# **SOLID PRINCIPLES**

Martin Fowler is a founder of SOLID principles in 2000x

|  |  |  |
| --- | --- | --- |
| ID | PRINCIPLE | DESK |
| 1 | **Single Responsibility Principle** | Принцип единственной ответственности |
| 2 | **Open Closed Principle** | Принцип открытости/закрытости) |
| 3 | **Liskov’s Substitution Principle** | Принцип подстановки Барбары Лисков |
| 4 | **Interface Segregation Principle** | Принцип разделения интерфейса |
| 5 | **Dependency Inversion Principle** | Принцип инверсии зависимостей |

## **1.PRINCIPLE – SINGLE RESPONSIBILITY**

Class or method should be responsible only for one purpose

+:

Easy test and modify

## **2.PRINCIPLE – OPEN/CLOSED**

Class should be closed for changes and open for extension

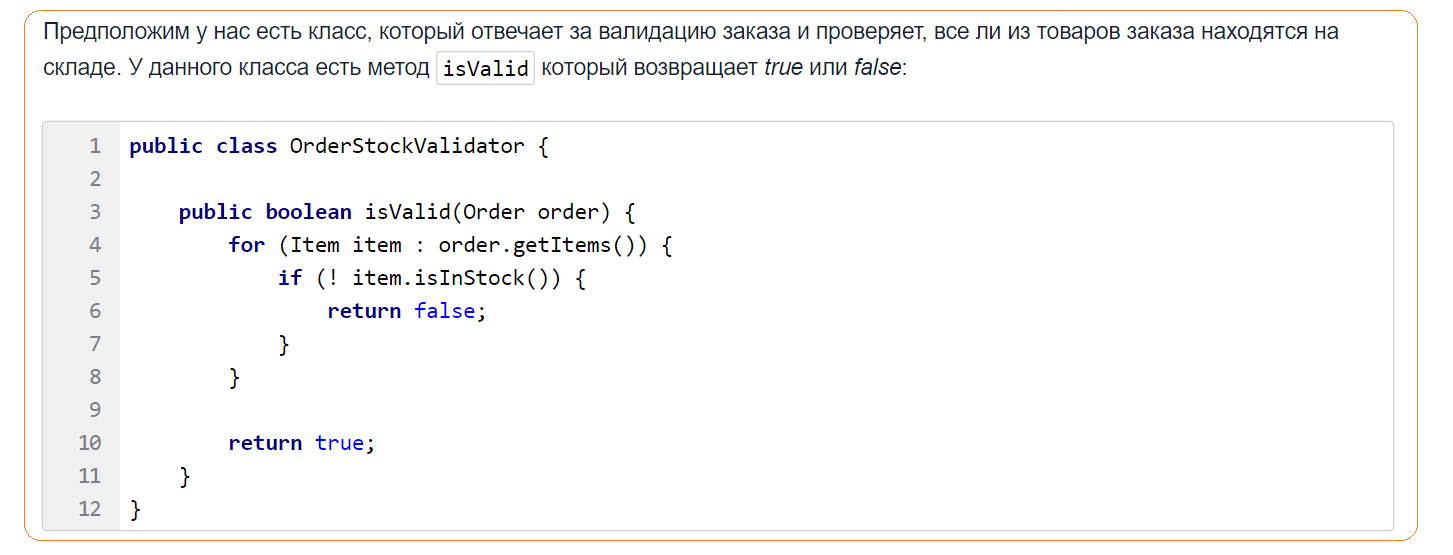
It means if you need to change base class you should create another class and extend from that instead of modifying

