SOLID PRINCIPLES OF OBJECT-ORIENTED DESIGN

from Robert Martin aka Uncle Bob

S

Single Responsibility Principle. Принцип единственной ответственности

Объект должен иметь одну обязанность, одну причину для существования. И эта причина должна быть полностью инкапсулирована в одном классе. Объект может иметь несколько форм поведения, но все они должны быть связаны с главной причиной его существования и всё, что он делает должно поддерживать это.

Из этого принципа следует, что не следует создавать “божественные объекты”, представляющие несколько сущностей реального мира. Но и разделять поведение одной сущности между несколькими объектами - плохая идея.

Объект должен всегда отвечать сам за себя.

O

Open/Closed Principle. Принцип открытости/закрытости

Программные сущности должны быть открыты для расширения, но закрыты для изменения. Сущности - это классы, модули, функции и т. д.

Этот принцип означает, что если после написания некоторого рабочего кода из-за изменения требований нужно добавить новое поведение - следует писать новый код, а не изменять тот, который уже работает.

Пример с наследованием. Допустим, у нас есть какой-то рабочий код. И мы получаем новое требование. И мы можем реализовать его добавив новый класс. Если этот класс должен поддерживать дополнительные формы поведения, следует не изменять исходный супер-класс (родительский класс), а написать новый код. Сохранение уже готового рабочего кода и означает, что система открыта для расширения, но закрыта для изменения.

L

Liskov Substitution Principle. Принцип подстановки Барбары Лисков

Этот принцип требует, чтобы объекты в программе можно было бы заменять экземплярами их подтипов, иначе говоря, подклассов и производных классов, без нарушения работы программы.

Данный принцип расширяет идею наследования. Это означает, что если создан целый ряд производных, дочерних классов или подклассов, мы всегда должны быть способны работать с ними как с объектами супер-класса (родительского класса). Не должно быть ситуации, когда нужно обрабатывать какой-либо отдельный объект специальным образом. Если это требуется - подкласс спроектирован плохо, с нарушением этого принципа

I

Interface Segregation Principle. Принцип разделения интерфейсов

Интерфейс - это не пользовательский интерфейс, его ещё называют протоколом. Несколько специфических протоколов лучше, чем один протокол общего назначения. Интерфейс, это формализованные списки методов, которые могут быть реализованы в классе. Суть в том, что эти интерфейсы (списки методов) должны быть как можно меньше. Если они чрезмерно увеличиваются - их следует разделить на меньшие интерфейсы.

Поскольку в классе можно реализовать несколько меньших интерфейсов, никакой класс не должен поддерживать огромные списки методов, которые ему не нужны.

D

Dependency Inversion Principle. Принцип инверсии зависимостей

Этот принцип требует, чтобы в основе зависимостей лежали абстракции, а не конкретные детали. Он подчеркивает, что не следует навязывать конкретные объекты непосредственно, а проектировать их так, чтобы они работали с абстракциями. Это нужно для того, чтобы свести к минимуму зависимость между обектами.

Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.

Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций. Это позволяет заменять конкретные классы, гибко расширяя возможности приложения, без изменения объекта Store. Следует иметь ввиду, что создавая слишком много уровней абстракции, нарушается принцип YAGNI. Стремитесь к разумному компромиссу.