ОНЛАЙН-ОБРАЗОВАНИЕ

OTUS

Подготовительный курс Java

Модуль 2.2
«Концепции ООП»

Виталий Чибриков
преподаватель

Проблема проектирования

- Повторное использование кода
- Модульность
- Тестируемость

Концепции ООП

- Наследование
- Полиморфизм
- Инкапсуляция
- Абстракция

Наследование

Новый тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

Между наследником и базовым классом существует отношение “is a”

Otus is a bird

Полиморфизм

```
Object[] array = new Object[4];
array[0] = new Object();
array[1] = “Hello!”;
array[2] = 1;
array[3] = new MyClass();

for (int i = 0; i < array.length; i++){
    array[i].toString(); // вызов метода наследника, если переопределен
}
```

Инкапсуляция

Класс разрешает доступ к своим внутренним переменным только тем методам, которые она сам определил.

Пример: пользователи класса Bird не могут поменять состояние isFlying.

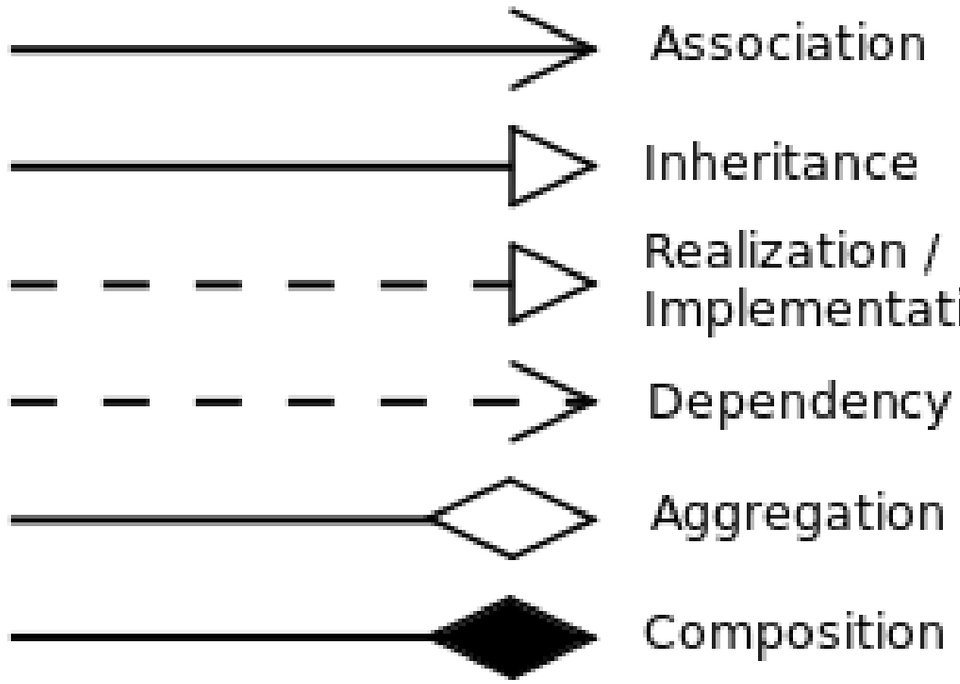
Птицу нельзя заставить взлететь.

Абстракция

Bird абстрактный класс. Нельзя создать объект класса Bird.

Абстрактные классы могут содержать абстрактные методы.
Наследники абстрактных классов обязаны определить эти методы.

