

Шаблоны проектирования

- Изменяются

- пожелания пользователя
- взгляд разработчика
- внешние условия

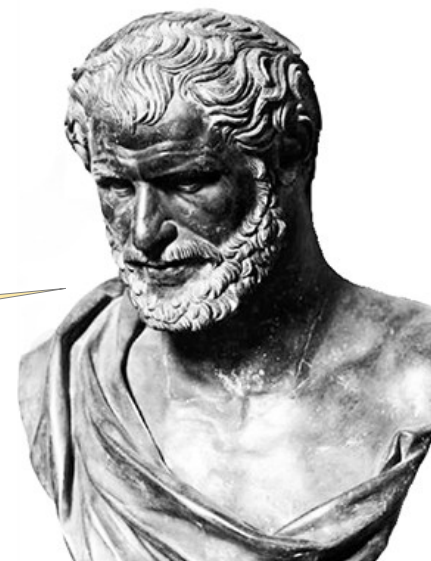
Программы меняются



Овидий

omnia mutantur

πάντα ρεῖ



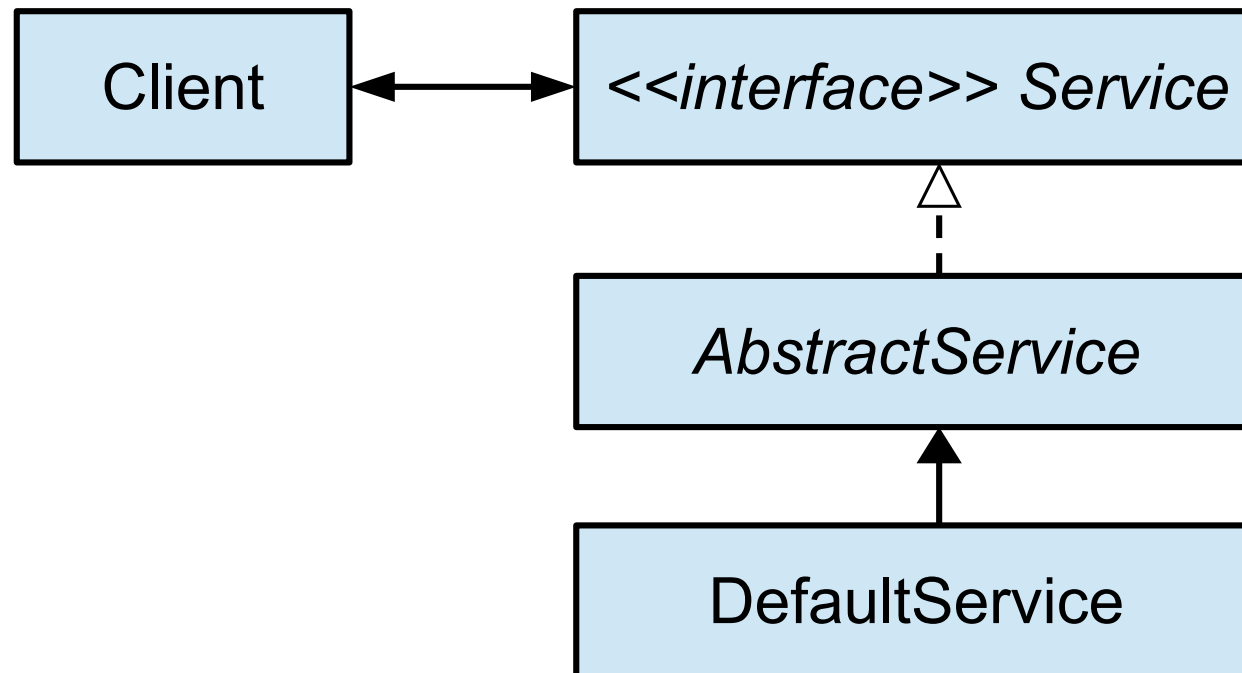
Гераклит

- Кто-то когда-то уже делал что-то похожее
- Потом пришлось вносить изменения
- Стало понятно, как надо было делать

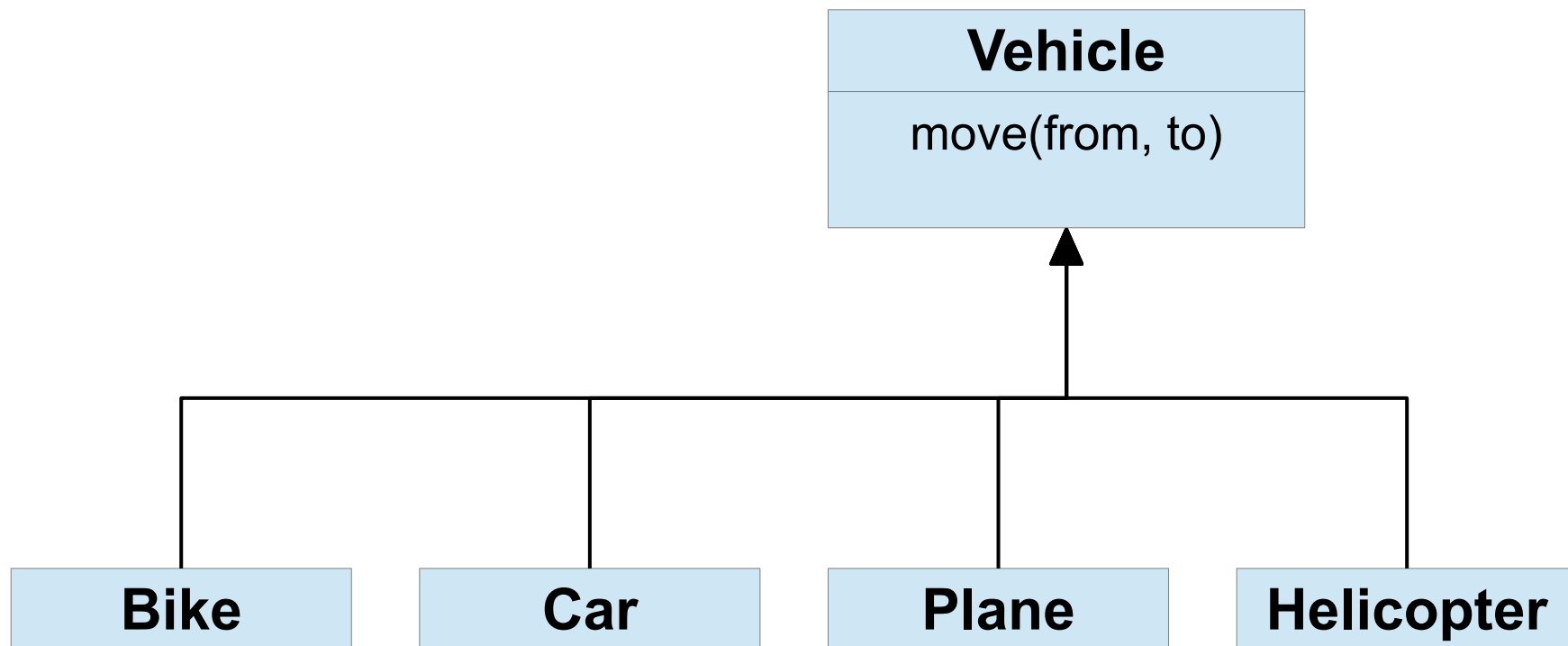


GoF book

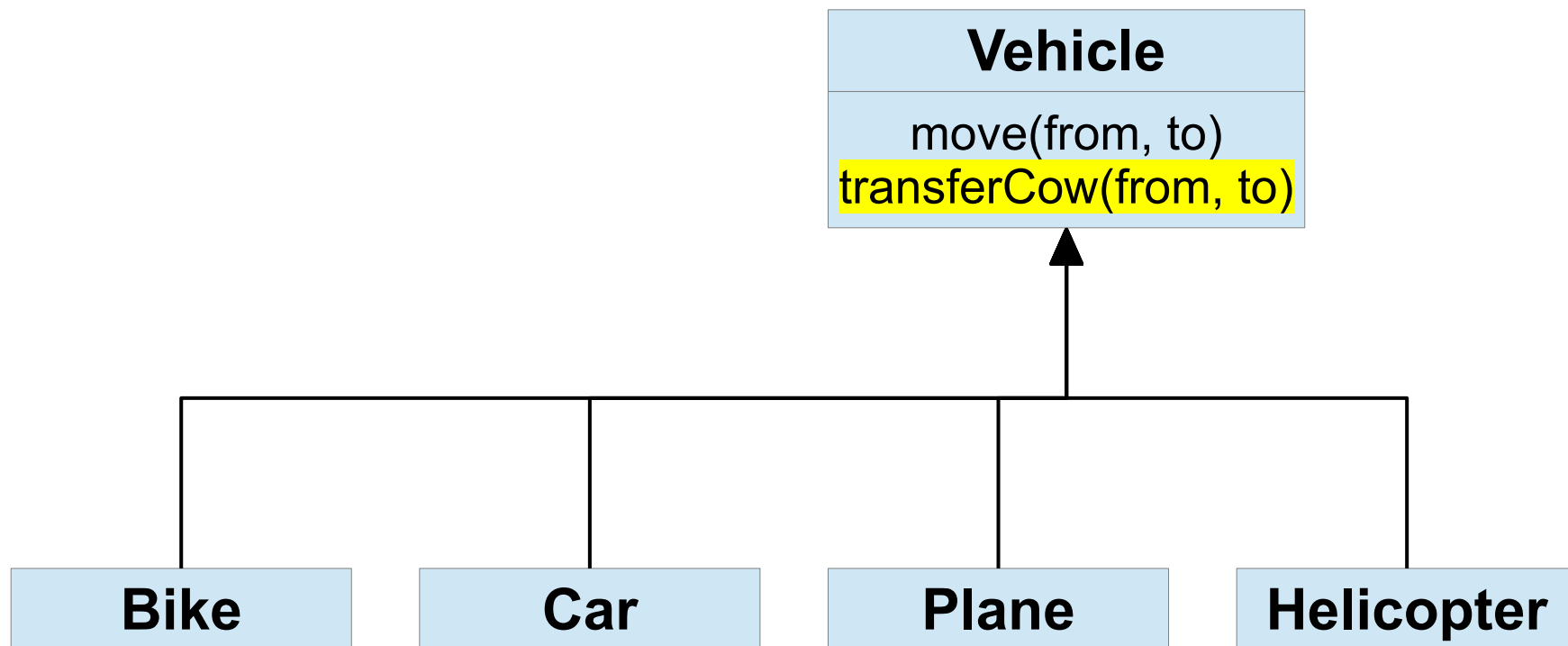
- Взаимодействие с абстракцией - ИНТЕРФЕЙС



- Умеет предок - умеют потомки
- Полиморфизм



- Перевозка коровы - наследование??

