СБЕРБАНК ТЕХНОЛОГИИ

Good Code

ОБСУДИМ

СБЕРБАНК ТЕХНОЛОГИИ

- Что такое хороший код
- Code smells
- Паттерны проектирования
- Инкапсуляция
- Рефакторинг

ХОРОШИЙ КОД

СБЕРБАНК ТЕХНОЛОГИИ

- Легко читается не только автором
- Легко расширяется
- Легко переиспользуется
- Не содержит дублирования логики

- Покрыт тестами

ЛЕГКО ЧИТАЕТСЯ

СБЕРБАНК ТЕХНОЛОГИИ

- Используются общепринятые naming convention
- Используются общепринятые правила форматирования кода
- Имена классов, переменных, методов хорошо объясняют свое назначение
- Каждый класс делает ровно одно дело

- Написан по стандартным паттернам и принципам

НАДО СОБЛЮДАТЬ JAVA NAMING CONVERSION

СБЕРБАНК ТЕХНОЛОГИИ

Это важно!

http://www.oracle.com/technetwork/java/codeconventions-135099.html

ФОРМАТИРОВАНИЕ КОДА

СБЕРБАНК ТЕХНОЛОГИИ

Не забывайте форматировать код!

cntl+alt+L (Idea Windows)

СБЕРБАНК ТЕХНОЛОГИИ

Рефакторинг — это контролируемый процесс улучшения кода, без написания новой функциональности.

Рефакторинг применяется, чтобы сделать плохой код хорошим

https://refactoring.guru/ru

Существуют признаки, как понять что ваш код плохой, и стандартные техники рефакторинга для таких случаев.

ОСМЫСЛЕННЫЕ НАЗВАНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

```
//Плохо
class MyThreadPool implements ThreadPool {
    public MyThreadPool(int z, int k) {
    }
}

//Хорошо
class FixedThreadPool implements ThreadPool {
    public FixedThreadPool(int maxThreadCount) {
    }
}
```

НЕ ИСПОЛЬЗОВАТЬ НАСЛЕДОВАНИЕ ТАМ ГДЕ ОНО НЕ НУЖНО

СБЕРБАНК ТЕХНОЛОГИИ

```
//Примеры плохого наследования в JDK

public class Properties extends
                        Hashtable<Object,Object>


public class Stack<E> extends Vector<E>
```

ПРИ НАСЛЕДОВАНИИ ДОЛЖНО ВЫПОЛНЯТЬСЯ ОТНОШЕНИЕ IS A

СБЕРБАНК ТЕХНОЛОГИИ

Плохие примеры:

```
public class Mp3Converter extends Thread

public class Calculator extends Layout

public class WidgetCache extends Map<String, Widget>
```

НЕПЛОХИЕ ПРИМЕРЫ НАСЛЕДОВАНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

```
public class ArrayList<E> extends AbstractList<E>

public class Mp3Converter extends AbstractMusicConverter{}
```

ПРЕДПОЧИТАЙТЕ КОМПОЗИЦИЮ НАСЛЕДОВАНИЮ

СБЕРБАНК ТЕХНОЛОГИИ

```
//Плохо
public class EncryptedSet extends HashSet {
    public boolean add(Object o) {
        return super.add(encrypt(o));
    }
}
```

ПРЕДПОЧИТАЙТЕ КОМПОЗИЦИЮ НАСЛЕДОВАНИЮ

СБЕРБАНК ТЕХНОЛОГИИ

```
public class EncryptedSet implements Set {
    private final Set container;

    public EncryptedSet(Set set) {
        this.container = set;
    }

    public boolean add(Object o) {
        return container.add(encrypt(o));
    }

    public boolean addAll(Collection c) {
        return conatainer.add(encrypt(c));
    }
```

ПРЕДПОЧИТАЙТЕ КОМПОЗИЦИЮ НАСЛЕДОВАНИЮ

СБЕРБАНК ТЕХНОЛОГИИ

Наследование очень редко когда оправдано, лучше использовать композицию

http://javarevisited.blogspot.com/2013/06/why-favor-composition-over-inheritance-java-oops-design.html

