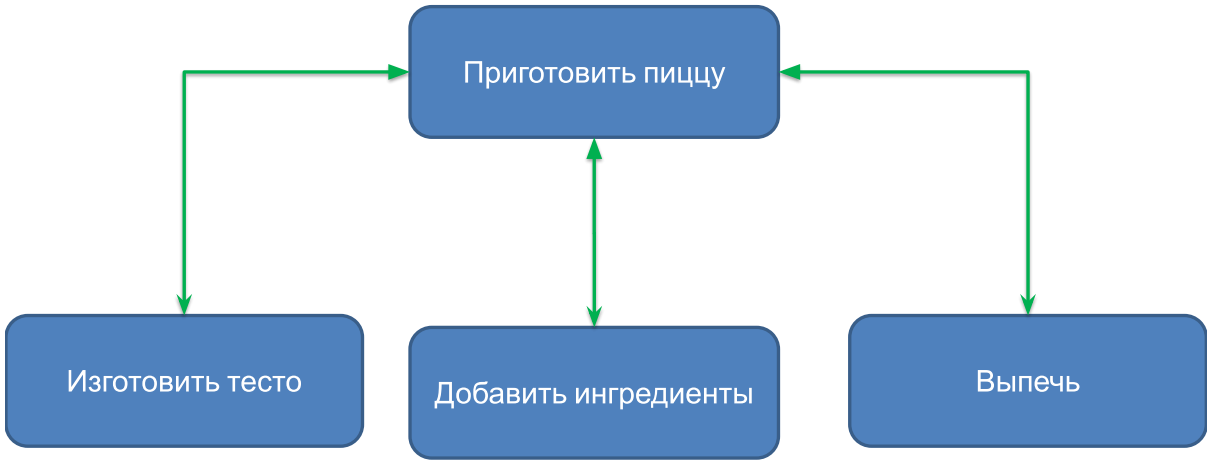
OOP

Problem

Структурное программирование

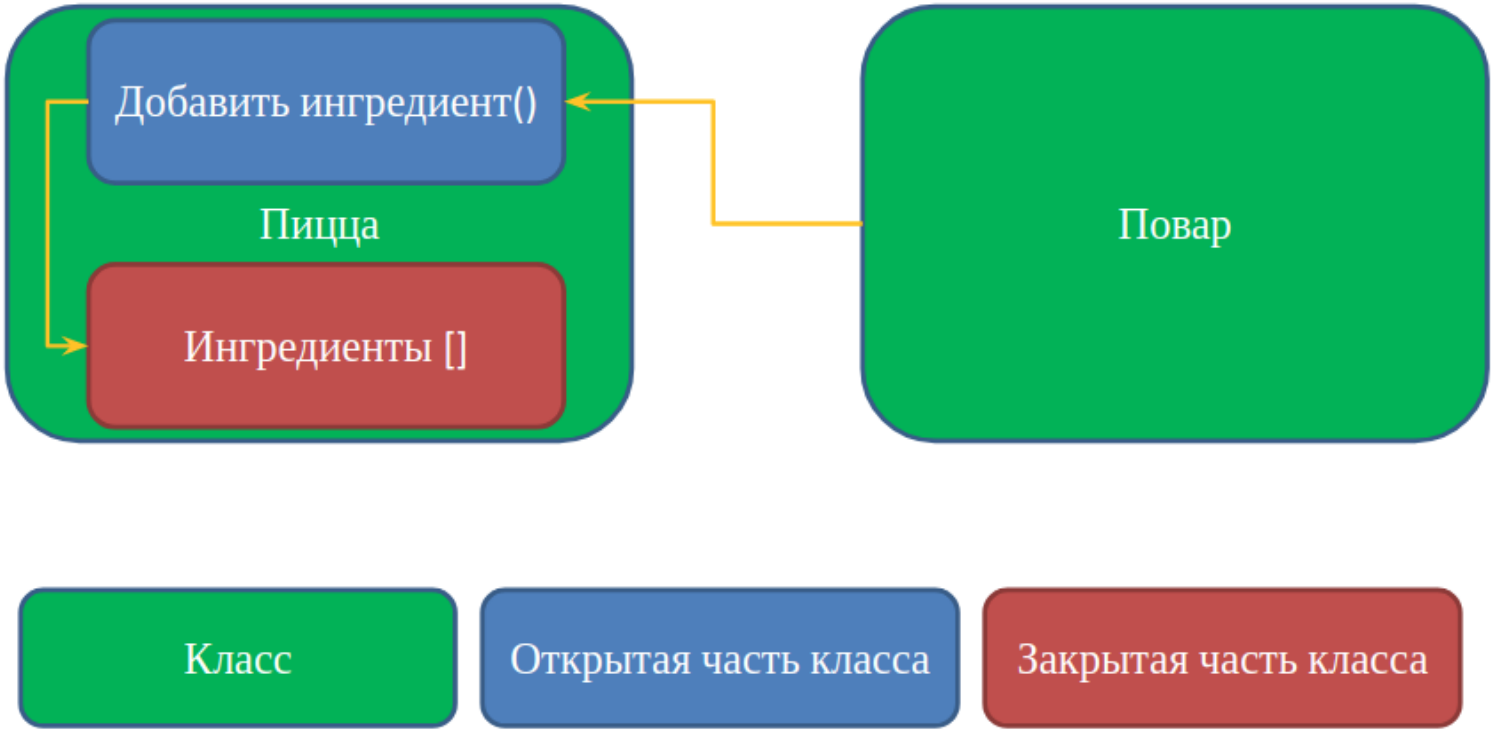
