УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Дата и время

ITMO more than a UNIVERSITY

Использование даты и времени

- ☑ Дата и время — варианты представления:
  - Человеческое время — часы, минуты, дни, недели, месяцы
  - Машинное время — (милли)секунды от нулевой точки отсчета (epoch)
    - Java/Unix epoch: 01-01-1970, 00:00:00.000
- ☑ Локальное и поясное время
- ☑ Летнее время, солнечное время, атомное время, ...

Традиционные классы. Date

- ☑ Date (Java 1.0)
  - Единственный класс даты
  - человеческое и машинное представление
  - форматирование
- ☑ Конструкторы
  - Date
  - Date(long)
- ☑ Date (Java 1.1)
  - Момент времени
  - большинство методов — deprecated
- ☑ Методы
  - long getTime()
  - boolean after(Date)
  - boolean before(Date)

TimeZone

- ☑ abstract class TimeZone — смещение от времени нулевого меридиана
  - до 1972: GMT = Greenwich Mean Time
  - после 1972: UTC (coordinated universal time / temps universel coordonné)
- ☑ class SimpleTimeZone extends TimeZone
- ☑ Методы
  - getDefault()
  - getAvailableIDs()
  - getRawOffset() - смещение без учета летнего времени
  - getOffset(long date) — с учетом летнего времени

Calendar

- ☑ abstract class Calendar — представление человеческого времени
- ☑ class GregorianCalendar extends Calendar
  - 2 календаря (григорианский и юлианский) с датой перехода
- ☑ Методы
  - Calendar getInstance()
  - add(int field, int amount);
  - roll(int field, int amount);
  - set(int field, int value);
  - Date getTime()
  - setTime(Date)

Date/Time API

- ☑ java.time - дата, время, периоды
  - Instant, Duration, Period
  - LocalDateTime, 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)

УНИВЕРСИТЕТ ИТМО

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

Java

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

Изменения

☑ Все меняется

- требования заказчика
- пожелания заказчика
- версии библиотек
- операционные системы
- новые устройства
- взгляды разработчика
- внешние условия
- находятся баги

} Программы меняются

Шаблоны проектирования

УНИВЕРСИТЕТ ИТМО

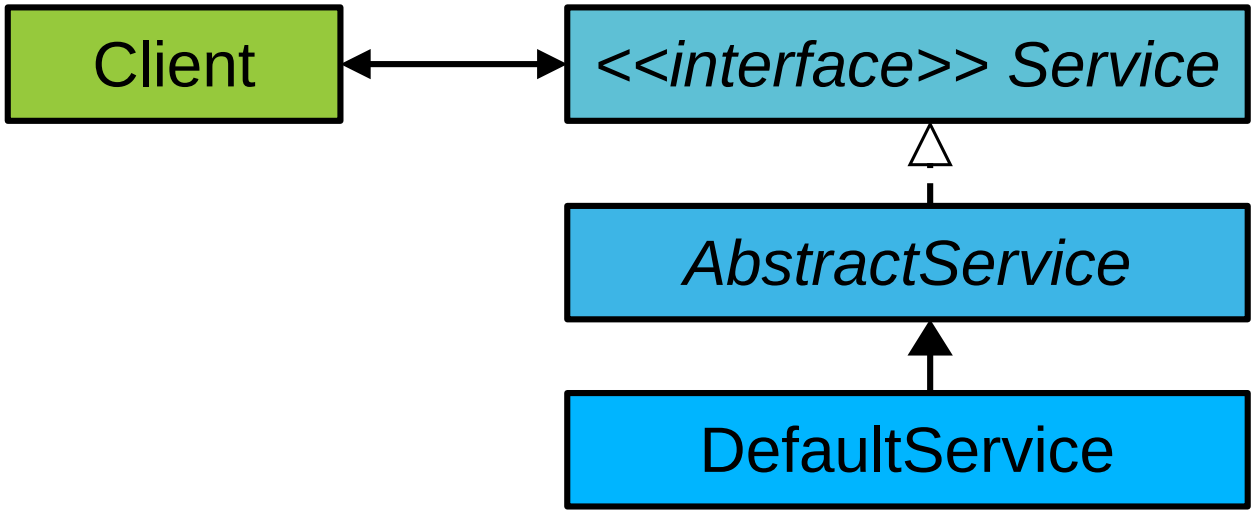
- ☑ Кто-то когда-то уже делал что-то похожее
- ☑ Пришлось вносить изменения - возникли проблемы
- ☑ Нужно сразу было делать по-другому!

GoF book

Java

Базовые принципы - интерфейсы

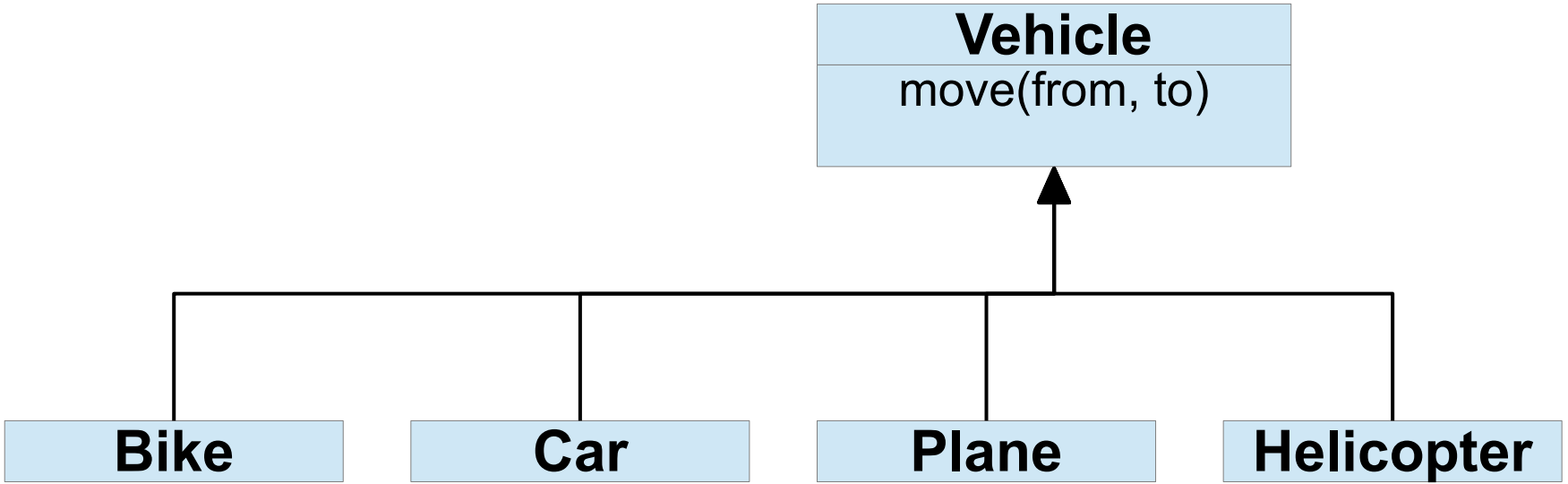
- Взаимодействие с абстракцией - ИНТЕРФЕЙС
- Интерфейс - абстрактный класс - реализация



Java

Наследование

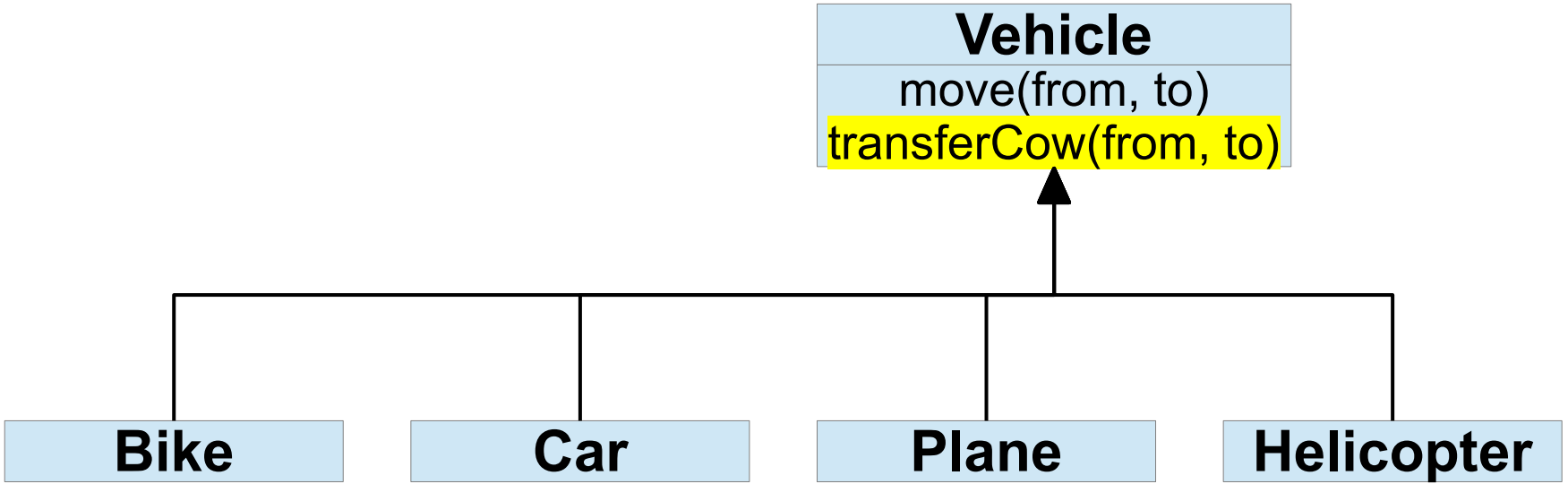
- ☑ Умеет предок - умеют потомки
- ☑ Полиморфизм



Java

Наследование

☑ Перевозка коровы - наследование??



Java

Перевозка коровы

(С) х/ф «Мимино»

Перевозка коровы

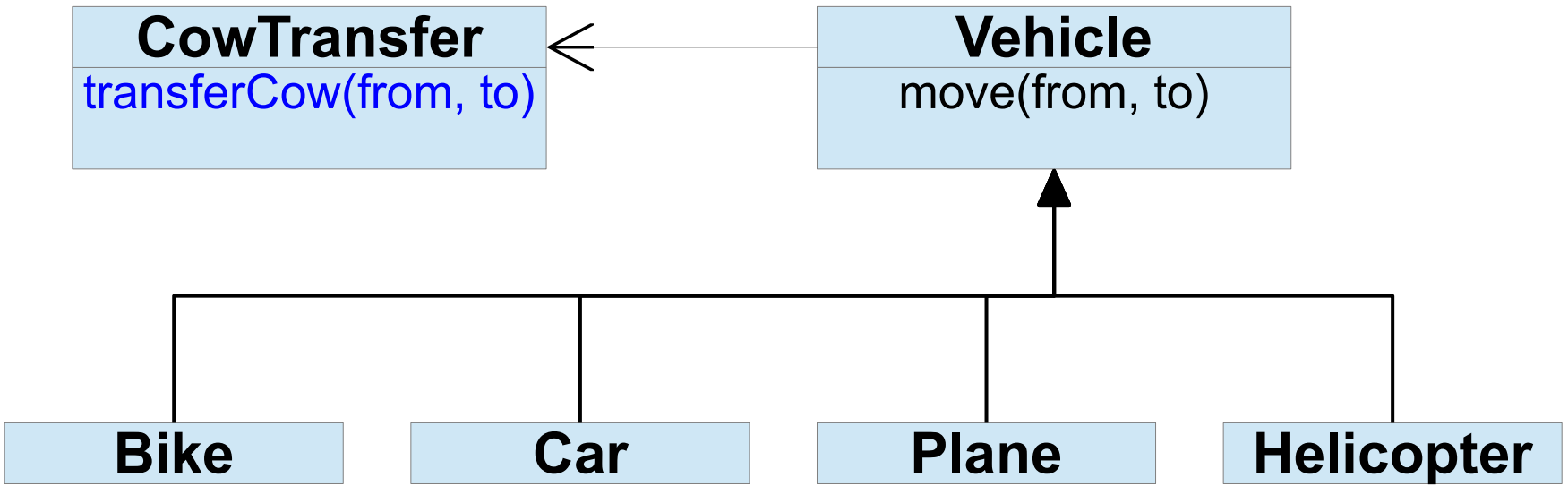
(С) х/ф «Особенности национальной охоты»

Перевозка коровы

Java

Наследование / делегирование

☑ Перевозка коровы - делегирование



Java

Базовые принципы - делегирование

☑ Наследование

- статическое отношение
- классификация

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

class Cat extends Animal {
}

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

☑ Делегирование

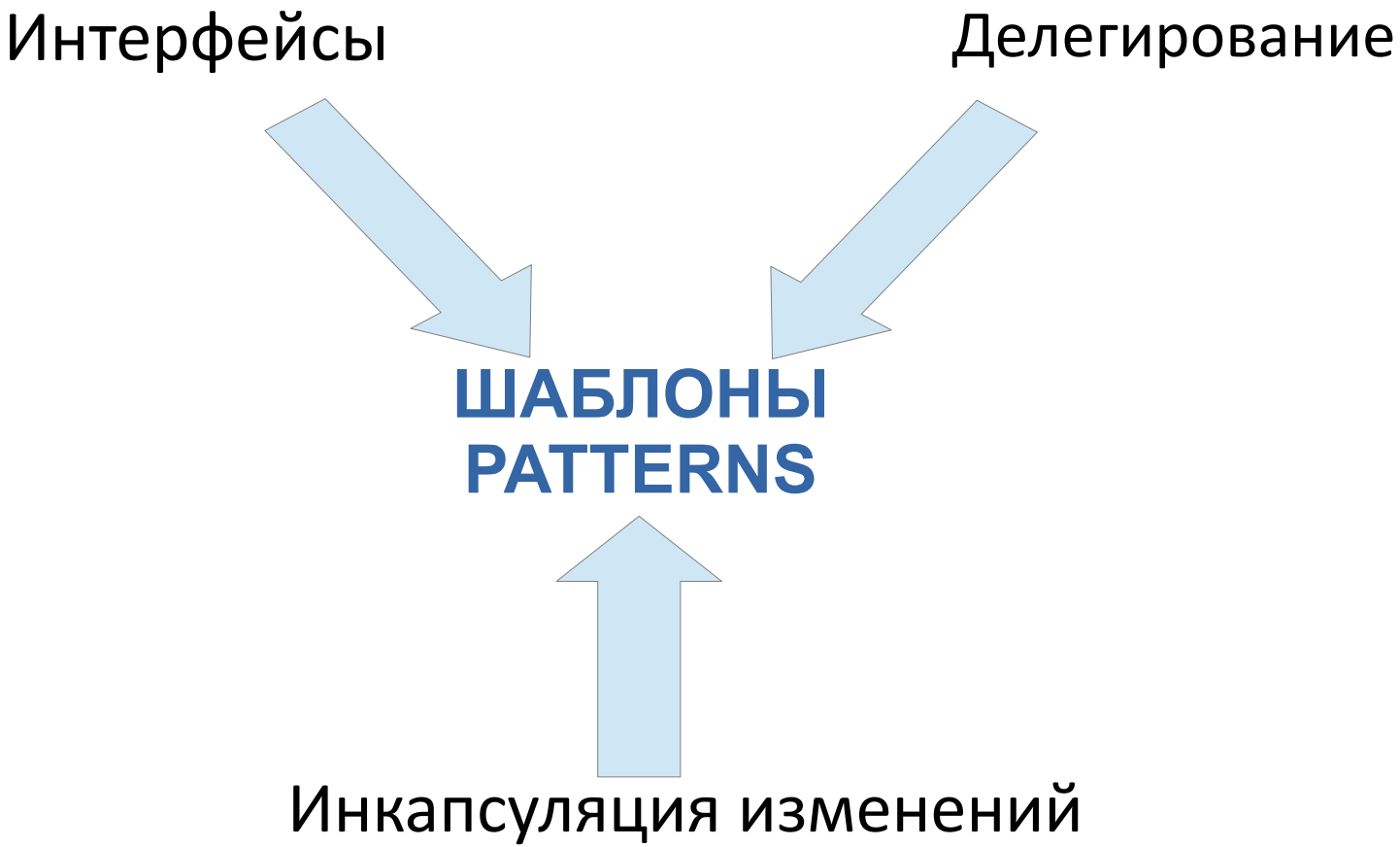
- динамическое отношение
- роль

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

Инкапсуляция изменений

- ☑ Выделить изменчивые сущности
- ☑ Инкапсулировать изменения

Как справиться с изменениями?