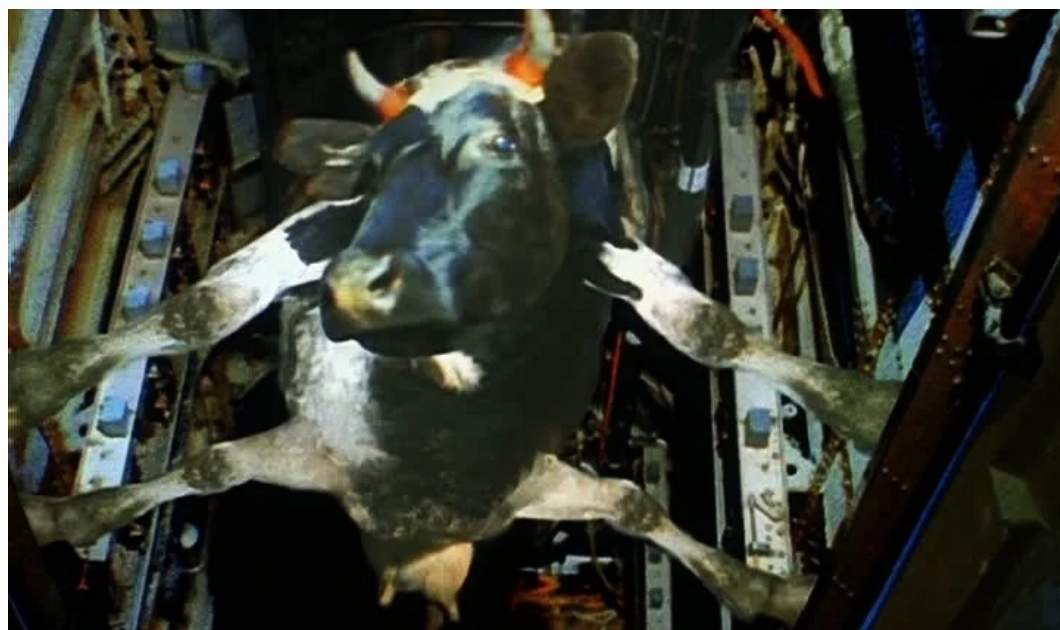


Перевозка коровы



(С) х/ф «Мимино»

Перевозка коровы



(С) х/ф «Особенности национальной охоты»

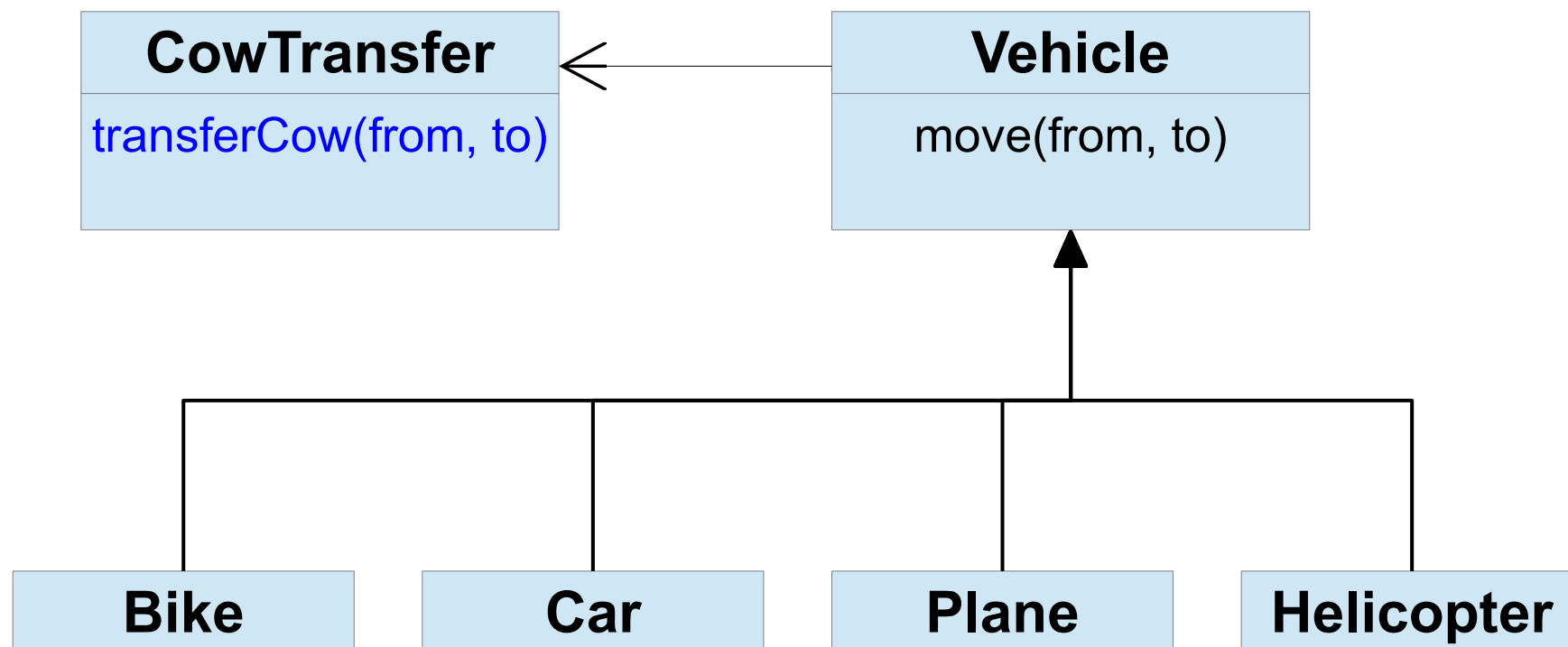
Перевозка коровы



Перевозка коровы



- Перевозка коровы - делегирование



- **Наследование** - статическое отношение - классификация
- **Делегирование** - динамическое отношение - роль

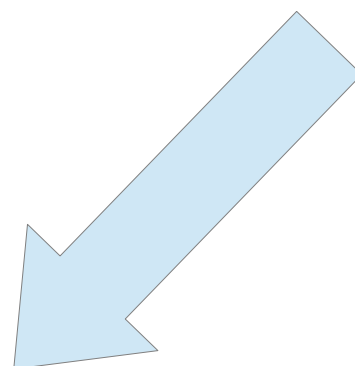
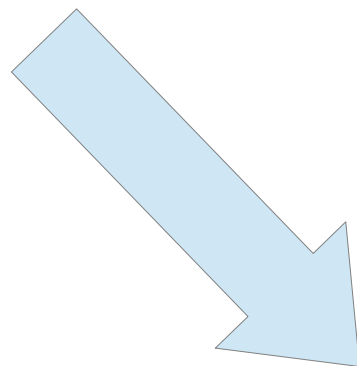
```
class Animal {  
    sleep() { ... }  
}  
  
class Cat extends Animal {  
}  
  
Animal animal = new Cat();  
animal.sleep();
```

```
class Flying {  
    fly() { ... }  
}  
  
class UFO {  
    Flying move = new Flying();  
    public fly() {  
        move.fly();  
    }  
}  
  
UFO ufo = new UFO();  
ufo.fly();
```

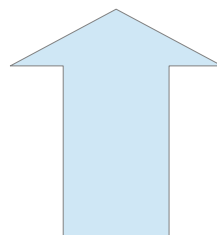
Как справиться с изменениями?

- Интерфейсы

- Делегирование



**ШАБЛОНЫ
PATTERNS**



- Отделить изменяющееся от постоянного
 - Инкапсулировать изменяющееся

Порождающие шаблоны проектирования

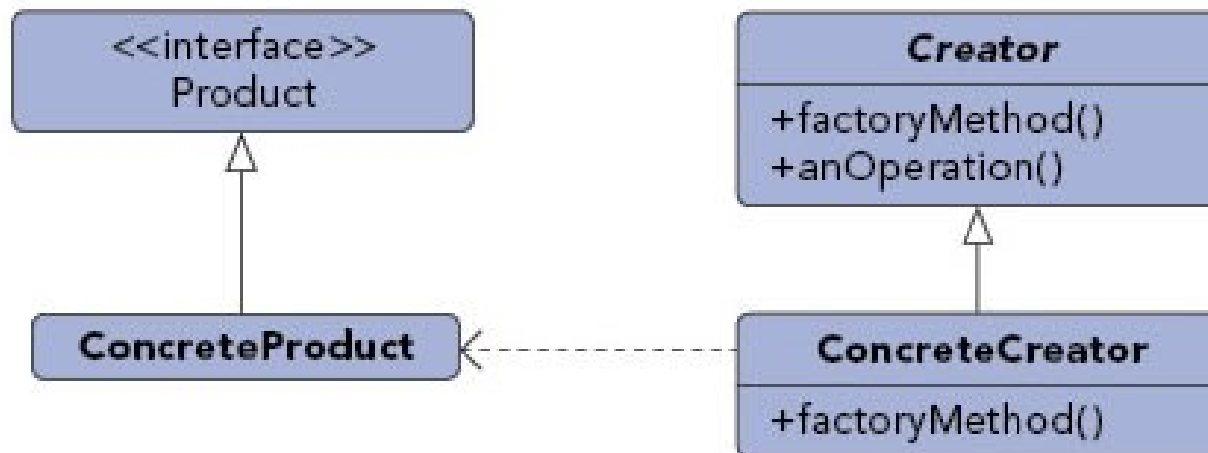
- Factory Method — Фабричный метод
- Abstract Factory — Абстрактная фабрика
- Builder — Строитель
- Prototype — Прототип
- Singleton - Одиночка
- Object Pool — Пул объектов

Factory Method

- Возможность создавать объекты разных типов с помощью фабрики

- Примеры:

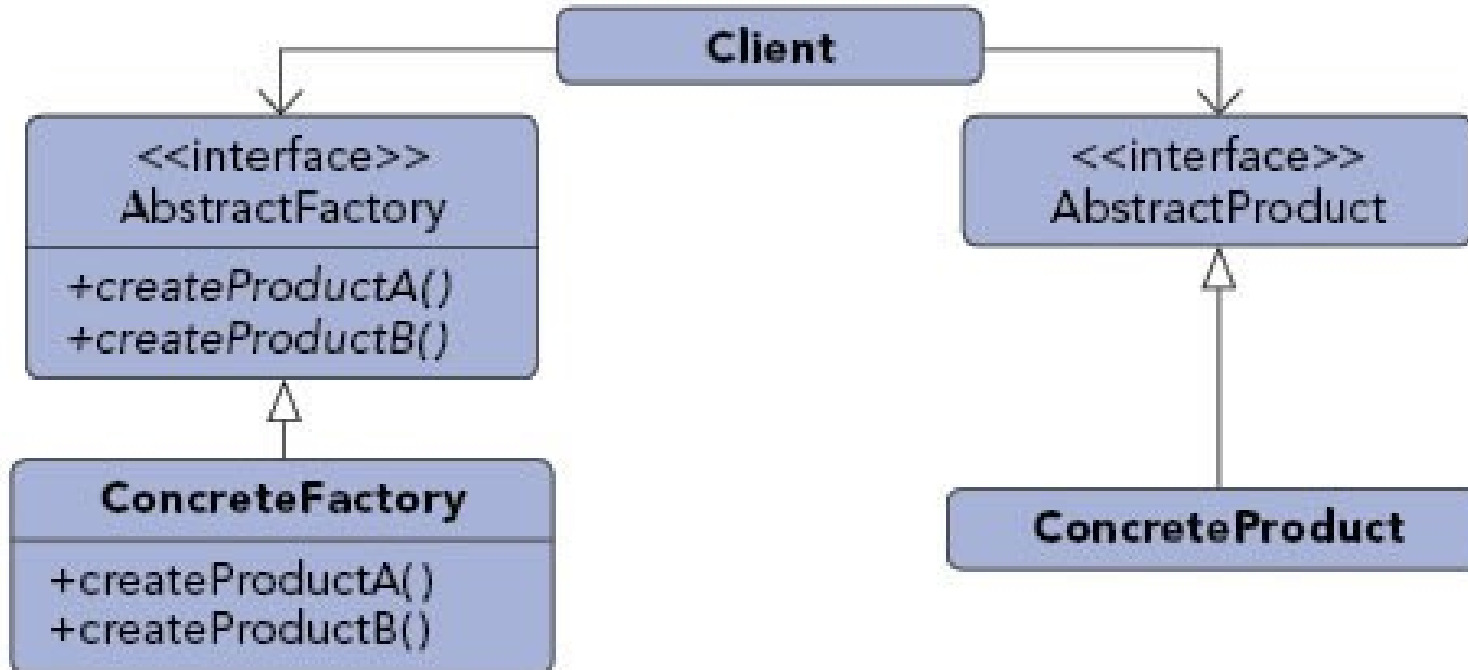
```
Connection conn = DriverManager.getConnection(args);  
Statement stat = conn.createStatement();
```