Принцип: код воздействует на данные



Solution

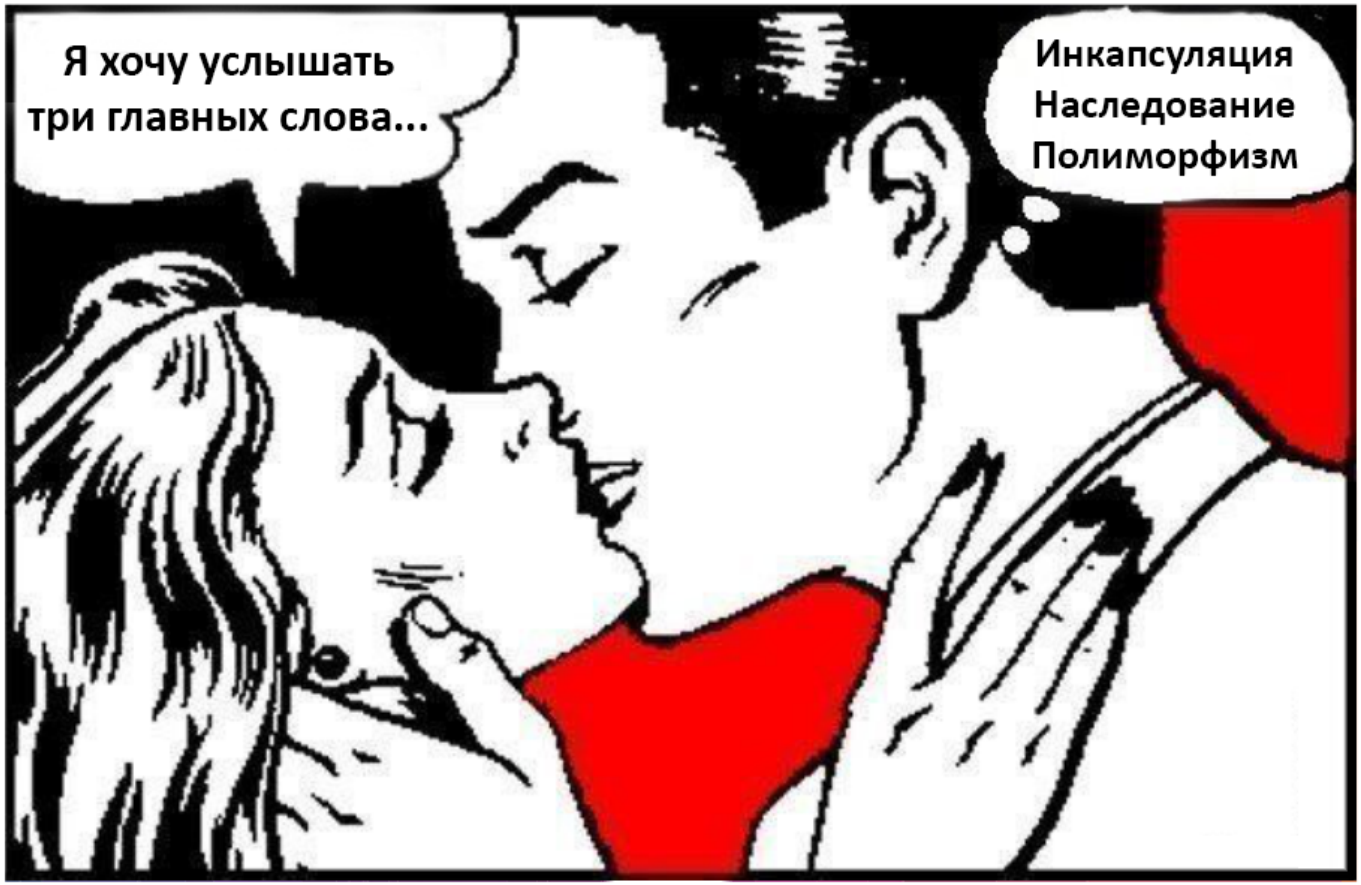
Объектно-ориентированное программирование