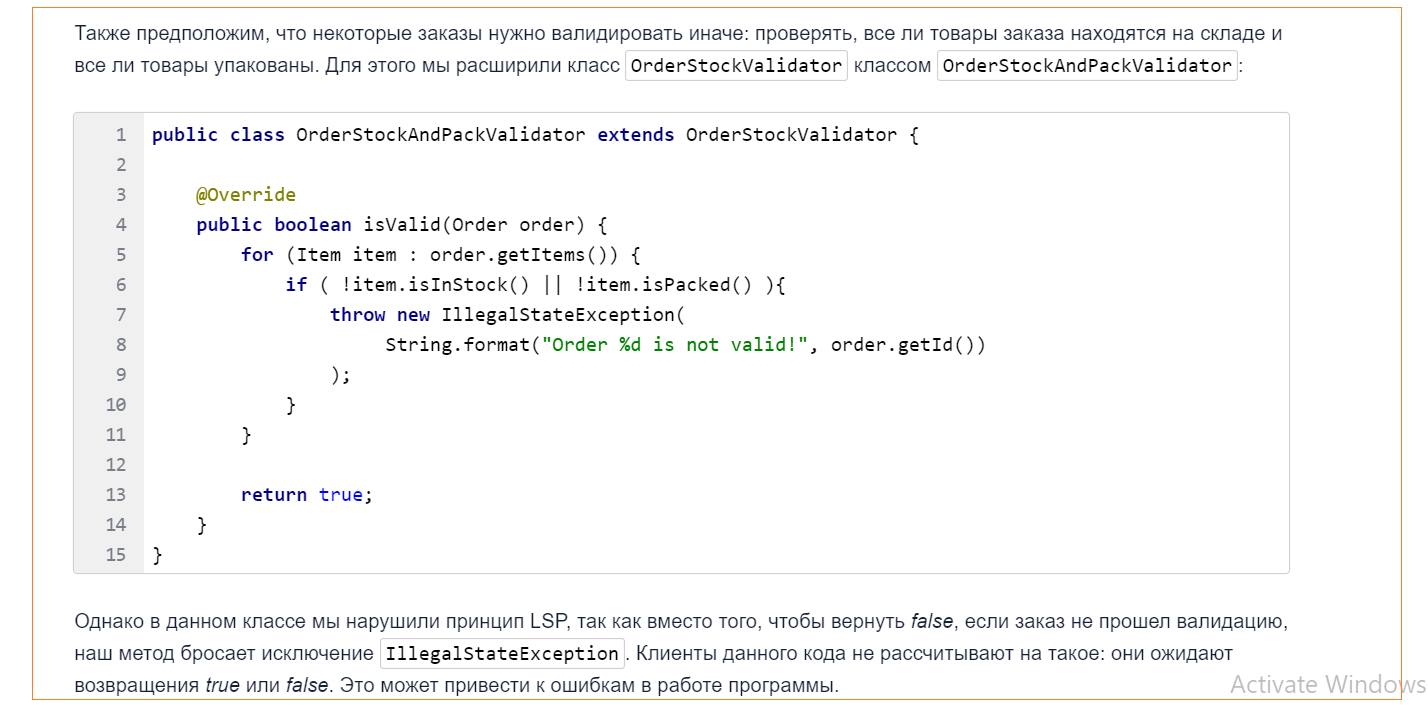
## **3.PRINCIPLE – LISKOV PRICNCIPLE**

Это вариация принципа открытости/закрытости, о котором говорилось ранее. Его можно описать так: объекты в программе можно заменить их наследниками без изменения свойств программы.

Это означает, что класс, разработанный путем расширения на основании базового класса, должен переопределять его методы так, чтобы не нарушалась функциональность с точки зрения клиента. То есть, если разработчик расширяет ваш класс и использует его в приложении, он не должен изменять ожидаемое поведение переопределенных методов.

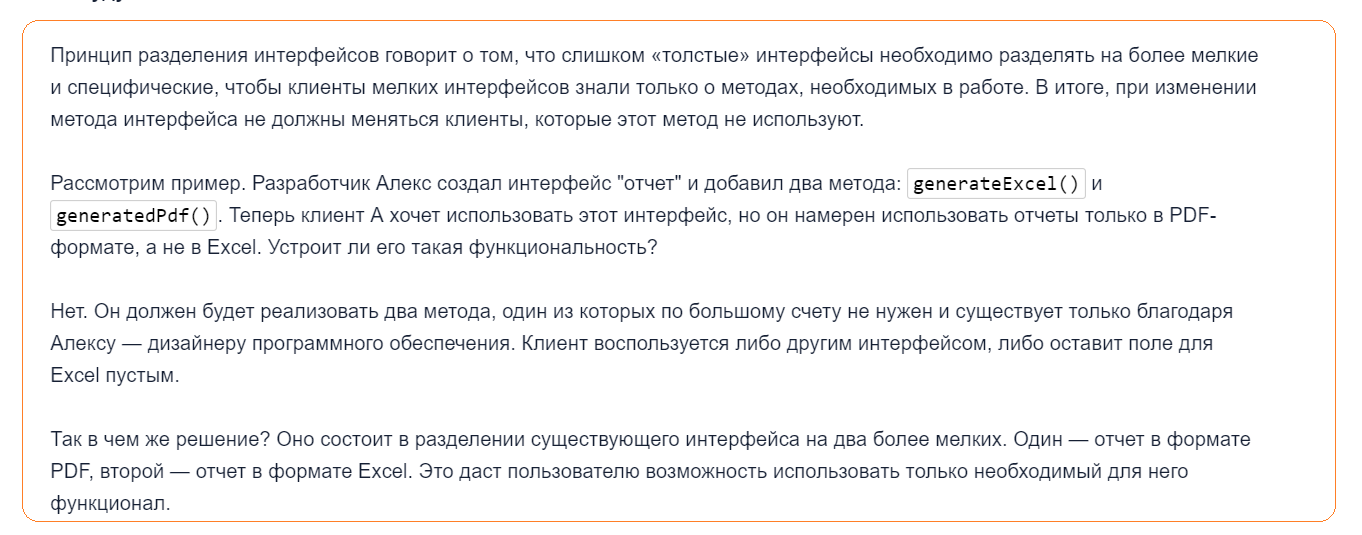
Подклассы должны переопределять методы базового класса так, чтобы не нарушалась функциональность с точки зрения клиента.





## **4.PRINCIPLE – INTERFACE SEGREGATION**

клиенты не должны быть вынуждены реализовывать методы, которые они не будут использовать.клиенты не должны быть вынуждены реализовывать методы, которые они не будут использовать.Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более мелкие и специфические, чтобы клиенты мелких интерфейсов знали только о методах, необходимых в работе.



## **5.PRINCIPLE – DEPENDENCY IMVERSION**

зависимости внутри системы строятся на основе абстракций. Модули верхнего уровня не зависят от модулей нижнего уровня. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Классическое применение этого принципа — Spring framework. В рамках Spring framework все модули выполнены в виде отдельных компонентов, которые могут работать вместе. Они настолько автономны, что могут быть быть с такой же легкостью задействованы в других программных модулях помимо Spring framework.

