, 클래스는 그 **책임을 완전히 캡슐화**해야 함을 일컫는다. 클래스가 제공하는 모든 기능은 이 책임과 주의 깊게 부합해야 한다.(응집성을 높이고 결합도를 낮춘다.)

**2. 개방-폐쇄 원칙 (Open/closed principle)**

두문자 : O 약어 : OCP

소프트웨어 개체(클래스, 모듈, 함수 등등)는 **확장에 대해 열려 있어야 하고, 수정에 대해서는 닫혀 있어야**한다는 프로그래밍 원칙이다.

**- 확장에 대해 열려 있다.**

이것은 모듈의 동작을 확장할 수 있다는 것을 의미한다. 애플리케이션의 요구 사항이 변경될 때, 이 변경에 맞게 **새로운 동작을 추가해 모듈을 확장**할 수 있다. 즉, 모듈이 하는 일을 변경할 수 있다.

**- 수정에 대해 닫혀 있다**

모듈의 **소스 코드나 바이너리 코드를 수정하지 않아도** 모듈의 기능을 확장하거나 변경할 수 있다. 그 모듈의 실행 가능한 바이너리 형태나 링크 가능한 라이브러리(예를 들어 윈도의 DLL이나 자바의 .jar)를 건드릴 필요가 없다.

**3. 리스코프 치환 원칙 (Liskov substitution principle)**

두문자 : L 약어 : LSP

치환성(substitutability)은 객체 지향 프로그래밍 원칙이다. 컴퓨터 프로그램에서**자료형 S가 자료형 T의 하위형이라면** 필요한 **프로그램의 속성(정확성, 수행하는 업무 등)의 변경 없이**자료형 T의 객체를 자료형 S의 객체로 **교체(치환)할 수 있어야** 한다는 원칙이다.

이게 좀 헷갈린다.http://static.se2.naver.com/static/img/emoticon/1_48.gif

**4. 인터페이스 분리 원칙 (Interface segregation principle)**

두문자 : I 약어 : ISP

인터페이스 분리 원칙은 클라이언트가 자신이**이용하지 않는 메서드에 의존하지 않아야** 한다는 원칙이다. 인터페이스 분리 원칙은 큰 덩어리의 **인터페이스들을 구체적이고 작은 단위들로 분리**시킴으로써 클라이언트들이**꼭 필요한 메서드들만** 이용할 수 있게 한다. 이와 같은 작은 단위들을 역할 인터페이스라고도 부른다. 인터페이스 분리 원칙을 통해 시스템의 내부 의존성을 약화시켜 리팩토링, 수정, 재배포를 쉽게 할 수 있다.

**5. 의존관계 역전 원칙 (Dependency inversion principle)**

두문자 : D 약어 : DIP

객체 지향 프로그래밍에서 의존 관계 역전 원칙은 소프트웨어 모듈들을 분리하는 특정 형식을 지칭한다. 이 원칙을 따르면, 상위 계층(정책 결정)이 하위 계층(세부 사항)에 의존하는 전통적인 의존 관계를 반전(역전)시킴으로써 **상위 계층이 하위 계층의 구현으로부터 독립되게**할 수 있다. 이 원칙은 다음과 같은 내용을 담고 있다.

첫째, 상위 모듈은 하위 모듈에 의존해서는 안된다.**상위 모듈과 하위 모듈 모두 추상화에 의존**해야 한다.

둘째, **추상화는 세부 사항에 의존해서는 안된다**. **세부사항이 추상화에 의존**해야 한다.

이 원칙은 '상위와 하위 객체 모두가 동일한 추상화에 의존해야 한다'는 객체 지향적 설계의 대원칙을 제공한다.

**reference**[**https://ko.wikipedia.org/wiki**](https://ko.wikipedia.org/wiki)

[#객체지향](http://blog.naver.com/PostListByTagName.nhn?blogId=lhm0812&logType=mylog&tagName=%B0%B4%C3%BC%C1%F6%C7%E2)[#5원칙](http://blog.naver.com/PostListByTagName.nhn?blogId=lhm0812&logType=mylog&tagName=5%BF%F8%C4%A2)[#SOLID](http://blog.naver.com/PostListByTagName.nhn?blogId=lhm0812&logType=mylog&tagName=SOLID)

http://blogimgs.naver.net/imgs/nblog/spc.gif [인쇄](http://blog.naver.com/PostView.nhn?blogId=lhm0812&logNo=220677064489&beginTime=0&jumpingVid=&from=search&redirect=Log&widgetTypeCall=true&topReferer=http%3A%2F%2Fsearch.naver.com%2Fsearch.naver%3Fwhere%3Dnexearch%26query%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B5%25EC%259B%2590%25EC%25B9%2599%26sm%3Dtop_sug.pre%26fbm%3D1%26acr%3D7%26acq%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B%26qdt%3D0%26ie%3Dutf8%26url%3Dhttp%253A%252F%252Fblog.naver.com%252Flhm0812%253FRedirect%253DLog%2526logNo%253D220677064489%26ucs%3DMzZII98wooKj)

* [블로그](http://blog.naver.com/PostView.nhn?blogId=lhm0812&logNo=220677064489&beginTime=0&jumpingVid=&from=search&redirect=Log&widgetTypeCall=true&topReferer=http%3A%2F%2Fsearch.naver.com%2Fsearch.naver%3Fwhere%3Dnexearch%26query%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B5%25EC%259B%2590%25EC%25B9%2599%26sm%3Dtop_sug.pre%26fbm%3D1%26acr%3D7%26acq%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B%26qdt%3D0%26ie%3Dutf8%26url%3Dhttp%253A%252F%252Fblog.naver.com%252Flhm0812%253FRedirect%253DLog%2526logNo%253D220677064489%26ucs%3DMzZII98wooKj)
* [카페](http://blog.naver.com/PostView.nhn?blogId=lhm0812&logNo=220677064489&beginTime=0&jumpingVid=&from=search&redirect=Log&widgetTypeCall=true&topReferer=http%3A%2F%2Fsearch.naver.com%2Fsearch.naver%3Fwhere%3Dnexearch%26query%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B5%25EC%259B%2590%25EC%25B9%2599%26sm%3Dtop_sug.pre%26fbm%3D1%26acr%3D7%26acq%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B%26qdt%3D0%26ie%3Dutf8%26url%3Dhttp%253A%252F%252Fblog.naver.com%252Flhm0812%253FRedirect%253DLog%2526logNo%253D220677064489%26ucs%3DMzZII98wooKj)

**[[출처]](http://blog.naver.com/PostView.nhn?blogId=lhm0812&logNo=220677064489&beginTime=0&jumpingVid=&from=search&redirect=Log&widgetTypeCall=true&topReferer=http%3A%2F%2Fsearch.naver.com%2Fsearch.naver%3Fwhere%3Dnexearch%26query%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B5%25EC%259B%2590%25EC%25B9%2599%26sm%3Dtop_sug.pre%26fbm%3D1%26acr%3D7%26acq%3D%25EA%25B0%259D%25EC%25B2%25B4%25EC%25A7%2580%25ED%2596%25A5%2B%26qdt%3D0%26ie%3Dutf8%26url%3Dhttp%253A%252F%252Fblog.naver.com%252Flhm0812%253FRedirect%253DLog%2526logNo%253D220677064489%26ucs%3DMzZII98wooKj" \o "포스트 보내기" \t "_blank)**[객체지향 5원칙 SOLID](http://blog.naver.com/lhm0812/220677064489)|**작성자**[lhm0812](http://blog.naver.com/lhm0812)