УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Дата и время

ITMO more than a UNIVERSITY

Использование даты и времени

- ☑ Дата и время — 2 варианта представления:
  - Человеческое время — часы, минуты, дни, недели, месяцы
  - Машинное время — миллисекунды от нулевой точки отсчета
    - 1 января 1970 года, 00:00:00

Традиционные классы. Date

- ☑ Date
  - в версии 1.0 — единственный класс даты
  - человеческое и машинное представление
  - форматирование даты
  - версия 1.1 — Date — момент времени
  - почти все методы объявлены deprecated
- ☑ Конструкторы
  - Date
  - Date(long)
- ☑ Методы
  - long getTime()
  - boolean after(Date)
  - boolean before(Date)

TimeZone

- ☑ Временная зона — смещение от стандартного:
- ☑ до 1972 года - Гринвич (GMT)
- ☑ после 1972 — UTC — всемирное координированное
- ☑ Методы
  - getDefault()
  - getAvailableIDs()
  - getRawOffset() - смещение без учета летнего времени
  - getOffset(long date) — с учетом летнего времени

- ☑ Класс SimpleTimeZone — реализованный потомок

Calendar

- ☑ Абстрактный класс — преобразование из машинных в человеческие единицы
  - Calendar getInstance()
  - add(int field, int amount);
  - roll(int field, int amount);
  - set(int field, int value);
  - Date getTime()
  - setTime(Date)
- ☑ реализованный класс GregorianCalendar
  - сочетает 2 календаря (григорианский и юлианский)

Date/Time API

- ☑ java.time - дата, время, периоды
  - Instant, Duration, Period, LocalDate, LocalTime, LocalDateTime, OffsetTime, OffsetDateTime, ZonedDateTime
- ☑ java.time.chrono - календарные системы
- ☑ java.time.format - форматирование даты и времени
- ☑ java.time.temporal - единицы измерения и отдельные поля
- ☑ java.time.zone - временные зоны и правила

Дни недели и месяцы (enums)

- ☑ enum DayOfWeek (1 (MONDAY) — 7 (SUNDAY))
- ☑ enum Month (1 (JANUARY) — 12 (DECEMBER))
- ☑ метод getDisplayName(style, locale)
- ☑ стиль — FULL, NARROW, SHORT / STANDALONE

Представление даты и времени

- ☑ Year
- ☑ YearMonth
- ☑ MonthDay
- ☑ LocalDate
- ☑ LocalTime
- ☑ LocalDateTime

Соглашения по именам методов

- ☑ Статические
  - of — создает экземпляр на основе входных параметров
    - LocalDate.of(year, month, day), ofYearDay(year, dayOfYear)
  - from — конвертирует экземпляр из другого типа
    - LocalDate.from(LocalDateTime)
  - parse — создает экземпляр из строкового представления
    - LocalDate.parse("2022-02-22")

Соглашения по именам методов

☑ Методы экземпляра

- format — форматирует объект в строку
- get — возвращает поля объекта // getHours()
- with — возвращает копию с изменением // withYear(2021)
- plus — возвращает копию с добавлением // plusDays(2)
- minus — возвращает копию с убавлением // minusWeeks(3)
- to — преобразует объект в другой тип // toLocalTime()
- at — комбинирует объект с другим // date.atTime(time)

Временная зона

- ☑ ZoneId — идентификатор зоны
  - Europe/Moscow
- ☑ ZoneOffset — разница со стандартным временем
  - UTC+01:00, GMT-2
- ☑ OffsetTime = LocalTime + ZoneOffset
- ☑ OffsetDateTime = LocalDateTime + ZoneOffset
- ☑ ZonedDateTime = LocalDateTime + ZoneId
  - использует java.time.zone.ZoneRules

Момент времени

- ☑ Класс Instant
  - now()
  - plusNanos()
  - plusMillis()
  - plusSeconds()
  - minusNanos()
  - minusMillis()
  - minusSeconds()

Разбор и форматирование

- ☑ java.time.format.DateTimeFormatter
  - формат можно выбрать из констант:
    - BASIC_ISO_DATE
    - ISO_DATE/TIME/DATETIME
    - ISO_LOCAL_DATE/TIME/DATETIME
    - ISO_OFFSET_DATE/TIME/DATETIME
    - ISO_ZONED_DATETIME
    - ISO_INSTANT
  - задать шаблон
    - ofPattern()
  - методы format() и parse()

Периоды даты и времени

- ☑ Duration — продолжительность в часах и менее
  - toNanos(), toMillis(), toSeconds(), toMinutes(), toHours(), toDays()
- ☑ Period — период в днях и более
  - getDays(), getMonths(), getYears()
- ☑ .between()
- ☑ .plus
- ☑ .minus

Старые и новые классы

Соответствия:

- Date — Instant
- GregorianCalendar — ZonedDateTime
- TimeZone — ZoneId, ZoneOffset

☑ Методы:

- Calendar.toInstant()
- GregorianCalendar.toZonedDateTime()
- GregorianCalendar.fromZonedDateTime()
- Date.fromInstant()
- Date.toInstant()
- TimeZone.toZoneId()

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Шаблоны проектирования

ITMO more than a UNIVERSITY

Основные принципы проектирования

- ☑ Повторное использование кода
  - DRY - Don't repeat yourself
  - стандартные библиотеки
  - фреймворки
- ☑ Расширяемость
  - Учет возможных будущих изменений

Все изменяется

☑ Изменяются

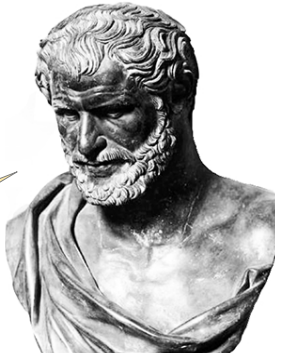
- пожелания пользователя
- взгляд разработчика
- внешние условия

} Программы меняются

Овидий

omnia mutantur

πάντα ρεῖ



Гераклит

Шаблоны проектирования

УНИВЕРСИТЕТ ИТМО

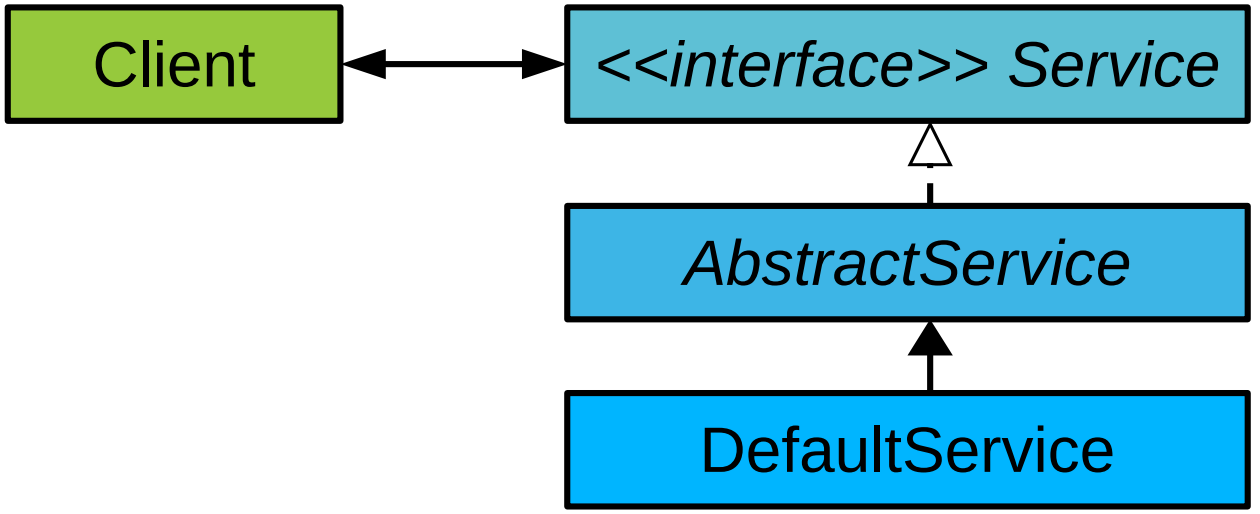
- ☑ Кто-то когда-то уже делал что-то похожее
- ☑ Пришлось вносить изменения - возникли проблемы
- ☑ Нужно сразу было делать по-другому!

GoF book

Java

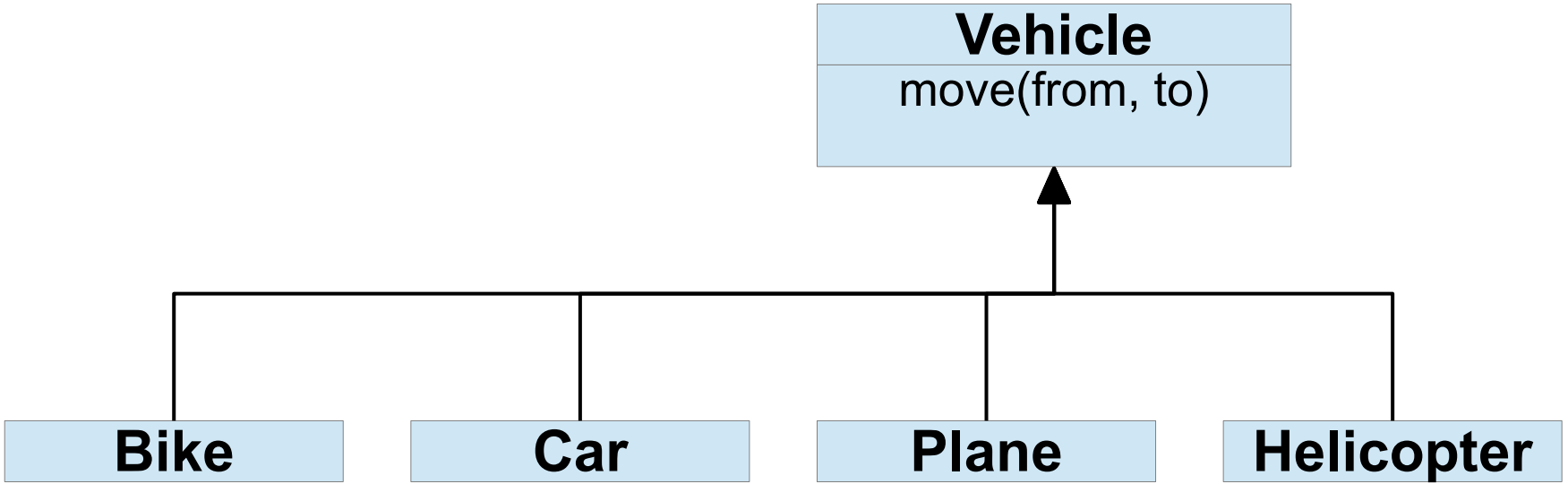
Базовые принципы - интерфейсы

- Взаимодействие с абстракцией - ИНТЕРФЕЙС



Наследование

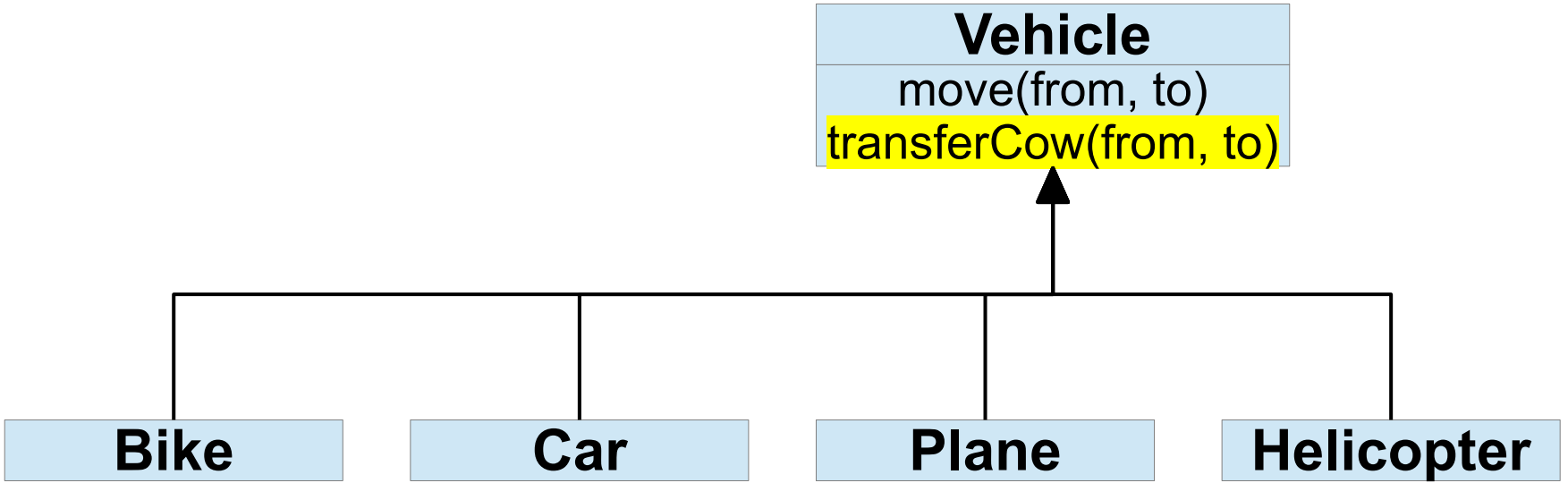
- ☑ Умеет предок - умеют потомки
- ☑ Полиморфизм



Java

Наследование

☑ Перевозка коровы - наследование??