DEPENDENCY INJECTION

СБЕРБАНК ТЕХНОЛОГИИ

Класс не должен сам создавать свои зависимости, они должны передаваться из вне (через конструктор).
Это дает возможность конфигурировать объекты передавая разные реализации

```
public class DefaultCarImpl implements Car {
    private final Engine engine = new DefaultEngineImpl();

    public double getSpeed() {
        return engine.getEngineRotation() *…;
    }
}
```

DEPENDENCY INJECTION

СБЕРБАНК ТЕХНОЛОГИИ

```
public class DefaultCarImpl implements Car {
    private final Engine engine;

    public DefaultCarImpl(Engine engine) {
        this.engine = engine;
    }

    public double getSpeed() {
        return engine.getEngineRotation() *…;
    }
}
```

ИНКАПСУЛЯЦИЯ

СБЕРБАНК ТЕХНОЛОГИИ

Работа всегда должна идти через интерфейсы.

Класс не должен делать предположения о внутренней реализации другого сервиса(должен работать через контракт интерфейса).

Это дает возможность свободно подменять реализации.

IMMUTABLE OBJECTS

СБЕРБАНК ТЕХНОЛОГИИ

Всегда делайте все поля final. Это правило можно нарушить, только если у вас есть веская причина для этого.

IMMUTABLE OBJECTS

СБЕРБАНК ТЕХНОЛОГИИ

С неизменяемыми объектами можно безопасно работать с многопоточной среде.

Их можно кешировать или передавать в другие методы, не боясь что их состояние случайно поменяется.

Неизменяемый объект всегда имеет одно полностью инициализированное состояние.

ПОКРЫТИЕ ТЕСТАМИ

СБЕРБАНК ТЕХНОЛОГИИ

Код должен быть покрыт юнит тестами.

Это позволит развивать и рефакторить код с большей степенью уверенности, что ничего не сломалось.

SINGLE RESPONSIBILITY PRINCIPLE

СБЕРБАНК ТЕХНОЛОГИИ

Хороший класс делает ровно одно дело.

Поэтому он должен уменьшаться ~ в 1 экран текста. Лучше меньше.
Это примерно 10-100 строк кода.

БОЛЬШОЙ КЛАСС

СБЕРБАНК ТЕХНОЛОГИИ

Если класс слишком большой, скорее всего он делает слишком много всего.

Надо декомпозировать на классы.

МНОГО ПОЛЕЙ

СБЕРБАНК ТЕХНОЛОГИИ

Если у класса много полей, скорее всего он делает слишком много всего. Надо декомпозировать.

~количество полей до 4.

ХОРОШИЙ МЕТОД

СБЕРБАНК ТЕХНОЛОГИИ

Хороший метод делает ровно одно дело.

Поэтому он занимает мало места. Это примерно 2-10 строк кода.

АРГУМЕНТЫ МЕТОДА

СБЕРБАНК ТЕХНОЛОГИИ

Если у метода много аргументов, скорее всего он делает слишком много всего. Надо декомпозировать на методы/классы.

~количество аргументов до 3.

СЛОЖНЫЙ МЕТОД

СБЕРБАНК ТЕХНОЛОГИИ

Если вам кажется, что нужен комментарий внутри(не Java-Doc), возможно есть проблема в коде.

БОЛЬШИЕ КЛАССЫ/МЕТОДЫ – ПРИЧИНА ПОЯВЛЕНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

В метод/класс всё время что-то добавляется, но ничего не выносится.

Так как писать код намного проще, чем читать, эта проблема долго остаётся незамеченной

ПЕРЕИСПОЛЬЗОВАНИЕ

СБЕРБАНК ТЕХНОЛОГИИ

Небольшие классы/методы используется как строительные кирпичики их легче переиспользовать.