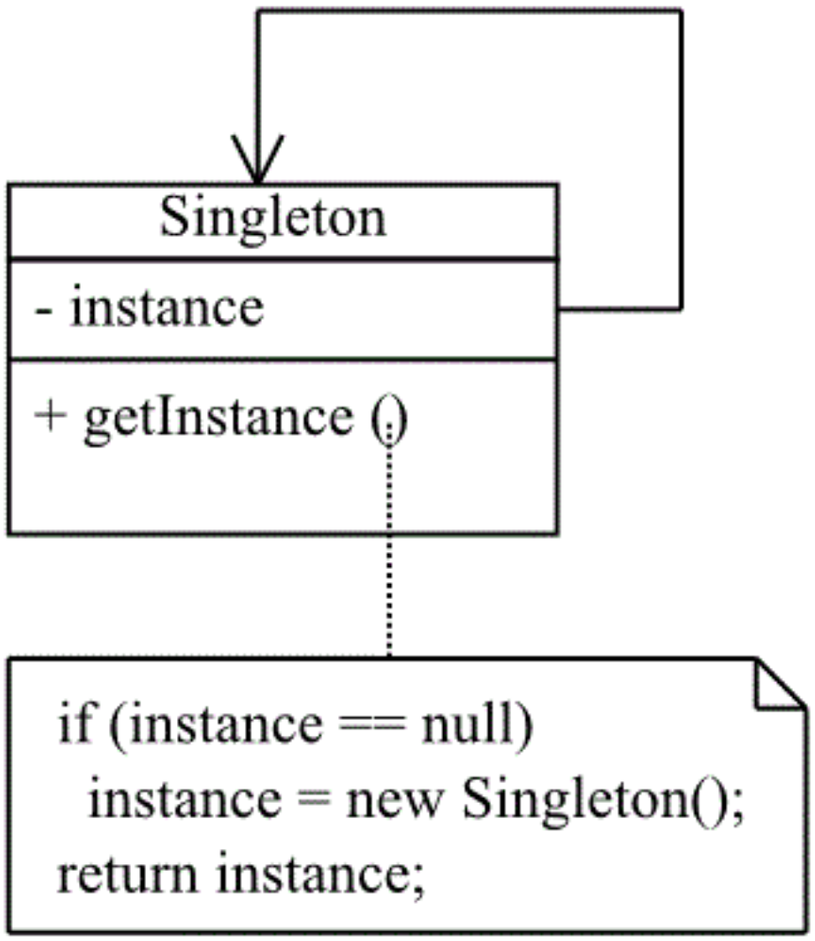
Нотация UML


Association
Inheritance
Realization /
Implementation
Dependency
Aggregation
Composition