Принцип: данные управляют доступом к коду



OOP

OOP principles



OOP principles

- **Encapsulation (Инкапсуляция)**
- **Inheritance (Наследование)**
- **Polymorphism (Полиморфизм)**
- **Abstraction (Абстракция)**

Classes and objects

Classes and objects

- **Класс** (**Class**)- это *шаблон* для создания объектов
- **Объект** (**Object**) - это *экземпляр* класса
- *Функция*, созданная внутри класса, называется **метод** (**method**)
- *Переменная*, созданная внутри класса, называется **поле** (**field**)

Class definition: syntax

```
class Person {
    // class content
}
```

Class definition: example

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

    void displayInfo() {
        System.out.printf("Name: %s \tAge: %d\n", name, age);
    }
}

class Program {
    public static void main(String[] args) {
        Person tom;
    }
}
```

Constructor

Constructor (Конструктор)

- Конструктор и метод внешне похожи
- Конструктор имеет имя как у класса
- В конструкторе не должно быть лишней логики
- Если конструктор не указан – компилятор создаст конструктор по умолчанию
- Если создали свой конструктор – конструктор по умолчанию не создаётся

Default Constructor: example

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

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

Default Constructor: example

```
public class Program {
    public static void main(String[] args) {
        Person tom = new Person();
        tom.displayInfo();
        tom.name = "Tom";
        tom.age = 34;
        tom.displayInfo();
    }
}
```

Constructors: example

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

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

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

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

    void displayInfo() {
```

Constructors: example

```
public class Program {
    public static void main(String[] args) {
        Person bob = new Person();
        bob.displayInfo();
        Person tom = new Person("Tom");
        tom.displayInfo();
        Person sam = new Person("Sam", 25);
        sam.displayInfo();
    }
}
```

Keyword `this`

Keyword `this`: example

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

    Person() {
        this("Undefined", 18);
    }

    Person(String name) {
        this(name, 18);
    }

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

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

Keyword `this`: example

```
public class Program {
    public static void main(String[] args) {
        Person undef = new Person();
        undef.displayInfo();
        Person tom = new Person("Tom");
        tom.displayInfo();
        Person sam = new Person("Sam", 25);
        sam.displayInfo();
    }
}
```

Initializers

Initializers (блок инициализации): example

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

    {
        this.name = "Undefined";
        this.age = 18;
    }

    Person() {
    }

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

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

Initializers: example

```
public class Program {
    public static void main(String[] args) {
        Person undef = new Person();
        undef.displayInfo();
        Person tom = new Person("Tom");
        tom.displayInfo();
    }
}
```

Objects as parameters of methods

Objects as parameters of methods: example

```
class Person {
    private String name;

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

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

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

Objects as parameters of methods: example

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

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

Objects as parameters of methods: example 2

```
class Person {
    private String name;

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

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

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

Objects as parameters of methods: example 2

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

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

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

Packages

Package definition: syntax

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

Package definition: example

```
package com.rakovets;

class Person {
    String name;
    int age;

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

    void displayInfo() {
        System.out.printf("Name: %s \t Age: %d \n", name, age);
    }
}
```

Package definition: example

```
package com.rakovets;

public class Program {
    public static void main(String[] args) {
        Person kate = new Person("Kate", 32);
        kate.displayInfo();
    }
}
```

