Слайд 1.
Программные системы должны быть легко поддерживаемыми и легко расширяемыми в долгосрочной перспективе. Хорошие архитекторы и программисты всегда имеют в виду, что код нужно писать так, чтобы в будущем можно было легко добавлять новые функции, так как когда-нибудь обязательно возникнут новые требования. А когда к существующему коду добавляется новая функциональность, это изменение потенциально может привести к поломке кода или появлению дефектов.

Слайд 2.
Разве не было бы здорово, если бы мы разрабатывали код таким образом, чтобы никогда не изменять существующий код, а добавлять новый код только тогда, когда требуется новая функциональность?

Слайд 3.Определение open closed принципа

ПРОГРАММНЫЕ ОБЪЕКТЫ, ТАКИЕ КАК КЛАССЫ, МОДУЛИ, ФУНКЦИИ И Т.П. ДОЛЖНЫ БЫТЬ ОТКРЫТЫ ДЛЯ РАСШИРЕНИЯ, НО ЗАКРЫТЫ ДЛЯ МОДИФИКАЦИИ.

Слайд 4.
Суть принципа Open Closed заключается в том, что однажды написанный код не должен быть изменен повторно в будущем, то есть он должен быть закрыт для внесения изменений.

Если необходимо реализовать новую функциональность, то это следует делать путем расширения кода, а не путем изменения существующего кода. Таким образом, мы всегда можем быть уверены, что существующий код по-прежнему работоспособен, поскольку он никогда не изменялся.

Слайд 5.
Реализация этого в реальных проектах является самой сложной среди всех принципов SOLID. Это достигается с помощью самых фундаментальных аспектов объектно-ориентированного программирования — наследования и абстракции.

Слайд 6 - 7.
Давайте возьмем пример, нарушающий open closed принцип.
В приведенном примере класс DrawShapes не придерживается принципа Open Closed. Причина в том, что любое введение нового типа фигуры потребует изменений в методе draw() — появится дополнительный оператор if/else для обработки новой фигуры.

Слайд 8.
Теперь давайте посмотрим на ту же функциональность, но на этот раз придерживаясь принципа.

Слайд 9 - 11.
В приведенном коде метод draw() был выделен в новый абстрактный класс. Затем каждый из отдельных типов классов фигур расширяет абстрактный класс и предоставляет собственную реализацию метода рисования. Класс DrawShapes использует только абстракции. Он остается нетронутым при добавлении нового типа фигуры, поскольку не касается деталей реализации каждого типа фигуры.

Слайд 12.
Принцип открытости и закрытости зависит от основ объектно-ориентированного проектирования — наследование, абстракция и инкапсуляция. В нашем случае нужна только

новая реализация класса для новой фигуры. Никакой существующий код при этом не изменяется.

Слайд 13. Еще одним фундаментальным аспектом объектно-ориентированного программирования, который помогает обеспечить соответствие кода принципу открытости и закрытости, является инкапсуляция.

1. Приватные поля класса. Когда поле в классе является публичным или защищенным, его можно изменить из других классов и подклассов. Следовательно, разработчик должен пытаться сделать все что можно приватным.

Слайд 14.

2. Final поля. Существуют поля, которые никогда не следует изменять. Их необходимо объявить final, чтобы они были закрыты для изменения.

Слайд 15.

3. Никаких глобальных переменных. Не следует оставлять глобальные переменные доступными для изменения. Все глобальные переменные должны быть объявлены как static final.