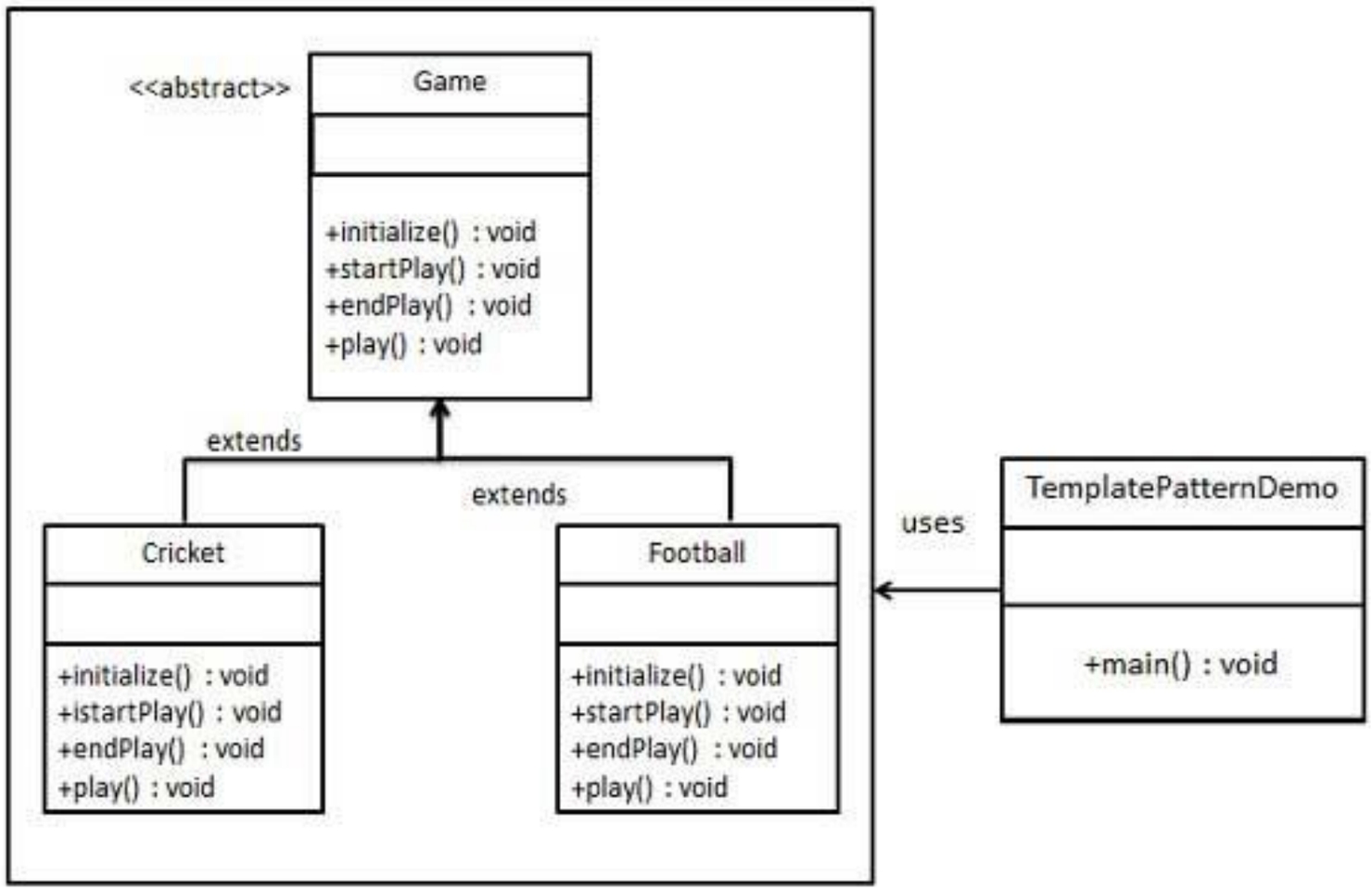
Базовые паттерны

- Singleton
- Template method
- Factory method
- Adapter
- Observer