Java

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Порождающие шаблоны (Creational Patterns)

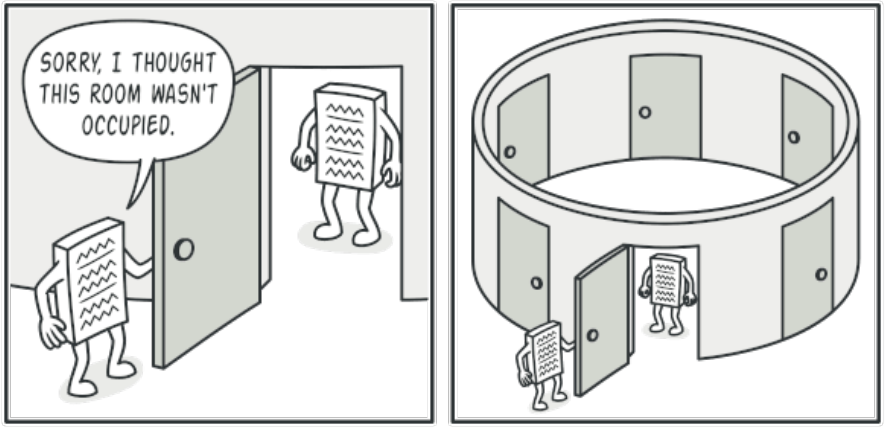
ITMO more than a UNIVERSITY

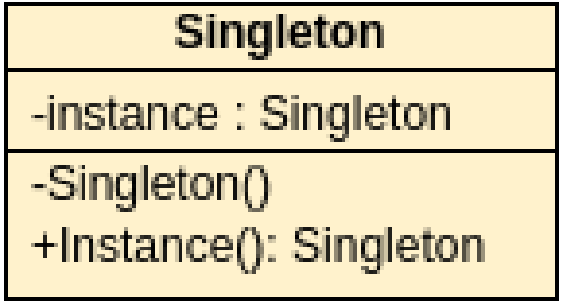
Порождающие шаблоны

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

Singleton

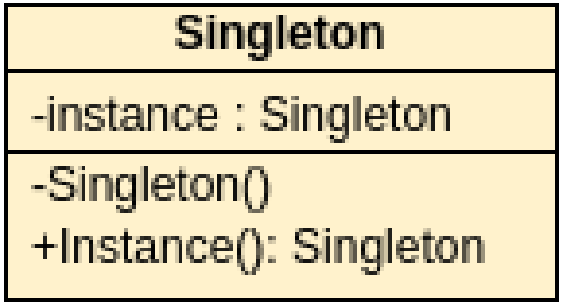
- ☑ Одиночка
  - Гарантирует наличие единственного экземпляра класса
  - Предоставляет глобальную точку доступа





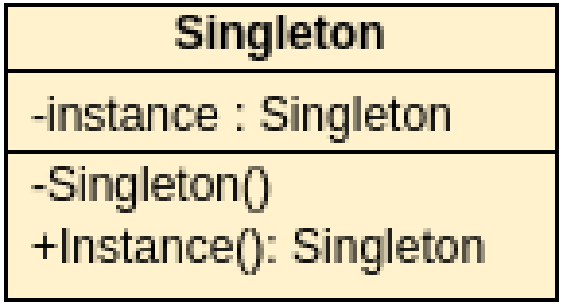
Singleton


```
class Singleton {
  private static Singleton instance = new Sungleton();

  private Singleton() { }

  public static Singleton getInstance() {
    return instance;
  }
}
```



Singleton

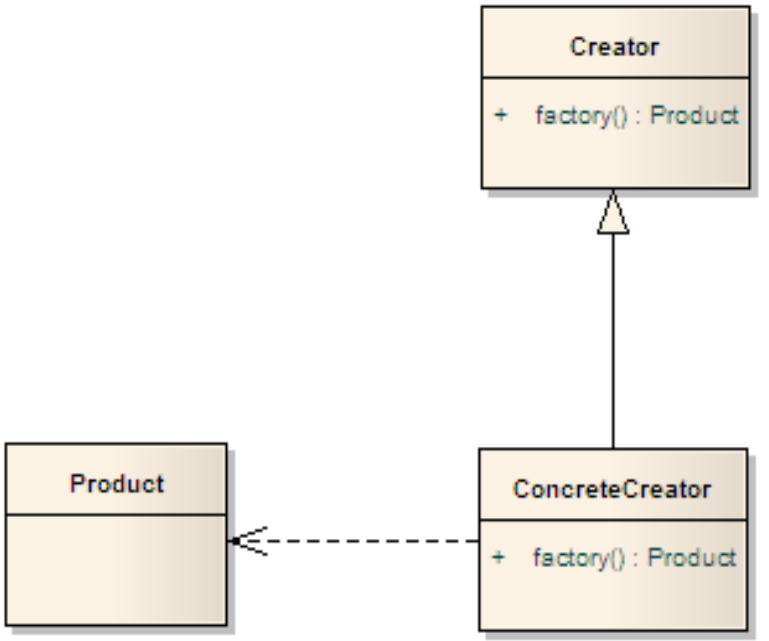
- ☑ Варианты
  - Ленивая инициализация
  - Синхронизация при многопоточности
- ☑ Использование
  - логгер
  - конфигуратор
  - фабрика
  - java.lang.Runtime



Simple Factory

- ☑ Простая фабрика (не совсем паттерн)
  - Объект, создающий другие объекты

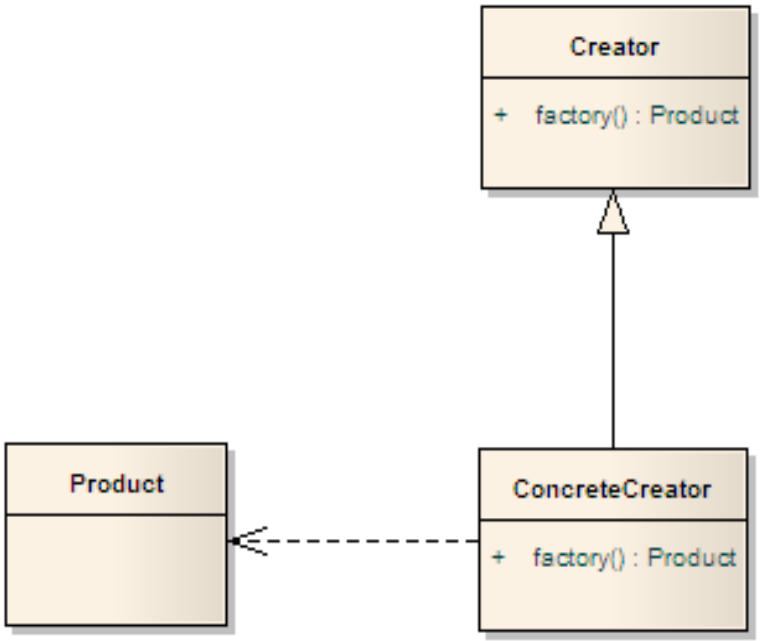
```
Animal[] animals = new Animal[2];
animals[0] = new Dog();
animals[1] = new Cat();
for (Animal a : animals) {
  a.sleep();
  a.makeSound();
}
```



Simple Factory

- ☑ Простая фабрика (не совсем паттерн)
  - Объект, создающий другие объекты

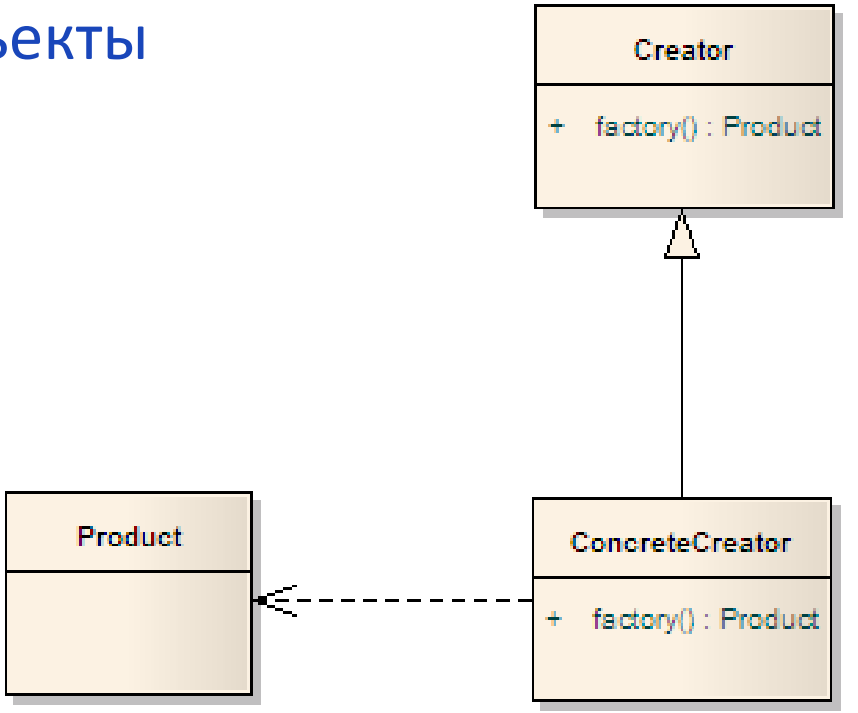
```
class Creator {
  public static Product get(int type) {
    return switch(type) {
      case 1 -> product1;
      case 2 -> product2;
  }
}
```



Simple Factory

- ☑ Простая фабрика (не совсем паттерн)
  - Объект, создающий другие объекты

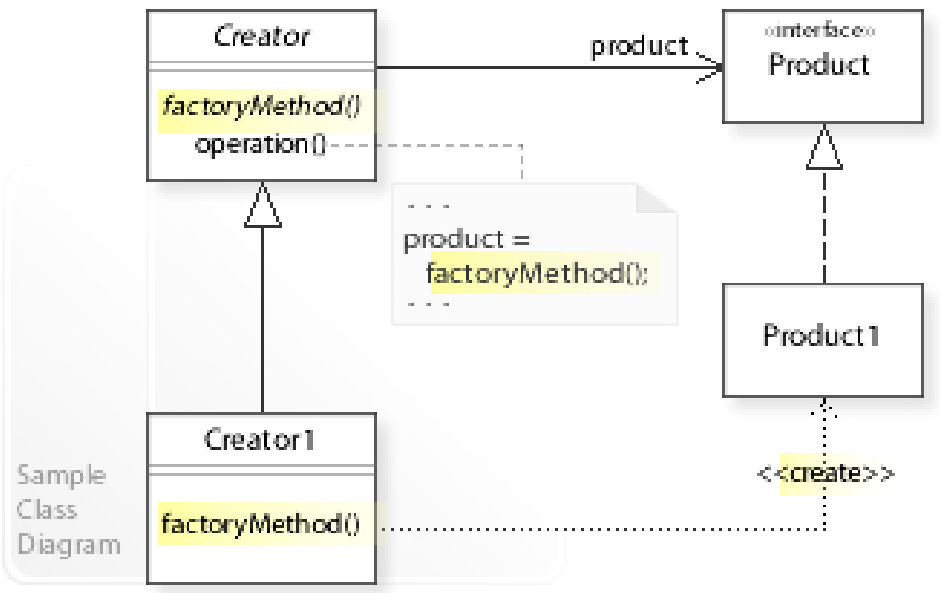
```
Animal[] animals = new Animal[2];
animals[0] = animalFactory.get(1);
animals[1] = animalFactory.get(2);
for (Animal a : animals) {
  a.sleep();
  a.makeSound();
}
```

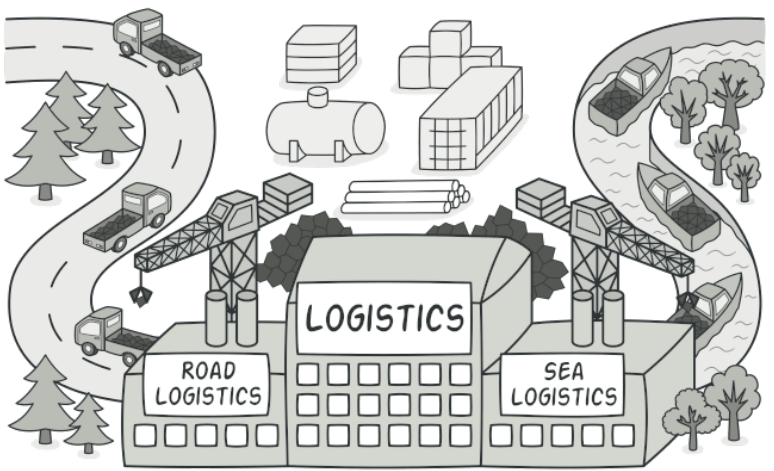


Factory Method

☑ Фабричный метод

- Создание объектов отдаем подклассам





Factory Method

- ☑ Продукт определяет общий интерфейс объектов
- ☑ Конкретные продукты отличаются реализацией, имеют общий интерфейс
- ☑ Создатель объявляет (абстрактный) фабричный метод. Он возвращает объект общего интерфейса продуктов
- ☑ Создатель также имеет другой код работы с продуктом
- ☑ Конкретные создатели реализуют фабричный метод, создавая конкретные продукты

Factory Method

☑ Фабричный метод

- Statement stat = connection.createStatement()

```
class Creator {
  protected abstract Product getProduct();
}

class ConcreteCreator1 {
  public Product getProduct() {
    return new ConcreteProduct1();
  }
}
```

Factory Method

УНИВЕРСИТЕТ ИТМО

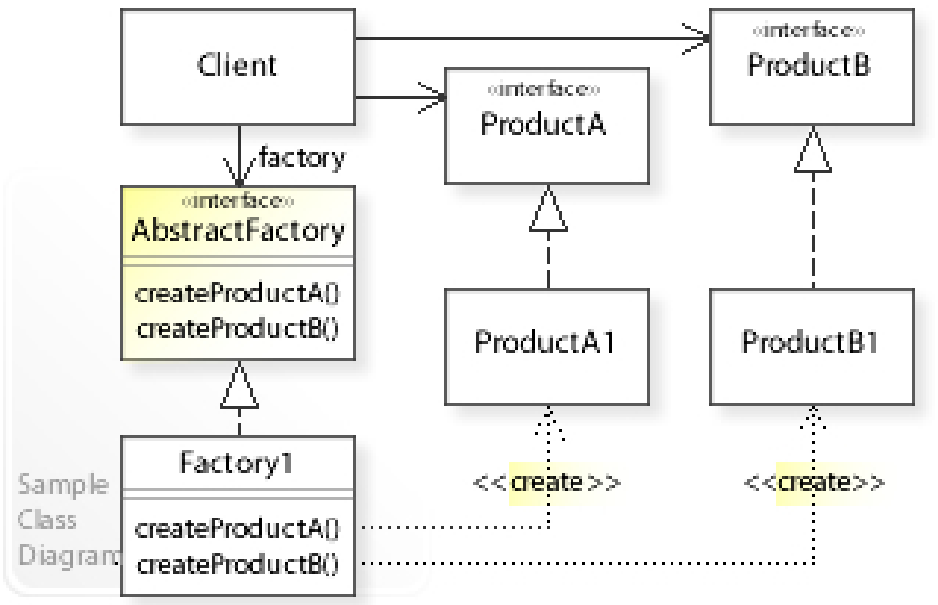
- ☑ Убирает зависимость от конкретных продуктов
- ☑ Упрощает добавление новых продуктов
- ☑ На каждый продукт нужен свой создатель

Java

Abstract Factory

☑ Абстрактная фабрика

- Фабрика для создания семейства продуктов



Abstract Factory

- ☑ Абстрактные продукты — интерфейсы продуктов разных видов (кола, апельсин, лимон)
- ☑ Конкретные продукты — классы различных видов и семейств (миринда, спрайт)
- ☑ Абстрактная фабрика объявляет методы для создания продуктов разного вида
- ☑ Конкретные фабрики умеют создавать все виды продуктов одного семейства (фабрика пепси)

Abstract Factory

☑ Абстрактная фабрика

- Фабрика для создания семейства продуктов

```
Factory f1 = Provider.getFactory("Coca-Cola");
Drink d1 = f1.getOrangeDrink(); // Фанта

Factory f2 = Provider.getFactory("Pepsi");
Drink d2 = f2.getLemonDrink(); // 7-up
```

Abstract Factory

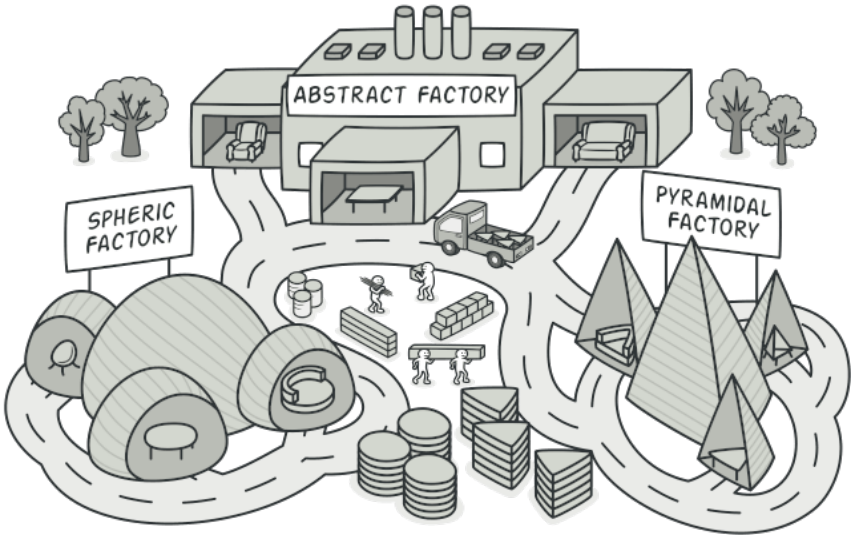
☑ Абстрактная фабрика

- Connection conn = DriverManager.getConnection(args)

```
Factory f1 = Provider.getFactory("Coca-Cola");
Drink d1 = f1.getOrangeDrink(); // Фанта

Factory f2 = Provider.getFactory("Pepsi");
Drink d2 = f2.getLemonDrink(); // 7-up

Factory f3 = Provider.getFactory("Добрый");
Drink d3 = f3.getColaDrink(); // Добрый кола
```