ПЛОХОЙ МЕТОД

СБЕРБАНК ТЕХНОЛОГИИ

```
public void generateAndSendHtmlSalaryReport(
 Connection dbConnection,
 String departmentId, LocalDate date,
 String recipients, JavaMailSender sender) {
}
```

EXTRACT METHOD

СБЕРБАНК ТЕХНОЛОГИИ

```
void printOwing() {
    printBanner();

    //print details
    System.out.println("name: " + name);
    System.out.println("amount: " + getOutstanding());
}
```

EXTRACT METHOD

СБЕРБАНК ТЕХНОЛОГИИ

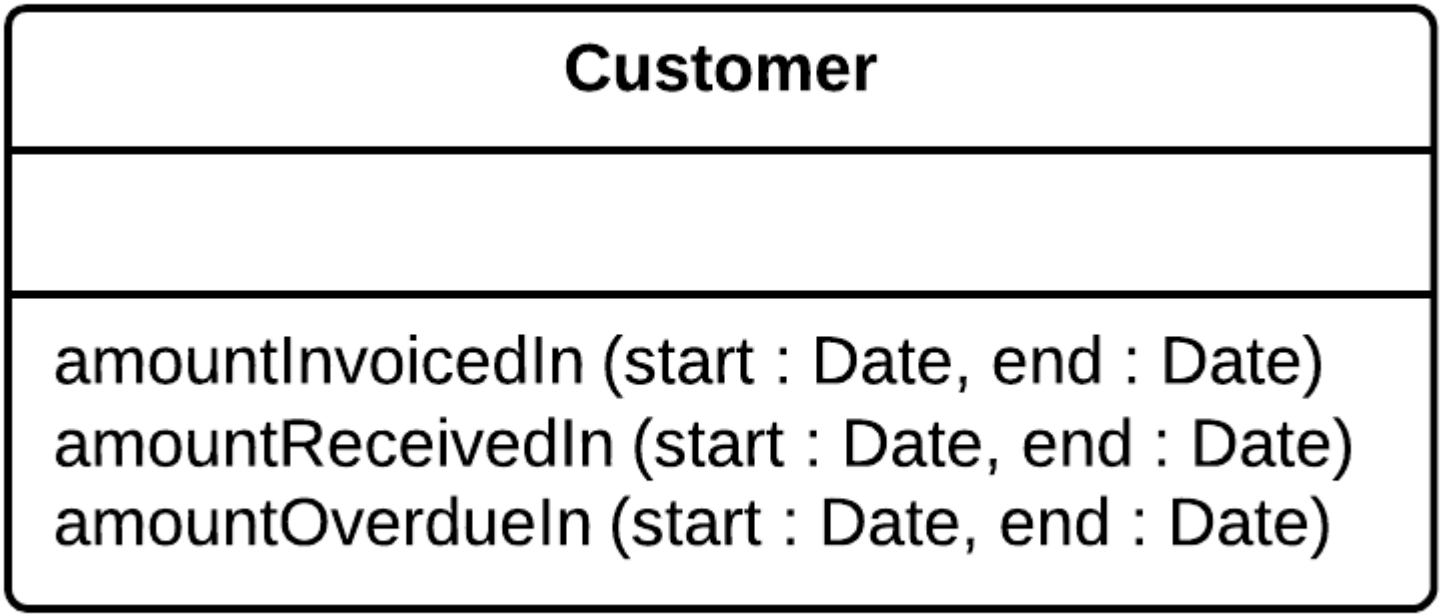
```
void printOwing() {
    printBanner();
    printDetails(getOutstanding());
}

void printDetails(double outstanding) {
    System.out.println("name: " + name);
    System.out.println("amount: " + outstanding);
}
```