Java

Перевозка коровы

(С) х/ф «Мимино»

Перевозка коровы

(С) х/ф «Особенности национальной охоты»

Перевозка коровы

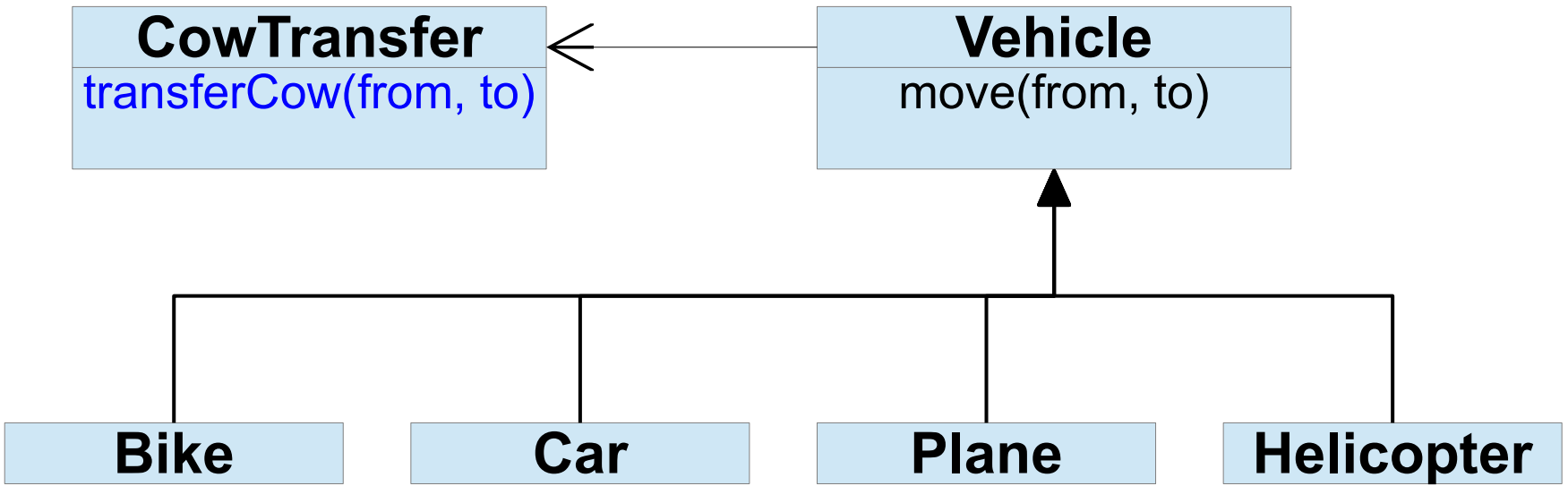
Java

Перевозка коровы

Java

Наследование / делегирование

☑ Перевозка коровы - делегирование



Java

Базовые принципы - делегирование

☑ Наследование

- статическое отношение
- классификация

```
class Animal {
  sleep() { ... }
}

class Cat extends Animal {
}

Animal cat = new Cat();
cat.sleep();
```

☑ Делегирование

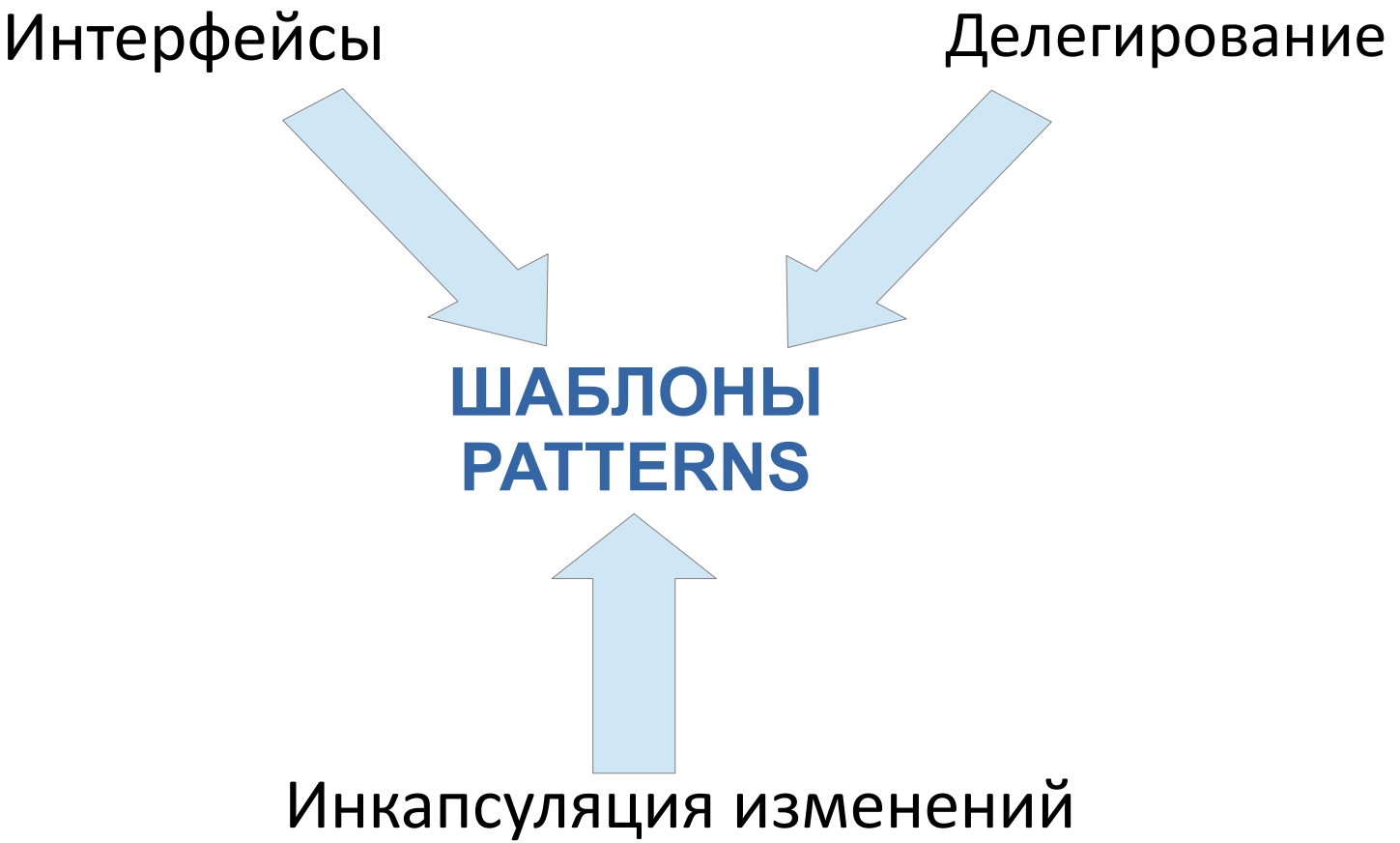
- динамическое отношение
- роль

```
class Vehicle {
  move(from, to) { ... }
}
class CowTransfer {
  Vehicle v; Cow c;
  transfer(from, to) {
    load(c); v.move(from, to);
    unload(c);
  }
}
```

Инкапсуляция изменений

- ☑ Отделить изменяющееся от постоянного
- ☑ Инкапсулировать изменяющееся

Как справиться с изменениями?



Java

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Порождающие шаблоны

ITMO more than a UNIVERSITY

Порождающие шаблоны

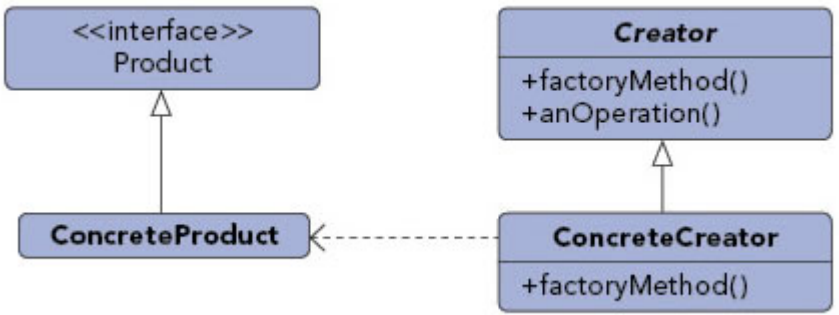
УНИВЕРСИТЕТ ИТМО

- ☑ Factory Method — Фабричный метод
- ☑ Abstract Factory — Абстрактная фабрика
- ☑ Builder — Строитель
- ☑ Prototype — Прототип
- ☑ Singleton - Одиночка
- ☑ Object Pool — Пул объектов

Java

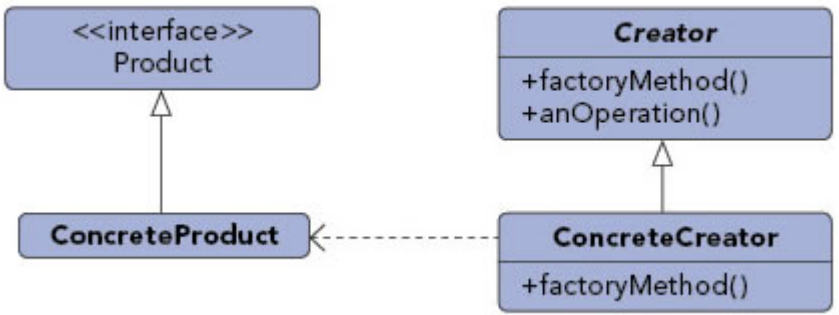
Factory Method

- ☑ Определяет интерфейс для создания объекта
- ☑ Делегирует создание экземпляров подклассам
- ☑ Инкапсулирует код создания конкретных продуктов



Factory Method

- ☑ Убирает зависимость от конкретных продуктов
- ☑ Упрощает добавление новых продуктов
- ☑ На каждый продукт нужен свой создатель



Factory Method

☑ Примеры:

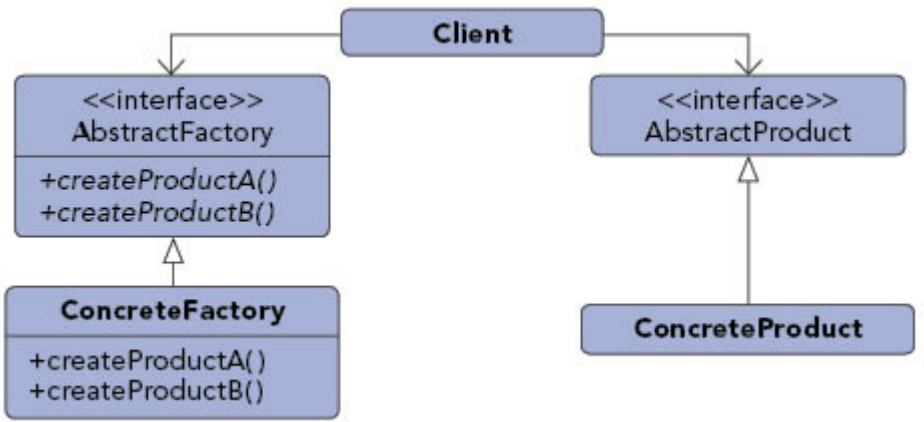
```
Connection conn  = DriverManager.getConnection(args);
Statement stat = conn.createStatement();
```

Abstract Factory

УНИВЕРСИТЕТ ИТМО

☑ Создание семейства объектов

```
Factory f1 = new PepsiFactory();
Product p1 = f.createColaDrink();
Factory f2 = new CocacolaFactory();
Product p2 = f.createOrangeDrink();
```

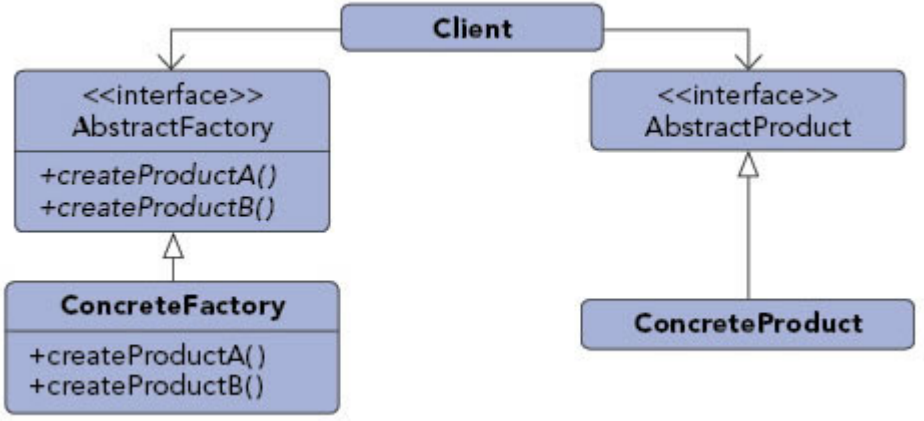


Java

Abstract Factory

УНИВЕРСИТЕТ ИТМО

- ☑ Устраняет зависимость от конкретных продуктов
- ☑ Гарантирует совместимость продуктов в наборе
- ☑ Требуется наличие всех типов продуктов в семействе
- ☑ Более сложный код



Java

Builder

☑ Создание объектов со сложным набором параметров

```
class Trip {
  public Trip(Date date, Location location)
  public Trip(Date date, Transport transport)
  public Trip(Date date, Location location, Transport transport)
  public Trip(Date date, Location location, int guests)
  public Trip(Date from, Date until, Location localtion)
  ....
}
```

Builder

УНИВЕРСИТЕТ ИТМО

☑ Создание объектов со сложным набором параметров

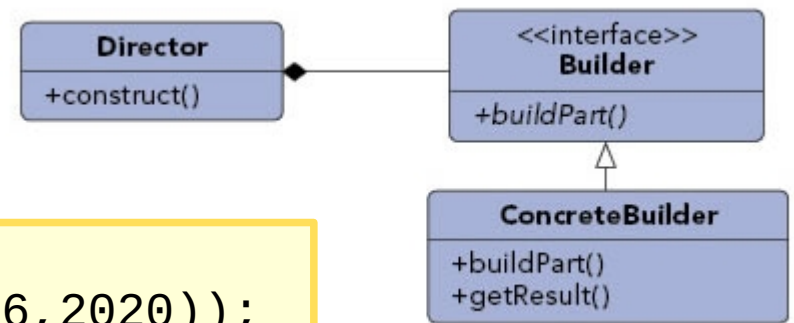
```
class Trip {
  public Trip()
  public setLocation(Location)
  public setDate(Date)
  public setTransport(Transport)
  ....
}
```

Builder

УНИВЕРСИТЕТ ИТМО

☑ Создание объектов со сложным набором параметров

```
Builder b = new Builder()
   .setWhere("Moscow")
   .setTransport("train")
   .setDate(1,6,2020);
Trip trip = b.create();
```

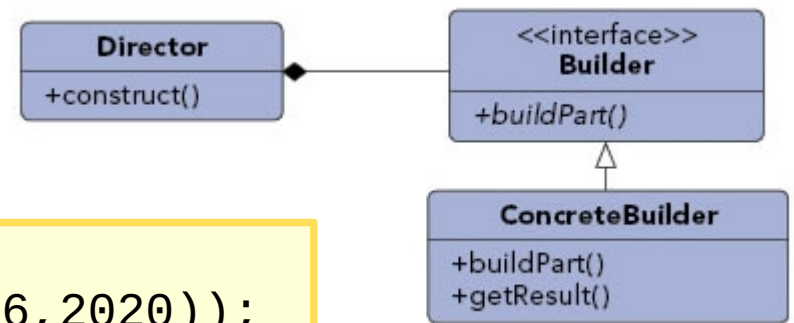



```
Director d = new Director();
p = d.makeTrainTrip("Moscow", new Date(1,6,2020));
```

Builder

- ☑ Изоляция сложной сборки от бизнес-логики
- ☑ Разрешает пошаговое создание
- ☑ Усложнение структуры программы (+ строитель)

```
Builder b = new Builder()
   .setWhere("Moscow")
   .setTransport("train")
   .setDate(1,6,2020);
Trip trip = b.create();
```

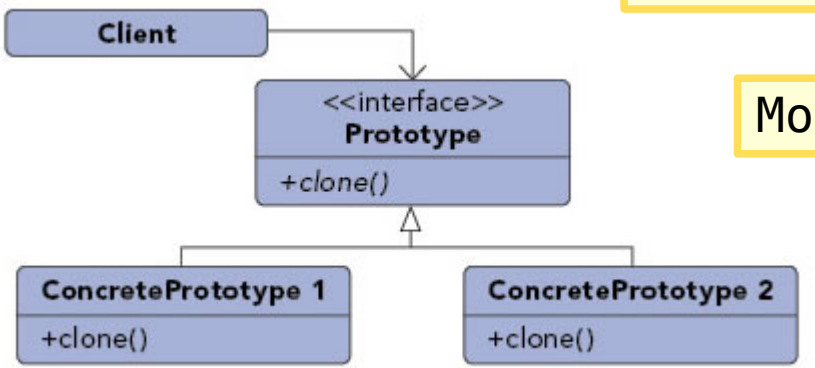



```
Director d = new Director();
p = d.makeTrainTrip("Moscow", new Date(1,6,2020));
```

Prototype

☑ Создание объектов клонированием

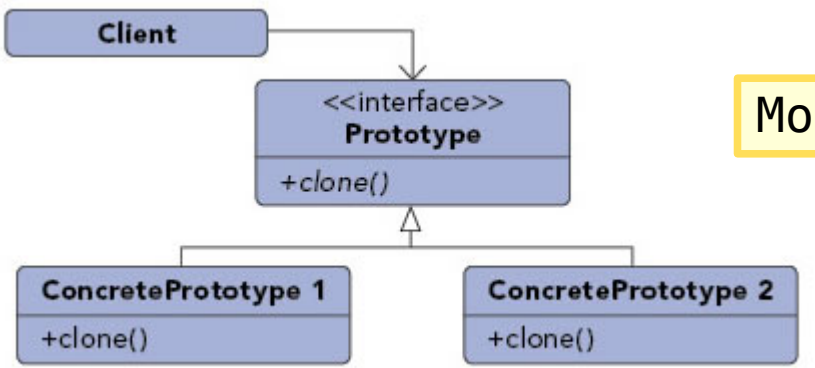
```
Monster boss = new Monster();
boss.setHP(10000);
boss.setSpeed(1000);
boss.setPower(5000);
boss.setAttack("Terminate");
```




```
Monster another = boss.clone();
```

Prototype

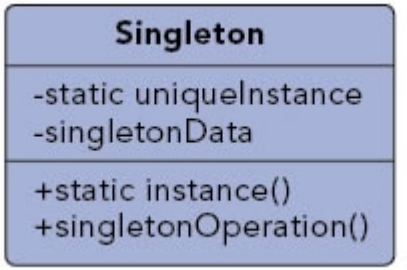
- ☑ Клонирование без зависимости от конкретных классов
- ☑ Позволяет хранить готовый набор сложных объектов
- ☑ Сложно реализовывать глубокое клонирование




```
Monster another = boss.clone();
```

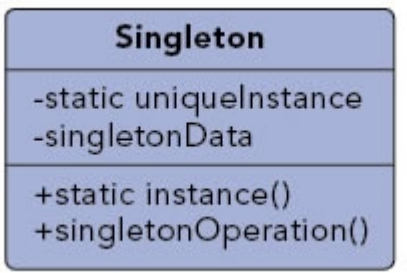
Singleton

- ☑ Ограничение количества экземпляров класса
  - private конструктор
  - статический метод для вызова конструктора
- ☑ 1 объект - Singleton
  - System.console()
- ☑ более 1 объекта — Object Pool
  - ThreadPool



Singleton

- ☑ Контроль количества объектов
- ☑ Возможность отложенной инициализации
- ☑ Неудобное наследование



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Структурные шаблоны

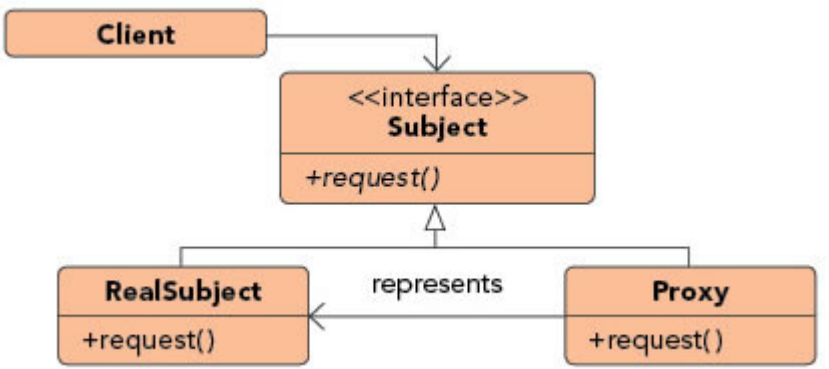
ITMO more than a UNIVERSITY

Структурные шаблоны

- ☑ Adapter — Адаптер
- ☑ Bridge — Мост
- ☑ Composite - Компоновщик
- ☑ Decorator - Декоратор
- ☑ Facade - Фасад
- ☑ Flyweight - Легковес
- ☑ Proxy - Заместитель

Proxy

- ☑ Делегирование с тем же интерфейсом
  - Создание объекта требует времени
  - Создание объекта может не потребоваться
  - Реальный объект может временно отсутствовать
  - Перехват методов

