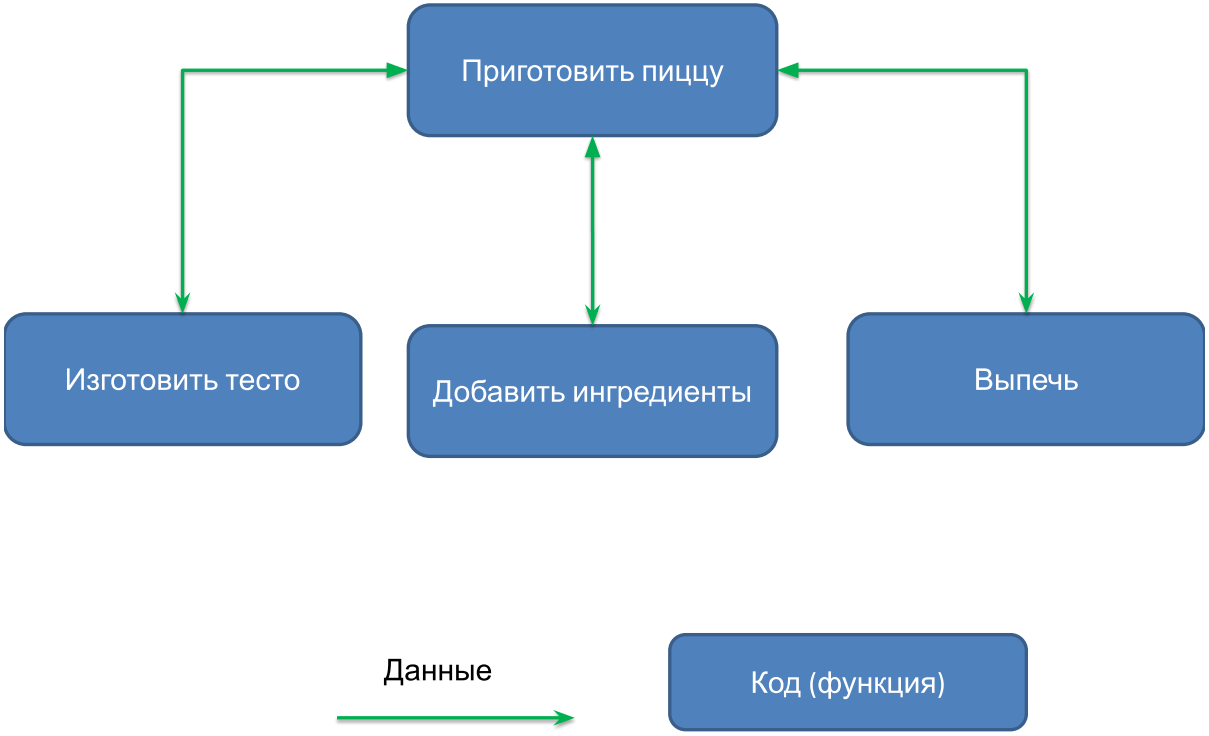
OOP. Classes and Objects

Intro

Problem

Структурное программирование

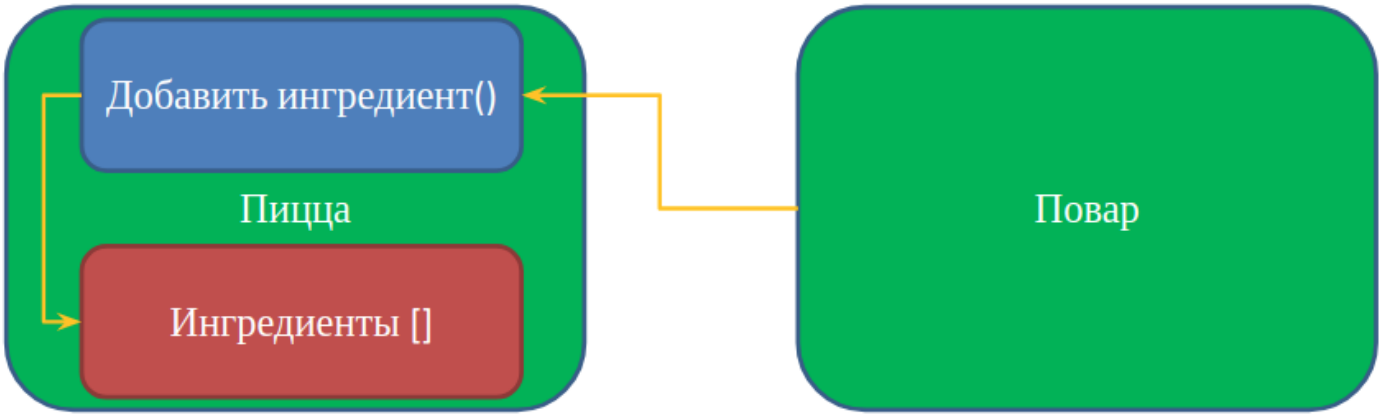
Принцип: код воздействует на данные.