Abstract Factory

- Создание семейства объектов

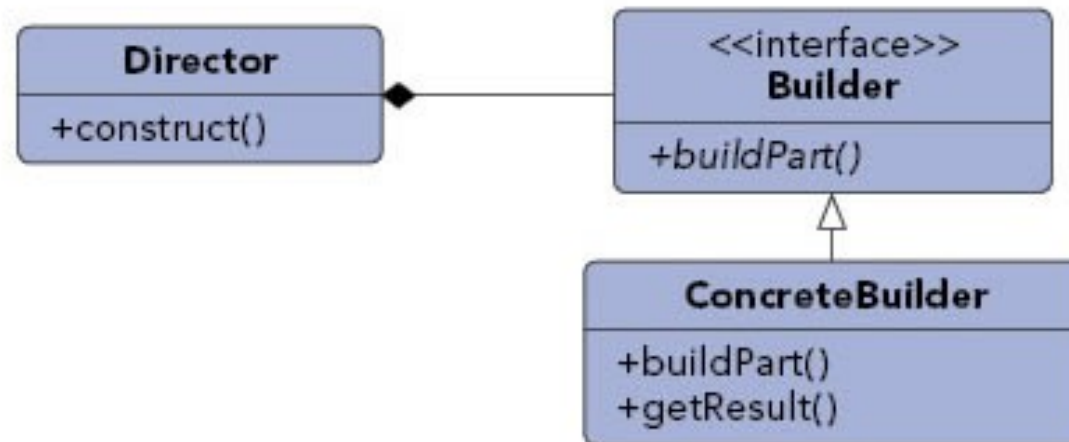
```
Factory f1 = new PepsiFactory();
Product p1 = f.createColaDrink();
Factory f2 = new CocacolaFactory();
Product p2 = f.createOrangeDrink();
```



- Создание объектов со сложным набором параметров

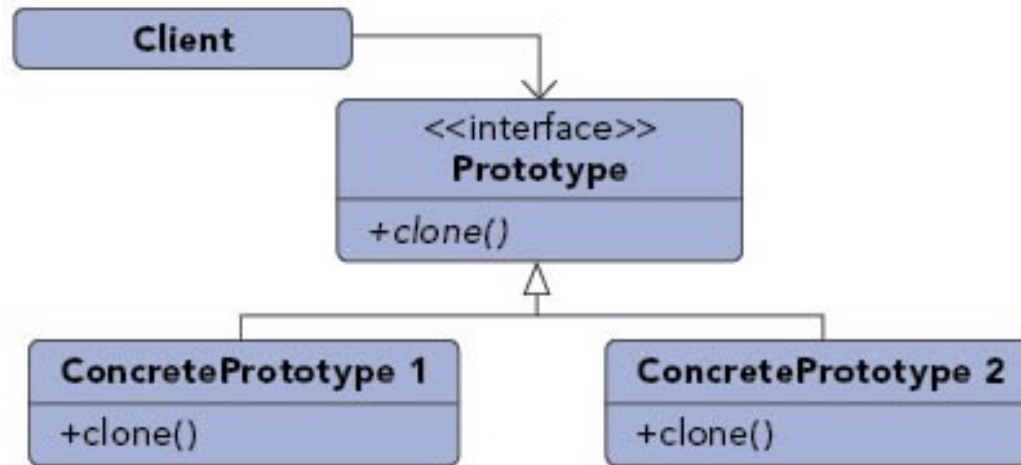
```
Builder b = new Builder()  
    .setWhere("Moscow")  
    .setTransport("train")  
    .setDate(1,6,2020);  
Product p = b.create();
```

```
Director d = new Director();  
p = d.makeTrainTrin("Moscow", new Date(1.6.2020));
```



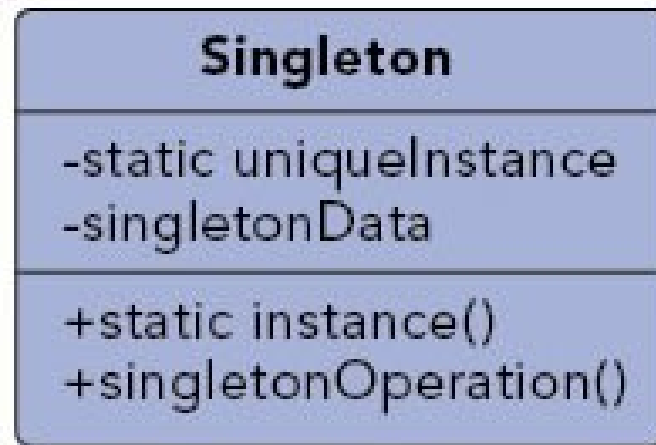
- Создание объектов клонированием

```
Cell cell = new HumanCell();  
Cell cell2 = cell.clone();
```



Singleton

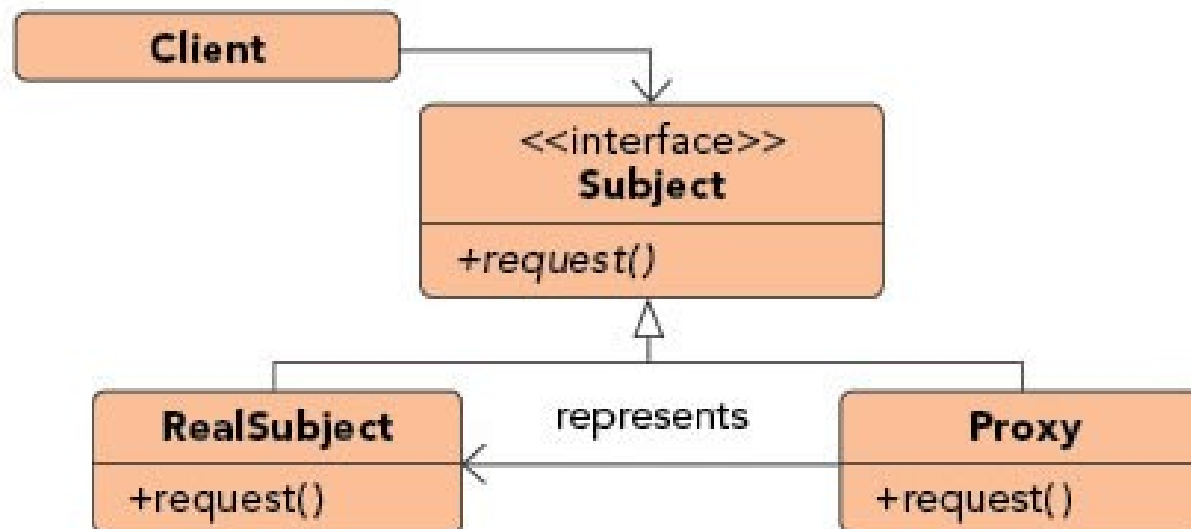
- Ограничение количества экземпляров класса
- private конструктор
- 1 объект - Singleton
 - System.console()
- > 1 объекта — Object Pool
 - ThreadPool



Структурные шаблоны проектирования

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

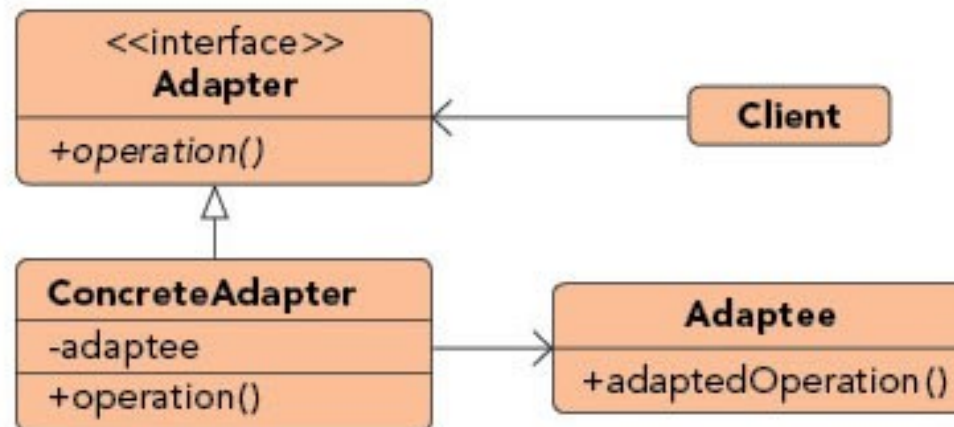
- Делегирование с тем же интерфейсом
 - Создание объекта требует времени
 - Создание объекта может не потребоваться
 - Реальный объект может временно отсутствовать
 - Перехват методов