Abstract Factory

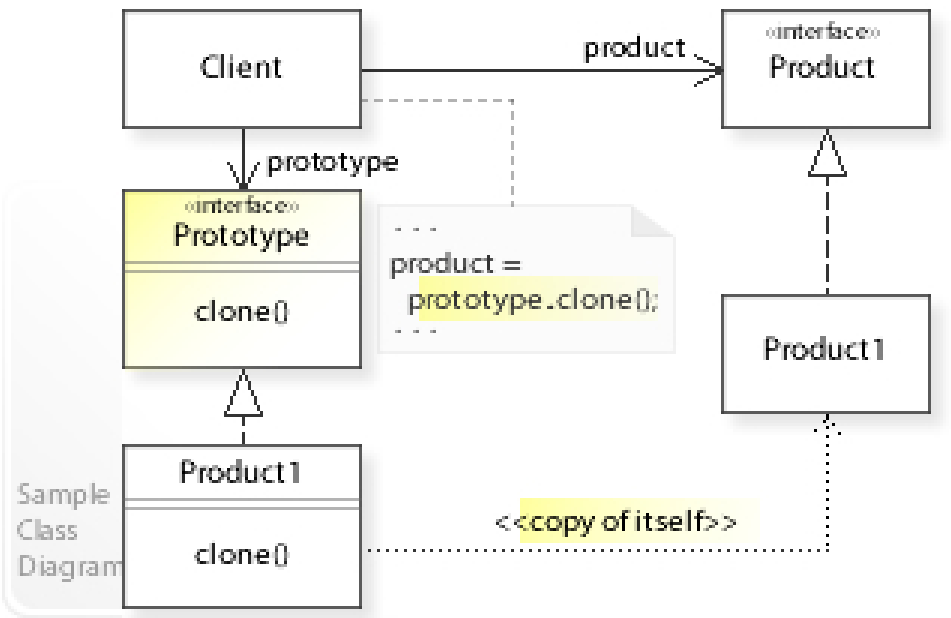
- ☑ Устраняет зависимость от конкретных продуктов
- ☑ Гарантирует совместимость продуктов в наборе
- ☑ Требуется наличие всех типов продуктов в семействе
- ☑ Более сложный код



Prototype

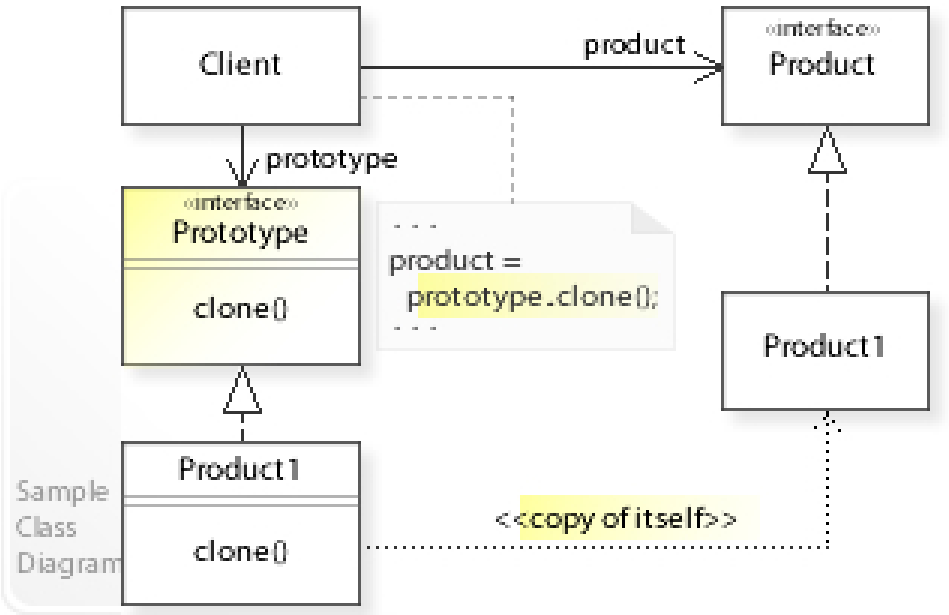
☑ Прототип

- Клонирование вместо создания



Prototype

- ☑ Интерфейс Prototype задает операцию клонирования
- ☑ Конкретный прототип реализует операцию клонирования самого себя
- ☑ Клиент создаёт копию объекта, вызывая метод клонирования через интерфейс прототипа
- ☑ Object.clone(), Cloneable



Prototype

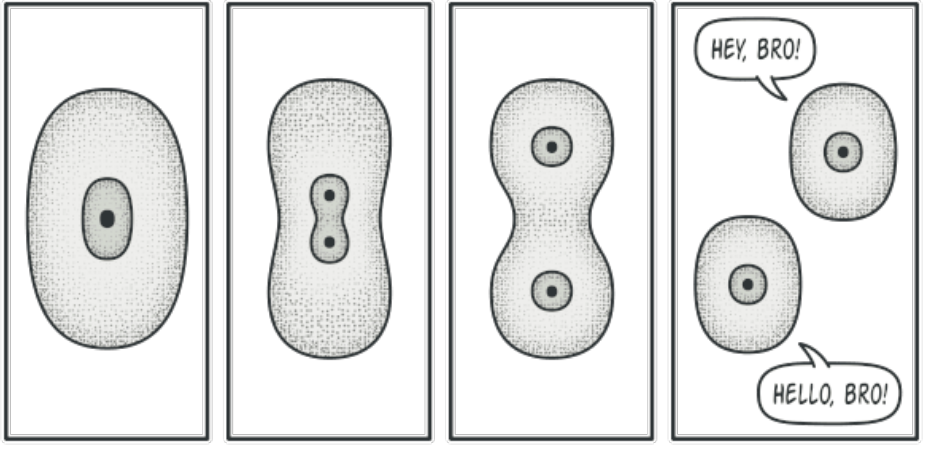
- ☑ Глубокая и неглубокая копия (deep / shallow copy)

```
Trooper trooper = new Trooper();
trooper.setHP(10000);
trooper.setSpeed(1000);
trooper.setPower(5000);
trooper.setAttack("Laser beam");
```

```
var army = new ArrayList<Trooper>();
for (i = 0; i < 10; i++) {
  var oneMore = trooper.clone();
  army.add(oneMore);
}
```

Prototype

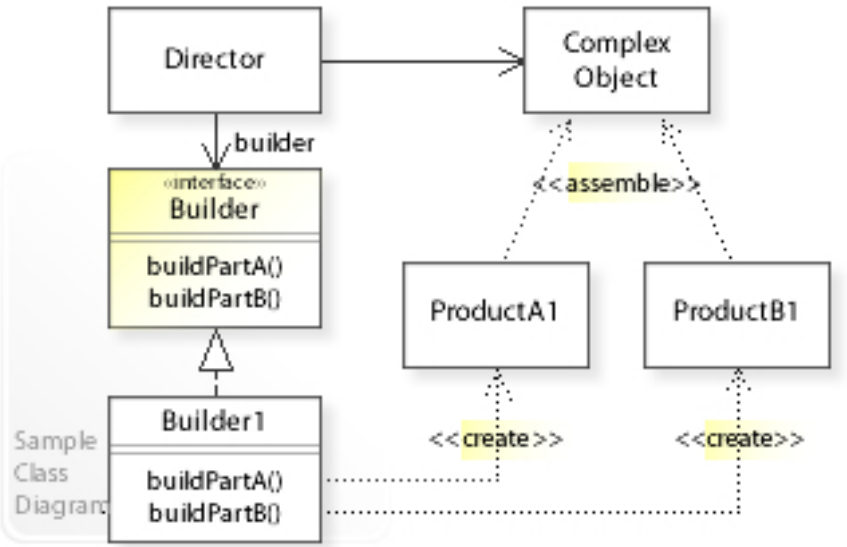
- ☑ Клонирование без зависимости от конкретных классов
- ☑ Позволяет хранить готовый набор сложных объектов
- ☑ Сложно реализовывать глубокое клонирование



Builder

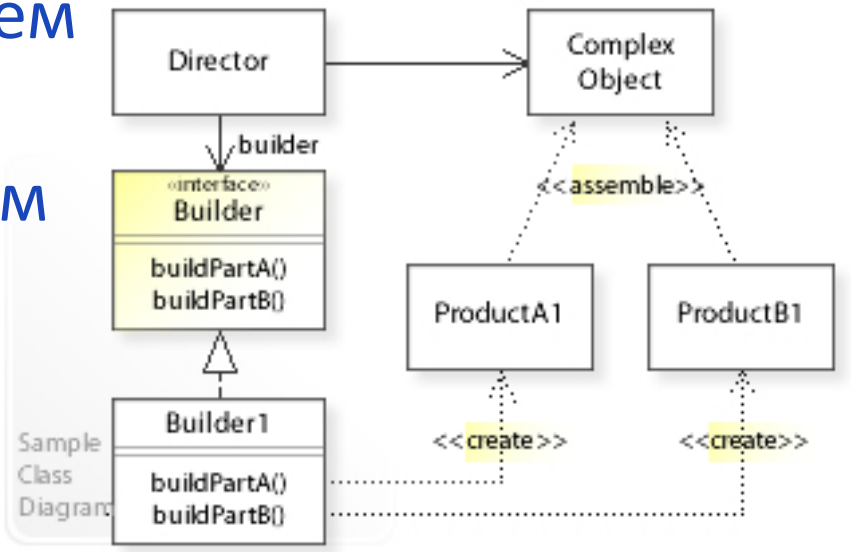
☑ Строитель

- Строит сложный объект по шагам



Builder

- ☑ Интерфейс Builder объявляет этапы создания продуктов
- ☑ Конкретные строители реализуют этапы создания
- ☑ Продукт — создается строителем при вызове метода create()
- ☑ Директор управляет строителем для производства нужной конфигурации продукта



Builder

- ☑ Класс со множеством полей

```
class Trip {
  private Location[] locations;
  private Transport transport;
  private int numberOfPersons;
  private Date startDate;
  private Date finalDate;
  private Hotel hotel;
  ...
}
```

Builder

☑ Телескопические конструкторы

```
class Trip {
  public Trip(Date date)
  public Trip(Location location)
  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

- ☑ Простой конструктор и набор сеттеров
- ☑ Объект не полностью готов в промежуточном состоянии

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

Builder

- ☑ Строитель создает объект по шагам
- ☑ Методы возвращают строителя

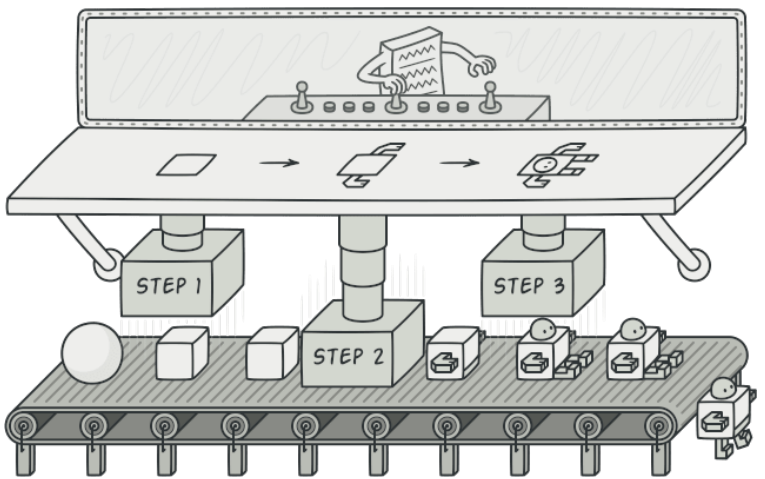
```
Builder b = new Builder()
   .setWhere("Moscow")
   .setTransport("train")
   .setDate(1,7,2023);
   ...

Trip trip = b.createTrip();
```

- ☑ Директор создает объект с помощью строителя одним методом

```
Director d = new Director();
Trip trip = d.makeTrainTrip("Moscow", new Date(1,7,2023));
```

Builder

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



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Структурные шаблоны (Structural Patterns)

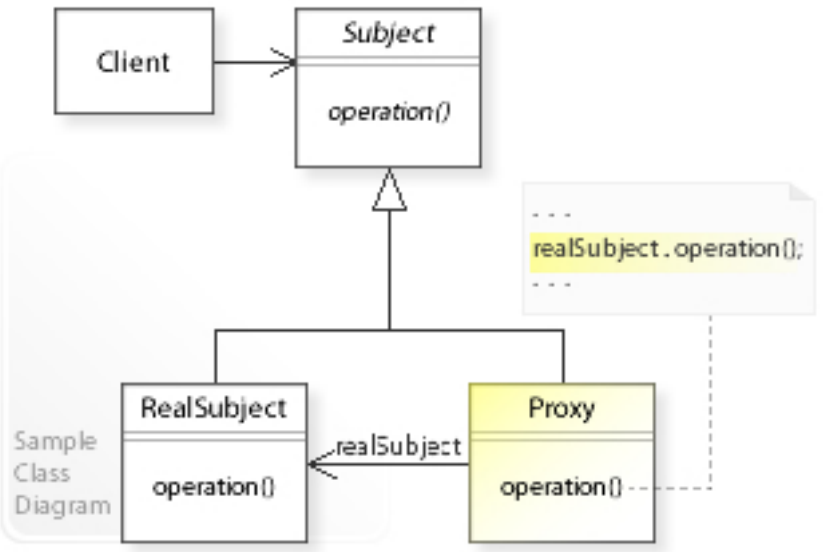
ITMO more than a UNIVERSITY

Структурные шаблоны

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

Proxy

- ☑ Заместитель - делегирование **с тем же интерфейсом**
- ☑ Подмена реального объекта заместителем
  - Virtual - ленивое создание
  - Remote - удаленный объект
  - Security - контроль доступа
  - Caching - кэширование
  - Smart - логирование

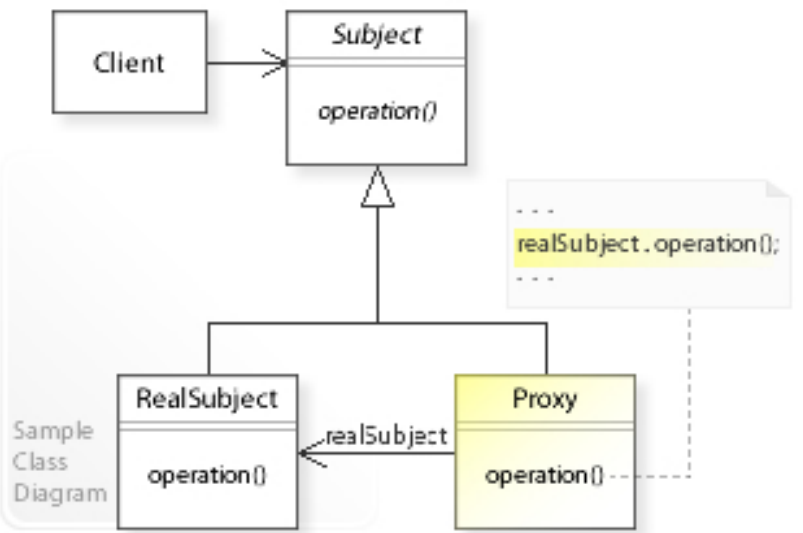


Proxy

☑ Заместитель

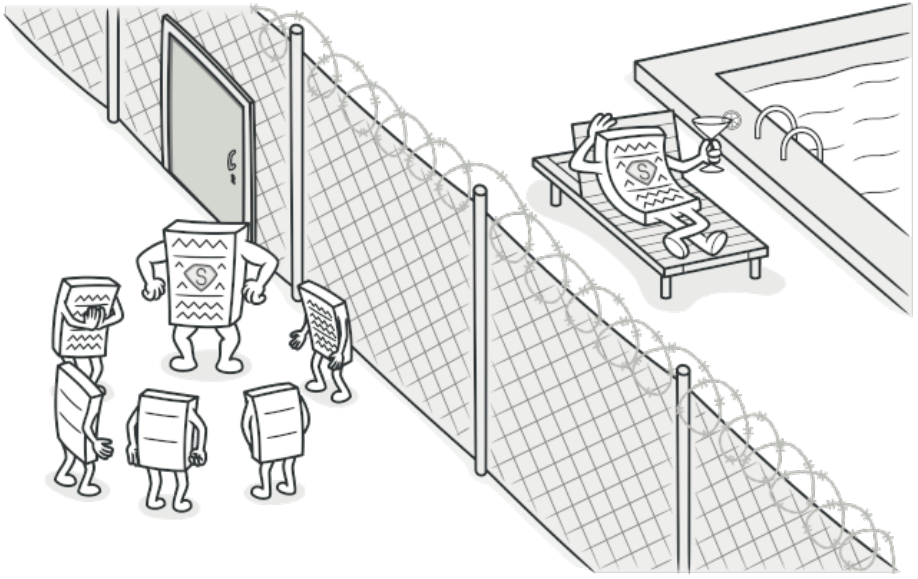
```
class Proxy implements Subject {
  RealSubject real;
  public request() {
    return real.request();
  }
}
```

