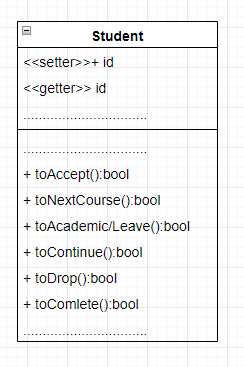
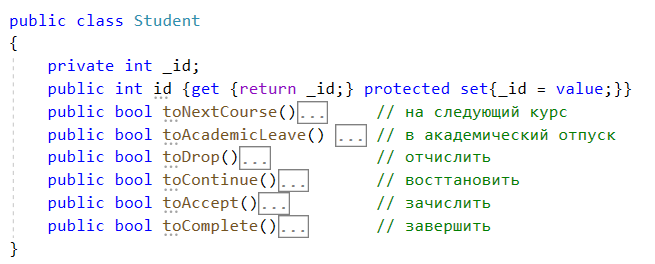
**Паттерны проектирования**

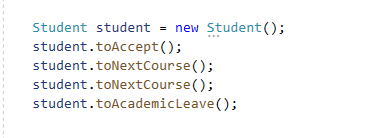
**ИСиТ,ПОИБМС-2**

**Лекция 2**

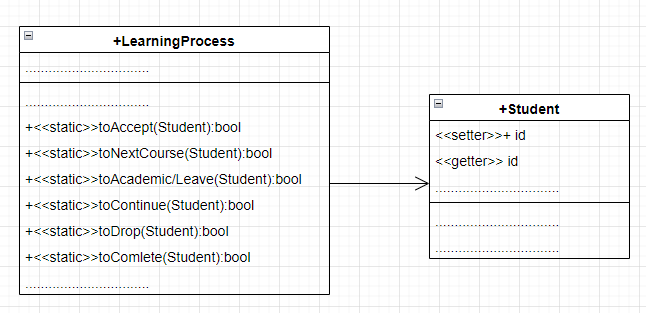
**Принципы SOLID**

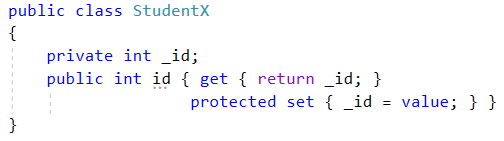
1. **SOLID:** Single Responsibility Principe(принцип единой ответственности)
2. **SOLID:** Роберт Мартин, книга «Agile Software Development, Principles, and Practices».
3. **Single** Responsibility Principe**:** принцип единой ответственности:

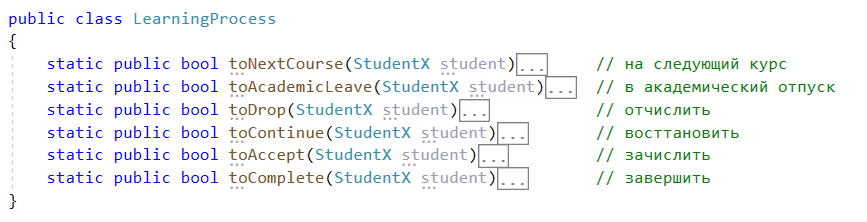
 

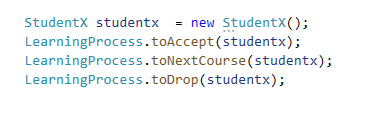


***Ассоциация(использует,взаимодействует,содержит)***

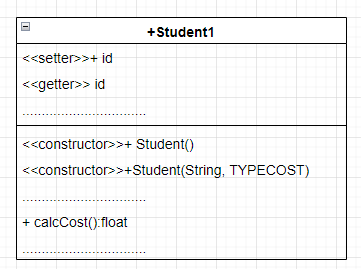


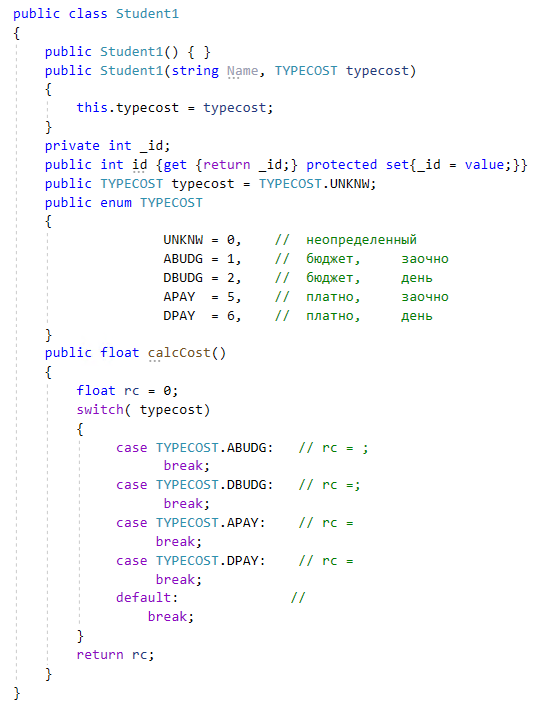






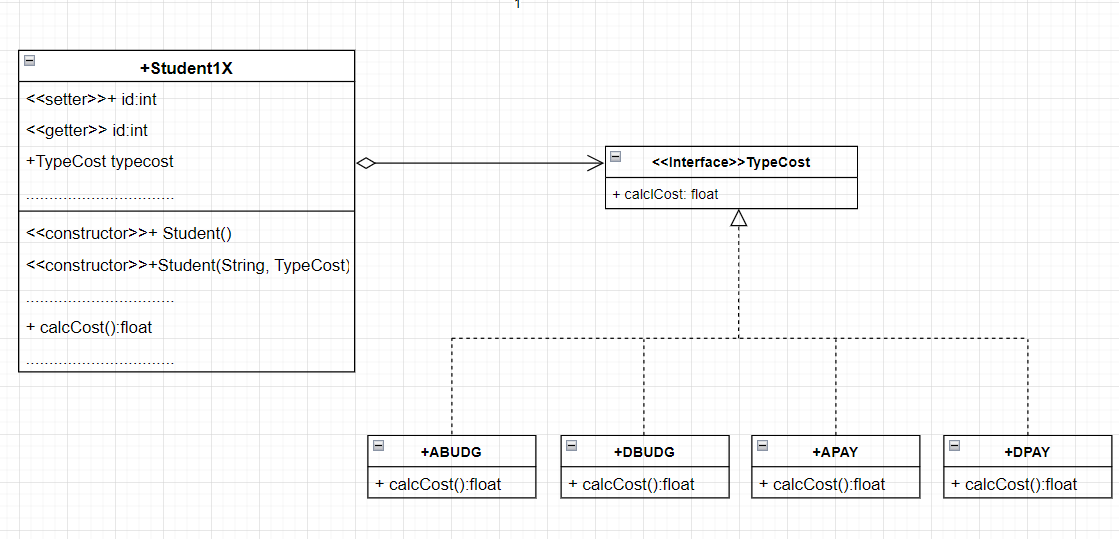
1. **Open/Closed** principle: классы должны быть расширяемы (наследование), но неизменяемы

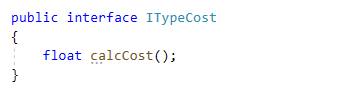


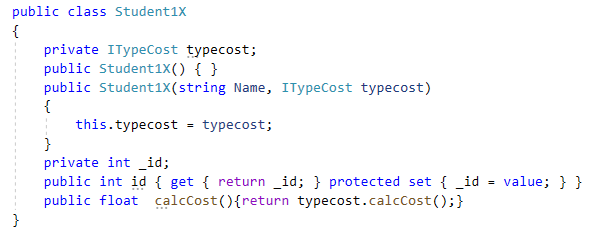


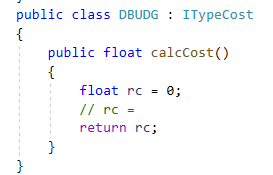
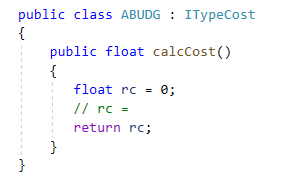


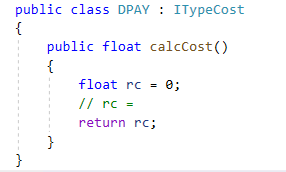
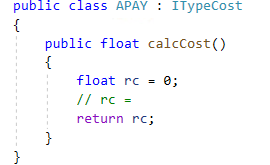
***паттерн «Стратегия», агрегация (включает)***