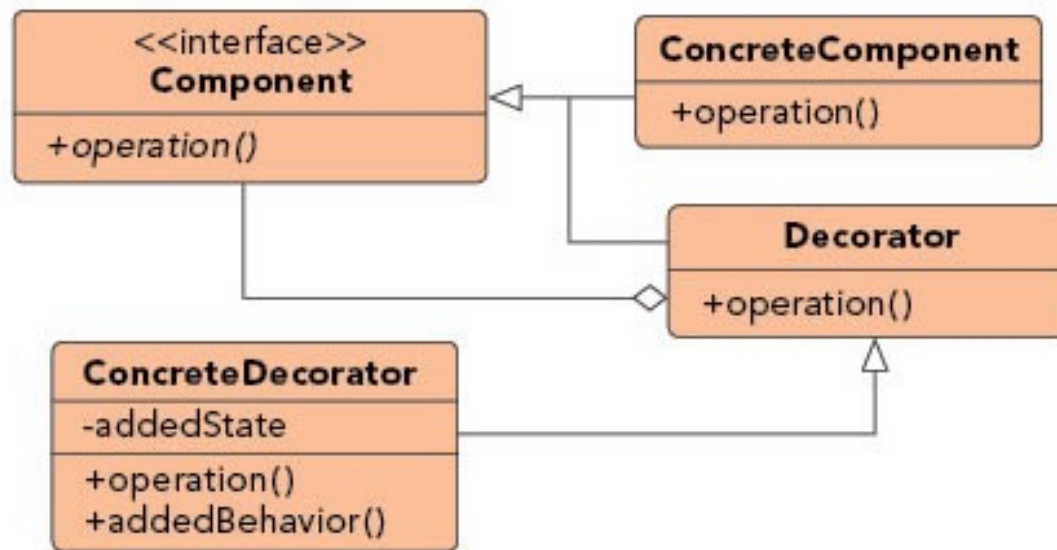
- Делегирование с заменой интерфейса

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

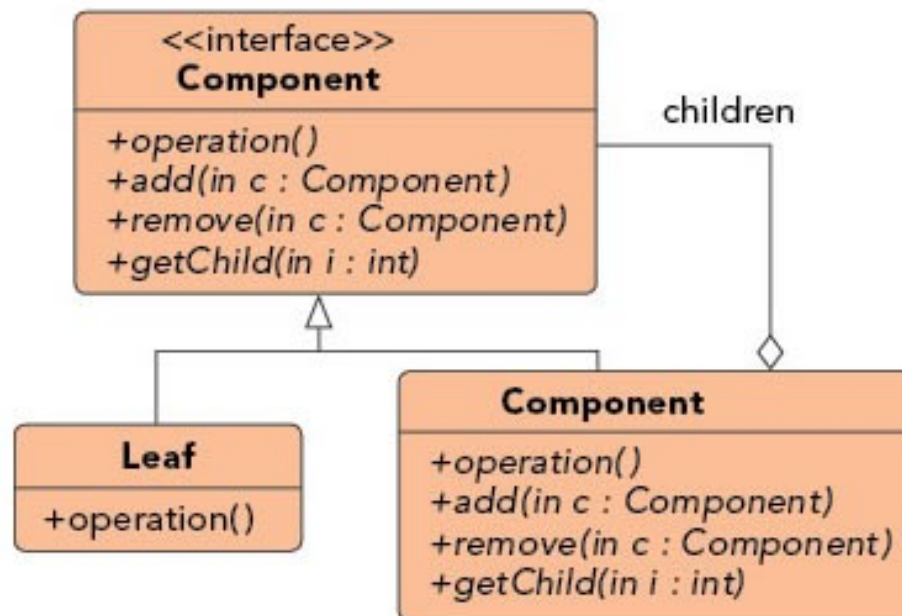
class Adapter implements EUSocket {
    UKSocket uks;
    getEUPower() {
        return uks.getUKPower();
    }
}
```



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



- Иерархические структуры
- GUI: контейнер — это компонент, на котором расположены другие компоненты



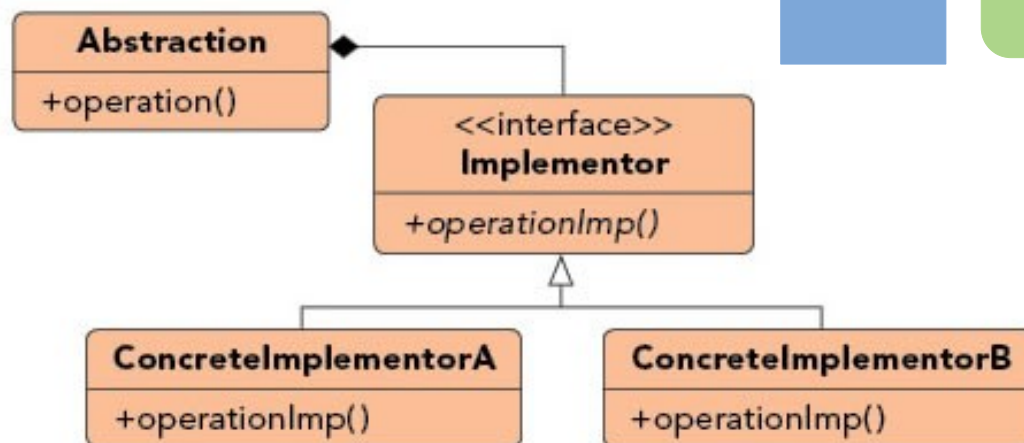
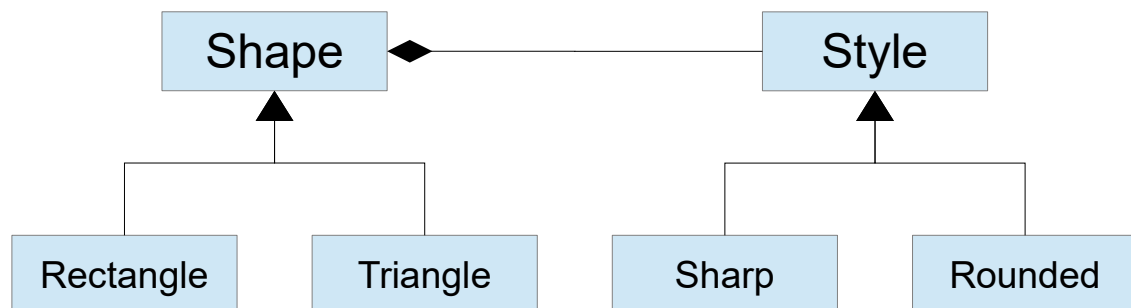
- Разделяет иерархии абстракций и реализаций
 - Абстракции - фигуры (треугольник, прямоугольник)
 - Реализации - стиль углов (обычные, закругленные)
- Поведение делегируется реализации



```

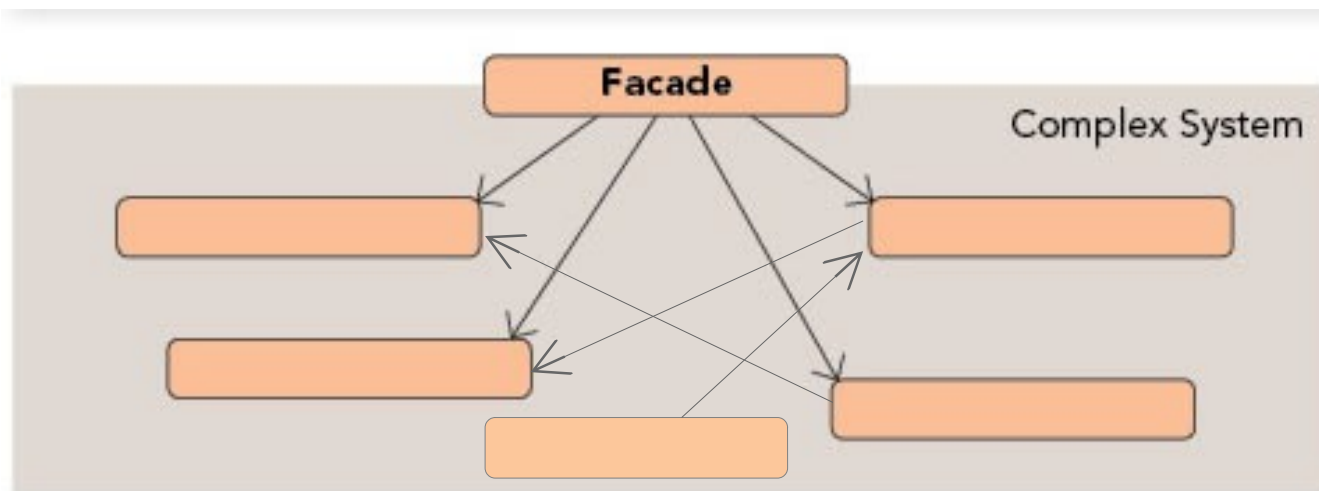
abstract class Shape {
    Style style;
    draw() {
        style.line();
    }
}

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



Facade

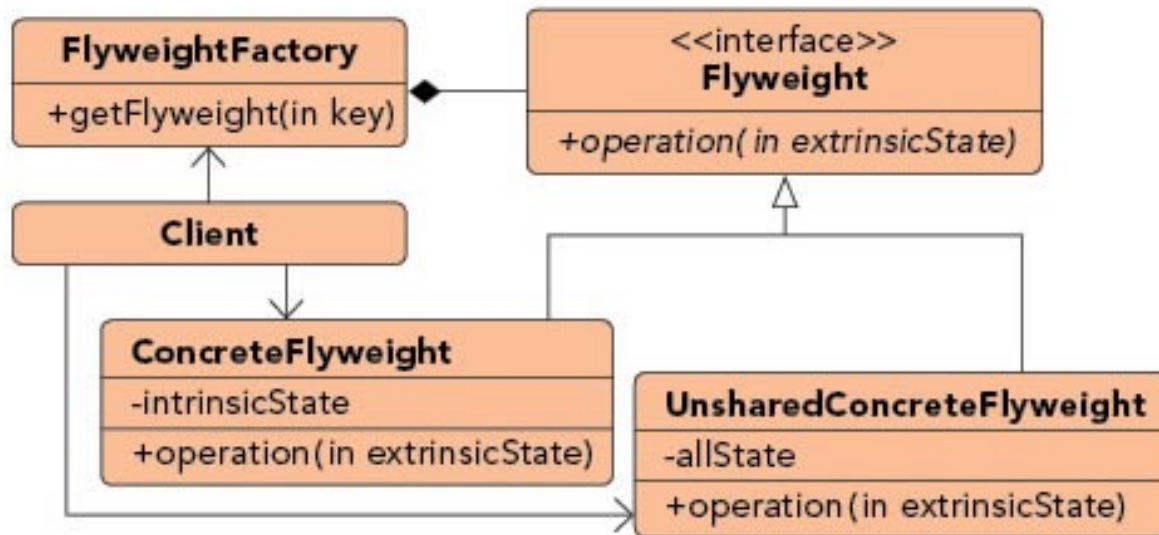
- Простой интерфейс к сложным системам
- Facade.start()



Flyweight

- Представление большого множества объектов одним
- Для экономии памяти

```
PhoneNumber number = new PhoneNumber("+79119876543");  
Phone.call("+79119876543");
```