```
Subject subject = new Proxy();
subject.request();
```



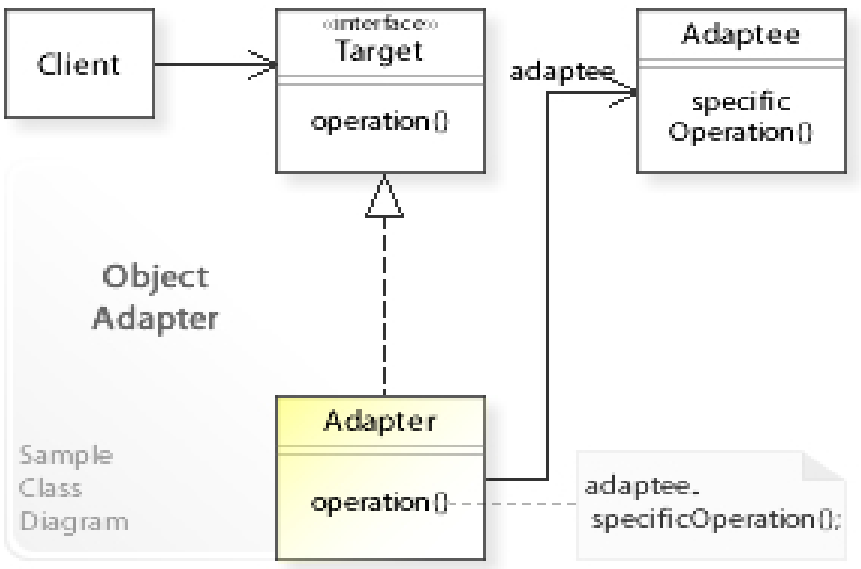
Proxy

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



Adapter

- ☑ Адаптер - делегирование **с заменой интерфейса**
- ☑ Адаптирует один интерфейс к другому



Adapter

☑ Адаптирует один интерфейс к другому

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


```
interface UKSocket {
  getUKPower()
}
```

Adapter

УНИВЕРСИТЕТ ИТМО

☑ Адаптирует один интерфейс к другому

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


```
interface UKSocket {
  getUKPower()
}
```

Adapter

☑ Адаптирует один интерфейс к другому

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

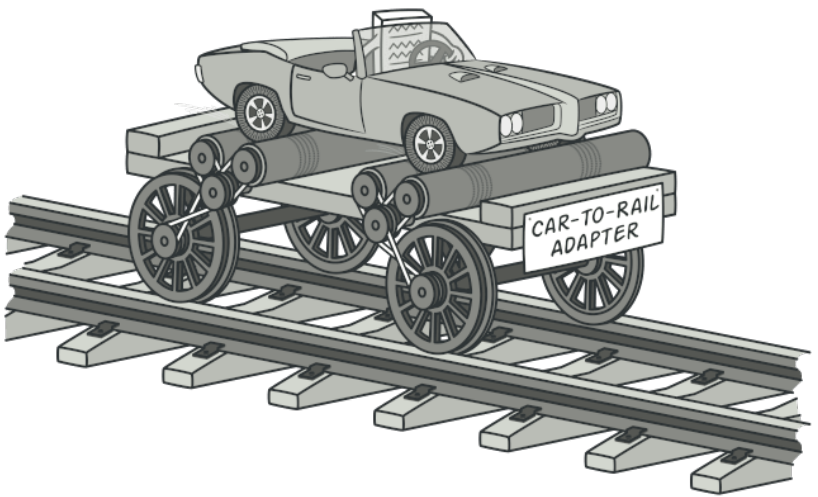

```
interface UKSocket {
  getUKPower()
}
```

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

Adapter

УНИВЕРСИТЕТ ИТМО

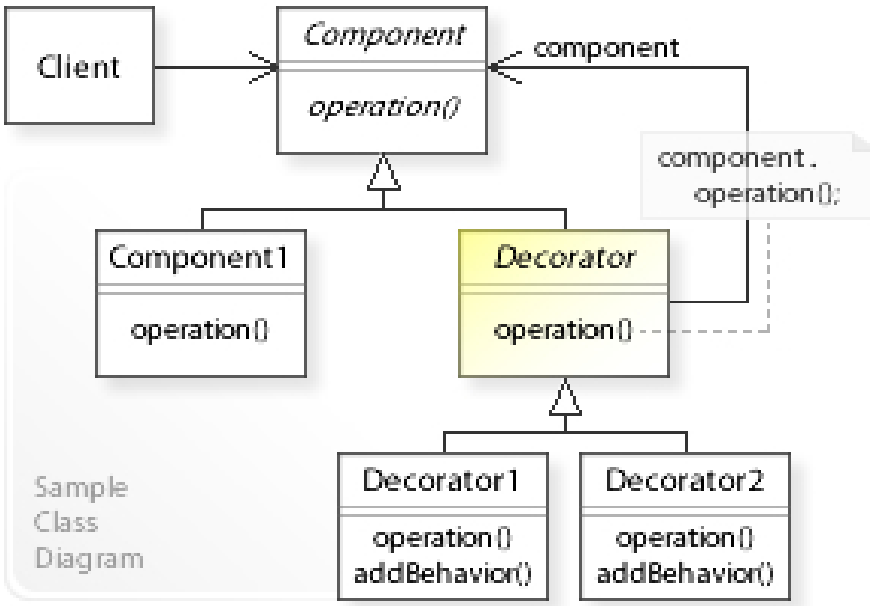
- ☑ Позволяет устранить несовместимость интерфейсов
- ☑ Не нужно переписывать существующий код
- ☑ Усложнение структуры (+ адаптер)



Java

Decorator

- ☑ Декоратор - делегирование с новой функциональностью
- ☑ Блины с добавками в Теремке
- ☑ InputStream
  - BufferedInputStream,
  - LineNumberInputStream,
  - …

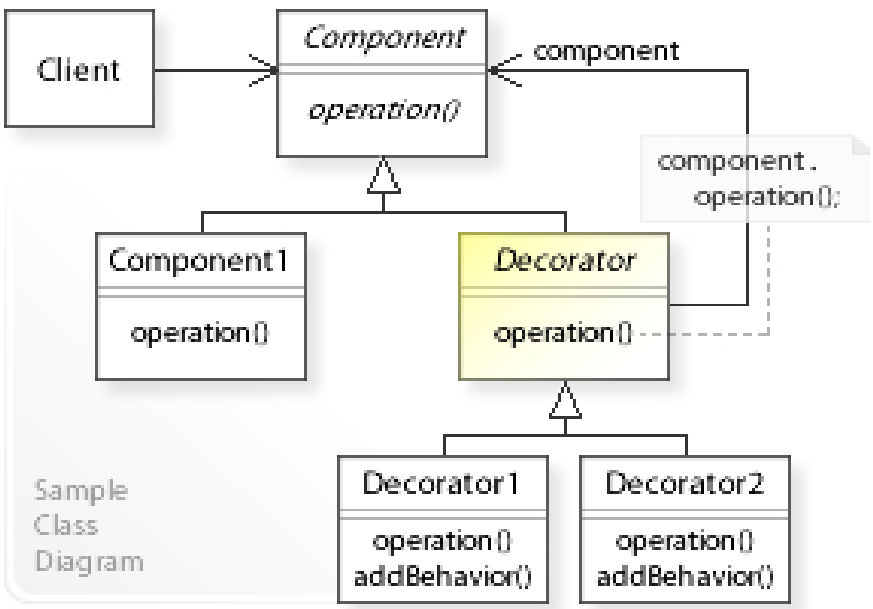


Decorator

УНИВЕРСИТЕТ ИТМО

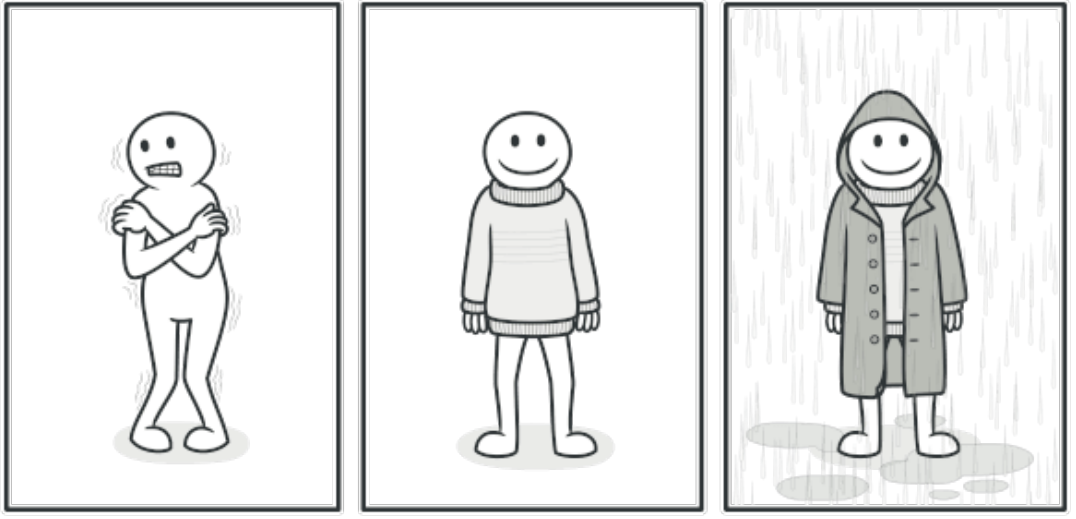
☑ Делегирование с расширением функциональности

```
class Decorator implements Component {
  Component1 component;
  public operation() {
    component.operation();
    addBehavior();
  }
}
```



Decorator

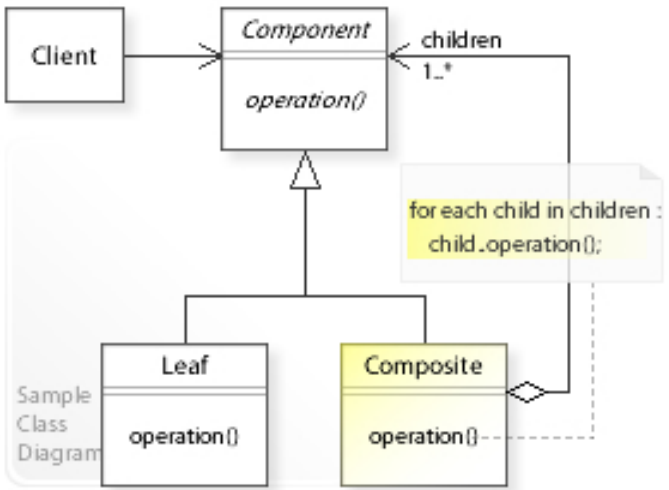
- ☑ Позволяет добавлять функциональность динамически
- ☑ Вместо большой иерархии - несколько декораторов
- ☑ Сложность конфигурирования

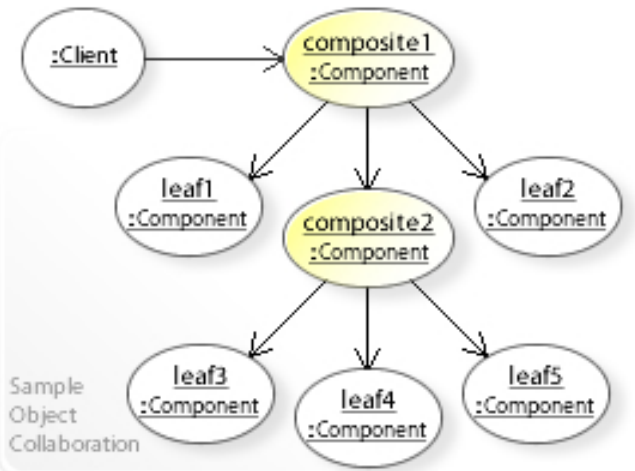


Composite

УНИВЕРСИТЕТ ИТМО

- ☑ Компоновщик - иерархическая древовидная структура
- ☑ Абстрактный компонент и его потомки: компонент-лист и компонент-контейнер





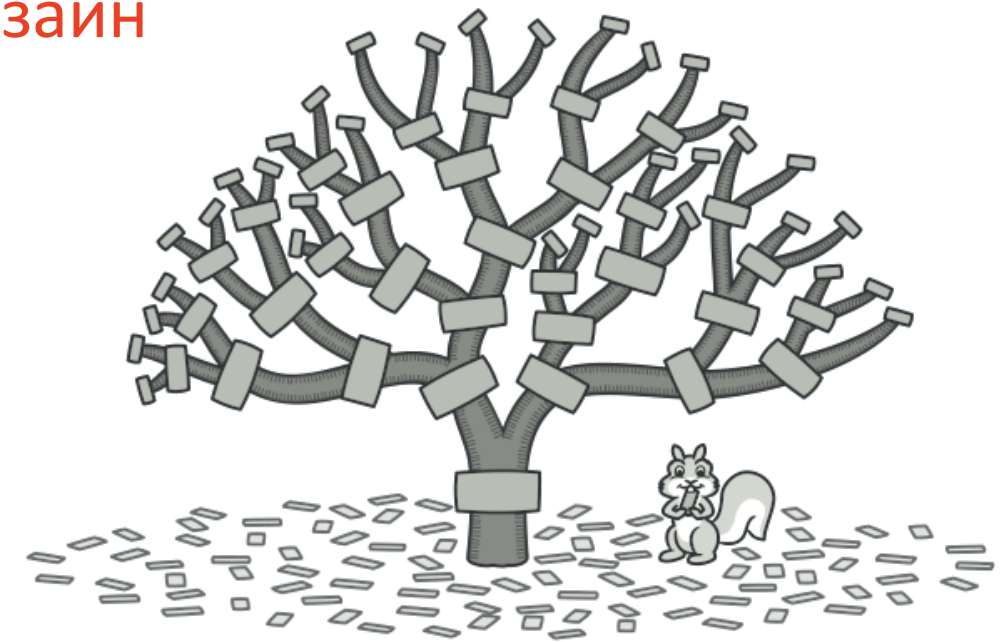
Composite

- ☑ Компоновщик - иерархическая древовидная структура
- ☑ Абстрактный компонент и его потомки: компонент-лист и компонент-контейнер

```
class Composite extends Component {
  List<Component> children;
  public operation() {
    for (c : children) {
      c.operation();
    }
  }
}
```

Composite

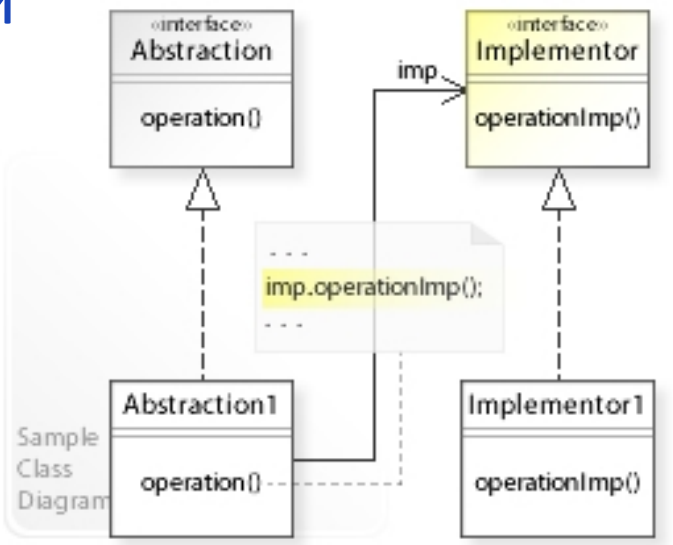
- ☑ Простая работа с деревом компонентов
- ☑ Простое добавление новых компонентов
- ☑ Слишком общий дизайн



Bridge

УНИВЕРСИТЕТ ИТМО

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

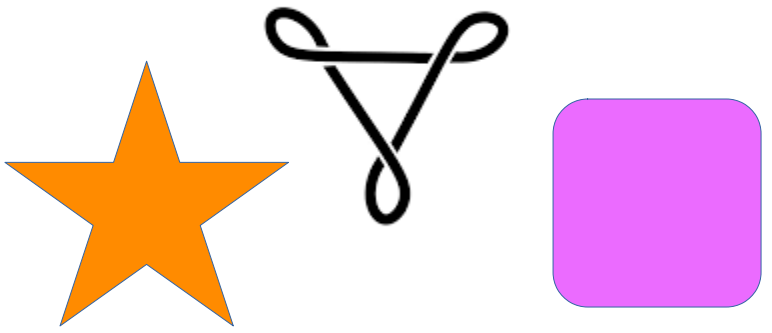


Bridge


```
class SharpTriangle
class RoundedTriangle
class SharpRectangle
class RoundedRectangle
```

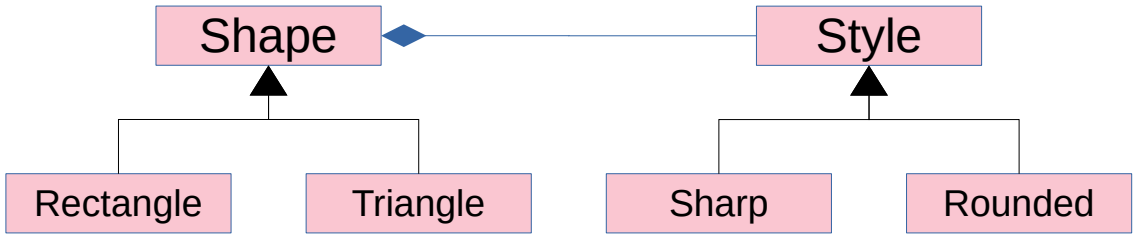
Bridge