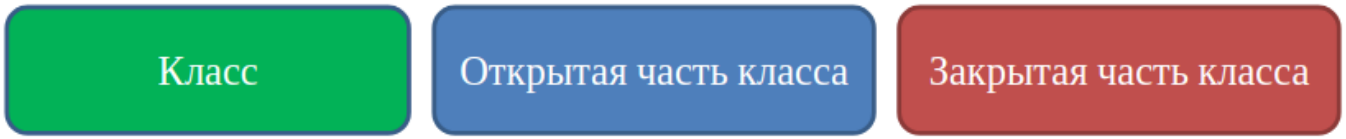


Solution

Объектно-ориентированное программирование

Принцип: данные управляют доступом к коду.





Classes and objects

OOP concepts

- **OOP** (**ООП**) — это **парадигма программирования** (**подход к программированию)**
- Главные понятия **OOP**:
  - **Classes** (**классы**)
  - **Objects** (**объекты**)
- **OOP** возникло в результате развития идей **процедурного программирования** (**ПП**)
- так как в **ПП данные** и **функции** их обработки формально не связаны

Classes and objects

- **Class** (**Класс**) — это **template** (**шаблон**) для создания **objects**.
- **Object** (**Объект**) — это **instance for class** (**экземпляр класса**).
- *Функция*, созданная внутри класса, называется **method** (**метод**).
- *Переменная*, созданная внутри класса, называется **field** (**поле**).

Class

- **Class** — это **template** для создания **objects**.
- **Class** определяет **state** (**состояние**) и **behavior** (**поведение**) **objects**.

```
class MyClass { // class
    // field, constructor, and
    // method declarations
}
```

Class

- **Class** состоит из **class members** (**членов класса**)
- Основные **class members**:
  - **fields**
  - **methods**
  - **constructors** (**конструкторы**)
  - **initialization blocks** (**блоки инициализации**)

Scope

- **Scope** (**Область видимости**) — часть текста программы, на протяжении которого к объекту/переменной/функции можно обращаться по его имени.
- В языках программирования выделяют:
  - **global** (**глобальную**) **scope**, если объявление происходит вне любой функции или блока кода и доступно в любой точке программы.
  - **local** (**локальную**) **scope**, если объявление происходит в теле функции или в пространстве имен. **Scope**: от объявления и до окончания блока кода.

Scope

- Основные **scopes** в Java:
  - **class** (**global**)
  - **method** (**local**)
  - **code block** (**local**)

Declaring Member Variables

- **Member variables** in a class—these are called **fields**.
- **Variables** in a *method* or *block of code* — these are called **local variables**.
- **Variables** in *method declarations* — these are called **parameters**.

Fields

- **Fields – class member**, которые будут содержать **state of object**, т.е. данные
- объявляются в **class** следующим образом:

```
modifier DataType variableName;
```

```
public int age; // public field
private String firstName; // private field
```

Methods

- **Methods** (**Методы**) – это обособленные **code blocks**, который отвечают за **behavior of object** данного **class**

```
modifier ReturnedDataType nameMethod(parameters) {
    // method body
}
```

```
public class Human { // class
    private double temperature; // field

    public boolean isIll() { // method
        // method body
        return temperature > 37.0 || temperature < 36.2;
    }
}
```

The signature of the method

- **Method signatures** (**сигнатура метода**) — это его название и тип данных **parameters**.

```
calculateAnswer(double, int, double, double)
```