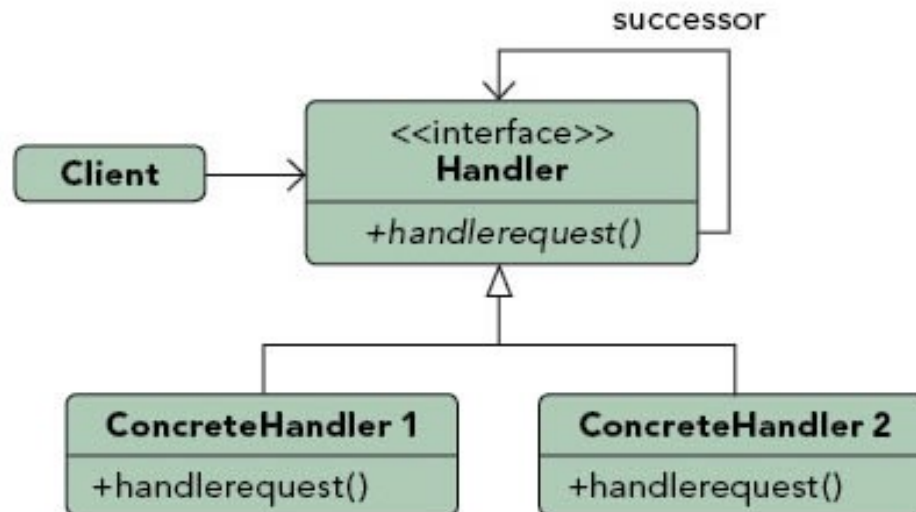


Поведенческие шаблоны проектирования

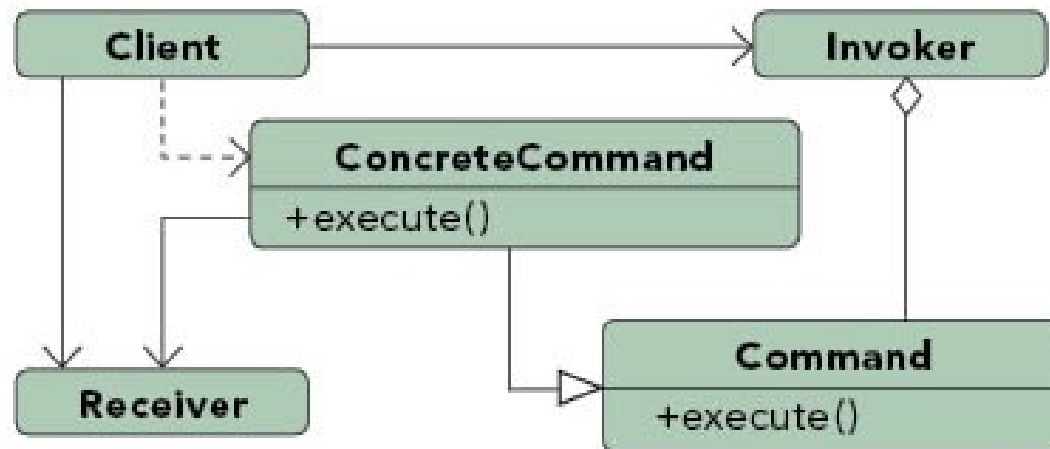
- Chain of Responsibility — Цепочка обязанностей
- Command - Команда
- Interpreter - Интерпретатор
- Iterator - Итератор
- Mediator - Посредник
- Memento - Хранитель
- Observer - Наблюдатель
- State - Состояние
- Strategy - Стратегия
- Template Method — Шаблонный метод
- Visitor - Посетитель

Chain of Responsibility

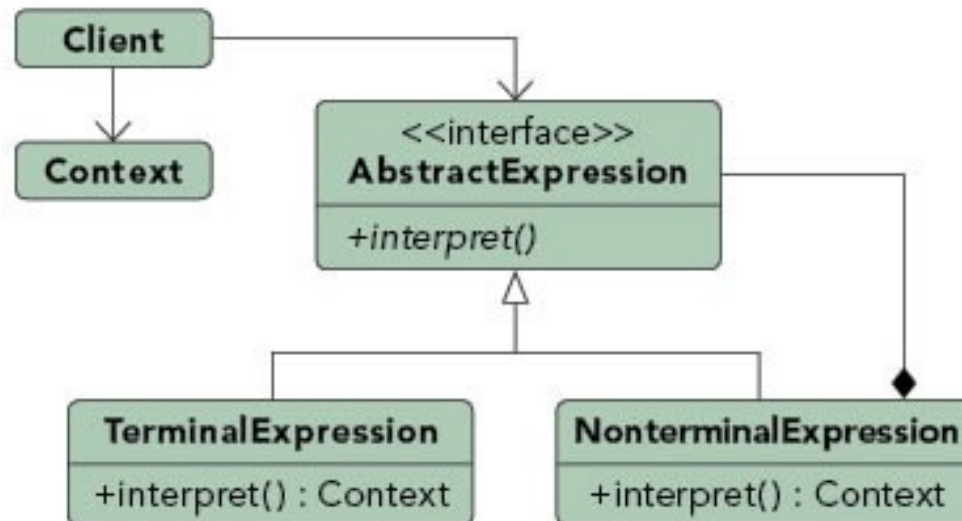
- Передача запроса по цепочке обработчиков
- Звонок в тех. поддержку
 - 1) взять кредит — нажмите 1, застряла карта — нажмите 2
 - 2) нажмите кнопку "сброс" на банкомате 2 раза
 - 3) стойте рядом, сейчас приедем



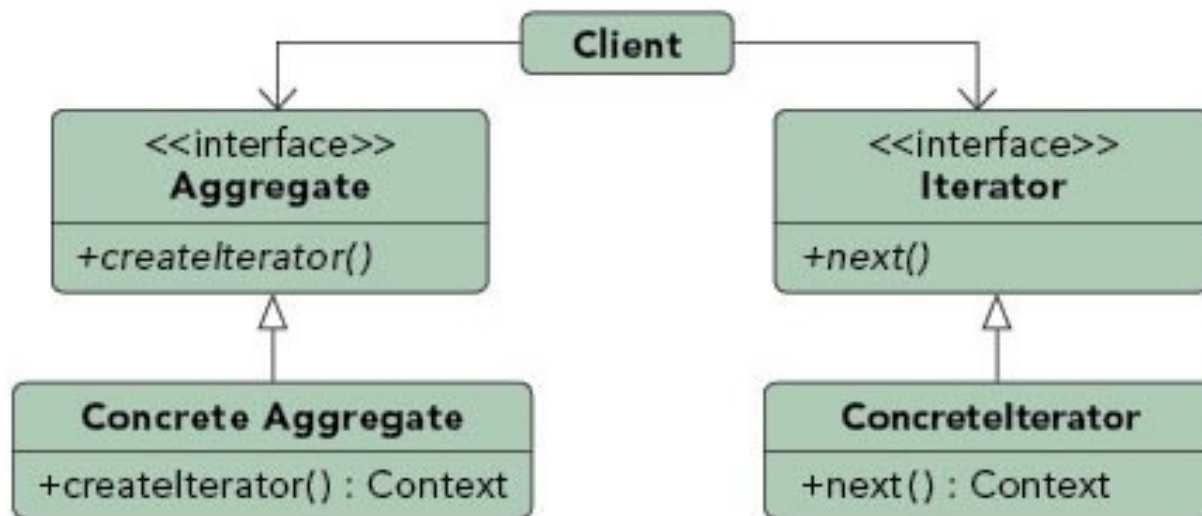
- Управление командами
- Разделение вызова и исполнения команд
- Можно организовать очередь команд и макрокоманды



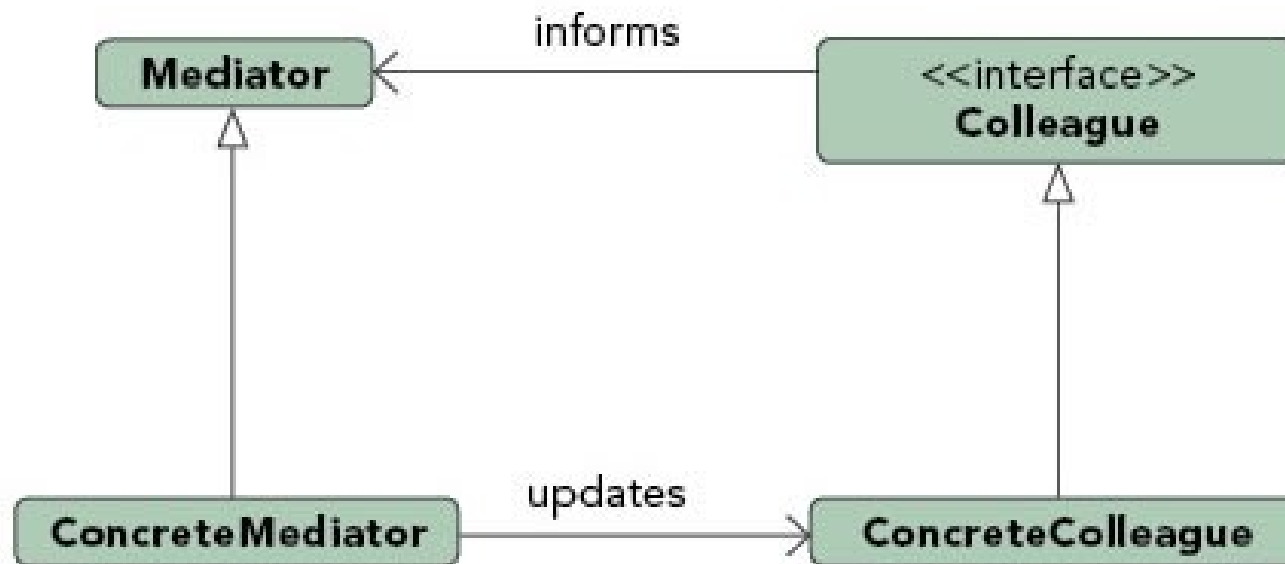
- Позволяет управлять поведением с помощью простого языка
 - Регулярные выражения
 - Форматирование строк



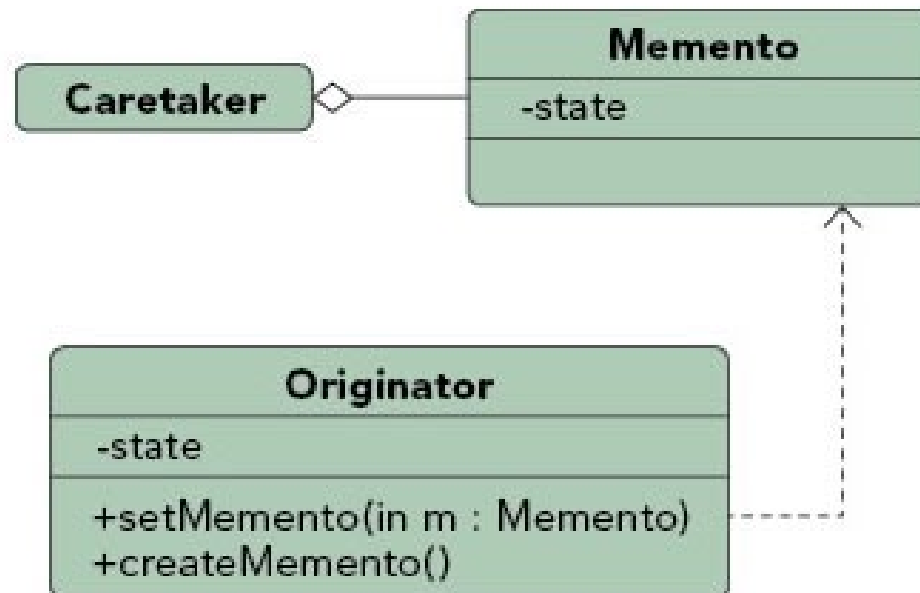
- Последовательный доступ к элементам коллекции
- Экскурсия
 - Реальный гид
 - Аудиогид
 - Путеводитель