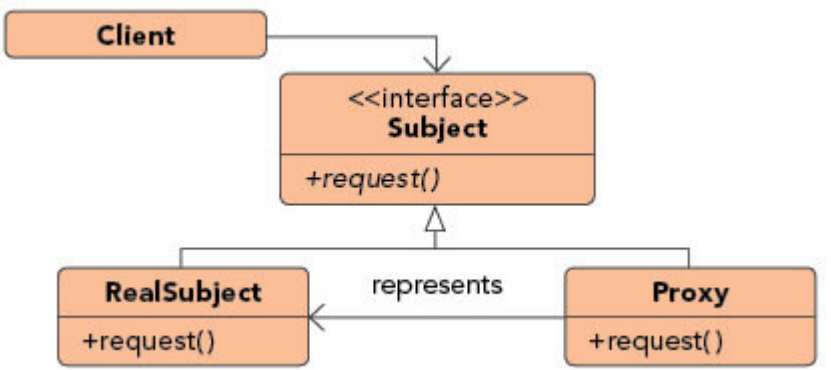


Proxy

☑ Делегирование с тем же интерфейсом

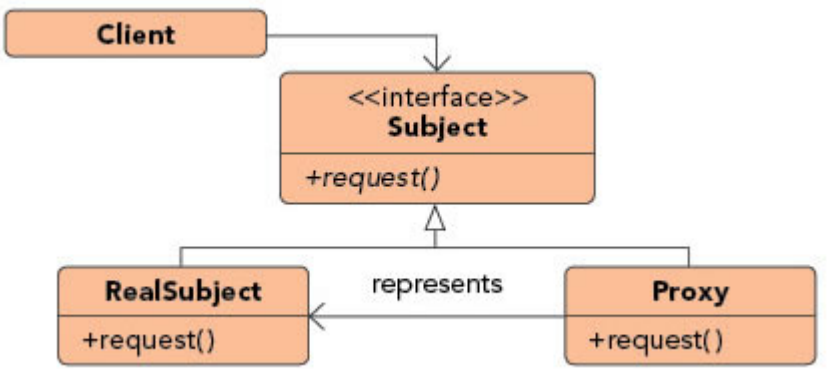
```
class Proxy implements Subject {
  RealSubject real;
  public request() {
    return real.request();
  }
}
```

```
client:
subject.request();
```



Proxy

- ☑ Контроль работы с реальным объектом
- ☑ Клиент не знает, используется прокси или нет
- ☑ Замедление и усложнение доступа



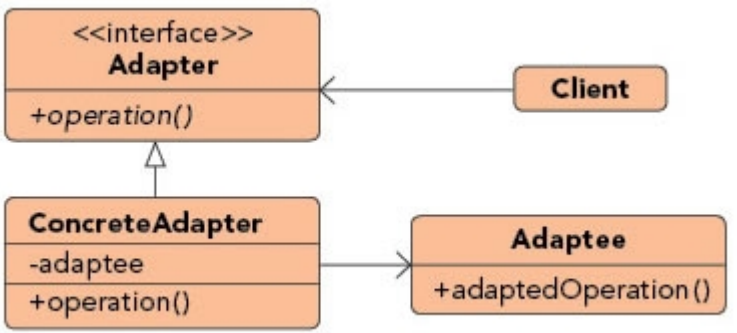
Adapter

УНИВЕРСИТЕТ ИТМО

☑ Делегирование с заменой интерфейса

```
class EUPlug {
  connect(EUSocket s) {
    s.getEUPower();
  }
}

interface UKSocket {
  getUKPower()
}
```



Java

Adapter

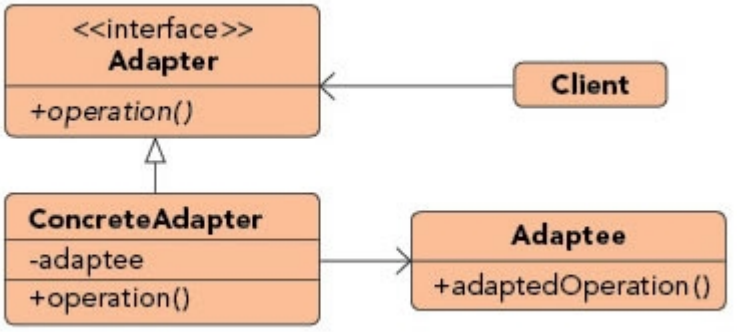
УНИВЕРСИТЕТ ИТМО

☑ Делегирование с заменой интерфейса

```
class EUPlug {
  connect(EUSocket s) {
    s.getEUPower();
  }
}

interface UKSocket {
  getUKPower()
}
```

```
class Adapter implements EUSocket {
  UKSocket uks;
  getEUPower() {
    return uks.getUKPower();
  }
}
```

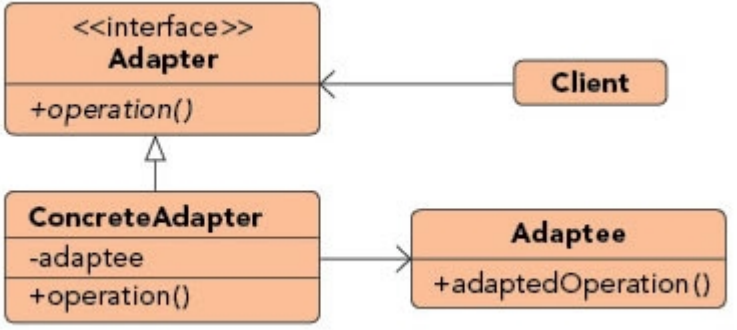


Java

Adapter

УНИВЕРСИТЕТ ИТМО

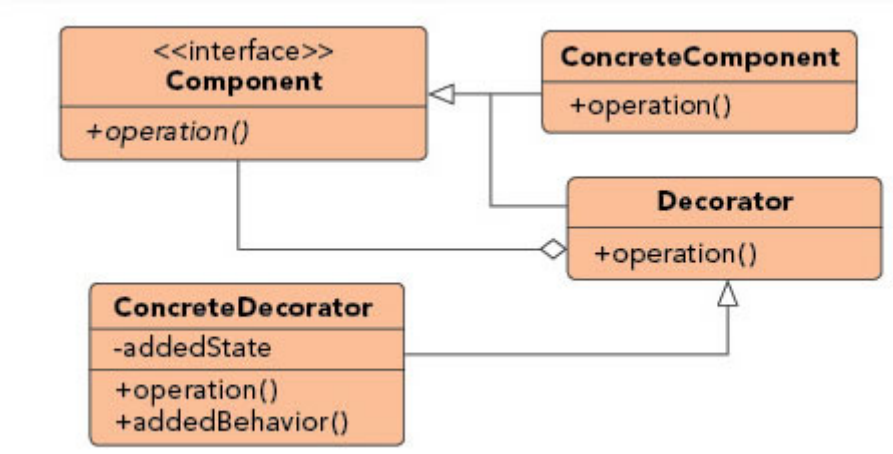
- ☑ Позволяет устранить несовместимость интерфейсов
- ☑ Не нужно переписывать существующий код
- ☑ Усложнение структуры (+ адаптер)



Java

Decorator

- ☑ Делегирование с расширением функциональности
- ☑ Блины с добавками в Теремке
- ☑ InputStream
  - BufferedInputStream,
  - LineNumberInputStream,
  - …

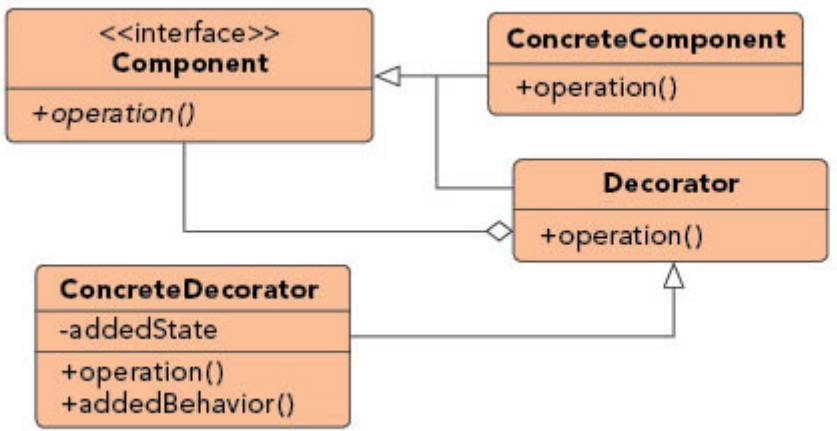


Decorator

УНИВЕРСИТЕТ ИТМО

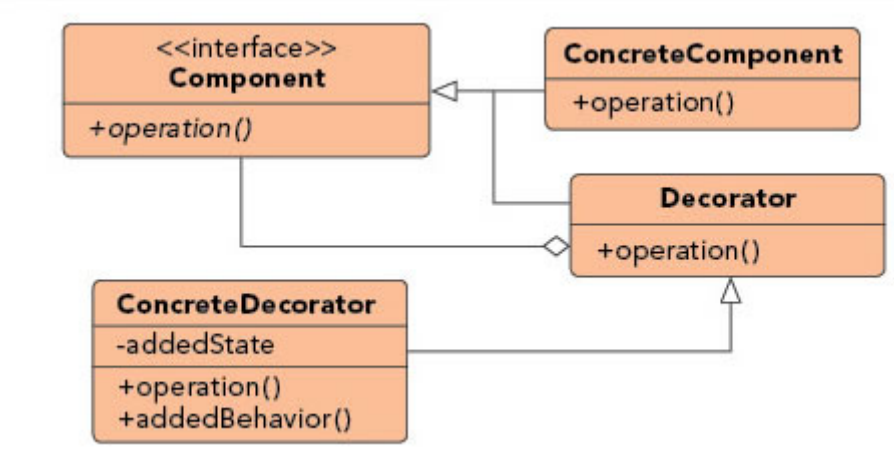
☑ Делегирование с расширением функциональности

```
class Decorator implements Component {
  ConcreteComponent cmp;
  public operation() {
    cmp.operation();
    doMore();
  }
}
```



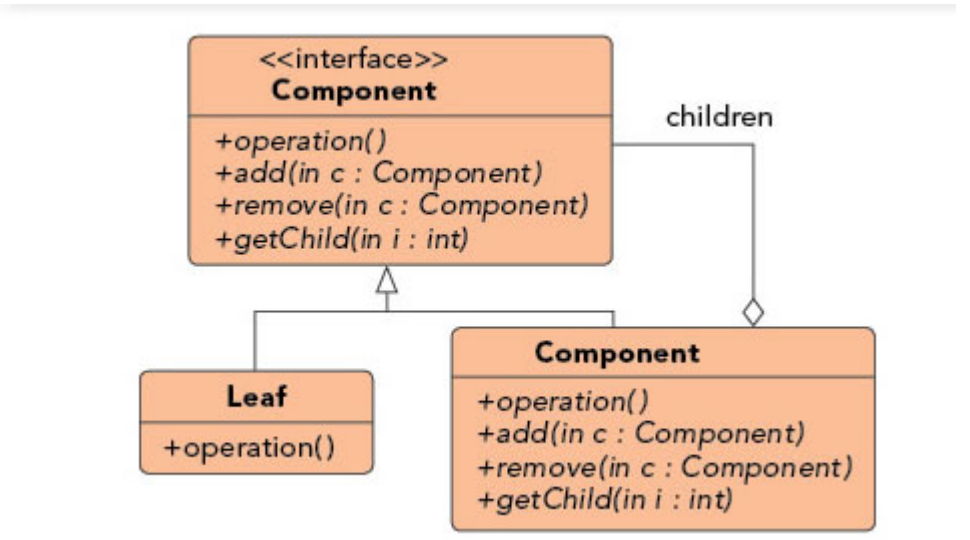
Decorator

- ☑ Позволяет добавлять функциональность динамически
- ☑ Вместо большой иерархии - несколько декораторов
- ☑ Сложность конфигурирования