- Если **signature of method** у нескольких **methods** совпадает, то произойдет **compile error**

Example: overloading Methods

```
public class DataArtist {
    public void draw(String s) {
        // method body
    }

    public void draw(int i) {
        // method body
    }

    public void draw(int i, double f) {
        // method body
    }
}
```

Overloading Methods

- В предыдущем пример все методы разные:
  - хотя они имеют одинаковое имя
  - но они имеют разные **parameters**
- такое написание **methods** называется **overloading methods** (**перегрузкой методов**)

Example: class

```
public class User {
    public String name;
    public int age;

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n", name,
    }
}
```

Example: how use it?

```
public class Example1SimpleEmptyClass {
    public static void main(String[] args) {
        User dmitry = new User();
        dmitry.tellAboutYourself();
    }
}
```

```
My name is null. I'm 0 years old.
```

Example: but it is empty?!

```
public class Example2SimpleClass {
    public static void main(String[] args) {
        User dmitry = new User();
        dmitry.name = "Dmitry";
        dmitry.age = 21;
        dmitry.tellAboutYourself();
    }
}
```

```
My name is Dmitry. I am 21 years old.
```

Objects

- **Object** – любой предмет из **domain** (**предметной области**).
- **Object** имеет четкую структуру.
- **Object** имеет смысл ТОЛЬКО в контексте данного **domain**.
- **Class** один, а **objects** данного **class** много.
- **Objects** имеют одинаковые **fields**, но независимые значения для них.
- Значения **fields** для **object** задают его **state**.
- **Methods** для **object** задают его **behavior**.

Objects

- **Object** создаются с помощью **keyword** (**ключевого слова**) `new`.

```
User rakovets = new User();
```

где:

- `User` - тип данных, т.е. **class**.
- `rakovets` - название для **variable** (**переменной**).
- `=` - оператор присваивания.
- `new` - **keyword** для создания **object**.
- `User()` - **constructor** (**конструктор**).

Constructor

Example

```
public class Person {
    public String name;
    public int age;

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n", name,
    }
}
```

Example

```
public class Example3Constructor {
    public static void main(String[] args) {
        Person rakovets = new Person();
        rakovets.tellAboutYourself(); (1)
        rakovets.name = "Dmitry";
        rakovets.age = 21;
        rakovets.tellAboutYourself(); (2)
    }
}
```

```
My name is null. I am 0 years old. (1)
My name is Dmitry. I am 21 years old. (2)
```

The compiler does the chore

```
public class Person {
    public String name;
    public int age;

    public Person() { (1)
    }

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n", name,
    }
}
```

- (1) - **default constructor**

Constructor

- **Constructor** фактически представляет собой **method**.
- Но:
  - **Constructor** имеет имя как у **class**.
  - **Constructor** не имеет типа возвращаемого результата.
  - **Constructor** должен только создавать **object**.
  - **Constructor** не должен содержать лишней логики.
- Если **constructor** не указан – **compiler** создаст **default constructor**.
- Если **constructor** создан – **default constructor** не создаётся.

Constructors: example

```
public class Person {
    public String name;
    public int age;

    public Person() {
        name = "Undefined";
        age = 18;
    }

    public Person(String n) {
        name = n;
        age = 18;
    }

    public Person(String n, int a) {
        name = n;
        age = a;
    }
```

Constructors: example

```
public class Example3Constructor {
    public static void main(String[] args) {
        Person guest = new Person();
        guest.tellAboutYourself(); (1)
        Person tom = new Person("Tom");
        tom.tellAboutYourself(); (2)
        Person dmitry = new Person("Dmitry", 21);
        dmitry.tellAboutYourself(); (3)
    }
}
```

```
My name is Guest. I am 18 years old. (1)
My name is Tom. I am 18 years old. (2)
My name is Dmitry. I am 21 years old. (3)
```

Initialization Blocks

Initialization Blocks

- При описании класса могут быть использованы **initialization blocks** (**блоки инициализации**).
- **Initialization block** это код, заключенный в фигурные скобки.
- **Initialization block** не принадлежат ни одному из **methods** текущего **class**.

```
{ /* код */ }
```