Packages and Terminal: example

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

`import` Packages and Classes: example

```java
package com.rakovets;

import java.util.Scanner;
import java.util.*;

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

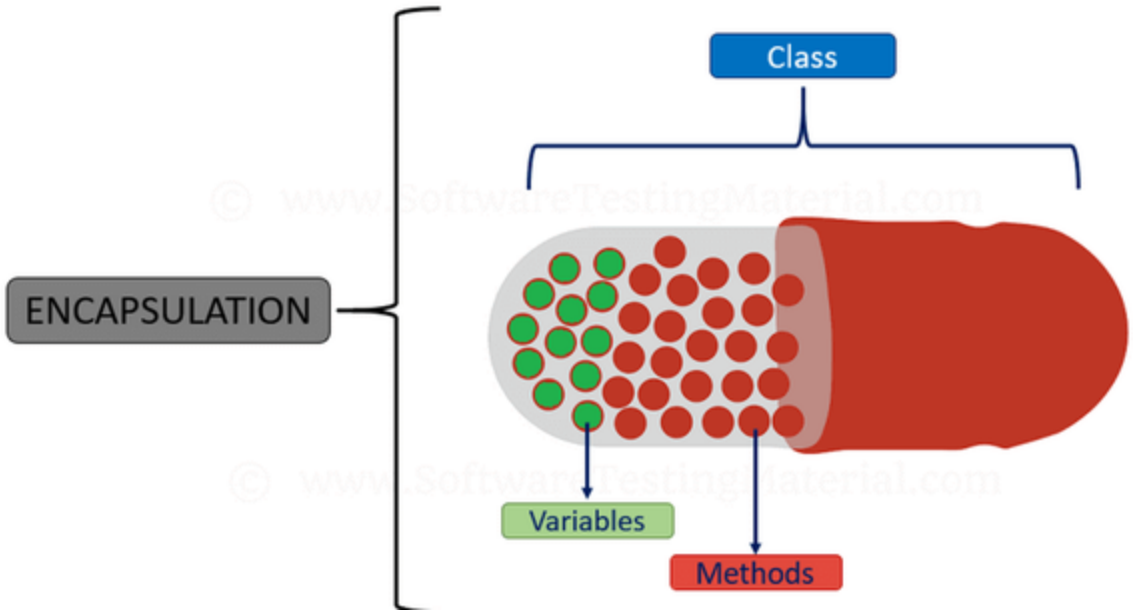
`import` Packages and Classes: example

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

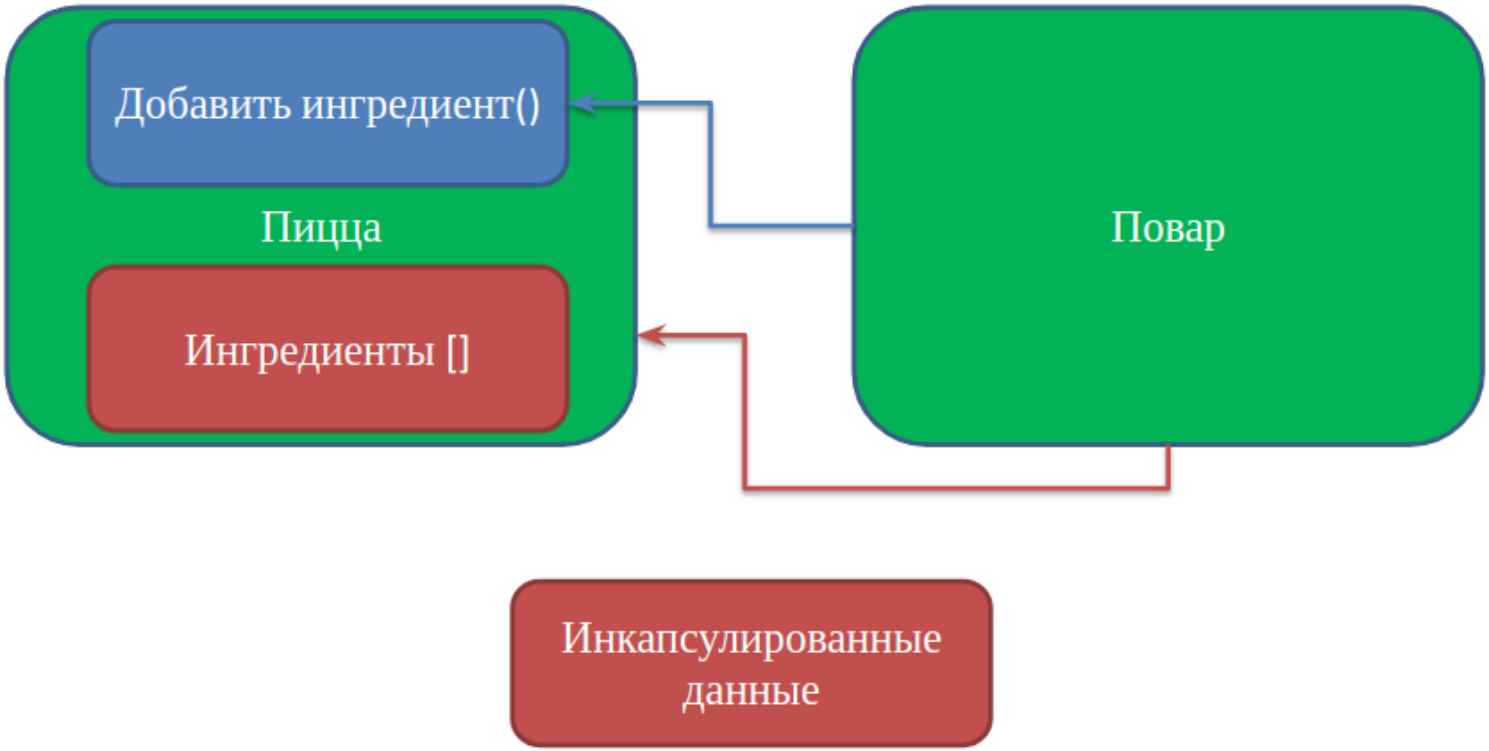
Encapsulation

Encapsulation

- **Encapsulation** (**Инкапсуляция**) - это процесс объединения кода и данных в единый блок.
- **Encapsulation** - это ограничение доступа одних компонентов программы к другим



Encapsulation



Access modifiers

Access modifiers (Модификаторы доступа)

Access modifiers

- `public`- доступно из любого места. Чаще всего для внешнего интерфейса.
- `protected`- внутри пакета и в дочерних классах
- *default* - доступно внутри пакета – использовать нежелательно
- `private` - доступно только внутри класса – для скрытия реализации (инкапсуляции)

Access modifiers

	private	default	ptotected	public
same class	+	+	+	+
same package subclass	-	+	+	+
same package non-subclass	-	+	+	+
different package subclass	-	-	+	+
different package non-subclass	-	-	-	+

Access modifiers

Bad practice:

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

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

Access modifiers

Bad practice:

```
public class Program {
    public static void main(String[] args) {
        Person kate = new Person("Kate", 30);
        System.out.println(kate.age);
        kate.age = 33;
        System.out.println(kate.age);
    }
}
```

Access modifiers

Good practice:

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

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

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

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

    public int getAge() {
        return this.age;
    }
```

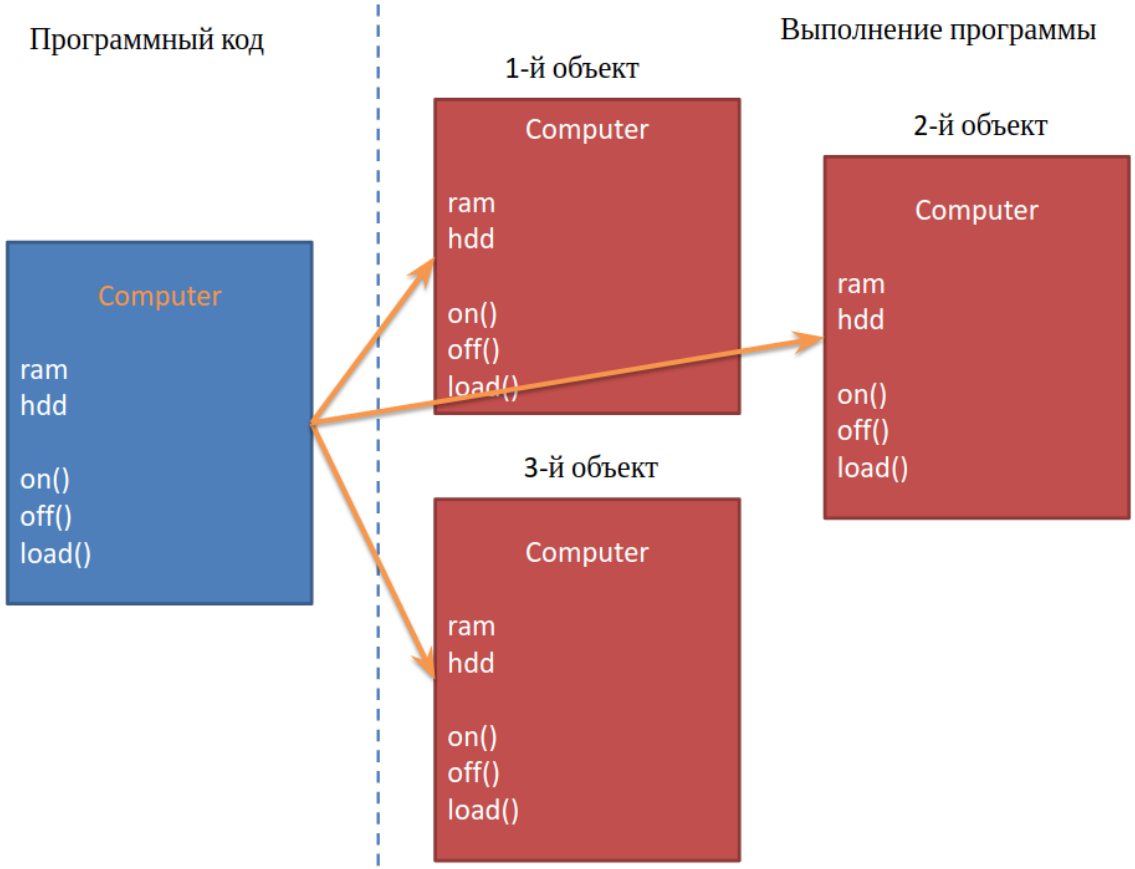
Access modifiers

Good practice:

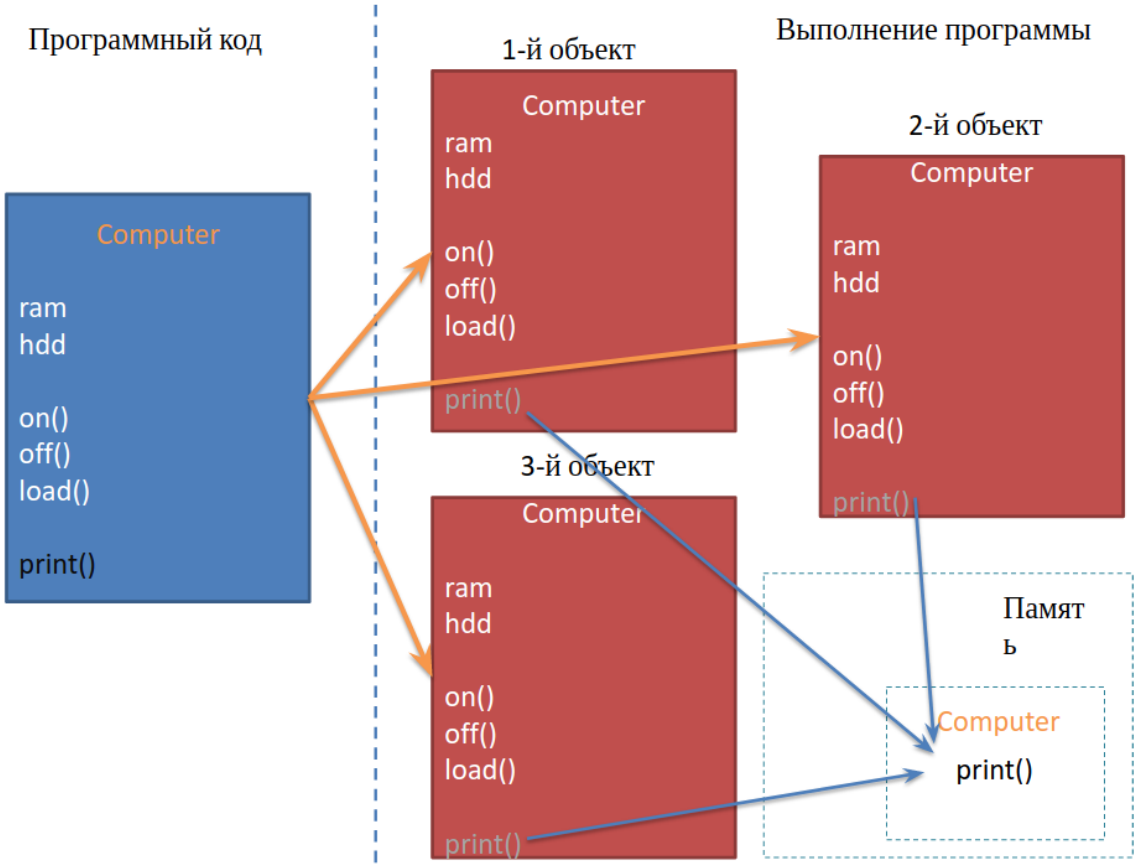
```
public class Program {
    public static void main(String[] args) {
        Person kate = new Person("Kate", 30);
        System.out.println(kate.getAge());
        kate.setAge(33);
        System.out.println(kate.getAge());
    }
}
```

Keyword `static`

non `static`


Программный код
Выполнение программы
1-й объект
2-й объект
3-й объект
Computer
ram
hdd
on()
off()
load()
Computer
ram
hdd
on()
off()
load()
Computer
ram
hdd
on()
off()
load()
Computer
ram
hdd
on()
off()
load()

static


Программный код
1-й объект
Выполнение программы
Computer
ram
hdd
on()
off()
load()
print()
Computer
ram
hdd
on()
off()
load()
print()
2-й объект
Computer
ram
hdd
on()
off()
load()
print()
3-й объект
Computer
ram
hdd
on()
off()
load()
print()
Память
Computer
print()

Keyword `static`: example

```
public static void main(String[] args) {
    // statements
}
```

`static` fields: example

```
class Person {
    private int id;
    static int counter = 1;

    Person() {
        id = counter++;
    }

    public void displayId() {
        System.out.printf("Id: %d \n", id);
    }
}
```

`static` fields: example

```
public class Program {
    public static void main(String[] args) {
        Person tom = new Person();
        Person bob = new Person();
        tom.displayId();
        bob.displayId();
        System.out.println(Person.counter);
        Person.counter = 8;
        Person sam = new Person();
        sam.displayId();
    }
}
```

`static` constants: example

```
public class Program {
    public static void main(String[] args) {
        double radius = 60;
        System.out.printf("Radisu: %f \n", radius);
        System.out.printf("Area: %f \n", Math.PI * radius);
    }
}

public class Math {
    public static final double PI = 3.14;
}
```

`static` methods: example

```
public class Operation {
    static int sum(int x, int y) {
        return x + y;
    }

    static int subtract(int x, int y) {
        return x - y;
    }

    static int multiply(int x, int y) {
        return x * y;
    }
}
```