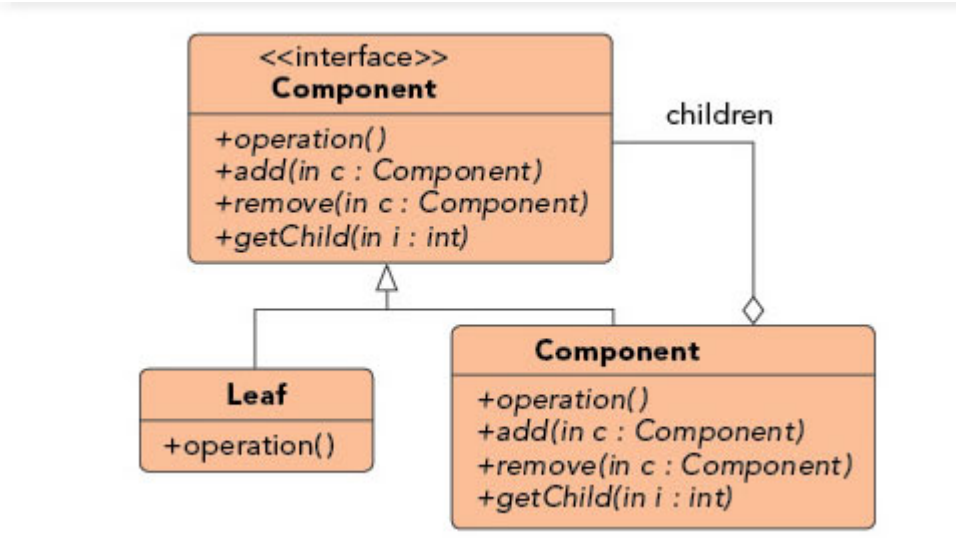
Composite

- ☑ Однообразная обработка древовидных структур
- ☑ GUI: контейнер — это компонент, на котором расположены другие компоненты



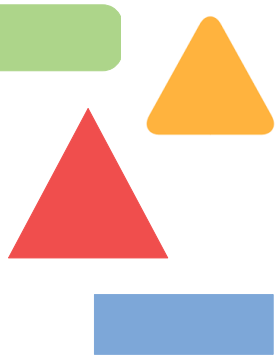
Composite

- ☑ Простая работа с деревом компонентов
- ☑ Простое добавление новых компонентов
- ☑ Слишком общий дизайн

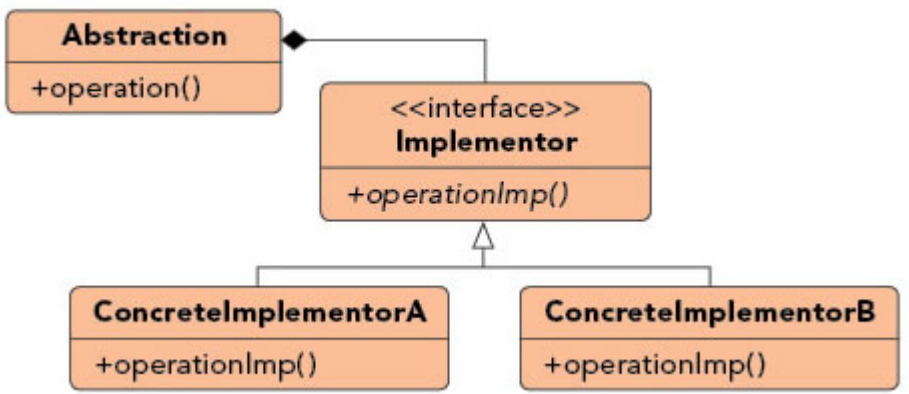


Java

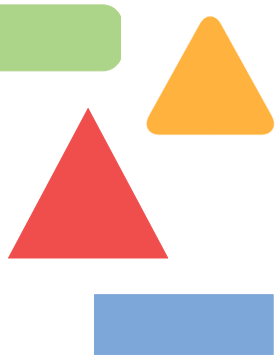
Bridge



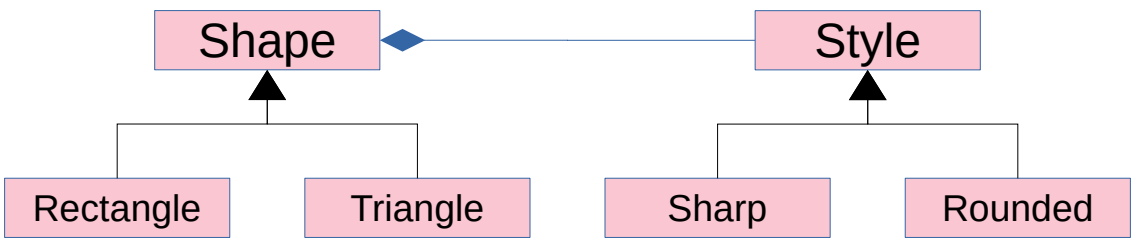
- ☑ Разделяет иерархии абстракций и реализаций
  - Абстракции - фигуры (треугольник, прямоугольник)
  - Реализации - стиль углов (обычные, закругленные)
- ☑ Поведение делегируется реализации



Bridge



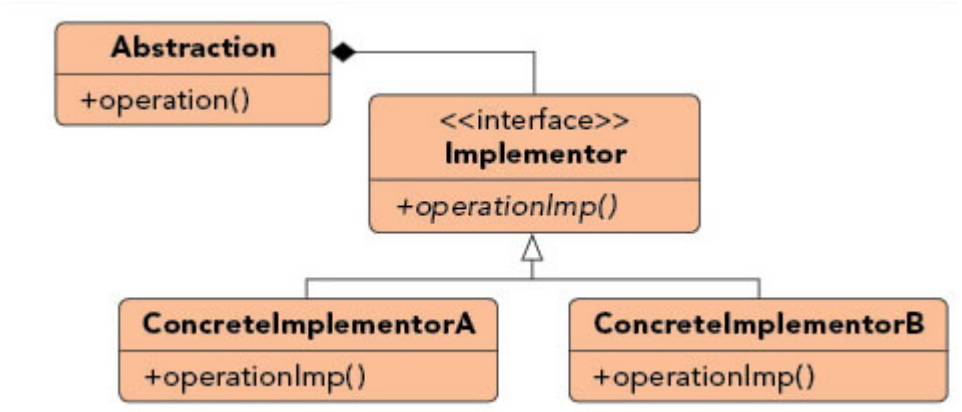
- ☑ Разделяет иерархии абстракций и реализаций
  - Абстракции - фигуры (треугольник, прямоугольник)
  - Реализации - стиль углов (обычные, закругленные)
- ☑ Поведение делегируется реализации




```
abstract class Shape {
 Style style;
 draw() {
   style.line();
 }
}
abstract class Style {
 line();
}
```

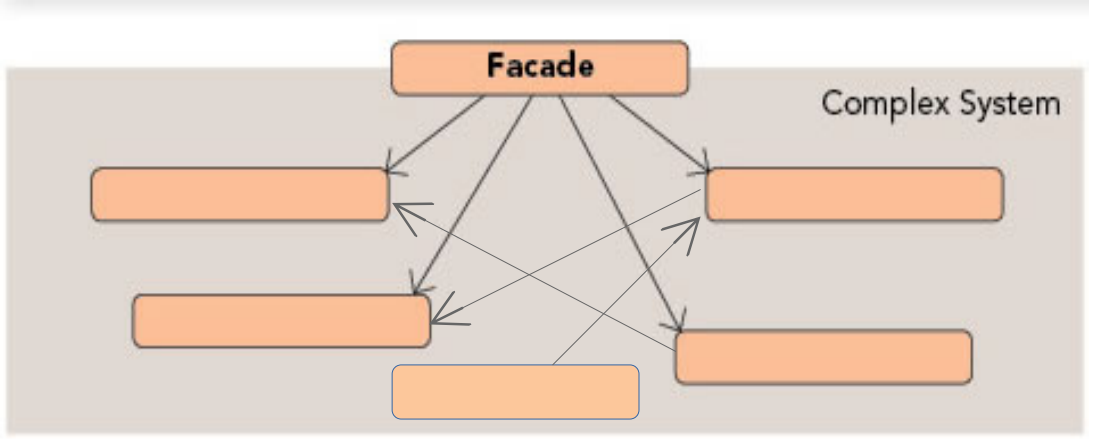
Bridge

- ☑ Повышение расширяемости
- ☑ Устраняет сложность поддержки нескольких иерархий
- ☑ Появление дополнительных классов



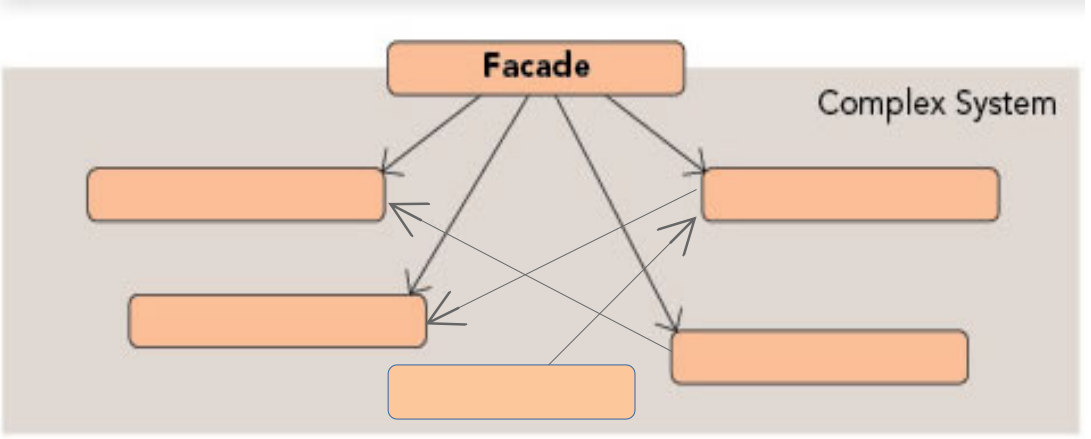
Facade

- ☑ Простой интерфейс к сложным системам
- ☑ Facade.start()



Facade

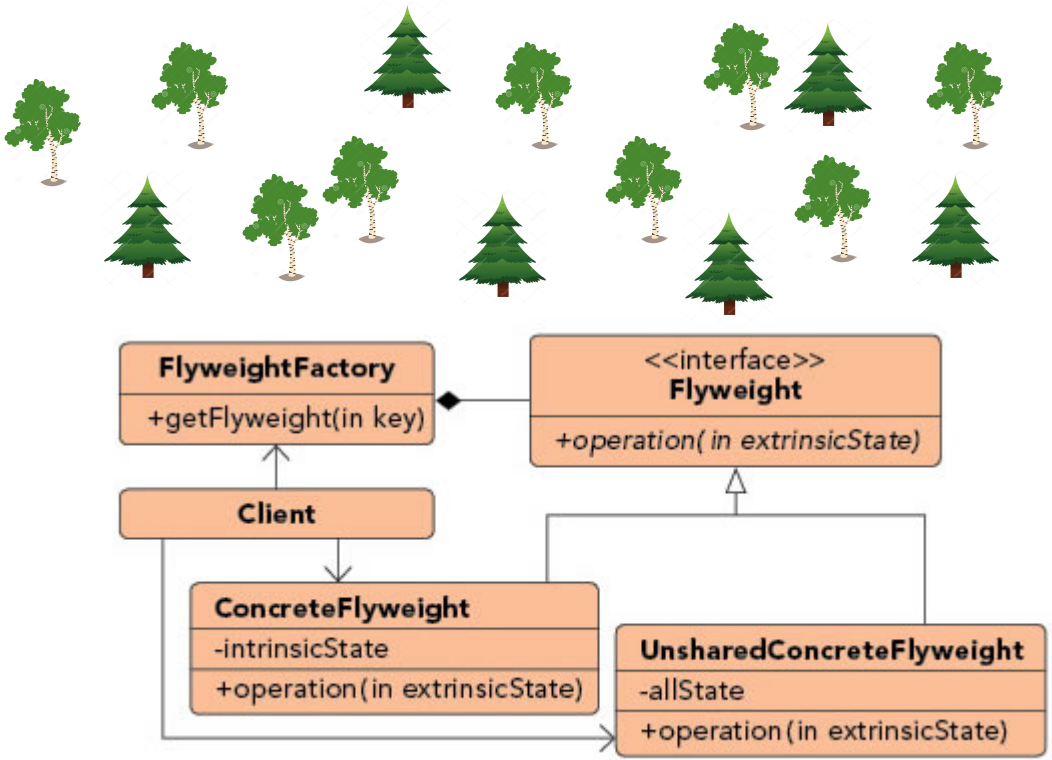
- ☑ Снижение зависимости клиента и компонентов системы
- ☑ Упрощает внешнее управление сложным объектом
- ☑ Есть опасность создать "Божественный объект"