Example

```
public class Person {
    public String name;
    public int age;

    {
        this.name = "Guest"; (1)
        this.age = 18;
    }

    public Person() { (2)
    }

    public Person(String name, int age) { (2)
        this.name = name;
        this.age = age;
    }

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n",
                name, age);
```

Example

```
public class Example3Constructor {
    public static void main(String[] args) {
        Person guest = new Person();
        guest.tellAboutYourself(); (1)
        Person dmitry = new Person("Dmitry", 21);
        dmitry.tellAboutYourself(); (2)
    }
}
```

```
My name is Guest. I am 18 years old. (1)
My name is Dmitry. I am 21 years old. (2)
```

Initialization Block

- **Initialization blocks** чаще всего используются для инициализации полей.
- **Initialization blocks** могут содержать вызовы методов как текущего класса, так и других.

Initialization Block

- При создании **object** какого-то **class**, **initialization blocks** вызываются:
  - последовательно
  - в порядке размещения
  - вместе с инициализацией **fields** как простая последовательность операторов
- только после выполнения всех **initialization blocks** будет вызван **constructor** для **class**.

Initialization Block

- Операции с **fields** для **class** внутри **блока инициализации** до явного объявления этого **field** возможны ТОЛЬКО при использовании ссылки `this`, представляющую собой ссылку на текущий **object**.
- **Блок инициализации** может быть объявлен со спецификатором `static`. В этом случае он вызывается только один раз в жизненном цикле приложения при создании **object** или при обращении к статическому **method/field** данного **class**.

How does it work?

```
public class Init {
    { (2)
        System.out.println("initializer order: 1, id=" + this.id);
    }

    public int id = 42; (3)

    public Init(int d) { (6)
        this.id = d;
        System.out.println("constructor: id=" + this.id);
    }

    { (4)
        System.out.println("initializer order: 2, id=" + this.id);
    }

    static { (1)
        System.out.println("static initializer");
    }
```

How does it work?

```
public class Example4InitializationBlock {
    public static void main(String[] args) {
        Init obj = new Init(7);
        System.out.println("Object state: id=" + obj.id);
    }
}
```

```
static initializer (1)
initializer order: 1, id=0 (2)
initializer order: 2, id=42 (3)
initializer order: 3, id=10 (4)
constructor: id=7 (5)
Object state: id=7 (6)
```

Keyword `this`

Example

```
public class Person {
    public String name;
    public int age;

    public Person(String name, int age) { (1)
        name = name; (2)
        age = age; (3)
    }

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n",
                name, age);
    }
}
```

Example

```
public class Example3Constructor {
    public static void main(String[] args) {
        Person dmitry = new Person("Dmitry", 21);
        dmitry.tellAboutYourself(); (1)
    }
}
```

```
My name is null. I am 0 years old. (1)
```

Example

```
public class Person {
    public String name;
    public int age;

    public Person(String name, int age) { (1)
        this.name = name; (2)
        this.age = age; (3)
    }

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n",
                name, age);
    }
}
```

Example

```
public class Example3Constructor {
    public static void main(String[] args) {
        Person dmitry = new Person("Dmitry", 21);
        dmitry.tellAboutYourself();
    }
}
```

```
My name is Dmitry. I am 21 years old.
```

Keyword `this`

- `this` (этот) — это ссылка на сам **object**
- С помощью его можно вызывать у текущего **object class**:
  - **fields**
  - **methods**
  - **constructors**

Example

```
public class Person {
    public String name;
    public int age;

    public Person() {
        this("Guest", 18); (1)
        this.tellAboutYourself(); (2)
    }

    public Person(String name, int age) {
        this.name = name; (3)
        this.age = age; (3)
    }

    public void tellAboutYourself() {
        System.out.printf("My name is %s. I am %d years old.\n",
                name, age);
    }
}
```

Getters and Setters

Example

```
public class Car {
    public String model;
    public int year;

    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }
}
```

Example

```
public class Example5GettersAndSetters {
    public static void main(String[] args) {
        Car bmw = new Car("X7", 2019); (1)
        bmw.model = "X5"; (2)
        System.out.printf("Car model: %s.\n", bmw.model); (3)
        System.out.printf("Car year: %d.\n", bmw.year); (4)
    }
}
```

```
Car model: X5 (3)
Car year: 2019 (4)
```