INTRODUCE PARAMETER OBJECT

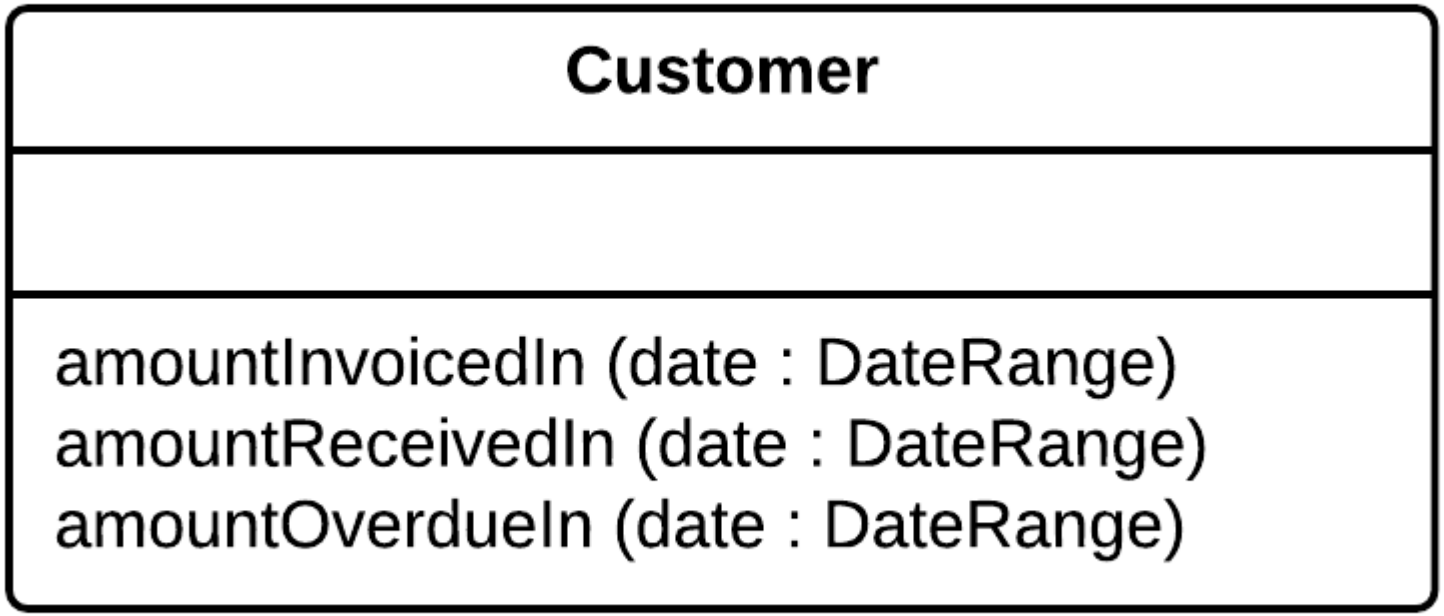
СБЕРБАНК ТЕХНОЛОГИИ

Некоторые аргументы всегда передаются вместе, можно для них сделать отдельный объект