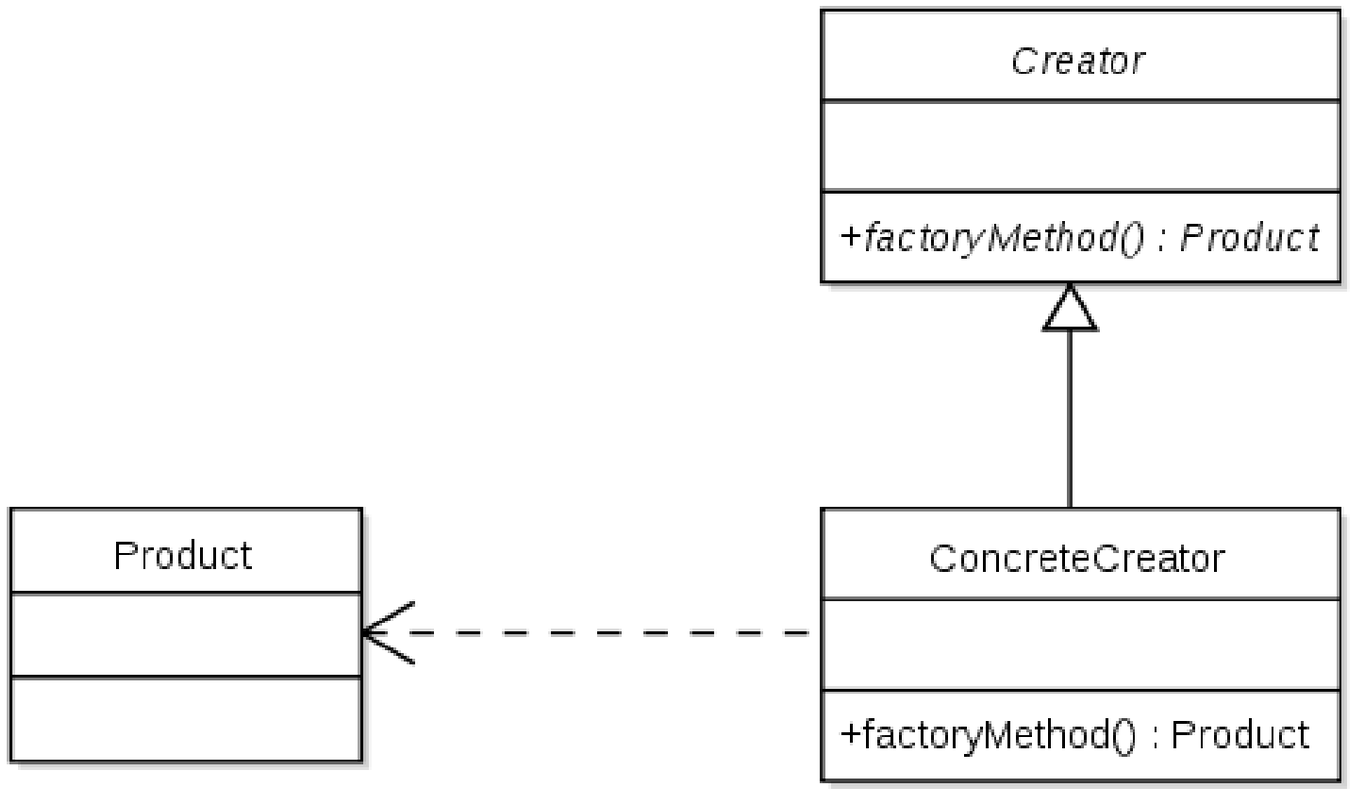
Singleton


Singleton
- instance
+ getInstance ()
if (instance == null)
instance = new Singleton();
return instance;