```
class SharpTriangle
class RoundedTriangle
class SharpRectangle
class RoundedRectangle
```



```
                +
class LoopedTriangle
class RoundedSquare
class SharpStar
                ...
```

Bridge

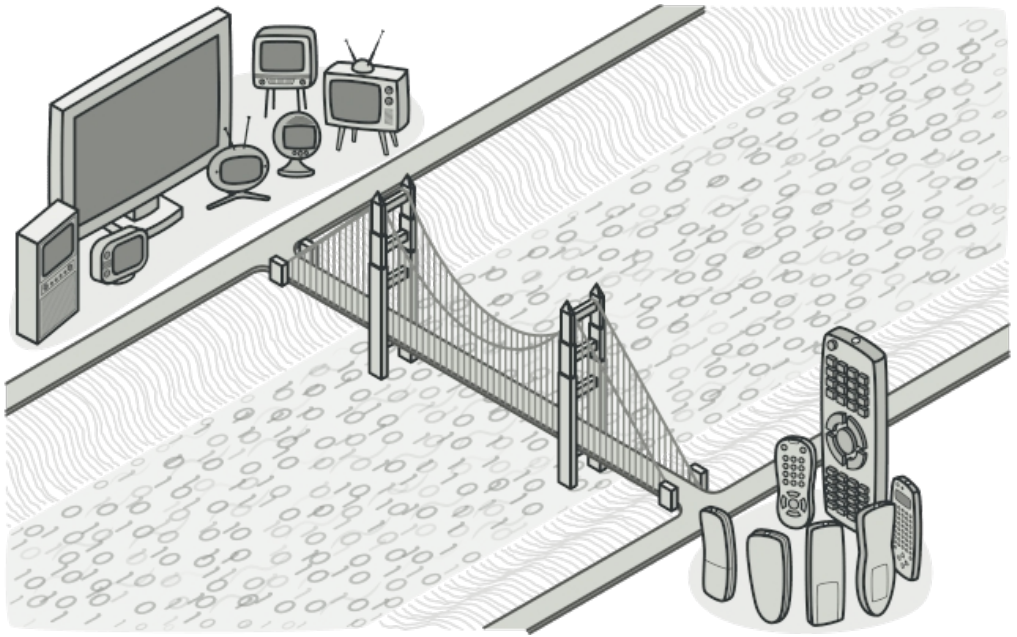


```
abstract class Shape {
  Style style;
  drawShape() {
    ...
    style.drawLine();
    ...
  }
}


abstract class Style {
  drawLine();
}
```

Bridge

УНИВЕРСИТЕТ ИТМО

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

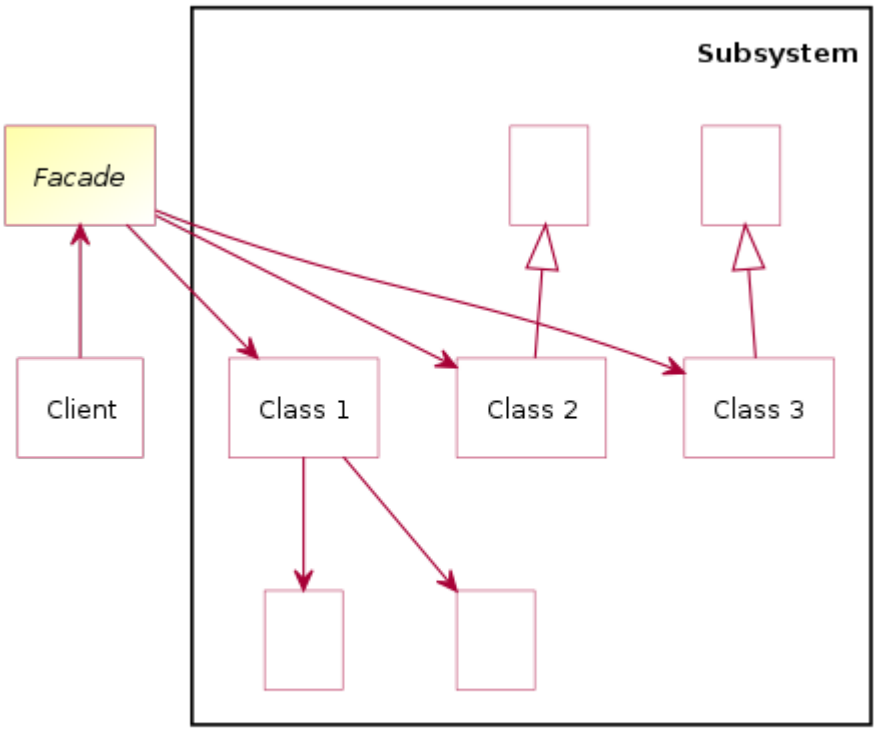


Java

Facade

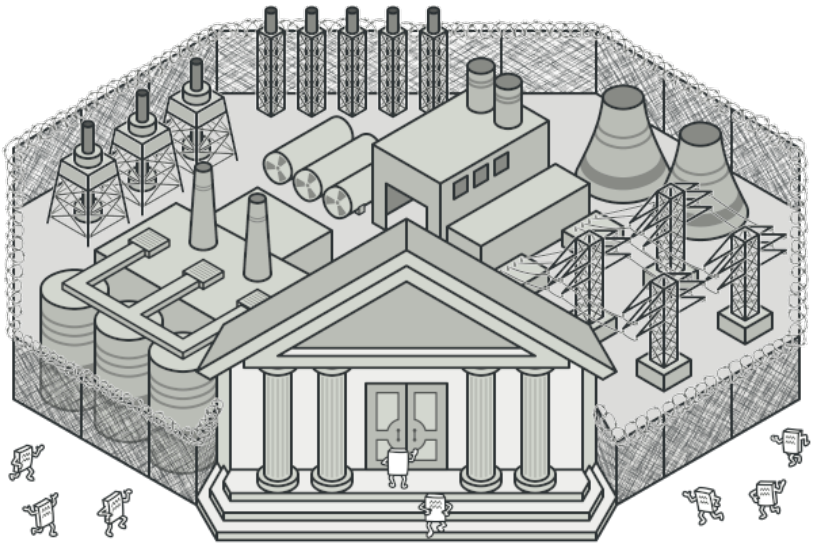
- ☑ Фасад - простой интерфейс к сложным системам
- ☑ Система может меняться
- ☑ Доступ через фасад

```
class Facade {
  public go() {
    ... // очень сложный код
  }
}
```



Facade

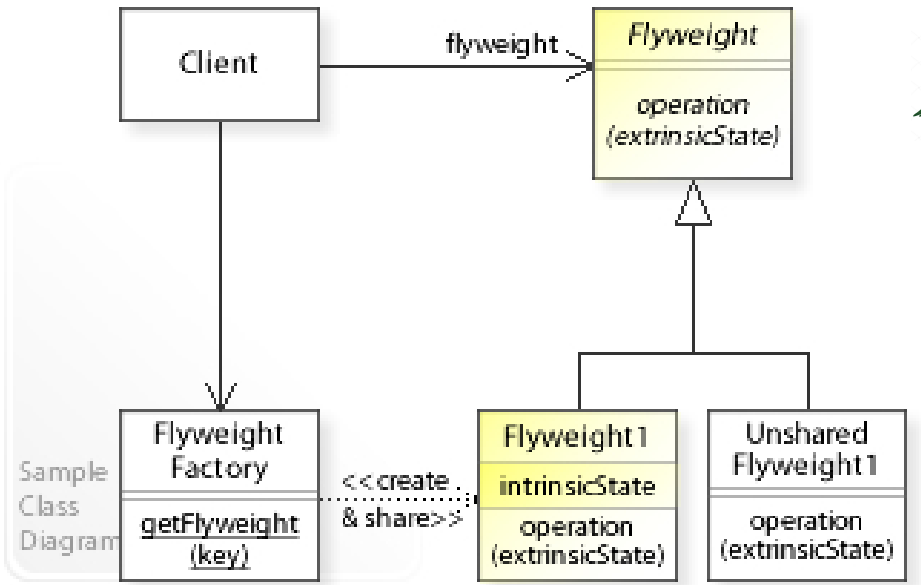
- ☑ Снижение зависимости клиента и компонентов системы
- ☑ Упрощает внешнее управление сложным объектом
- ☑ Есть опасность создать "Божественный объект"



Flyweight

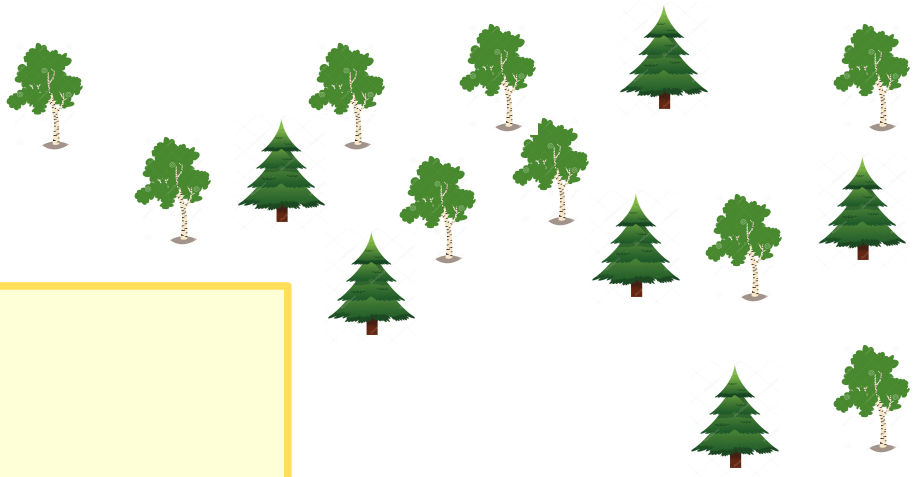
УНИВЕРСИТЕТ ИТМО

☑ Легковес - экономия памяти при большом числе объектов



Flyweight

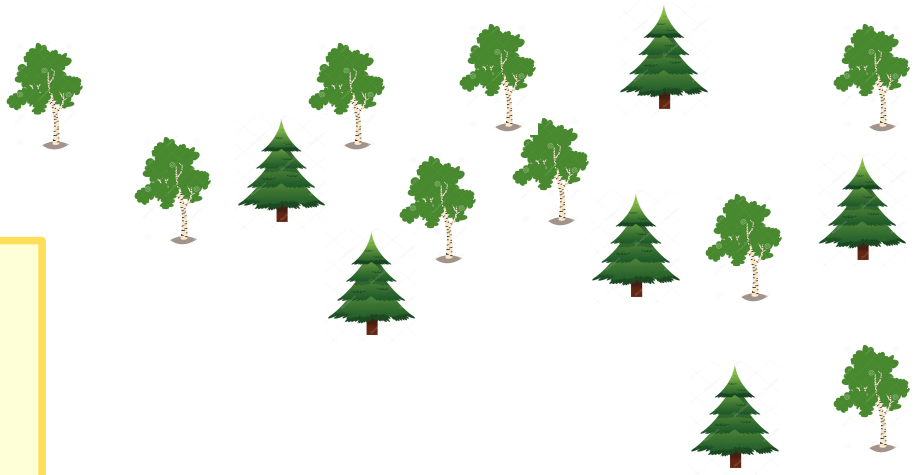
☑ Каждый объект - свое состояние



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

Flyweight

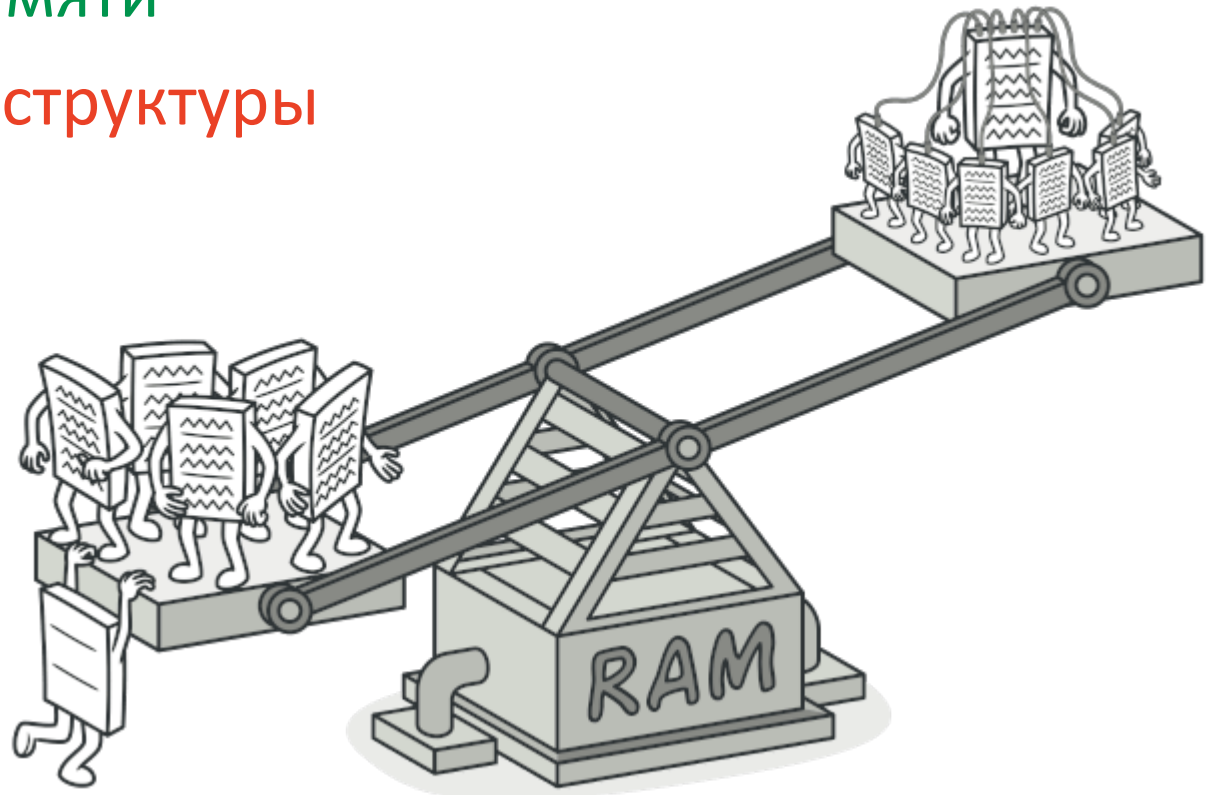
- ☑ Общее состояние храним в одном объекте
- ☑ Уникальное



