ВОЩИЛО ЮРИЙ

РАЗРАБОТКА КОРПОРАТИВНЫХ РЕШЕНИЙ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ JAVA

ОСНОВНЫЕ КОНЦЕПЦИИ И ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

1. S.O.L.I.D
2. Singleton
3. Factory, Factory Method
4. Command
5. Builder
6. Strategy

S.O.L.I.D

1. Принцип единственности ответственности (The **S**ingle Responsibility Principle)
2. Принцип открытости/закрытости (The **O**pen Closed Principle)
3. Принцип замещения Лисков (The **L**iskov Substitution Principle)
4. Принцип разделения интерфейса (The **I**nterface Segregation Principle)
5. Принцип инверсии зависимости (The **D**ependency Inversion Principle)

ПРИНЦИП ЕДИНСТВЕННОСТИ ОТВЕТСТВЕННОСТИ (THE SINGLE RESPONSIBILITY PRINCIPLE)

"не должно быть больше одной причины для изменения класса"

ПРИНЦИП ЕДИНСТВЕННОСТИ ОТВЕТСТВЕННОСТИ (THE SINGLE RESPONSIBILITY PRINCIPLE)

“не должно быть больше одной причины для изменения класса”

Если у объекта много ответственности, то и меняться он будет очень часто. Таким образом, если класс имеет больше одной ответственности, то это ведет к хрупкости дизайна и ошибкам в неожиданных местах при изменениях кода.

ПРИНЦИП ОТКРЫТОСТИ/ЗАКРЫТОСТИ (THE OPEN CLOSED PRINCIPLE)

“программные сущности (классы, модули, функции и т.д.) должны быть открыты для расширения, но закрыты для изменения”

ПРИНЦИП ОТКРЫТОСТИ/ЗАКРЫТОСТИ (THE OPEN CLOSED PRINCIPLE)

“программные сущности (классы, модули, функции и т.д.) должны быть открыты для расширения, но закрыты для изменения”

целью является разработка системы, которая будет достаточно просто и безболезненно меняться. Другими словами, система должна быть гибкой.

Например, внесение изменений в библиотеку общую для 4х проектов не должно быть долгим («долгим» является разным промежутком времени для конкретной ситуации) и уж точно не должно вести к изменениям в этих 4х проектах.

ПРИНЦИП ЗАМЕЩЕНИЯ ЛИСКОВ (THE LISKOV SUBSTITUTION PRINCIPLE)

“если для каждого объекта o1 типа S существует объект o2 типа T, который для всех программ P определен в терминах T, то поведение P не изменится, если o2 заменить на o1 при условии, что S является подтипом T.”

“Функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом.”

ПРИНЦИП ЗАМЕЩЕНИЯ ЛИСКОВ (THE LISKOV SUBSTITUTION PRINCIPLE)

Принципы проектирования взаимосвязаны.

Нарушение одного из принципов скорее всего приведет к нарушению одного или нескольких других принципов.

ПРИНЦИП РАЗДЕЛЕНИЯ ИНТЕРФЕЙСА (THE INTERFACE SEGREGATION PRINCIPLE)

"клиенты не должны зависеть от методов, которые они не используют"

ПРИНЦИП РАЗДЕЛЕНИЯ ИНТЕРФЕЙСА (THE INTERFACE SEGREGATION PRINCIPLE)

"клиенты не должны зависеть от методов, которые они не используют"

"необходимо избавиться от ненужных зависимостей в коде, сделать код легко читаемым и легко изменяемым"

- Лишняя абстракция в наследовании
- «Жирный» интерфейс

ПРИНЦИП ИНВЕРСИИ ЗАВИСИМОСТИ (THE DEPENDENCY INVERSION PRINCIPLE)

“Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракции.”

“Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.”

ОСНОВНЫЕ ШАБЛОНЫ

- Основные шаблоны
- Порождающие шаблоны

шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять инстанцируемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту.

- Структурные шаблоны

определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

- Поведенческие шаблоны

определяют взаимодействие между объектами, увеличивая таким образом его гибкость.

SINGLETON

порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

```
public class Singleton {

  public static final Singleton INSTANCE = new Singleton();
}
```

ФАБРИЧНЫЙ МЕТОД

порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.