`static` methods: example

```
public class Program {
    public static void main(String[] args) {
        System.out.println(Operation.sum(45, 23));
        System.out.println(Operation.subtract(45, 23));
        System.out.println(Operation.multiply(4, 23));
    }
}
```

`static` initializers: example

```
class Person {
    private int id;
    static int counter;

    static {
        counter = 105;
        System.out.println("Static initializer");
    }

    Person() {
        id = counter++;
        System.out.println("Constructor");
    }

    public void displayId() {
        System.out.printf("Id: %d \n", id);
    }
}
```

`static` initializers: example

```
public class Program {
    public static void main(String[] args) {
        Person tom = new Person();
        Person bob = new Person();
        tom.displayId();
        bob.displayId();
    }
}
```

Static `import`: example

```
package study;

import static java.lang.System.*;
import static java.lang.Math.*;

public class Program {
    public static void main(String[] args) {
        double result = sqrt(20);
        out.println(result);
    }
}
```

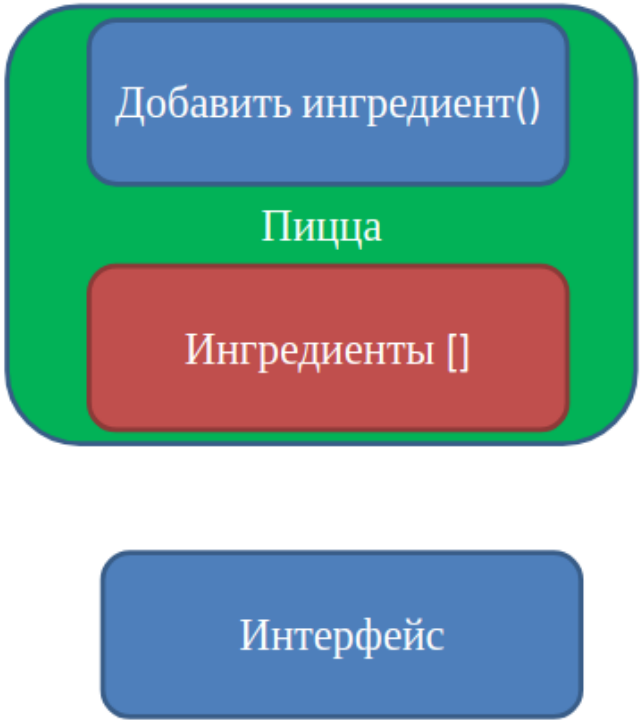
Ключевое слово `static`

- Статичному элементу запрещено использовать нестатичные переменные и методы класса.
- Статичные элементы не манипулируют свойствами объекта и не привязаны к конкретному объекту.
- Статичные методы и свойства можно вызывать:
    1. Через имя класса
    2. Через ссылку на экземпляр класса
- Чаще используется первый вариант
- Статичный элемент связан не с объектом, а с классом, следовательно его нельзя переопределить