Getters

```
public class Car {
    private String model;
    private int year;

    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }
}
```

Getters

```
public class Example5GettersAndSetters {
    public static void main(String[] args) {
        Car bmw = new Car("X7", 2019); (1)
        System.out.printf("Car model: %s.\n", bmw.getModel()); (2)
        System.out.printf("Car year: %d.\n", bmw.getYear()); (3)
    }
}
```

```
Car model: X7 (2)
Car year: 2019 (3)
```

Setters

```
public class Car {
    private String model;
    private int year;

    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }
```

Setters

```
public class Example5GettersAndSetters {
    public static void main(String[] args) {
        Car bmw = new Car("X7", 2019); (1)
        bmw.setYear(2020); (2)
        System.out.printf("Car model: %s.\n", bmw.getModel()); (3)
        System.out.printf("Car year: %d.\n", bmw.getYear()); (4)
    }
}
```

```
Car model: X7 (3)
Car year: 2020 (4)
```

Objects as parameters of methods

Example

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }
}
```

Example: reference

```
public class Program {
    public static void main(String[] args) {
        Person kate = new Person("Kate");
        System.out.println(kate.getName());
        changeName(kate); (1)
        System.out.println(kate.getName()); (3)
    }

    static void changeName(Person p) { (2)
        p.setName("Alice");
    }
}
```

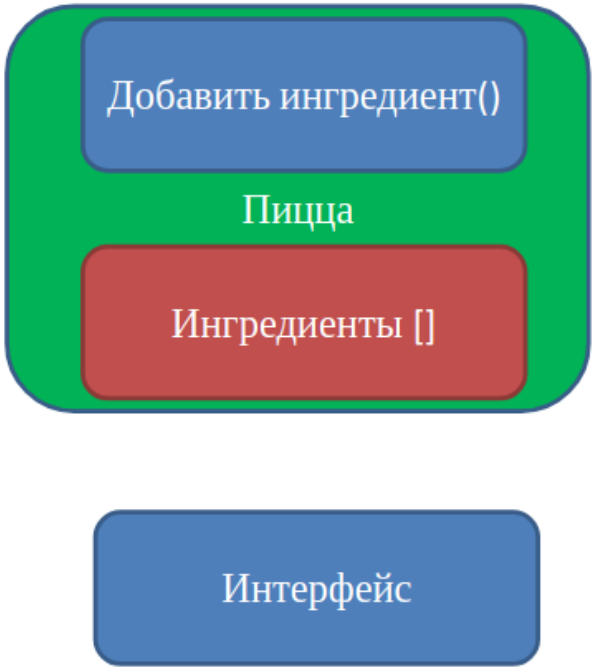
Example: rewrite reference

```
public class Program {
    public static void main(String[] args) {
        Person kate = new Person("Kate");
        System.out.println(kate.getName());
        changePerson(kate); (1)
        System.out.println(kate.getName()); (3)
    }