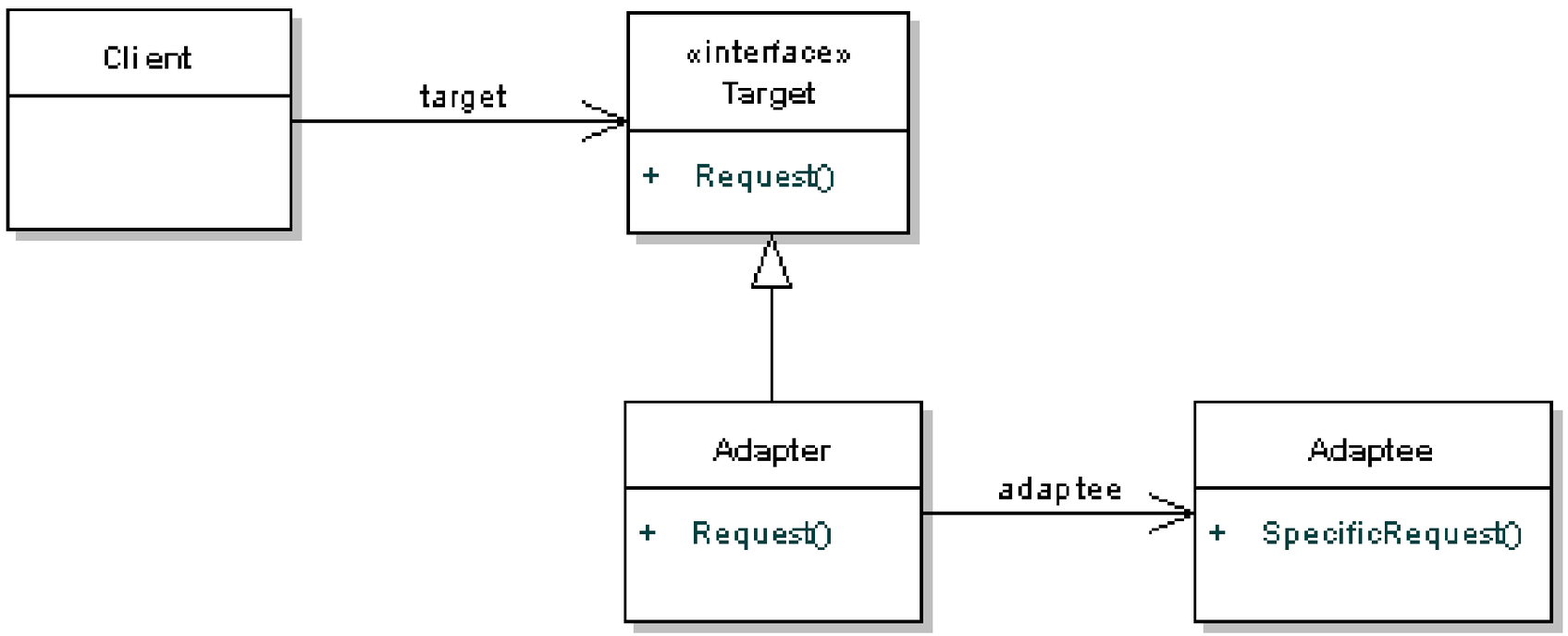
Template method



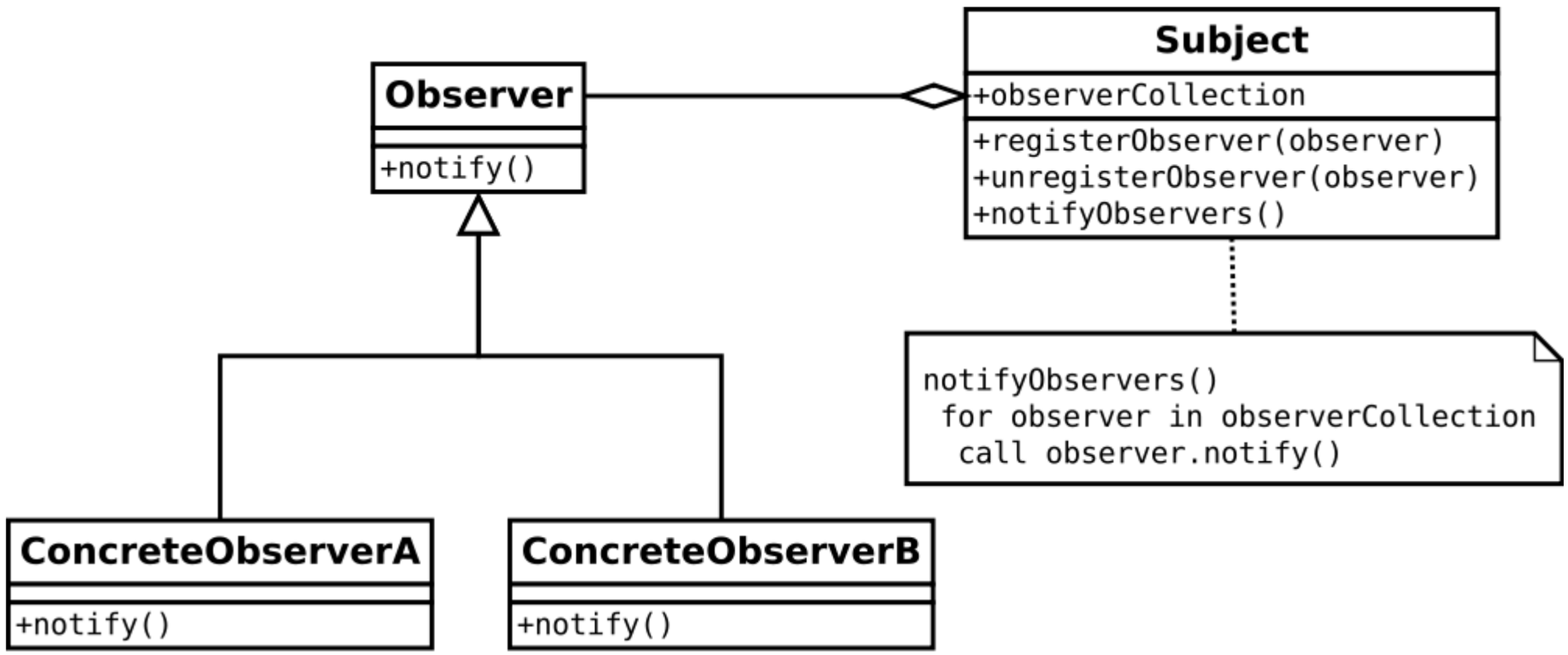
Factory method



Adapter


Client
target
«interface»
Target
+ Request()
Adapter
+ Request()
adaptee
Adaptee
+ SpecificRequest()

Observer


Subject
+observerCollection
+registerObserver(observer)
+unregisterObserver(observer)
+notifyObservers()
Observer
+notify()
notifyObservers()
for observer in observerCollection
call observer.notify()
ConcreteObserverA
+notify()
ConcreteObserverB
+notify()