```
class Tree {
 Color rgb;
 Image image;
 static List<Coordinate> position;
}
Tree birch = new Tree(...);
Tree pine = new Tree(...);
birch.addPosition(15,35);
pine.addPosition(20,40);
```

Flyweight

- ☑ Экономия памяти
- ☑ Усложнение структуры



УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Поведенческие шаблоны (Behavioral Patterns)

ITMO more than a UNIVERSITY

Поведенческие шаблоны

УНИВЕРСИТЕТ ИТМО

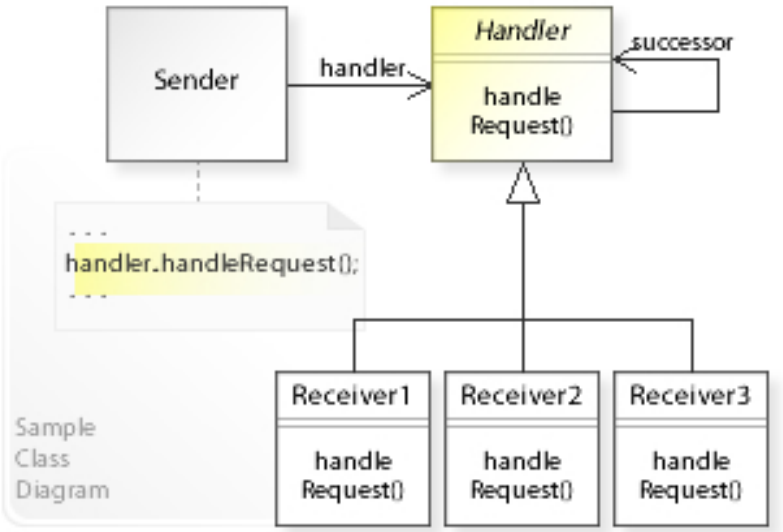
- ☑ Chain of Responsibility — Цепочка обязанностей
- ☑ Command - Команда
- ☑ Interpreter - Интерпретатор
- ☑ Iterator - Итератор
- ☑ Mediator - Посредник
- ☑ Memento - Хранитель
- ☑ Observer - Наблюдатель
- ☑ State - Состояние
- ☑ Strategy - Стратегия
- ☑ Template Method — Шаблонный метод
- ☑ Visitor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

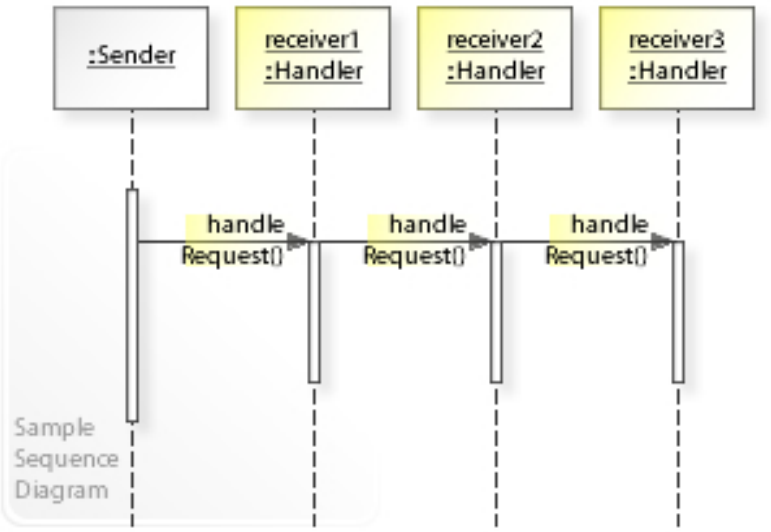
Java

Chain of Responsibility

УНИВЕРСИТЕТ ИТМО

- ☑ Передача запроса по цепочке обработчиков





Chain of Responsibility

- ☑ Передача запроса по цепочке обработчиков
- ☑ В банкомате застряла карта
  - техподдержка 8-800-111-11-11
    - бот: ваш звонок очень важен для нас,
      хотите кредит — нажмите 1, застряла карта — нажмите 2

Chain of Responsibility

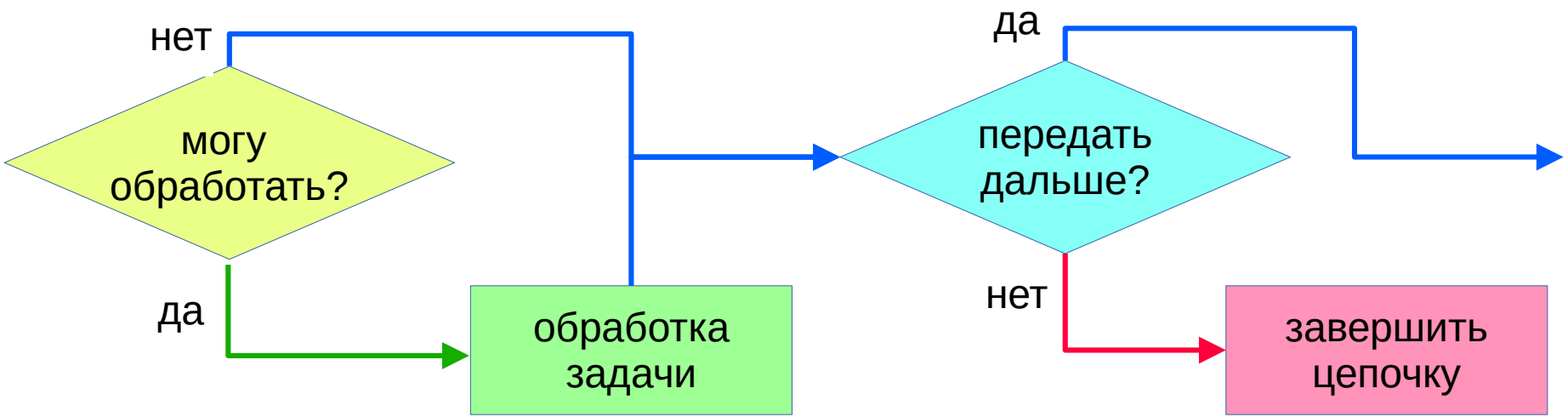
- ☑ Передача запроса по цепочке обработчиков
- ☑ В банкомате застряла карта
  - техподдержка 8-800-111-11-11
    - бот: ваш звонок очень важен для нас,
      хотите кредит — нажмите 1, застряла карта — нажмите 2
  - 1
    - другой бот: нажмите кнопку "отмена" на банкомате 4 раза

Chain of Responsibility

- ☑ Передача запроса по цепочке обработчиков
- ☑ В банкомате застряла карта
  - техподдержка 8-800-111-11-11
    - бот: ваш звонок очень важен для нас,
      хотите кредит — нажмите 1, застряла карта — нажмите 2
  - 1
    - другой бот: нажмите кнопку "отмена" на банкомате 4 раза
  - не помогает
    - оператор: стойте рядом, сейчас приедем

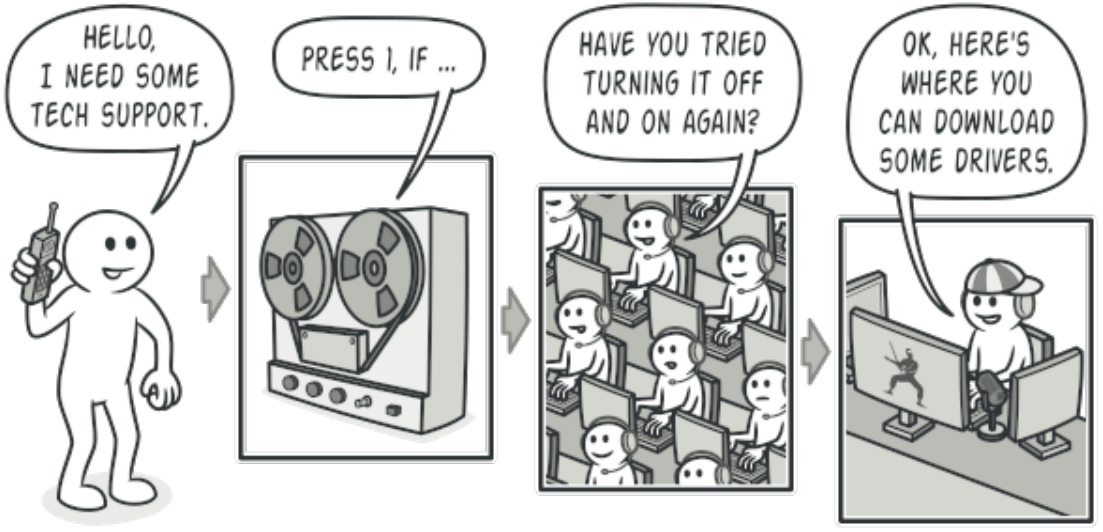
Chain of Responsibility

- ☑ Передача запроса по цепочке обработчиков
- ☑ Варианты действий



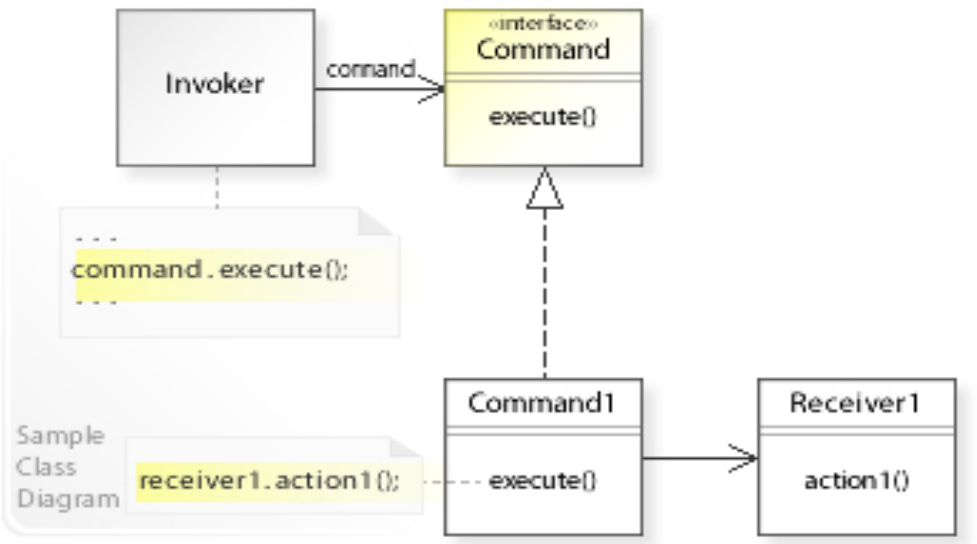
Chain of Responsibility

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



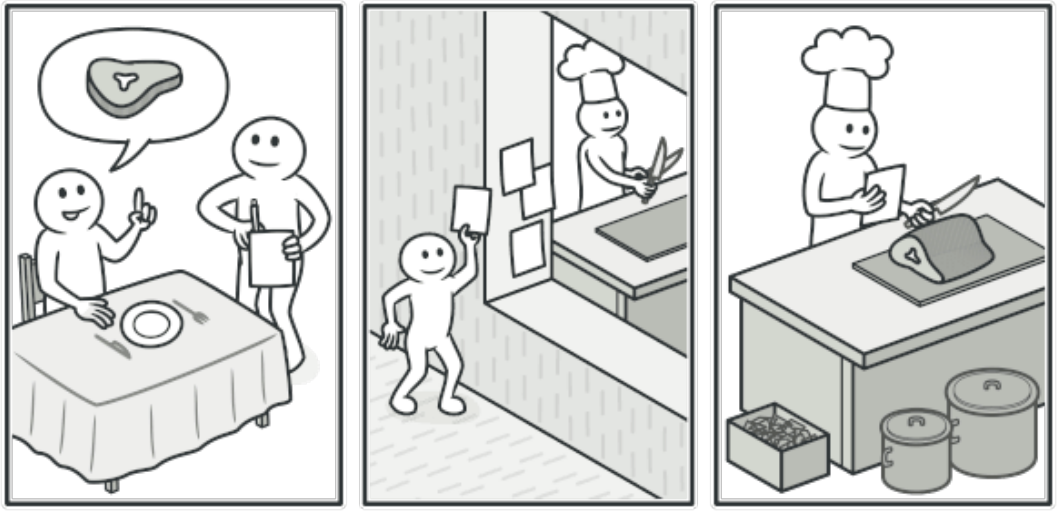
Command

- ☑ Команда - разделение вызова и исполнения команд
- ☑ Дополнительные возможности