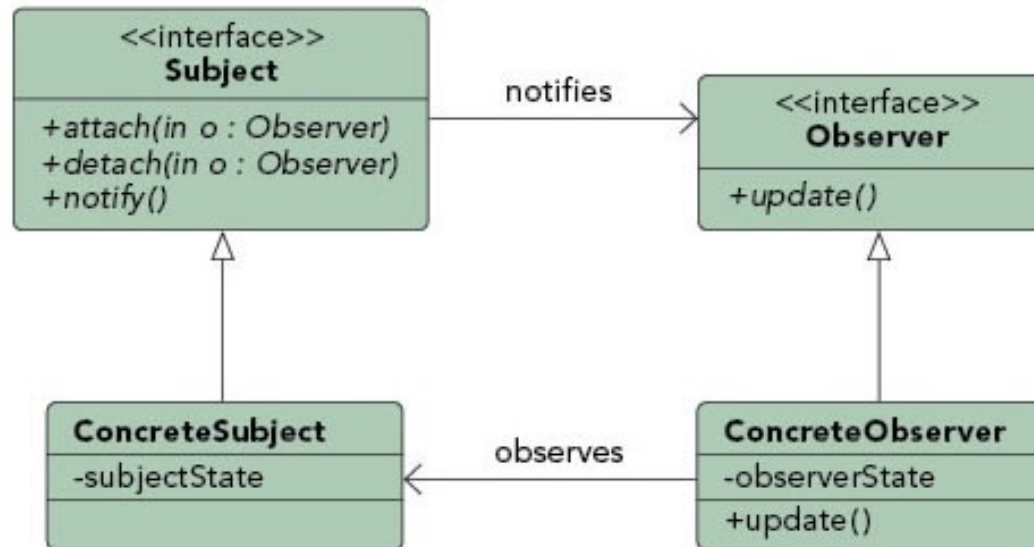
- Класс-посредник для управления изменением состояния других объектов



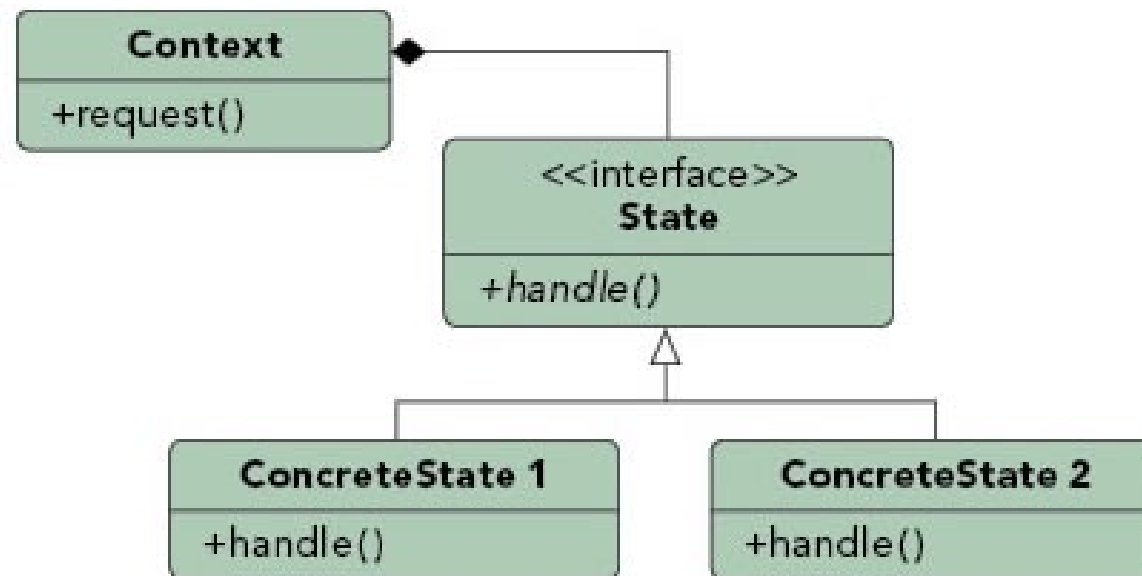
- Хранение и восстановление состояния объекта
- UNDO



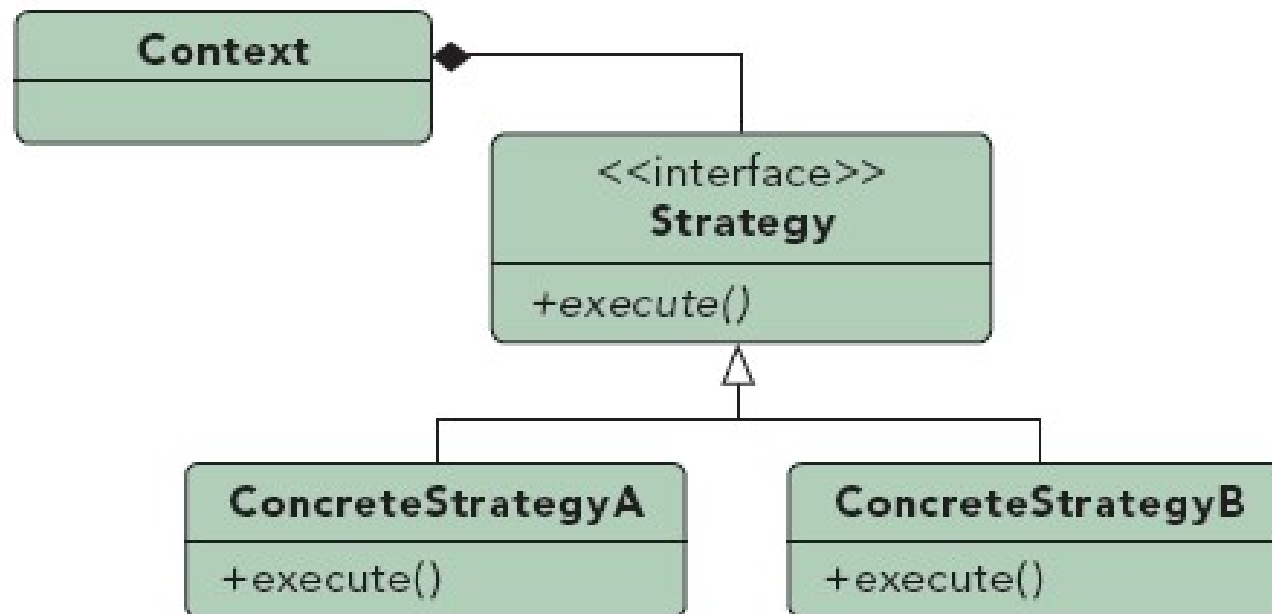
- Оповещение объектов об изменении состояния
- Обработка событий:
 - Источник — наблюдаемый
 - Обработчик - наблюдатель



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

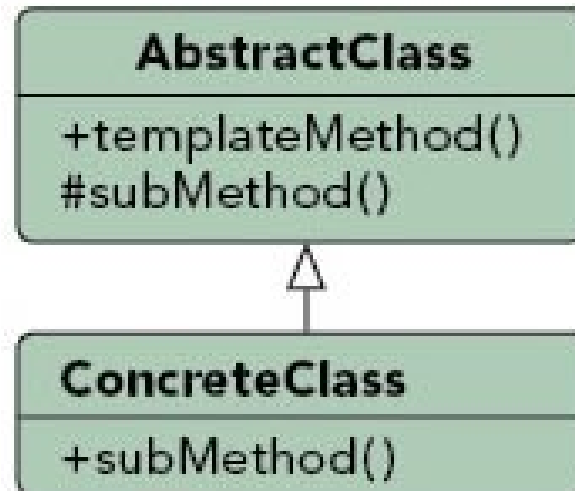


- Выбор одного из алгоритмов, реализованных в классе
 - Как добраться до аэропорта
 - ◊ Автобус
 - ◊ Такси
 - ◊ Пешком



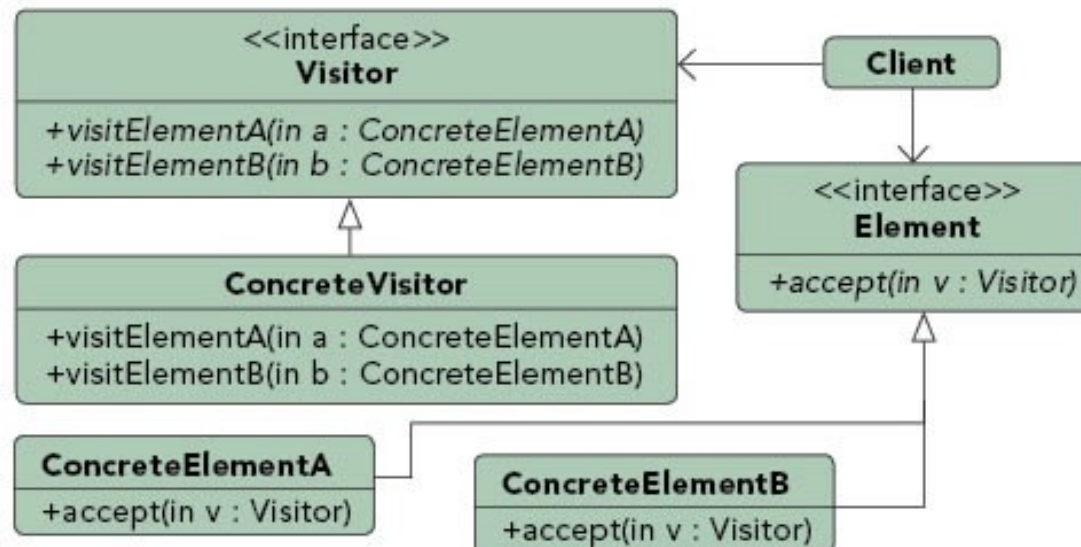
Template Method

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



Visitor

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



I_{10n} и i_{18n}

- Локализация — адаптация программных продуктов для определенного языка и/или определенной местности
 - Перевод текста
 - Использование соответствующих форматов данных
 - Замена звуковой и визуальной информации
 - localization = l10n

- 0005610 001 \0 \0 \0 h e l l
- 0005620 o , w o r l d \0
- 0005610 001 \0 \0 \0 п р и в
- 0005620 е т , м и р \0

- Интернационализация — проектирование программ таким образом, чтобы их локализация была возможна без конструктивных изменений
 - выделение текстовых данных из кода
 - отображение данных с учетом местных форматов
 - internationalization = i18n

- application.exe
- messages.txt
[hello]
en=hello

- application.exe
- messages.txt
 - [hello]
 - en=hello
 - ru=привет
 - it=ciao
 - es=hola
 - fr=salut
 - de=hallo
 - zh= 你好