Flyweight

- ☑ Представление множества объектов одним для экономии памяти

```
class Tree {
 int x, y;
 Color rgb;
 Image image;
}
new Tree(10, 20, (0, 256, 0),
'birch.png');
new Tree(20, 30, (0, 256, 0),
'pine.png');
```

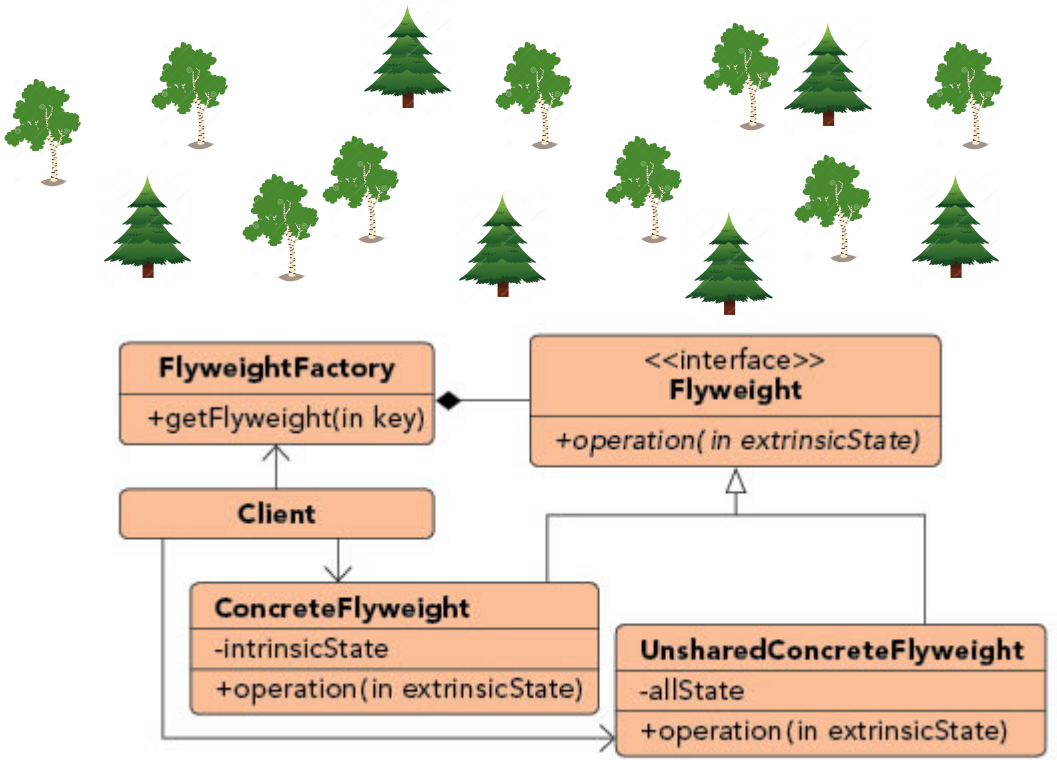


Flyweight

- ☑ Представление множества объектов одним для экономии памяти

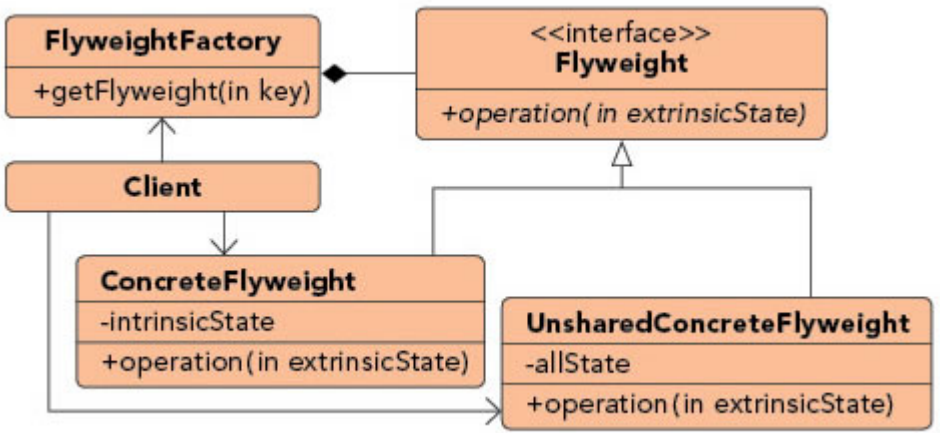
```
class Tree {
 Color rgb;
 Image image;
}
Tree birch = new Tree(...);
Tree pine = new Tree(...);

setTree(birch, 10, 20);
setTree(birch, 20, 30);
setTree(pine, 35, 45);
setTree(pine, 50,70);
```



Flyweight

- ☑ Экономия памяти
- ☑ Усложнение структуры



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Поведенческие шаблоны

ITMO more than a UNIVERSITY

Поведенческие шаблоны

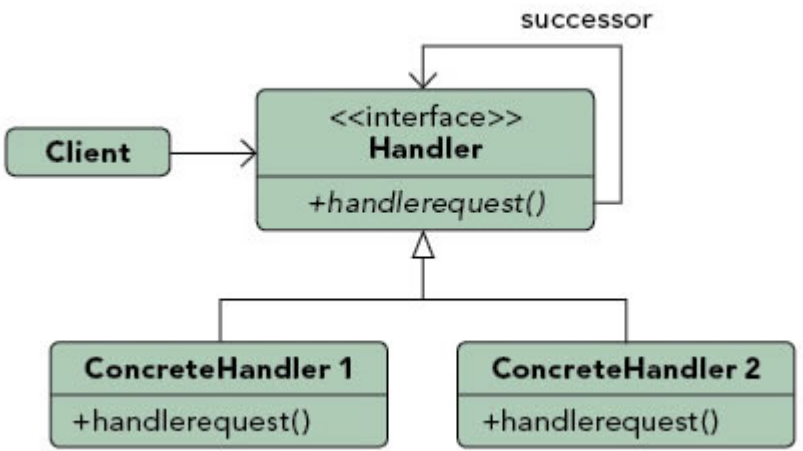
УНИВЕРСИТЕТ ИТМО

- ☑ Chain of Responsibility — Цепочка обязанностей
- ☑ Command - Команда
- ☑ Interpreter - Интерпретатор
- ☑ Iterator - Итератор
- ☑ Mediator - Посредник
- ☑ Memento - Хранитель
- ☑ Observer - Наблюдатель
- ☑ State - Состояние
- ☑ Strategy - Стратегия
- ☑ Template Method — Шаблонный метод
- ☑ Visitor - Посетитель

Java

Chain of Responsibility

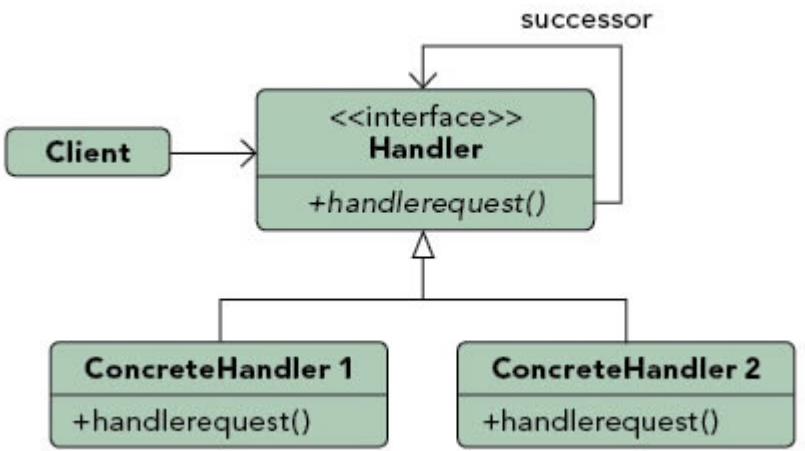
- ☑ Передача запроса по цепочке обработчиков
- ☑ Звонок в тех. поддержку



Java

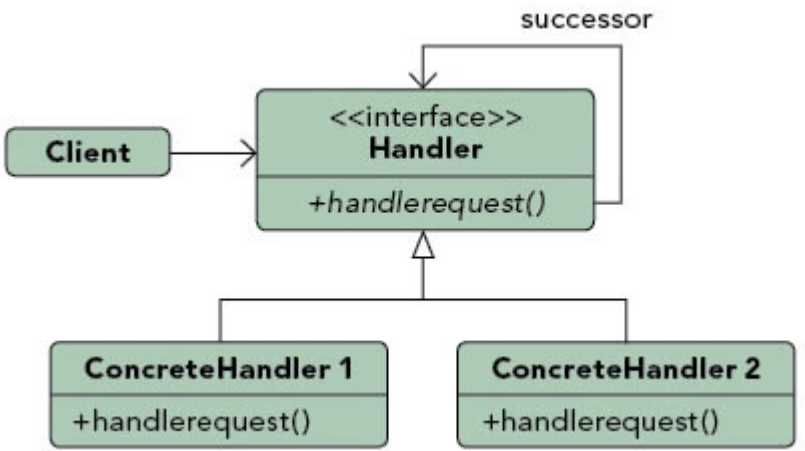
Chain of Responsibility

- ☑ Передача запроса по цепочке обработчиков
- ☑ Звонок в тех. поддержку
  1) взять кредит — нажмите 1, застряла карта — нажмите 2



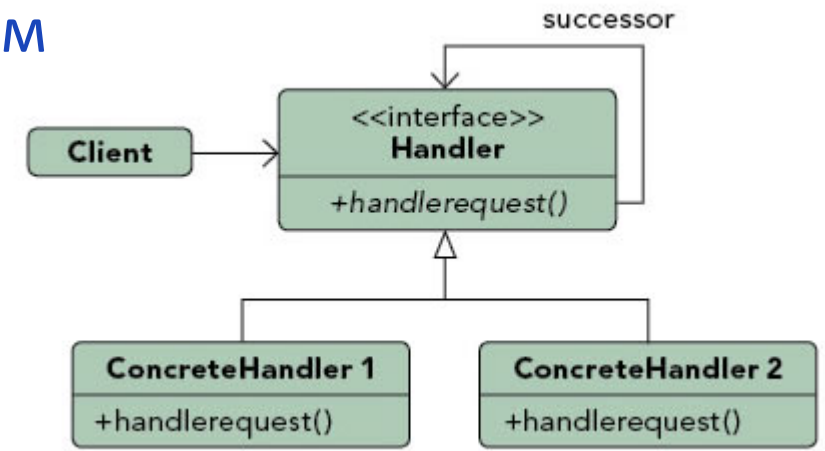
Chain of Responsibility

- ☑ Передача запроса по цепочке обработчиков
- ☑ Звонок в тех. поддержку
  1) взять кредит — нажмите 1, застряла карта — нажмите 2
  2) нажмите кнопку "сброс" на банкомате 2 раза



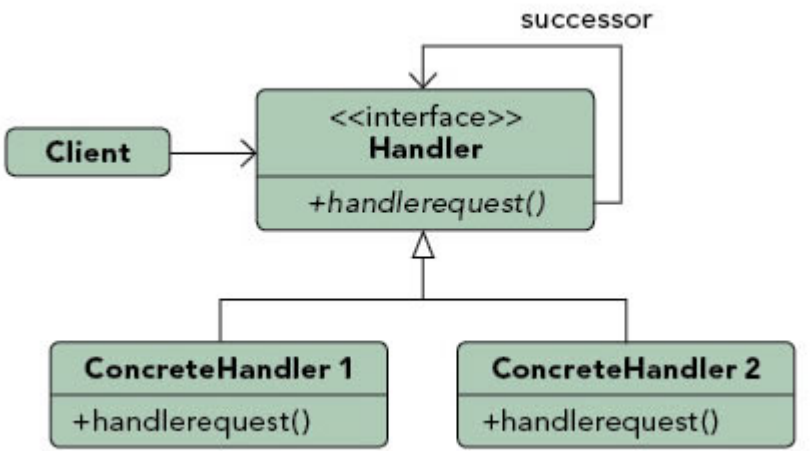
Chain of Responsibility

- ☑ Передача запроса по цепочке обработчиков
- ☑ Звонок в тех. поддержку
  1) взять кредит — нажмите 1, застряла карта — нажмите 2
  2) нажмите кнопку "сброс" на банкомате 2 раза
  3) стойте рядом, сейчас приедем