  - Очередь команд
  - Макрокоманды
  - История команд



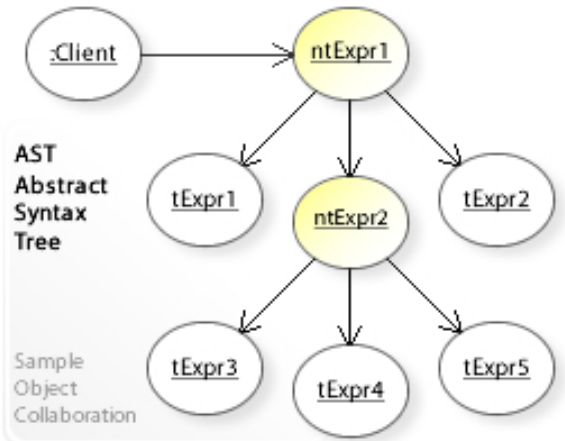
Command

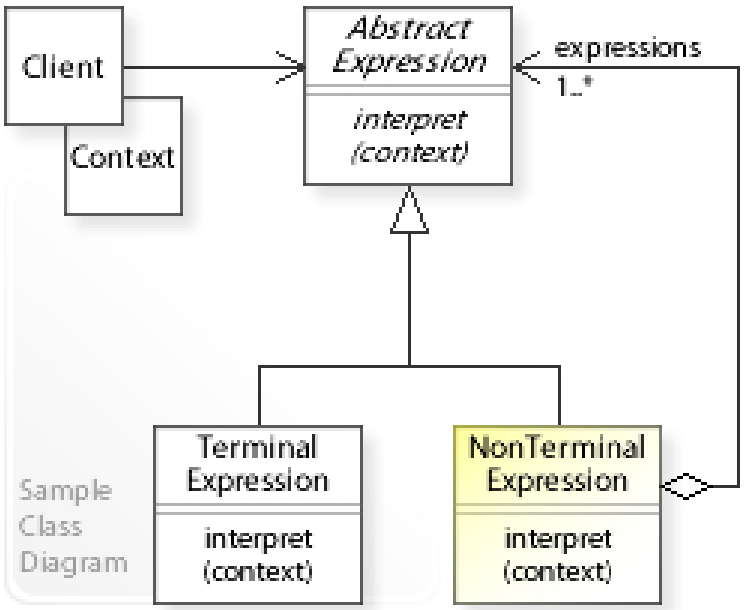
- ☑ Отделение кода исполнения команд от вызова
- ☑ Дополнительные возможности управления
- ☑ Усложнение кода (дополнительные классы)



Interpreter

- ☑ Интерпретатор языка для управления поведением
  - Регулярные выражения
  - Форматирование строк

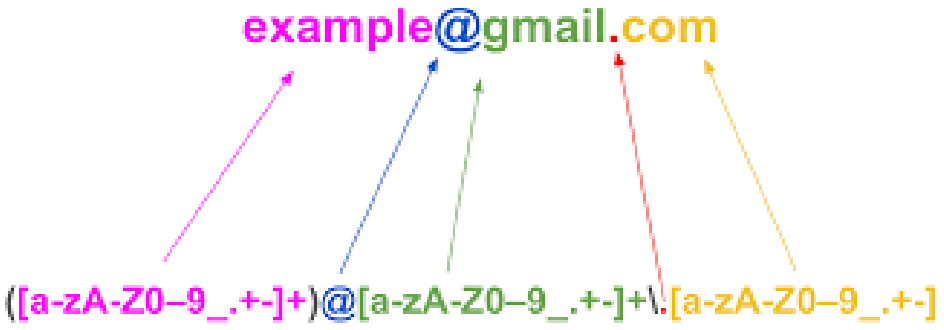


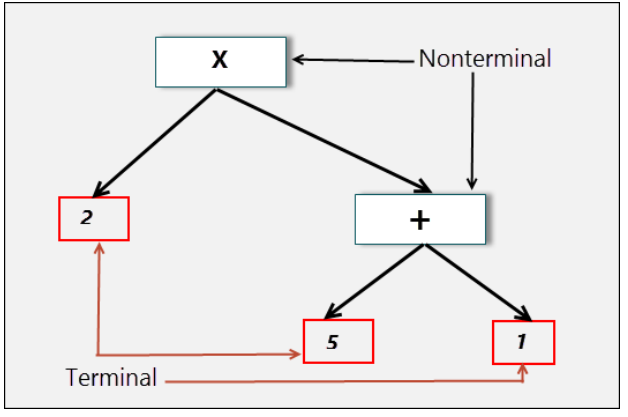


Interpreter

УНИВЕРСИТЕТ ИТМО

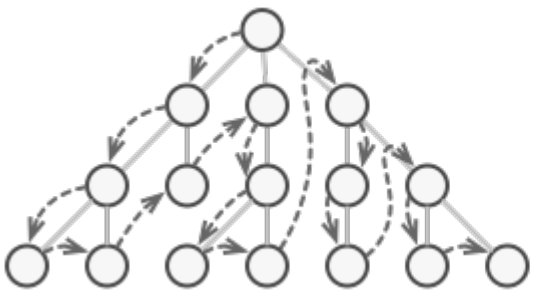
- ☑ Простота расширения и изменения языка
- ☑ Простота добавления новых способов
- ☑ Сложность сопровождения сложных грамматик

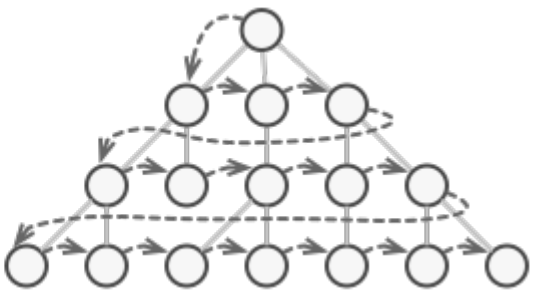




Iterator

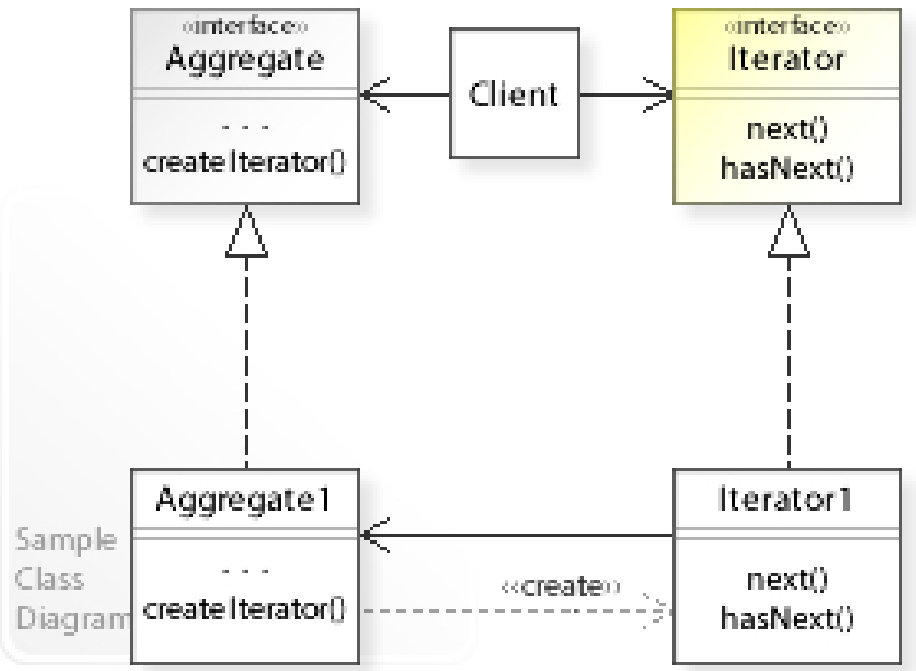
- ☑ Последовательный доступ к элементам коллекции
- ☑ Итератор знает внутреннюю структуру коллекции
- ☑ Разные способы обхода





Iterator

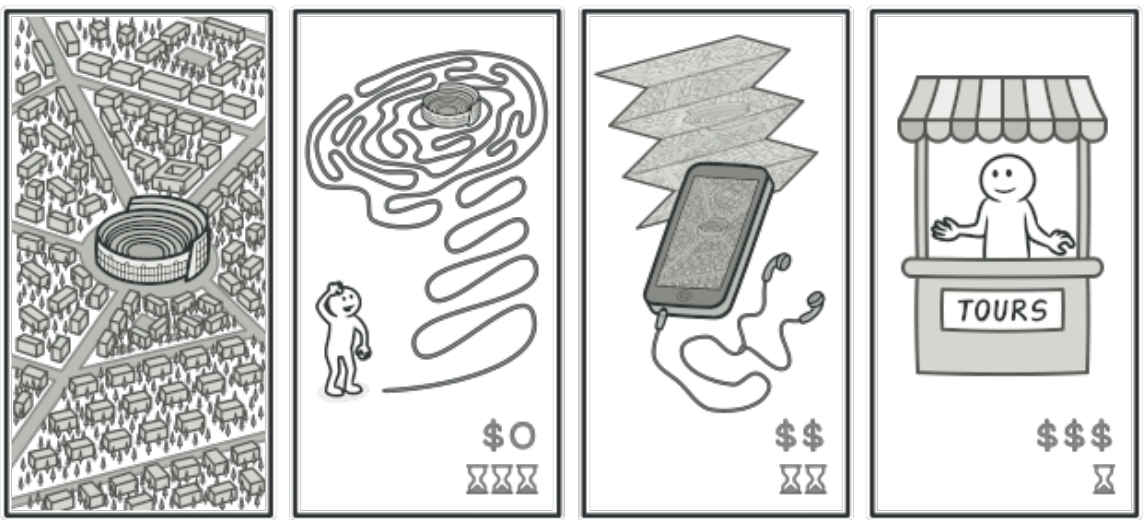
- ☑ Последовательный доступ к элементам коллекции
- ☑ Экскурсия
  - Реальный гид
  - Аудиогид
  - Путеводитель



Iterator

УНИВЕРСИТЕТ ИТМО

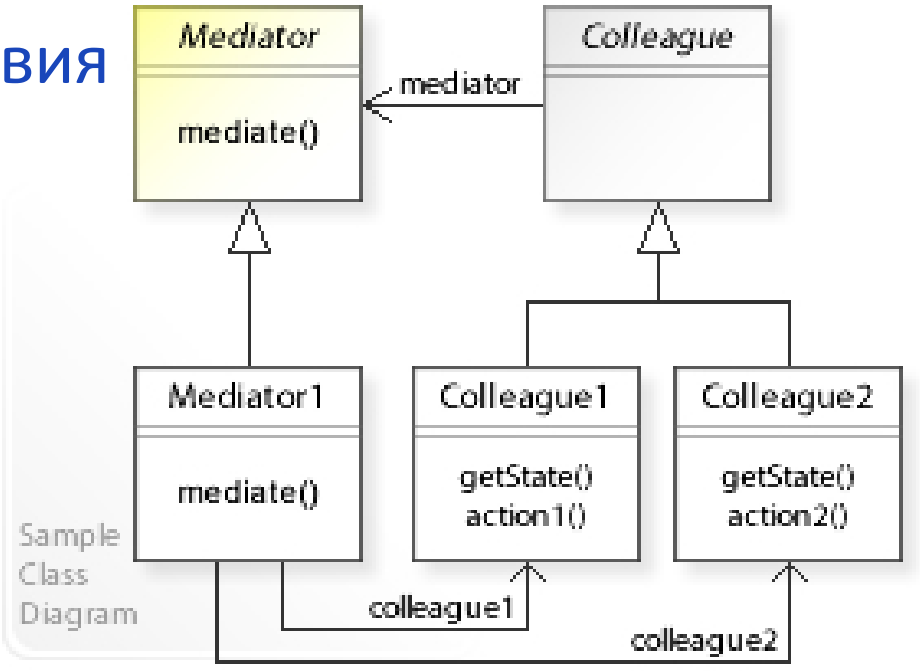
- ☑ Универсальный доступ ко всем коллекциям
- ☑ Гибкая реализация обхода структур
- ☑ Более сложный вариант, чем простой цикл



Java

Mediator

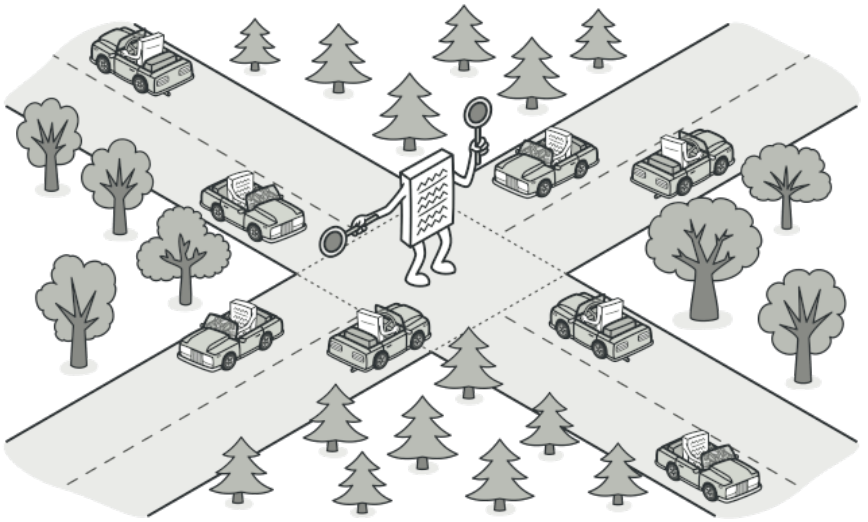
- ☑ Посредник для управления изменением состояния других объектов
- ☑ Вместо прямого взаимодействия объектов друг с другом они общаются через посредника



Mediator

УНИВЕРСИТЕТ ИТМО

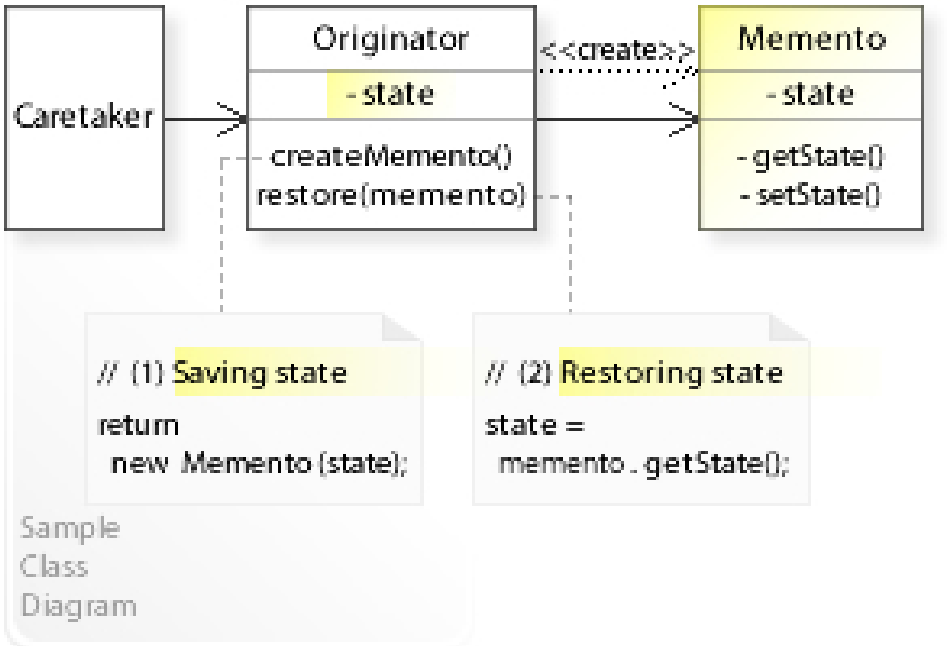
- ☑ Снижение зависимости между компонентами
- ☑ Простое централизованное управление
- ☑ Возможность разрастания посредника



Java

Memento

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



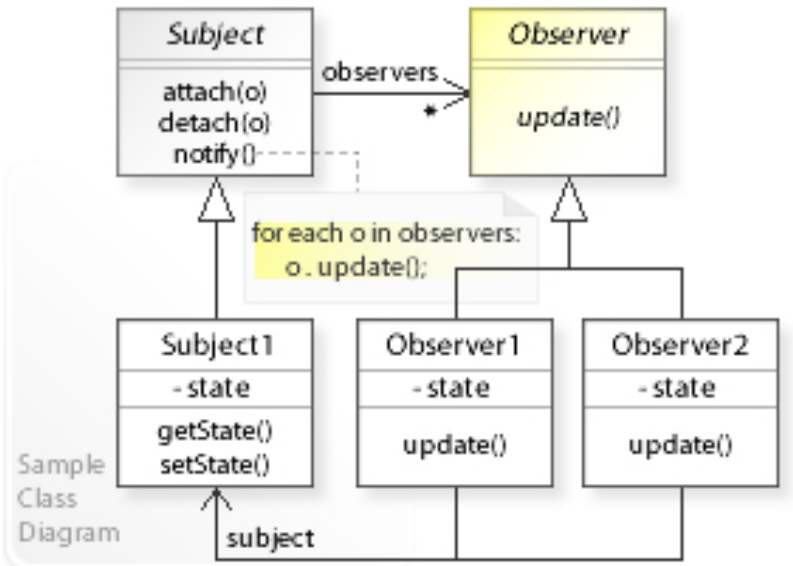
Memento

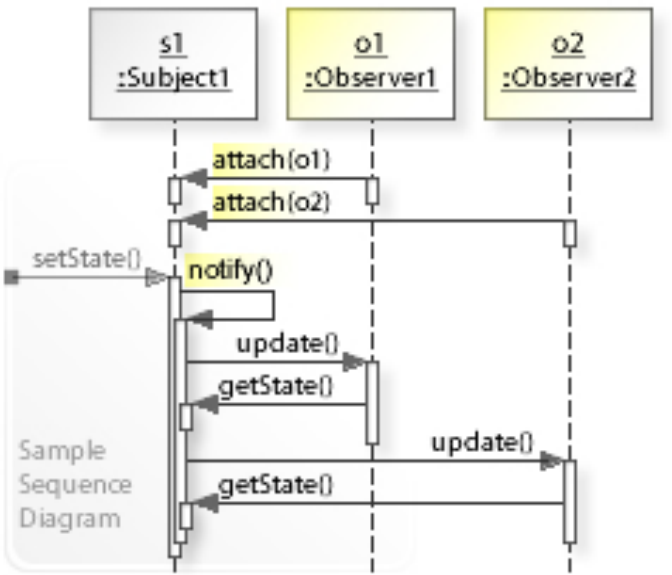
- ☑ Сохранение инкапсуляции исходного объекта
- ☑ Выделение отдельного хранилища снимков
- ☑ Может потребовать большого объема памяти

Observer

УНИВЕРСИТЕТ ИТМО

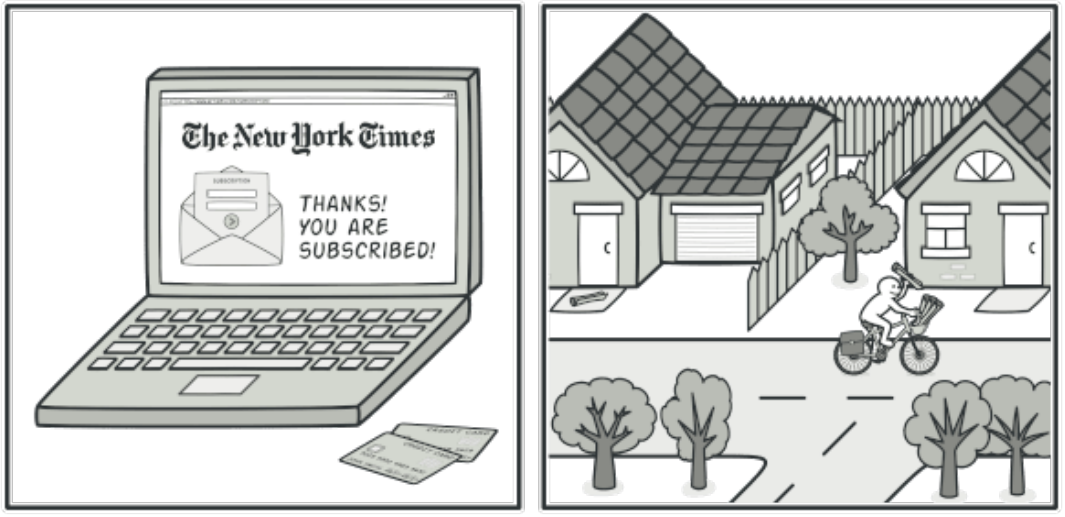
- ☑ Оповещение объектов об изменении состояния
- ☑ Подписка на извещения о событиях





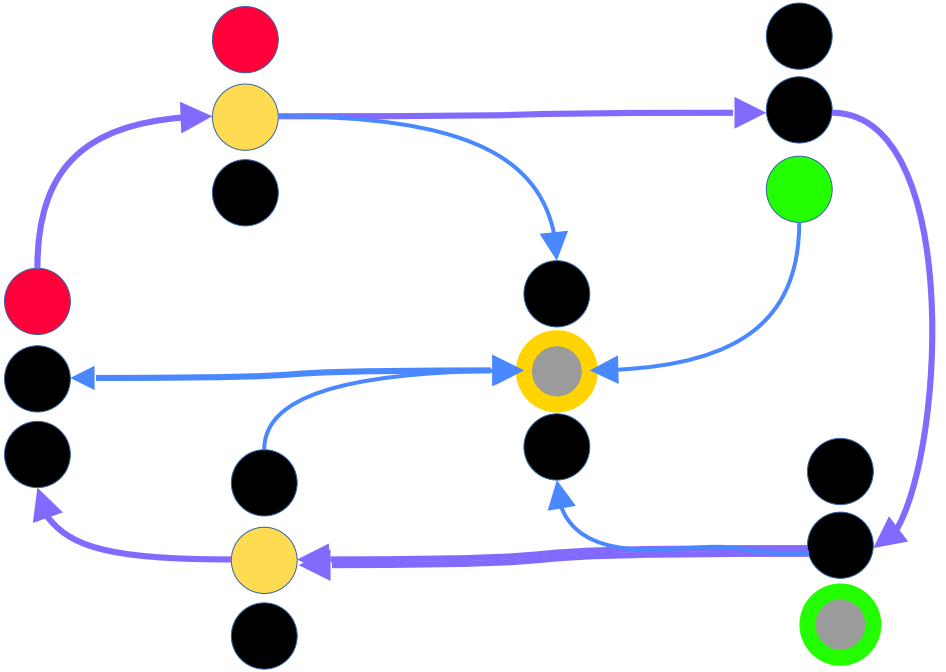
Observer

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



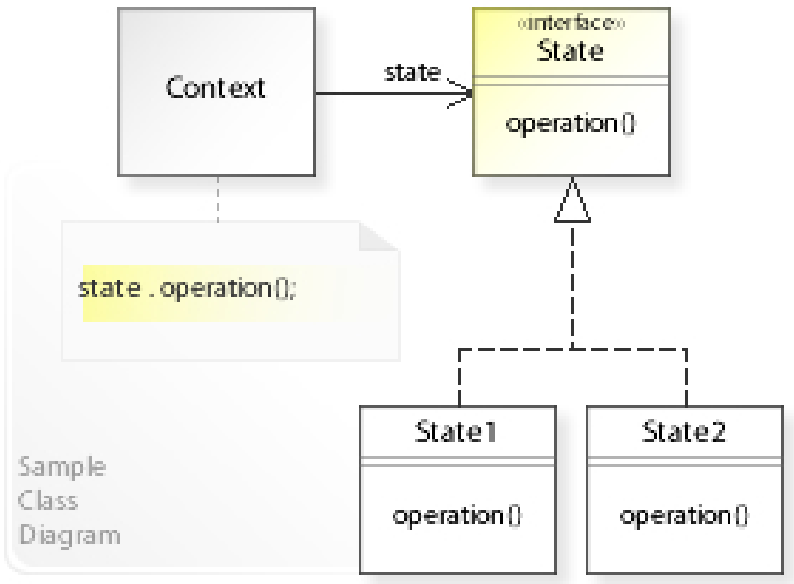