- Локаль — совокупность характеристик, определяющих географический, политический или культурный регион
- Класс `java.util.Locale`
- Элементы локали:
 - **язык** — 2 строчные буквы (иногда 3) (ru)
 - **страна** (регион) — 2 заглавные буквы (RU) или 3 цифры
 - **вариант** (например, кодировка для русской локали)
 - письменность — 4 буквы, первая заглавная (Cyril)
 - расширение

- Конструкторы класса `Locale`
 - `new Locale(String lang)`
 - `new Locale(String lang, String country)`
 - `new Locale(String lang, String country, String variant)`
- Класс `Locale.Builder`
 - `new Locale.Builder().setLanguage("ru").setRegion("RU").build()`
- Метод `forLanguageTag()`
 - `Locale.forLanguageTag("ru-RU");`
- Константы класса `Locale`
 - `Locale.FRENCH`
 - для русского константы нет

- Получение списка локалей и локали по умолчанию
 - `static Locale[] getAvailableLocales()`
 - `static Locale getDefault()`
- Преобразование в строку
 - `String toString() // ru_RU`
 - `String getDisplayName() // Russian (Russia)`

- Класс ResourceBundle
- Набор ресурсов вида "ключ-значение"
- 2 варианта:
 - Текстовый файл свойств *.property (PropertyResourceBundle)
 - Класс со списком (ListResourceBundle)

- | | |
|---------------------------|-------------------------|
| • GuiLabels_en.properties | GuiLabels_ru.properties |
| s1 = Yes | s1 = Да |
| s2 = No | s2 = Нет |

```
ResourceBundle r = ResourceBundle.getBundle("GuiLabels");  
JButton b1 = new JButton(r.getString("s1"));  
JButton b2 = new JButton(r.getString("s2"));
```

- + отдельные текстовые файлы
- - только String
- Кодировка ISO-8859-1 — необходима обработка с помощью native2ascii

```
public class GuiLabels_en extends ListResourceBundle {  
    public Object[][] getContents() { return contents; }  
    private Object[][] contents = { {"s1", "Yes"}, {"s2", "No"} };  
}  
public class GuiLabels_ru extends ListResourceBundle {  
    public Object[][] getContents() { return contents; }  
    private Object[][] contents = { {"s1", "Да"}, {"s2", "Нет"} };  
}  
ResourceBundle r = ResourceBundle.getBundle("GuiLabels");  
JButton b1 = new JButton(r.getString("s1"));  
JButton b2 = new JButton(r.getString("s2"));
```

- + любые типы объектов
- - нужна компиляция файлов

- Построение списка кандидатов

```
Locale loc = Locale.US; // Locale.getDefault() = ru_RU;
```

```
ResourceBundle rb = ResourceBundle.getBundle("GuiLabels", loc);
```

1) GuiLabels_en_US

2) GuiLabels_en

3) GuiLabels_ru_RU

4) GuiLabels_ru

5) GuiLabels

- Формирование набора (bundle) - .class .property
- MissingResourceException

- getString(key)
- getStringArray(key)
- getObject(key)

- Просматривается список кандидатов и возвращается первый найденный

- Класс `NumberFormat` — абстрактный
 - `NumberFormat nf = NumberFormat.getNumberInstance();`
 - `NumberFormat cf = NumberFormat.getCurrencyInstance();`
 - `NumberFormat pf = NumberFormat.getPercentInstance();`
 - `nf.format(new Float(999.8));`
- Класс `DecimalFormat`
 - `df = (DecimalFormat) nf;`
 - `df.applyPattern("##,##0.00");`
 - `df.format(new Float(888.7));`
- Класс `DecimalFormatSymbols`
 - `DecimalFormatSymbols ds = new DecimalFormatSymbols();`
 - `ds.setDecimalSeparator('=');`
 - `df.setDecimalFormatSymbols(ds);`
 - `df.format(new Float(777.6));`

- 0 — цифра, 0 отображается
- # — цифра, 0 не отображается
- . — разделитель десятичной дроби
- , — разделитель групп разрядов
- E — разделитель мантиссы и порядка
- — знак минус
- ; — разделитель подшаблонов
- % — умножить на 100 и отобразить как процент
- ‰ — умножить на 1000 и отобразить как промилле
- ₽ — СИМВОЛ валюты

- Класс `DateFormat` — абстрактный
 - `DateFormat df = DateFormat.getInstance(DateFormat.FULL)`
 - `DateFormat tf = DateFormat.getInstance(DateFormat.LONG)`
 - `DateFormat dtf = DateFormat.getInstance(DateFormat.SHORT)`
 - `df.format(new Date());`
- Класс `SimpleDateFormat`
 - `sdf = (SimpleDateFormat) df;`
 - `sdf.applyPattern("yyyy-MM-dd");`
 - `sdf.format(new Date());`
- Класс `DateFormatSymbols`
 - `DateFormatSymbols ds = new DateFormatSymbols();`
 - `ds.setShortWeekdays("пнд","втр","срд","чтв","птн","сбт","вск");`
 - `sdf.setDateFormatSymbols(ds);`
 - `sdf.format(new Date());`

Более 4 символов — полный формат, 3 — сокращенный, 2 - число

G — эра

y — год

M — месяц после числа

L — месяц (название)

d — число

E — название дня недели

H — часы

m — минуты

s — секунды

S — миллисекунды

z — временная зона

- Класс MessageFormat

- 9 марта 2016 в 1:58 произойдет полное солнечное затмение.
- Total solar eclipse will happen at 1:58 on March 9, 2016.

Solar_en.properties

msg = {0} solar eclipse will be at {1,time,short} on {1,date,short}.

full = Total

part = Partial

Solar_ru.properties

{1,date,short} в {1,time,short} будет {0} солнечное затмение

full = полное

part = частное

```
ResourceBundle r = ResourceBundle.getBundle("Solar");  
MessageFormat mf = new MessageFormat(r.getString("msg");  
Calendar x = new Calendar(); x.set(2016,3,9,1,58,0);  
Object[] args = {r.getString("full", x.getTime())};  
mf.format(args);
```

- Класс ChoiceFormat extends NumberFormat
 - 0 friends like it
 - 1 friend likes it
 - 1000 friends like it

Like_en.properties

```
msg = {0} it  
one = {0,number} friend likes  
many = {0,number} friends like
```

```
ResourceBundle r = ResourceBundle.getBundle("Like");  
MessageFormat mf = new MessageFormat(r.getString("msg"));  
double[] lms = { 0, 1, 2 };  
String o = r.getString("one");  
String m = r.getString("many");  
String[] msgs = { m, o, m };  
ChoiceFormat cf = new ChoiceFormat(lms, msgs);  
mf.setFormatByArgumentIndex(0, cf);  
Object[] args = { new Integer(15) };  
mf.format(args);
```

- Класс Collator - абстрактный
- Collator getInstance()
- int compare()

```
List<String> lst = Arrays.asList({"Fluor", "Chlor","Brom","Jod"});
```

```
Collator c1 = Collator.getInstance(Locale.EN);
```

```
Collator c2 = Collator.getInstance(new Locale("cz","CZ");
```

```
lst.sort(c1);    // Brom, Chlor, Flour, Jod
```

```
lst.sort(c2);    // Brom, Fluor, Chlor, Jod
```

```
// A, Á, B, C, Č, D, Ď, E, É, Ě, F, G, H, Ch, I, Í, J, K, L, M, N,
```

```
// Ň, O, Ó, P, Q, R, Ř, S, Š, T, Ť, U, Ú, Ů, V, W, X, Y, Ý, Z, Ž
```

- RuleBasedCollator

- Класс BreakIterator — поиск границ
- Методы
 - `getCharacterInstance()`
 - `getWordInstance()`
 - `getSentenceInstance()`
 - `getLineInstance()`
 - `int first()`
 - `int last()`
 - `int next()`
 - `int previous()`
 - `int following(int offset)`
 - `int preceding(int offset)`
 - `BreakIterator.DONE`

Дата и время

- 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

- 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)

- 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

- 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)

- Статические

- of — создает экземпляр на основе входных параметров
- from — конвертирует экземпляр из другого типа
- parse — создает экземпляр из строкового представления

- Методы экземпляра

- format — форматирует объект в строку
- get — возвращает частичное состояние объекта
- is — проверяет параметр объекта
- with — возвращает копию объекта с одним измененным элементом
- plus — возвращает копию объекта с добавленным временем
- minus — возвращает копию объекта с убавленным временем
- to — преобразует объект в другой тип
- at — комбинирует объект с другим

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

Классы для представления даты и времени

	год	месяц	день	час	минута	секунда	нано
Year	x						
YearMonth	x	x					
MonthDay		x	x				
LocalDate	x	x	x				
LocalTime				x	x	x	x
LocalDateTime	x	x	x	x	x	x	x

.of(year,month,day,hour,min,sec,nano)

.now()

YearMonth Year.**at**Month(Month)

LocalDate MonthDay.**at**Year(Year)

LocalDate YearMonth.**at**Day(day)

LocalDate Year.**at**MonthDay(MonthDay)

LocalDateTime LocalDate.**at**Time(LocalTime)

LocalDateTime LocalTime.**at**Date(LocalDate)

.get

.getNano()

.getSecond()

.getMinute()

.getHour()

.getDayOfMonth()

.getDayOfWeek()

.getMonth()

.getYear()

.with

.plus

.minus

.plusNanos(nano)

.plusSeconds(sec)

.plusMinutes(min)

.plusHours(hour)

.plusDays(day)

.plusWeeks(week)

.plusMonths(month)

.plusYears(year)

- 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`

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

- `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()`

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

Соответствия:

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

• Методы:

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

Рефлексия

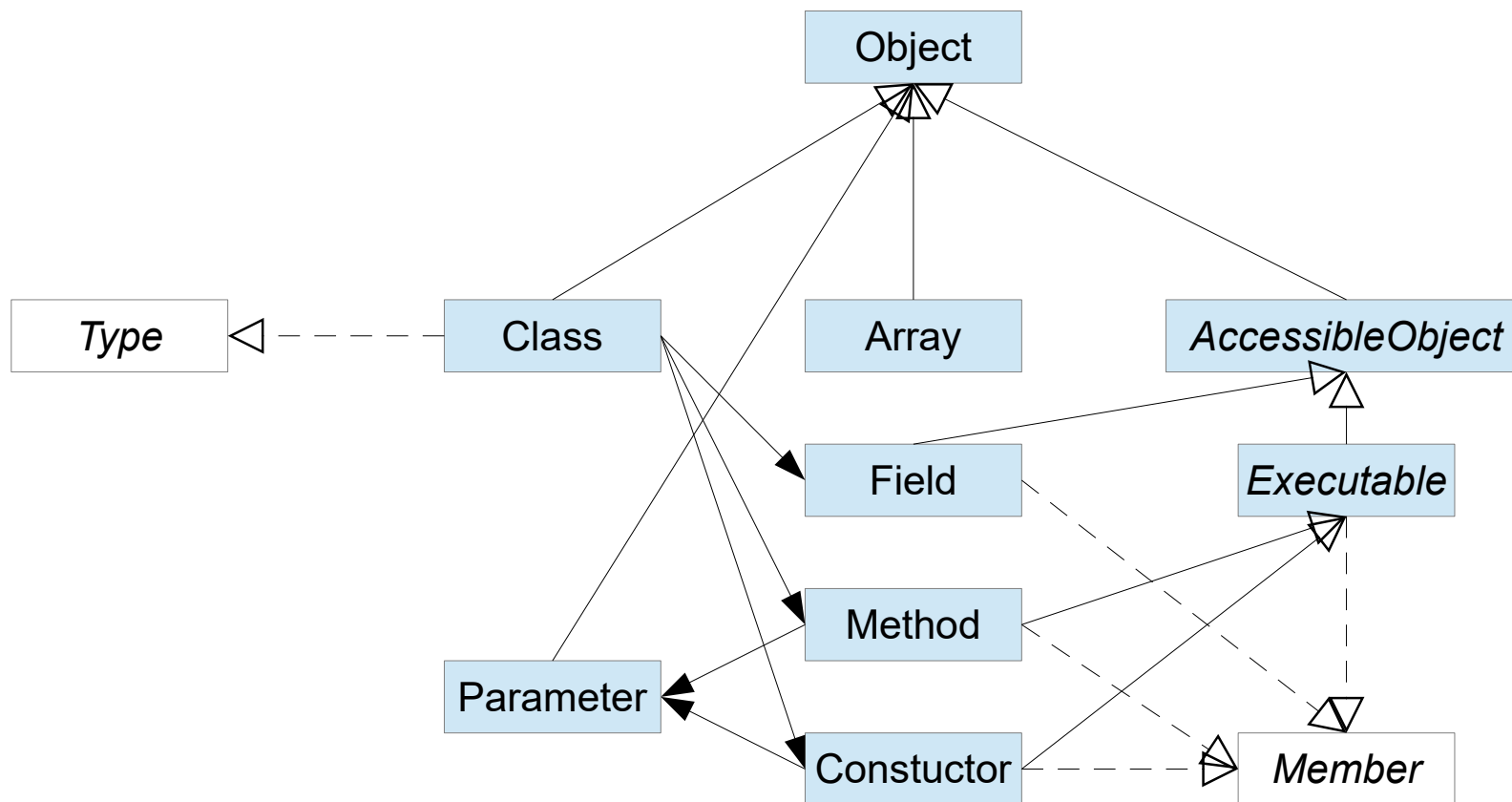
- Информация, которую содержит объект или класс — данные.
- Информация, которая описывает, как устроен класс — метаданные.
- Механизм работы с метаданными, позволяющий получить доступ к внутренней структуре классов и объектов — рефлексия.
- `java.lang.Class`
- `java.lang.reflect.*`

- Класс `Class<T>` - класс, представляющий классы
- Как получить объект класса `Class`?
 - `obj.getClass()` — по имеющемуся объекту
 - `T.class` — по любому имеющемуся типу данных
 - `Class.forName(String name)` — динамически по имени
 - `Integer.TYPE`

- `isInterface()`
- `isArray()`
- `isPrimitive()`
- `IsSynthetic()` - не объявленный явно или неявно
- `isEnum()`
- `isAnnotation()`
- `isMemberClass()`
- `isLocalClass()`
- `isAnonymousClass()`

- `Class getSuperclass()`
- `Class getEnclosingClass()`
- `Class getDeclaringClass()` - null для анонимных классов
- `Class[] getInterfaces()`
- `int getModifiers()`
- `TypeVariable[] getTypeParameters()`

- `boolean instanceof(Object o)`
- `T newInstance()`
- `T cast(Object o)`



- Object get(Object o)
- int getInt(Object o)
- ...
- void set(Object o, Object value)
- void setInt(Object o, int value)
- ...
- String getName()
- Class getType() / getGenericType()
- int getModifiers()
- setAccessible(true)

STRICT	ABSTRACT	INTERFACE	NATIVE	TRANSIENT	VOLATILE	SYNCHRONIZED	FINAL	STATIC	PROTECTED	PRIVATE	PUBLIC
0	0	0	0	0	0	0	0	0	0	0	0

- Modifier.PUBLIC ...
- boolean Modifier.isPublic() ...
- int Modifier.fieldModifiers() ...

```
if (field.getModifiers() &  
(Modifier.PUBLIC | Modifier.STATIC | Modifier.FINAL) != 0)
```

- newInstance(class, length)
- Object get(Object array, int index)
- int getInt(Object array, int index)
- ...
- void set(Object array, int index, Object value)
- void setInt(Object array, int index, int value)
- ...
- String getName()
- Class getComponentType() (метод класса Class)

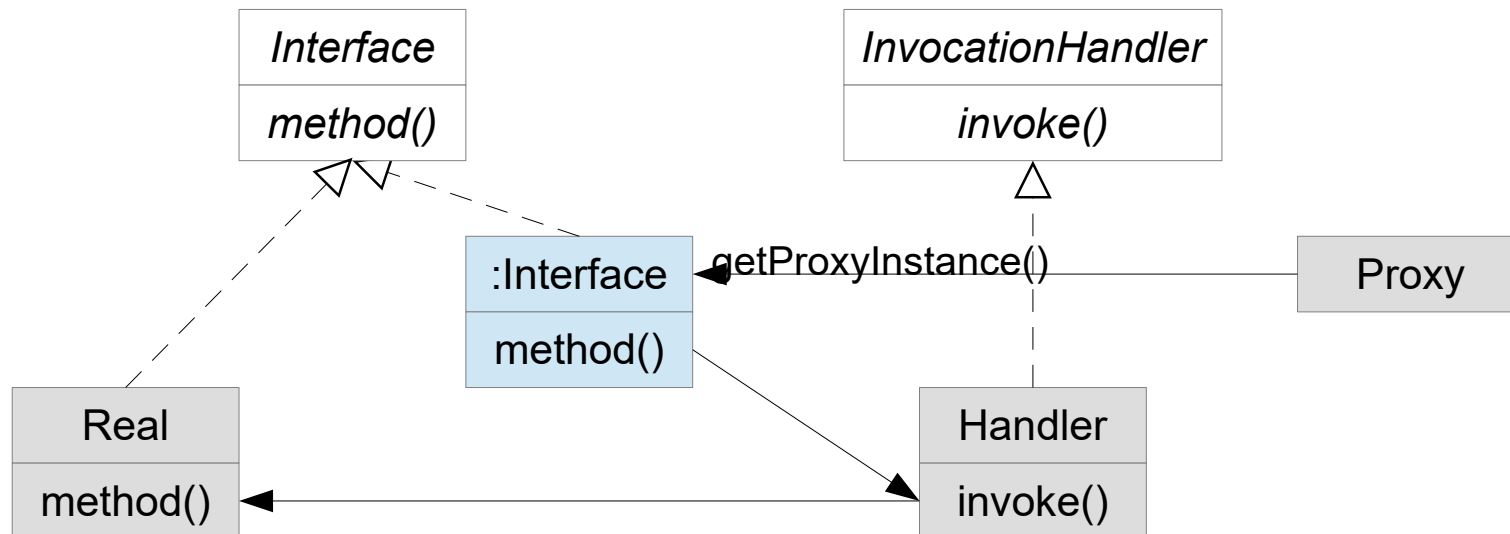
- boolean [Z
- byte [B
- char [C
- class or interface [*Lname*;
- double [D
- float [F
- int [I
- long [J
- short [S

- Object invoke(Object o, Object... args)
- String getName()
- Class getReturnType() / getGenericReturnType()
- Class[] getParameterTypes() / getGenericParameterTypes()
- Class[] getExceptionTypes() / getGenericExceptionTypes()
- int getModifiers()

- `T newInstance(Object... args)`
- `String getName()`
- `Class[] getParameterTypes() / getGenericParameterTypes()`
- `Class[] getExceptionTypes() / getGenericExceptionTypes()`
- `int getModifiers()`

- getField(name) / getFields()
- getMethod(name) / getMethods()
- getConstructor(name) / getConstructors()
 - Возвращают наследуемые элементы
 - Не возвращают приватные элементы
- getDeclaredField(name) / getDeclaredFields()
- getDeclaredMethod(name) / getDeclaredMethods()
- getDeclaredConstructor(name) / getDeclaredConstructors()
 - Возвращают приватные элементы
 - Не возвращают наследуемые элементы

- класс `java.lang.reflect.Proxy`
- интерфейс `java.lang.reflect.InvocationHandler`
- Создает динамический прокси-объект, реализующий заданные интерфейсы, при этом вызовы методов интерфейсов передаются обработчику, реализующему `InvocationHandler`



```
// интерфейс
interface Able { public void doIt(); }

// класс, реализующий интерфейс
public class Real implements Able { public doIt() { ... }

// создание прокси
Real real = new Real();
InvocationHandler handler = (proxy, method, args) -> {
// еще действия (логирование)
// можем преобразовать результат .toUpperCase()
return method.invoke(real, args);
};

Able proxy = (Able)Proxy.newProxyInstance(
    Able.class.getClassLoader(),
    new Class[] { Able.class },
    handler);

proxy.doIt();
```

- Аннотации — метаданные, добавляемые в исходный код, не влияющие семантически на программу, но используемые в процессе анализа кода, компиляции или во время исполнения
- Аннотация — это подвид интерфейса, все аннотации — потомки интерфейса `java.lang.annotation.Annotation`
- Обозначаются символом `@`

- `@Deprecated` — отмечает устаревший метод
- `@Override` — отмечает переопределенный метод
- `@SuppressWarnings` — запрещает компилятору выдавать определенные предупреждения
- `@FunctionalInterface` — показывает, что объявленный тип является функциональным интерфейсом
- `@SafeVarargs` — не выводить предупреждения об использовании нематериализуемого типа (`non-reifiable`) в методе или конструкторе с переменным числом аргументов (`varargs`)

`m(T[]...)`

- `@Retention(RetentionPolicy.SOURCE, CLASS, RUNTIME)` — время действия аннотации
- `@Target(ElementType.FIELD, METHOD, TYPE, CONSTRUCTOR, PACKAGE, LOCAL_VARIABLE, PARAMETER, ANNOTATION_TYPE)` — элемент, к которому можно применить аннотацию
- `@Inherited` — аннотация наследуется потомками
- `@Documented` — аннотированный элемент должен быть документирован с помощью javadoc
- `@Repeatable` — аннотация может повторяться для одного и того же элемента

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@interface DBTable {
    String name();
}

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
@interface PrimaryKey { }

@DBTable(name="person")
public class Human {
    @PrimaryKey int id;
    String name;
    LocalDate birthdate;
}
```

- интерфейс `java.lang.reflect.AnnotatedElement`
- реализуется классами `Class`, `Field`, `Method`, `Constructor`, `Parameter`
- `Annotation[] getAnnotations() / getDeclaredAnnotations()`
- `Annotation getAnnotation(Class a) / getDeclaredAnnotation`
- `boolean isAnnotationPresent(Class a)`

- Project Lombok
 - @Getter
 - @Setter
 - @ToString
 - @EqualsAndHashCode
- Checker Framework
 - @NotNull
 - @Initialized
 - @Regex
 - @Format