    static void changePerson(Person p) { (2)
        p = new Person("Alice");
        p.setName("Ann");
    }
}
```

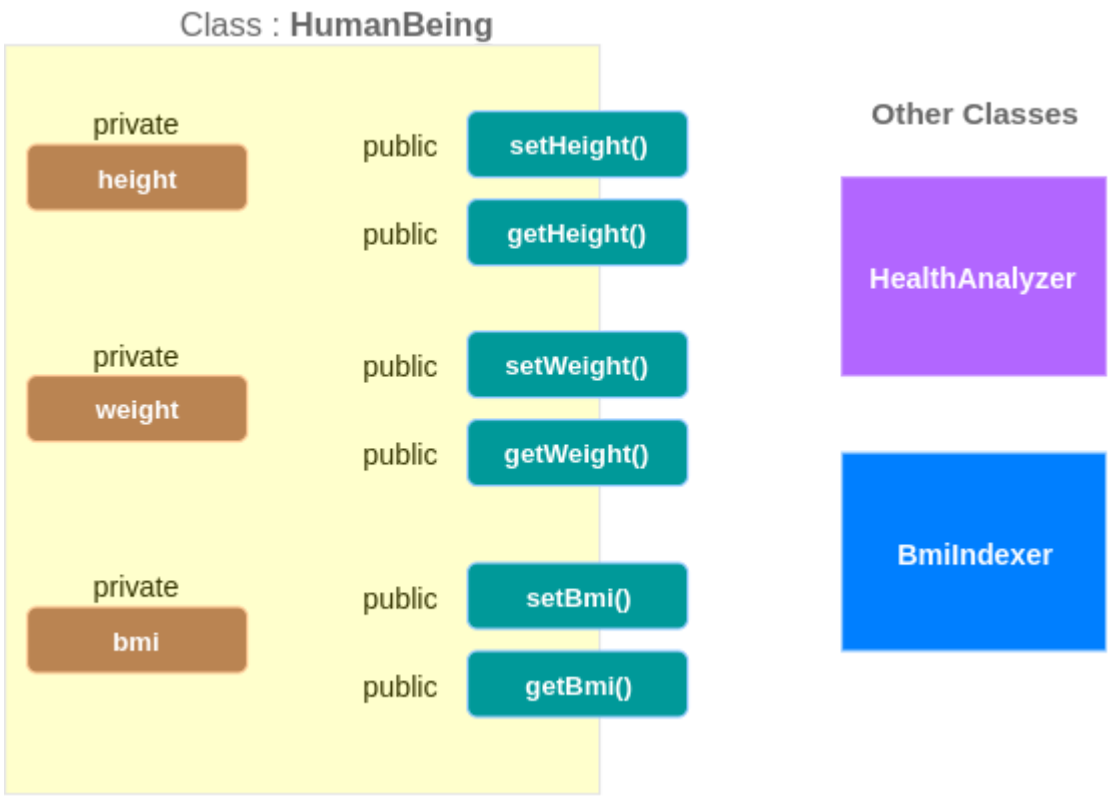
Interface

Interface (Интерфейс)

Открытая часть класса, с помощью которой другие классы могут с ним взаимодействовать



Interface


Class : HumanBeing
private
height
public
setHeight()
public
getHeight()
private
weight
public
setWeight()
public
getWeight()
private
bmi
public
setBmi()
public
getBmi()
Other Classes
HealthAnalyzer
BmiIndexer

Packages

Packages

- Для логического группирования множеств классов в связанные группы в Java применяется понятие **package** (**пакета**).
- **Packages** обеспечивают:
  - независимые пространства имён (**namespaces**)
  - ограничение доступа к классам
- **Packages** — это фактически обычная директория.

Packages

- **Packages** — это фактически обычная директория.

```
package your.package.which.can.has.any.name;
```

Package definition: example

```java
package com.rakovets;

public class User {
    public String name;

    public User(String name) {
        this.name = name;
    }

    void tellAboutYourself() {
        System.out.printf("Name: %s\n", name);
    }
}
```

Package definition: example

```
package com.rakovets;

public class Program {
    public static void main(String[] args) {
        User dmitry = new User("Dmitry");
        dmitry.tellAboutYourself();
    }
}
```

Packages and Terminal: example

```
cd D:\home\rakovets\dev
javac com\rakovets\Program.java
java com.rakovets.Program
```