Chain of Responsibility

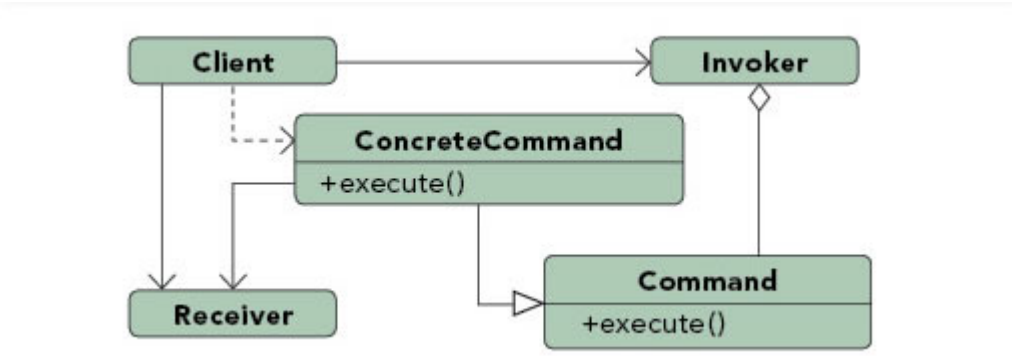
- ☑ Снижение зависимости между клиентом и обработчиками
- ☑ Разделение обязанностей
- ☑ Запрос может остаться не обработанным



Java

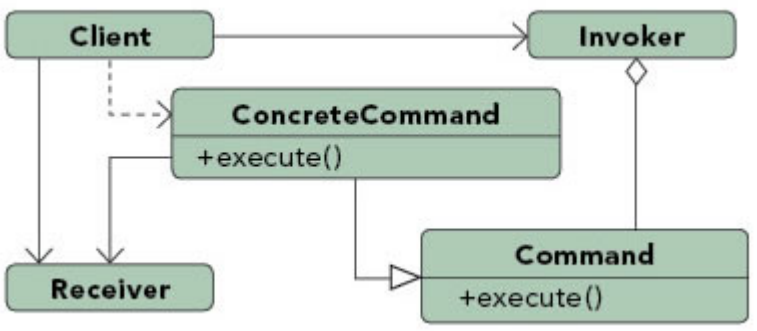
Command

- ☑ Управление командами
- ☑ Разделение вызова и исполнения команд
- ☑ Можно организовать очередь команд и макрокоманды



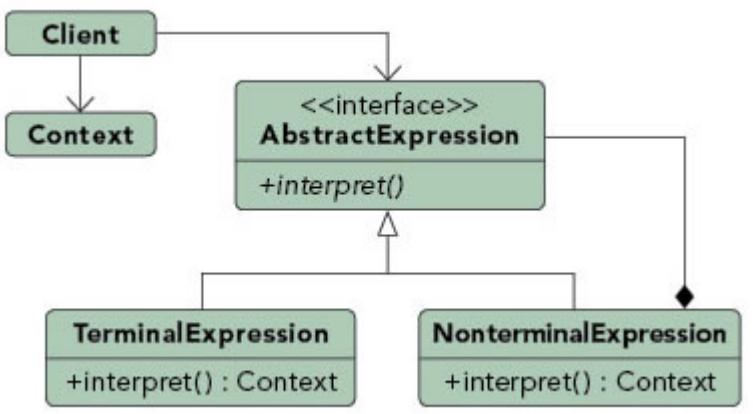
Command

- ☑ Отделение кода исполнения команд от вызова
- ☑ Дополнительные возможности управления
- ☑ Усложнение кода (дополнительные классы)



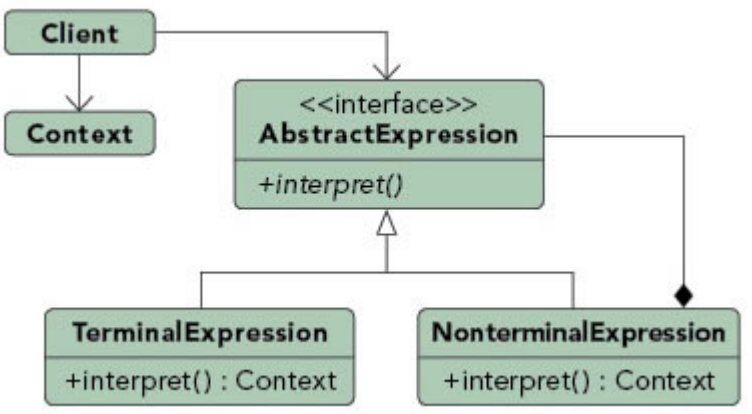
Interpreter

- ☑ Позволяет управлять поведением с помощью простого языка
  - Регулярные выражения
  - Форматирование строк



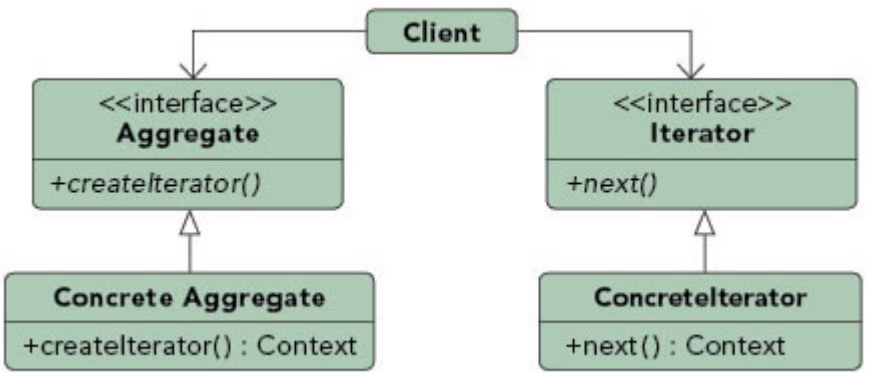
Interpreter

- ☑ Простота расширения и изменения языка
- ☑ Простота добавления новых способов
- ☑ Сложность сопровождения сложных грамматик



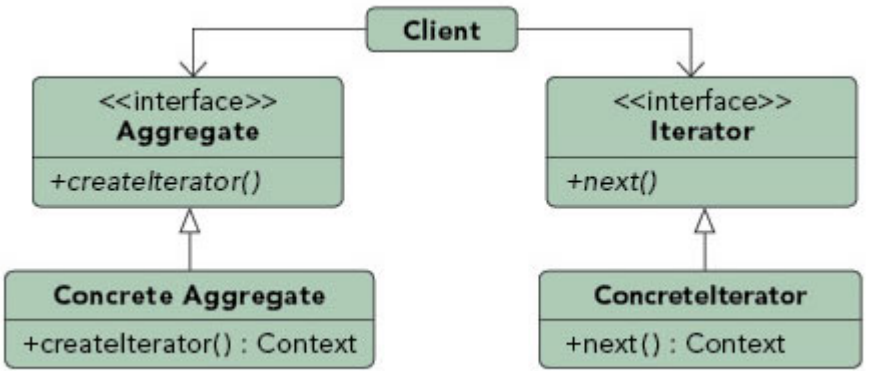
Iterator

- ☑ Последовательный доступ к элементам коллекции
- ☑ Экскурсия
  - Реальный гид
  - Аудиогид
  - Путеводитель



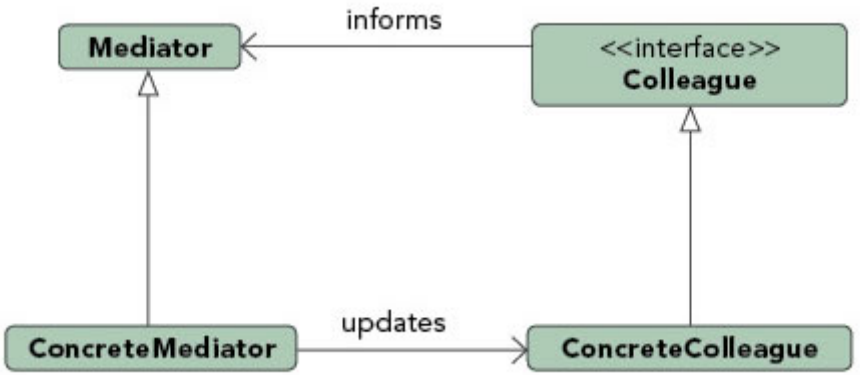
Iterator

- ☑ Универсальный доступ ко всем коллекциям
- ☑ Гибкая реализация обхода структур
- ☑ Более сложный вариант, чем простой цикл



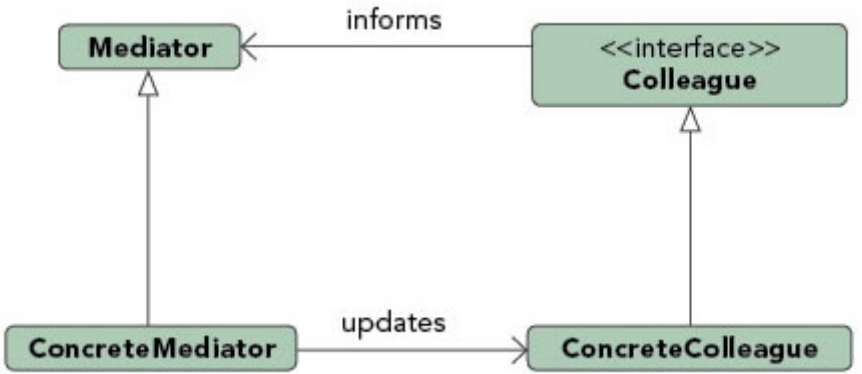
Mediator

- ☑ Класс-посредник для управления изменением состояния других объектов



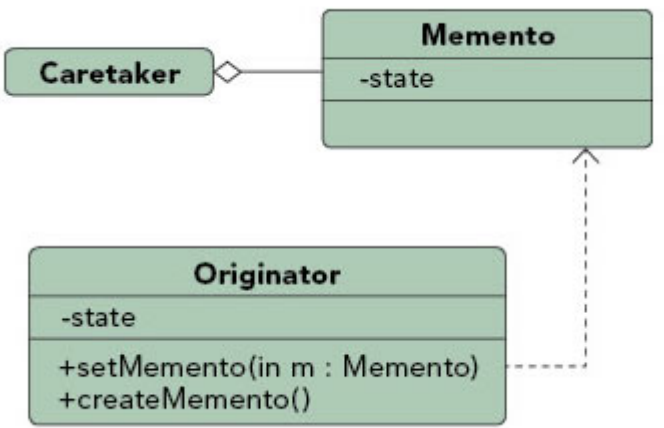
Mediator

- ☑ Снижение зависимости между компонентами
- ☑ Простое централизованное управление
- ☑ Возможность разрастания посредника