State

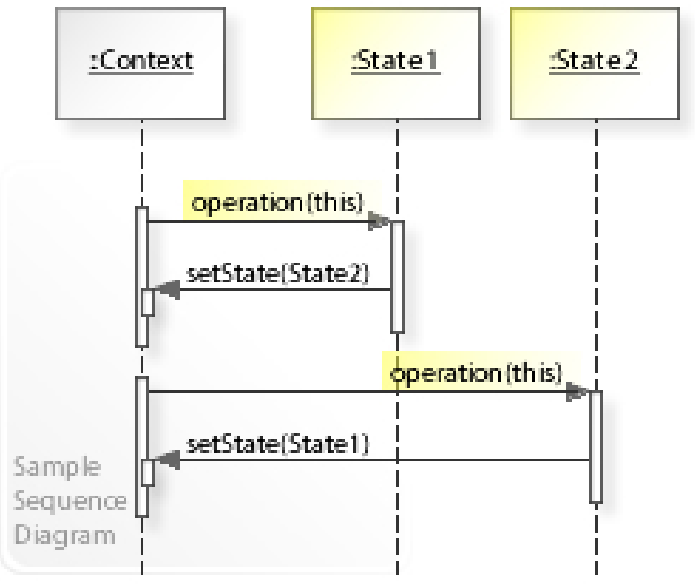
- ☑ Изменяет поведение объекта в зависимости от состояния — реализация конечного автомата



State

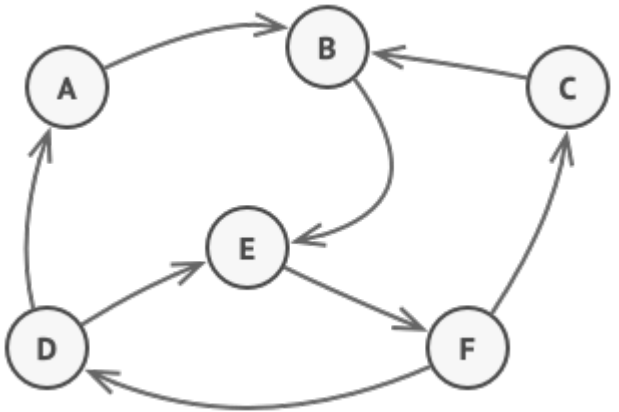
- ☑ Изменяет поведение объекта в зависимости от состояния — реализация конечного автомата





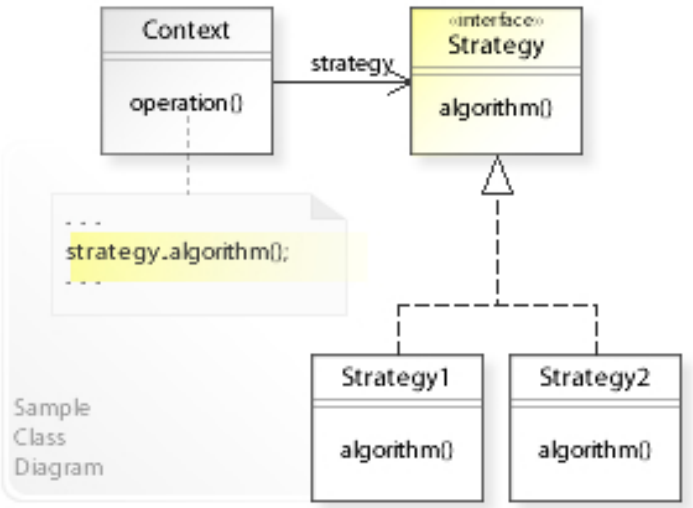
State

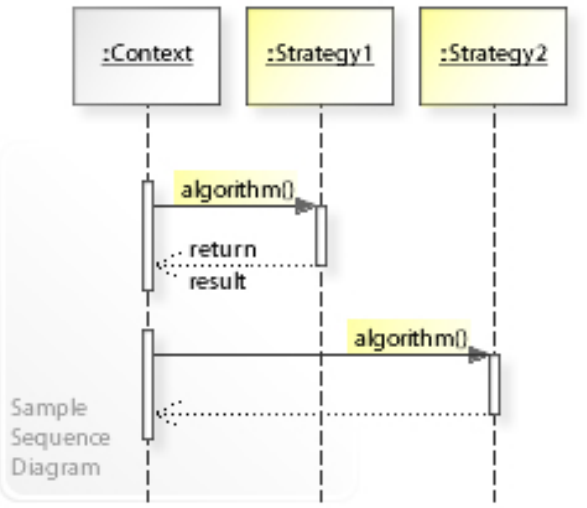
- ☑ Избавляет от множества операторов if
- ☑ Выделяет код для каждого состояния в одном месте
- ☑ В некоторых случаях слишком сложный код



Strategy

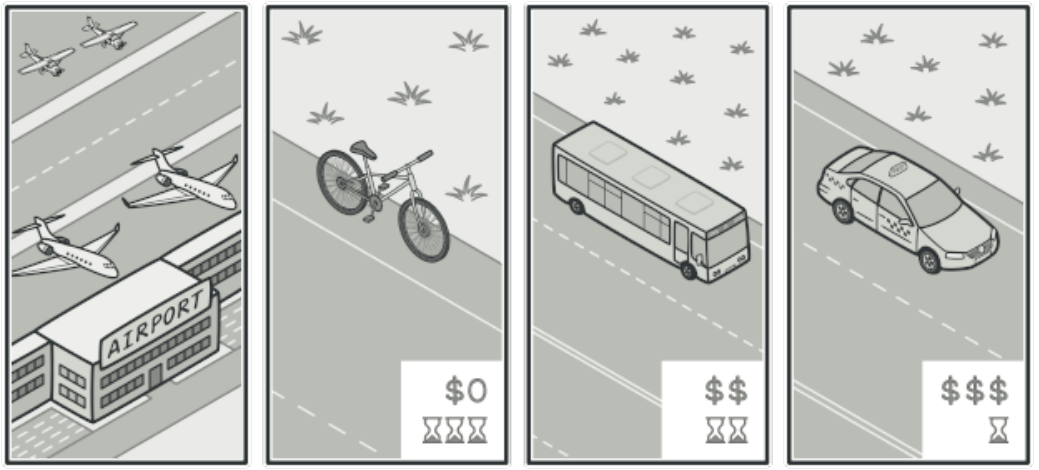
- ☑ Выбор одного из алгоритмов, реализованных в классе
  - Как добраться до аэропорта
    - Автобус
    - Такси
    - Пешком





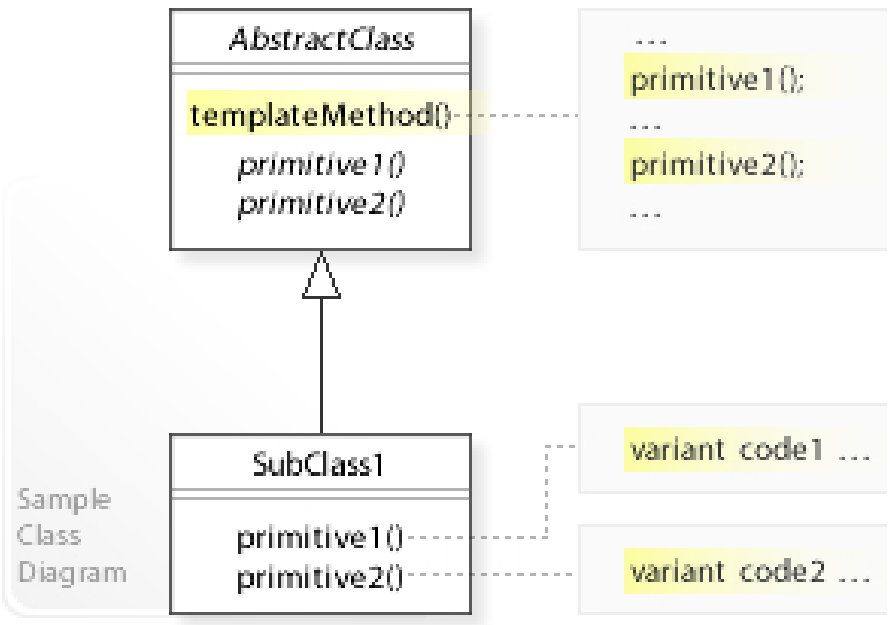
Strategy

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



Template Method

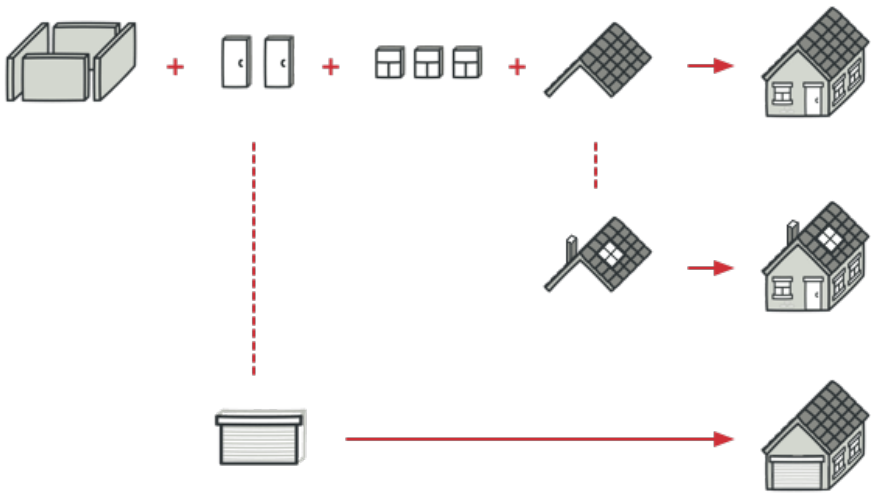
- ☑ Позволяет реализовать часть поведения в базовом классе, остальное реализуется в подклассах
- ☑ Покемоны
  - Move.applyOppDamage()

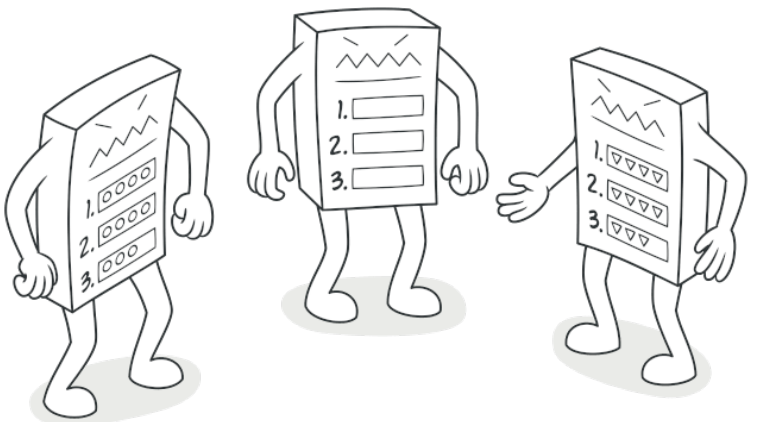


Template Method

УНИВЕРСИТЕТ ИТМО

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



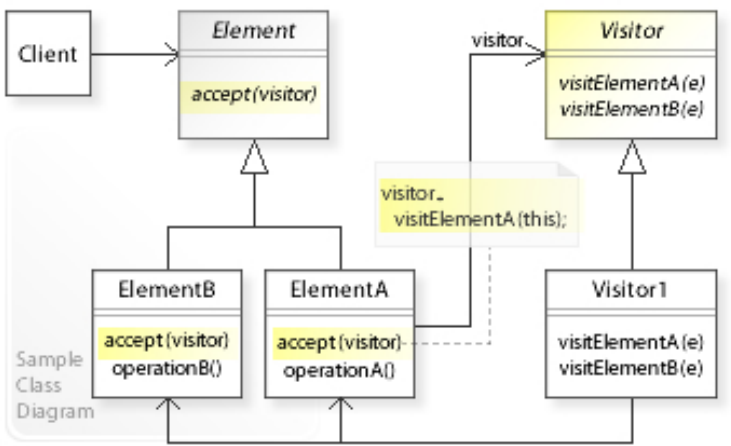


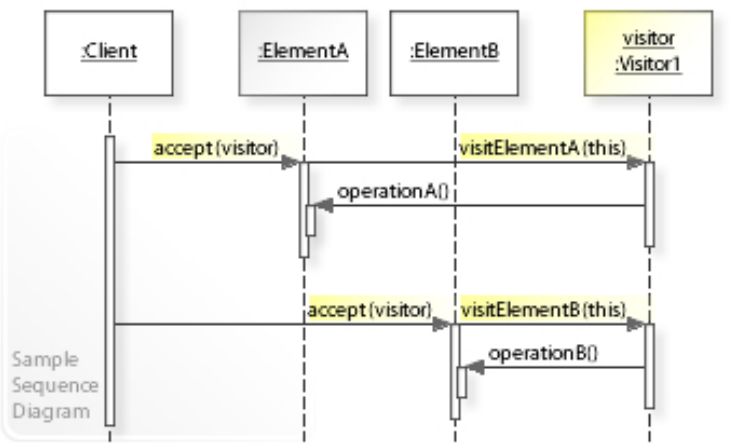
Java

Visitor

УНИВЕРСИТЕТ ИТМО

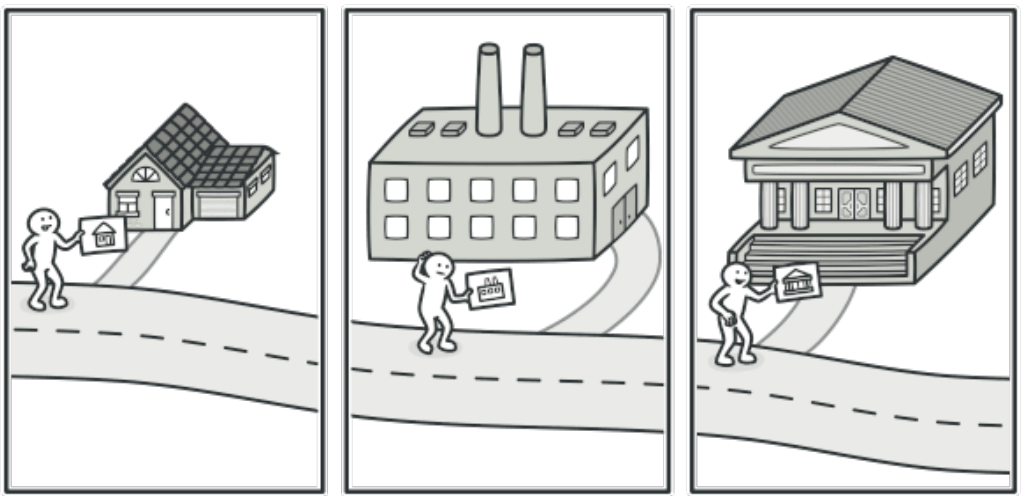
- ☑ Позволяет сгруппировать операции, выполняемые над структурой объектов
- ☑ Применяется если структура элементов более стабильна, чем набор операций



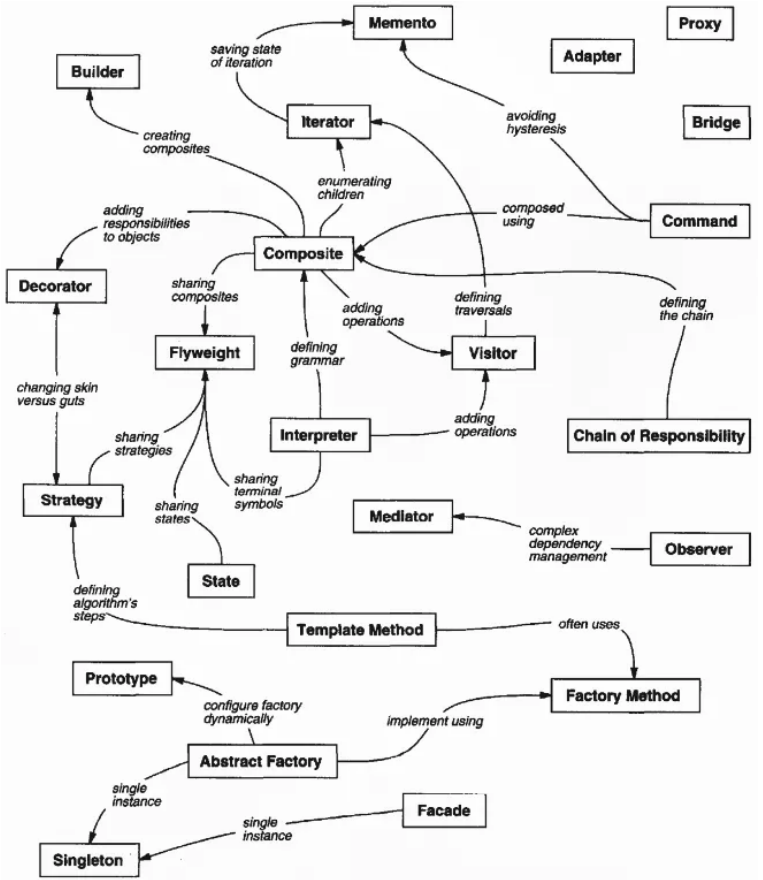


Visitor

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



Все вместе


Memento
Proxy
saving state of iteration
Builder
Adapter
Iterator
avoiding hysteresis
Bridge
creating composites
enumerating children
adding responsibilities to objects
composed using
Command
Composite
Decorator
sharing composites
adding operations
defining traversals
defining the chain
Flyweight
defining grammar
Visitor
changing skin versus guts
sharing strategies
Interpreter
adding operations
Chain of Responsibility
sharing terminal symbols
Strategy
sharing states
Mediator
complex dependency management
Observer
State
defining algorithm's steps
Template Method
often uses
Prototype
Factory Method
configure factory dynamically
implement using
Abstract Factory
single instance
Facade
single instance
Singleton