```
Name: Dmitry
```

`import` Packages and Classes: example

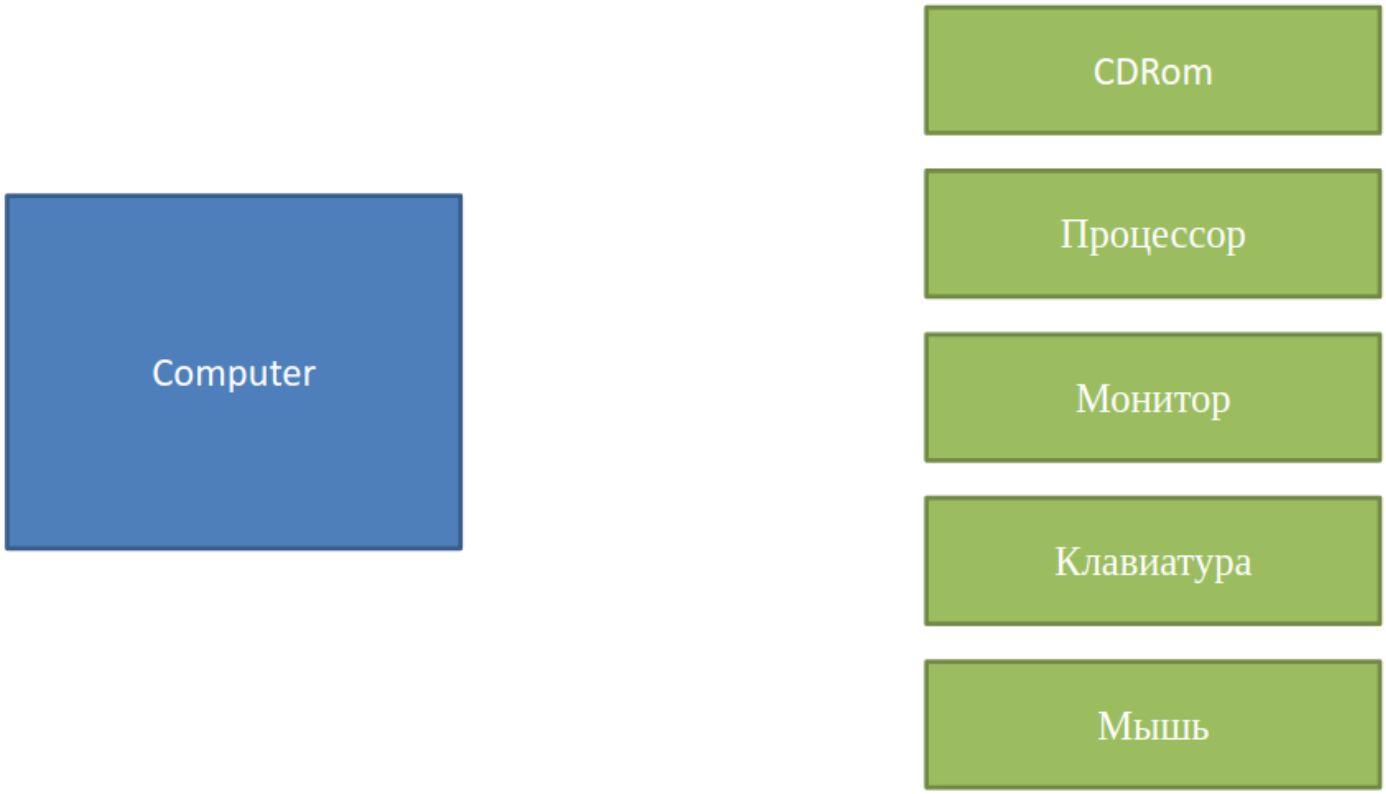
```
package com.rakovets;