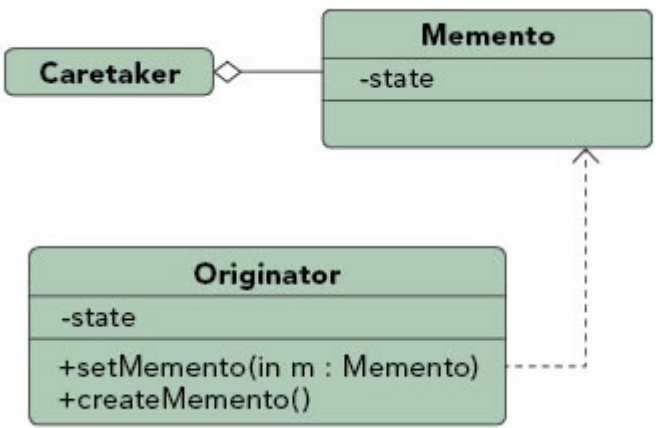
Memento

- ☑ Хранение и восстановление состояния объекта
- ☑ Реализация функции UNDO



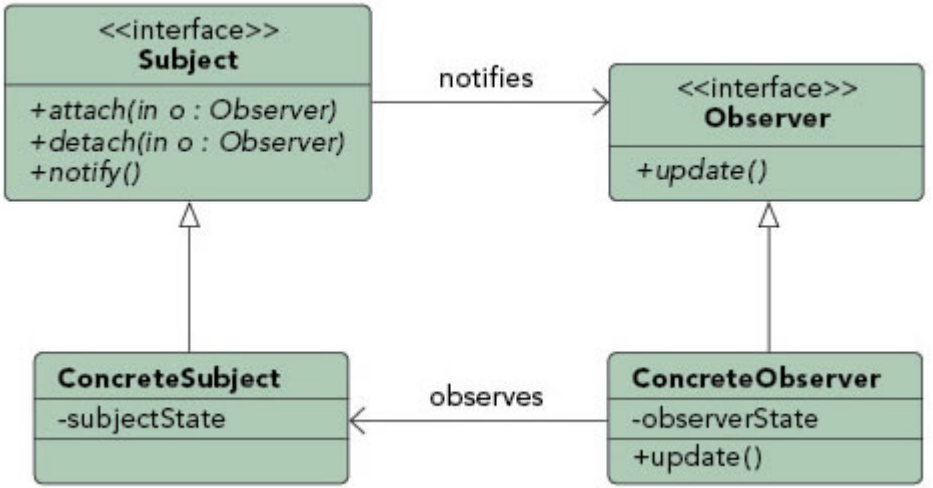
Memento

- ☑ Сохранение инкапсуляции исходного объекта
- ☑ Выделение отдельного хранилища снимков
- ☑ Может потребовать большого объема памяти



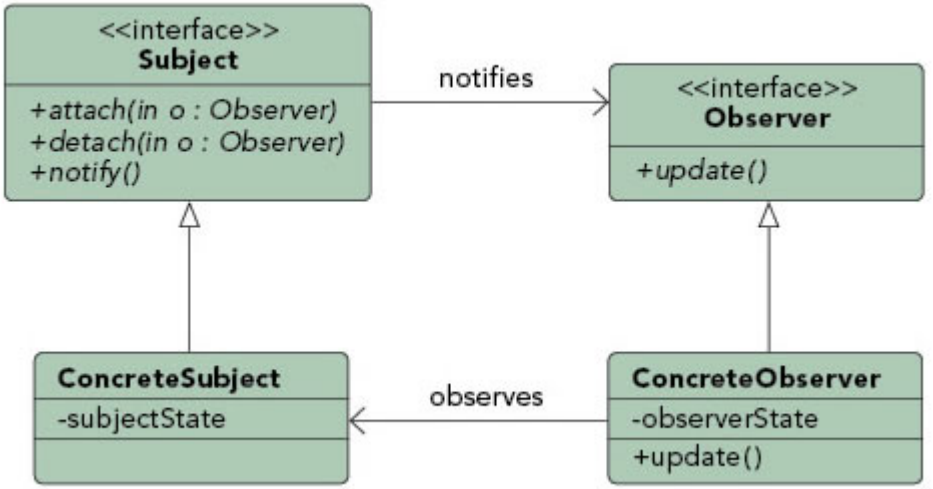
Observer

- ☑ Оповещение объектов об изменении состояния
- ☑ Обработка событий:
  - Источник — наблюдаемый
  - Обработчик - наблюдатель



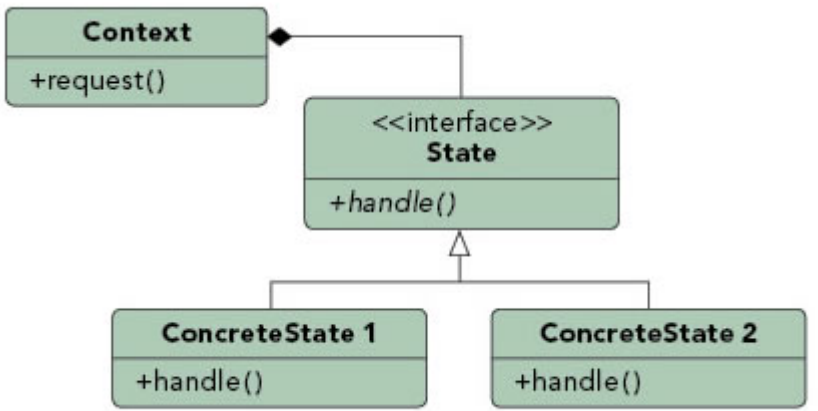
Observer

- ☑ Динамическая подписка и ее отмена
- ☑ Наблюдаемый объект не зависит от наблюдателей
- ☑ Сложность отладки



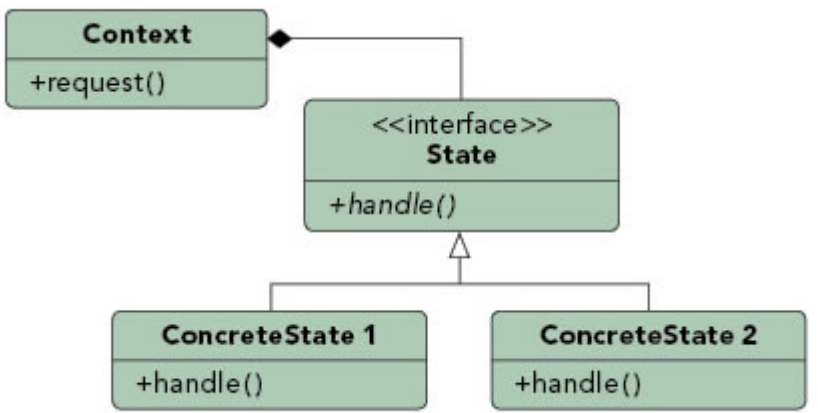
State

- ☑ Изменяет поведение объекта в зависимости от состояния — реализация конечного автомата



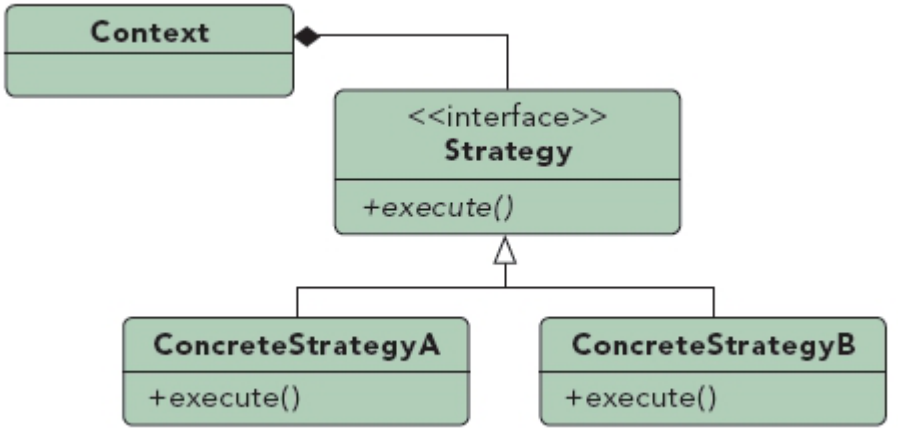
State

- ☑ Избавляет от множества операторов if
- ☑ Выделяет код для каждого состояния в одном месте
- ☑ В некоторых случаях слишком сложный код



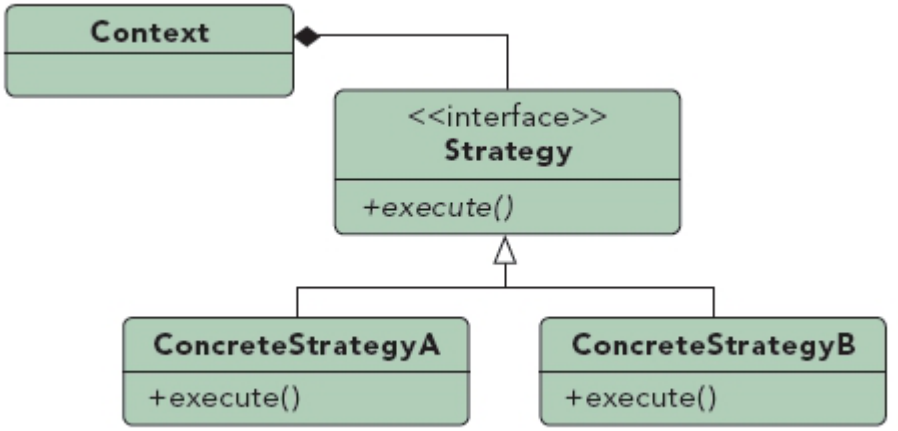
Strategy

- ☑ Выбор одного из алгоритмов, реализованных в классе
  - Как добраться до аэропорта
    - Автобус
    - Такси
    - Пешком



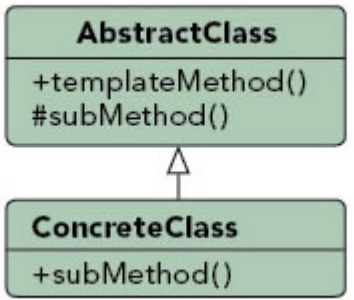
Strategy

- ☑ Изолирует алгоритм в своем классе
- ☑ Динамический выбор стратегии
- ☑ Усложнение программы (дополнительные классы)



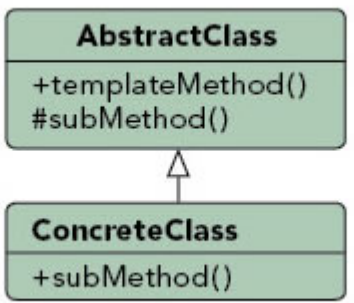
Template Method

- ☑ Позволяет реализовать часть поведения в базовом классе, остальное реализуется в подклассах
- ☑ Покемоны
  - Move.applyOppDamage()



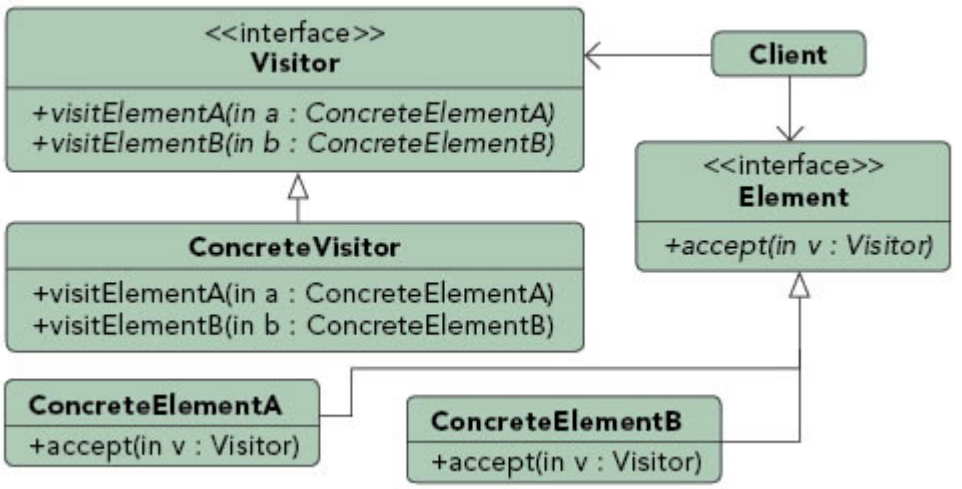
Template Method

- ☑ Упрощает повторное использование кода
- ☑ Жестко задает последовательность действий
- ☑ При большом количестве действий сложно поддерживать



Visitor

- ☑ Позволяет сгруппировать операции, выполняемые над структурой объектов
- ☑ Применяется если структура элементов более стабильна, чем набор операций



Visitor

- ☑ Упрощает добавление новых операций
- ☑ Отделяет код операций от структуры элементов
- ☑ Не позволяет гибко менять структуру