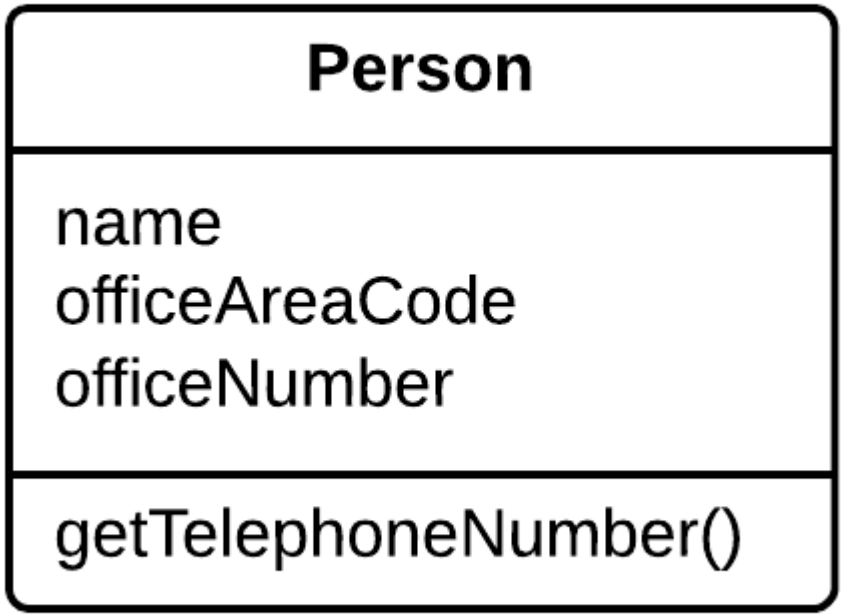
INTRODUCE PARAMETER OBJECT

СБЕРБАНК ТЕХНОЛОГИИ



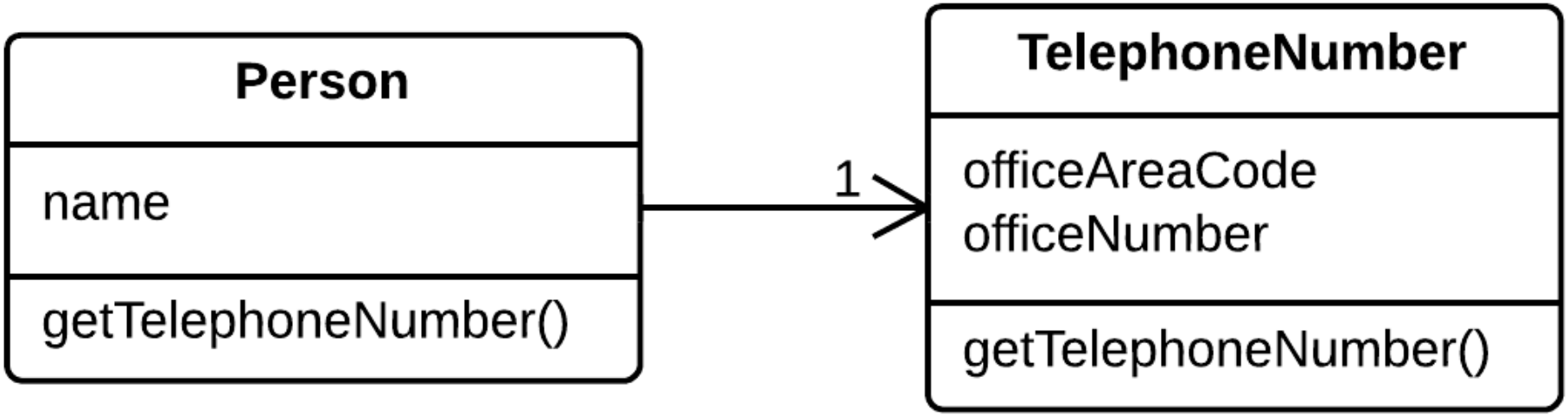
EXTRACT CLASS

СБЕРБАНК ТЕХНОЛОГИИ



EXTRACT CLASS

СБЕРБАНК ТЕХНОЛОГИИ



ДУБЛИРОВАНИЕ ЛОГИКИ

СБЕРБАНК ТЕХНОЛОГИИ

Не должно быть.

Extract method, Extract class, Introduce Parameter object

REPLACE CONDITIONAL WITH POLYMORPHISM

СБЕРБАНК ТЕХНОЛОГИИ

```
class Bird {
    //...
    double getSpeed() {
        switch (type) {
            case EUROPEAN:
                return getBaseSpeed();
            case AFRICAN:
                return getBaseSpeed() - getLoadFactor() *
                                        numberOfCoconuts;
            case NORWEGIAN_BLUE:
                return isNailed ? 0 : getBaseSpeed(voltage);
        }
        throw new RuntimeException("Should be unreachable");
    }
}
```

```
abstract class Bird {
    //...
    abstract double getSpeed();
}

class European extends Bird {
    double getSpeed() {
        return getBaseSpeed();
    }
}
class African extends Bird {
    double getSpeed() {
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
    }
}
class NorwegianBlue extends Bird {
    double getSpeed() {
        return isNailed ? 0 : getBaseSpeed(voltage);
    }
}
```