import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
    }
}
```

`import` Packages and Classes: example

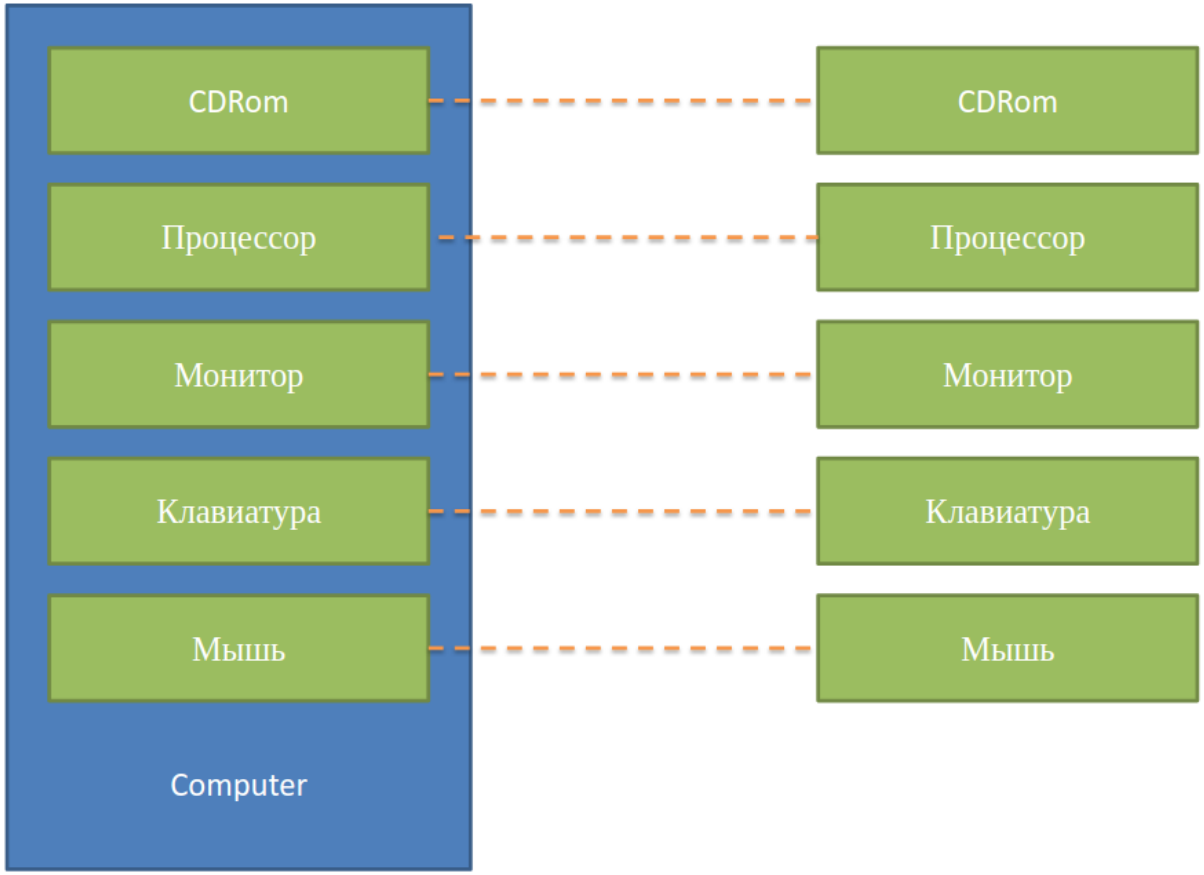
```
java.util.Date utilDate = new java.util.Date();
java.sql.Date sqlDate = new java.sql.Date();
```

Aggregation

Aggregation (Агрегация)



Aggregation



Example

```
public class CPU {
    private final String brand;
    private final String model;
    private final String socket;

    public CPU(String brand, String model, String socket) {
        this.brand = brand;
        this.model = model;
        this.socket = socket;
    }

    public void print() {
        System.out.printf("%s %s, %s\n", brand, model, socket);
    }
}
```

Example

```
public class MotherBoard {
    private final String brand;
    private final String model;
    private final String socket;

    public MotherBoard(String brand, String model, String socket) {
        this.brand = brand;
        this.model = model;
        this.socket = socket;
    }

    public void print() {
        System.out.printf("%s %s, %s\n", brand, model, socket);
    }
}
```

Example

```
public class PC {
    private final String codeName;
    private final CPU cpu;
    private final MotherBoard motherBoard;

    public PC(String codeName, CPU cpu, MotherBoard motherBoard) {
        this.codeName = codeName;
        this.cpu = cpu;
        this.motherBoard = motherBoard;
    }

    public void print() {
        System.out.printf("%s:\n", codeName);
        motherBoard.print();
        cpu.print();
    }
}
```

Example

```
public class Example6Composition {
    public static void main(String[] args) {
        CPU intel = new CPU("Intel", "i5 6400", "LGA 1151");
        MotherBoard asus = new MotherBoard("ASUS", "Z-170P", "LGA 1151
        PC dev = new PC("Dev", intel, asus);
        dev.print();
    }
}
```

```
Dev:
ASUS Z-170P, LGA 1151
Intel i5 6400, LGA 1151
```

Total

Нужно ли всегда создавать объекты?

- Всегда нужно создавать объекты (даже если программа простая).
- Всегда нужно писать код в стиле ООП.
- В проекте не должно быть лишних/неиспользуемых объектов.
- Никогда не давайте объекту чужие понятия и действия.