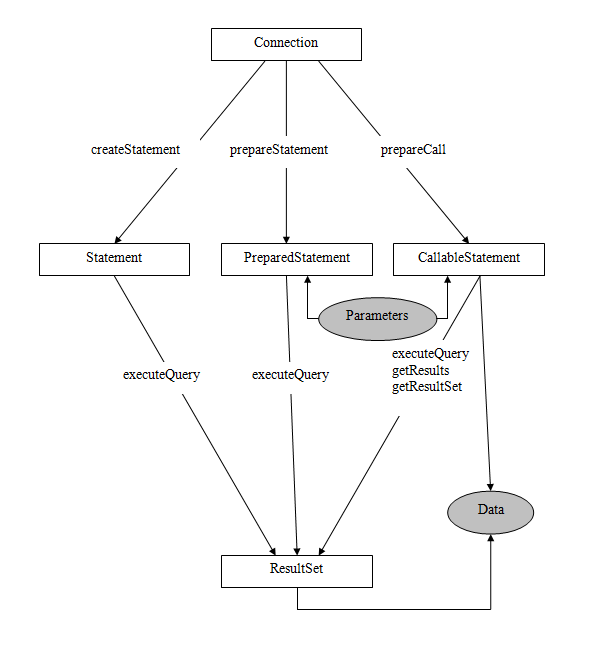
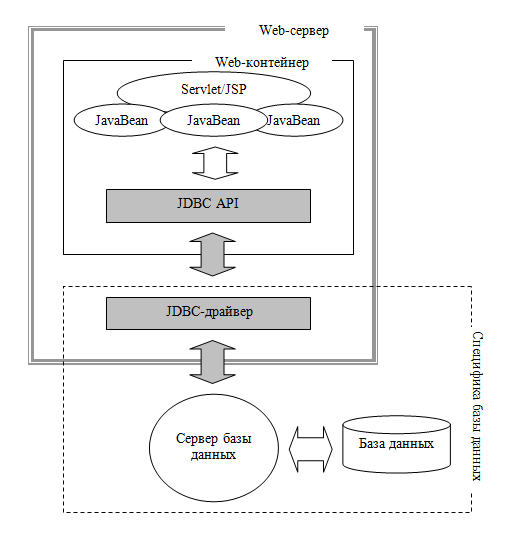


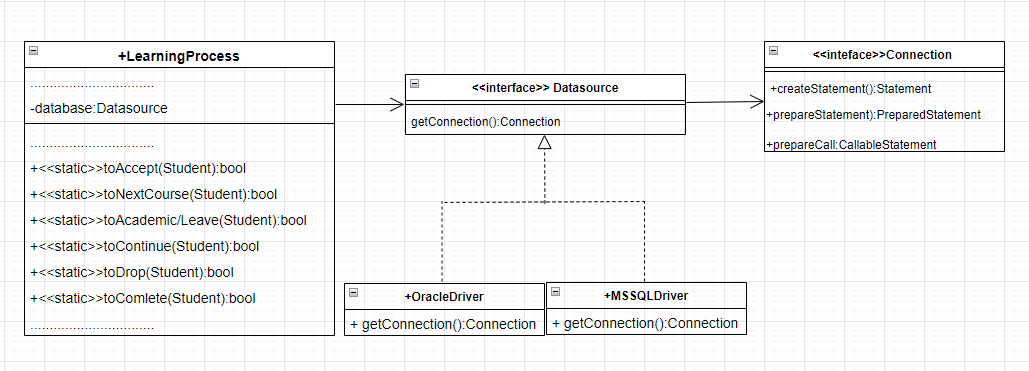






1. **Liskov Substitution Principe:** Барбара Лисков (1987) «Data Abstraction and hierarchy»
2. **Liskov Substitution Principe:** производные классы, должны быть разработаны таким образом, чтобы их можно было подставить в клиентский код вместо базового класса и при этом функциональность клиентского кода осталась бы корректной.
   * 1. типы параметров одноименных метода производного класса должны иметь такой же тип как у базового или более абстрактный;
     2. тип возвращаемого методом значения должен быть такой же как у базового или производным от возвращаемого типа методом базового класса;
     3. производный метод не должен выбрасывать исключения не свойственные базовому(можно выбрасывать производные);
     4. производный метод не должен ужесточать предусловия;
     5. метод не должен ослаблять постусловия;
     6. производный класс не изменяет инвариантов (определяющих свойств и методов), в идеале: производный класс только добавляет новые свойства или методы;
     7. производный класс не должен изменять приватные поля базового класса (рефлексия).
3. **Interface Segregation:** не должны быть интерфейсы слишком «толстые», интерфейс должен соответствовать принципу «Бритва Оккама» - «не стоит множить сущее без необходимости», можно создавать несколько интерфейсов; не должно быть реализаций интерфейсов с пустой реализацией.
4. **Dependency Inversion Principle:** инверсия зависимости, open/closed principle – с другой стороны, изменение базового класса не должно приводить к изменению производных. Надо зависимость вытаскивать выше по иерархии классов. Низкоуровневые операции – интерфейс, который применяет высокоуровневый класс. Модель: потребитель провайдер. Потребитель формирует требования, провайдер их реализует. Dependency Injection.





1. **КОНЕЦ**