ИЛИ ТАК

СБЕРБАНК ТЕХНОЛОГИИ

```
class Bird {
    private final BirdType;

    double getSpeed() {
        return type.getSpeed();
    }
}
```

REPLACE CONDITIONAL WITH POLYMORPHISM

СБЕРБАНК ТЕХНОЛОГИИ

```
class EuropeanType implements BirdType {
    double getSpeed() {
        return getBaseSpeed();
    }
}
class African implements BirdType {
    double getSpeed() {
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
    }
}
class NorwegianBlue implements BirdType {
    double getSpeed() {
        return isNailed ? 0 : getBaseSpeed(voltage);
    }
}
```

ВЫНЕСЕНИЕ УТИЛЬНОЙ ЛОГИКИ ИЗ КЛАССОВ

СБЕРБАНК ТЕХНОЛОГИИ

```
private void sendReport() {
    Date now = new Date();
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(now);
    calendar.roll(Calendar.DAY_OF_WEEK, 1);
    Date prevDate = calendar.getTime();

    … someLogic
}
```

ВЫНЕСЕНИЕ УТИЛЬНОЙ ЛОГИКИ ИЗ КЛАССОВ

СБЕРБАНК ТЕХНОЛОГИИ

```
private void sendReport() {
    Date prevDate = DateUtils.prevDate();
    //
}

private static Date prevDate() {
    Date now = new Date();
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(now);
    calendar.roll(Calendar.DAY_OF_WEEK, 1);
    return calendar.getTime();
}
```

REPLACE MAGIC NUMBER WITH SYMBOLIC CONSTANT

СБЕРБАНК ТЕХНОЛОГИИ

```
double potentialEnergy(double mass, double height) {
    return mass * height * 9.81;
}
```

REPLACE MAGIC NUMBER WITH SYMBOLIC CONSTANT

СБЕРБАНК ТЕХНОЛОГИИ

```
private static final double GRAVITATIONAL_CONSTANT = 9.81;

double potentialEnergy(double mass, double height) {
  return mass * height * GRAVITATIONAL_CONSTANT;
}
```

SIMPLIFYING CONDITIONAL EXPRESSIONS

СБЕРБАНК ТЕХНОЛОГИИ

```
public double getPayAmount() {
    double result;
    if (isDead){
        result = deadAmount();
    }
    else {
        if (isSeparated){
            result = separatedAmount();
        }
        else {
            if (isRetired){
                result = retiredAmount();
            }
            else{
                result = normalPayAmount();
            }
        }
    }
    return result;
}
```

SIMPLIFYING CONDITIONAL EXPRESSIONS

СБЕРБАНК ТЕХНОЛОГИИ

```
public double getPayAmount() {
    if (isDead) {
        return deadAmount();
    }
    if (isSeparated) {
        return separatedAmount();
    }
    if (isRetired) {
        return retiredAmount();
    }
    return normalPayAmount();
}
```

CONSOLIDATE DUPLICATE CONDITIONAL FRAGMENTS

СБЕРБАНК ТЕХНОЛОГИИ

```
if (isSpecialDeal()) {
    total = price * 0.95;
    send();
}
else {
    total = price * 0.98;
    send();
}
```

```
if (isSpecialDeal()) {
    total = price * 0.95;
}
else {
    total = price * 0.98;
}
send();
```

ДРУГИЕ ПРИМЕРЫ ПЛОХОГО КОДА

СБЕРБАНК ТЕХНОЛОГИИ

https://refactoring.guru/ru/smells/smells

ЗАРЕФАКТОРИТЬ

СБЕРБАНК ТЕХНОЛОГИИ

https://bitbucket.org/agoshkoviv/solid-homework/src/099989b0c76217689c4642242c87c1ac080dfc01/src/main/java/ru/sbt/bit/ood/solid/homework/SalaryHtmlReportNotifier.java?at=master&fileviewer=file-view-default

https://bitbucket.org/agoshkoviv/patterns-homework-1/src/69a61334ea43ff4c3fd950a00095377cf1e3bfd4/src/main/java/ru/sbt/test/refactoring/?at=master