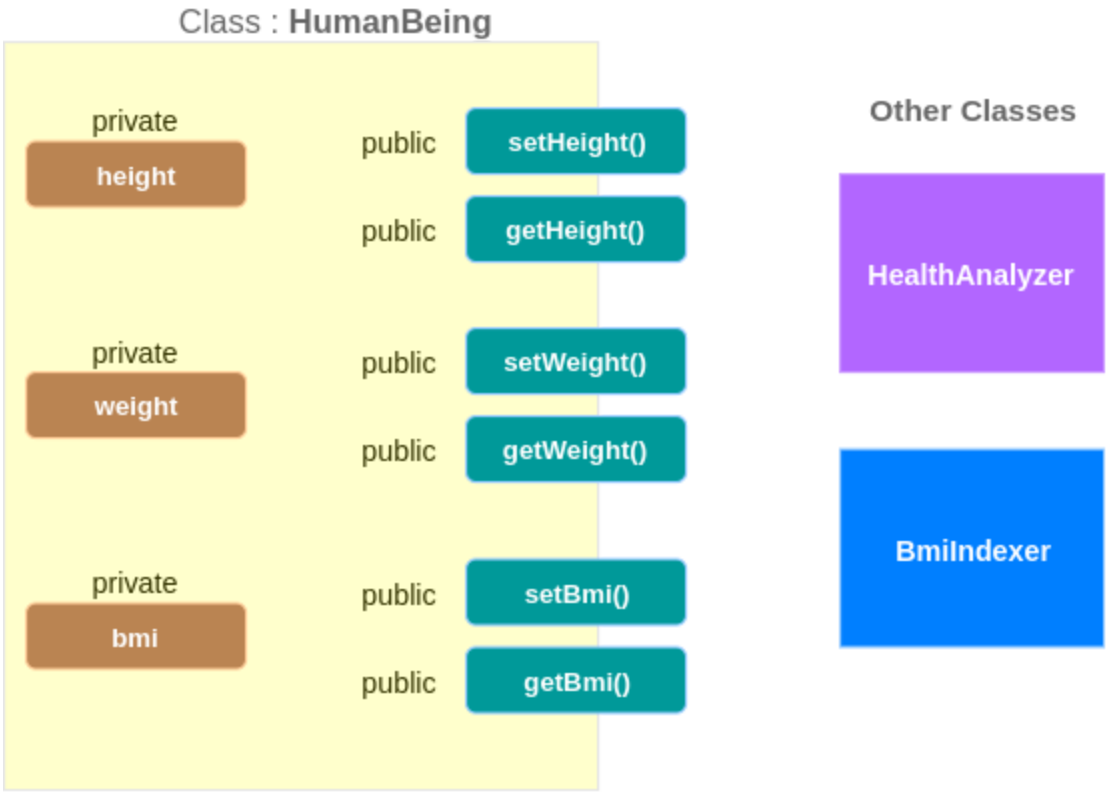
Interface

Interface (Интерфейс)

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

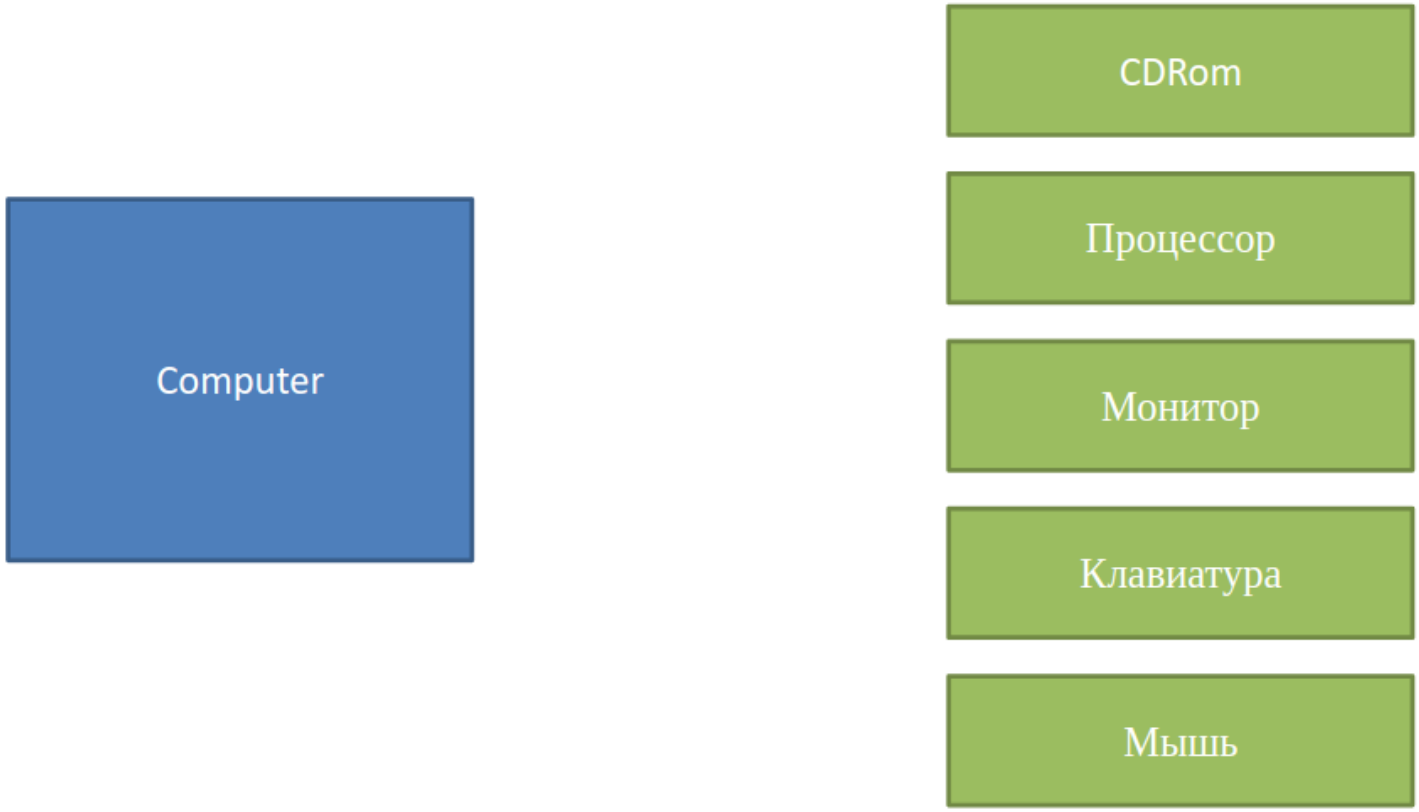


Interface

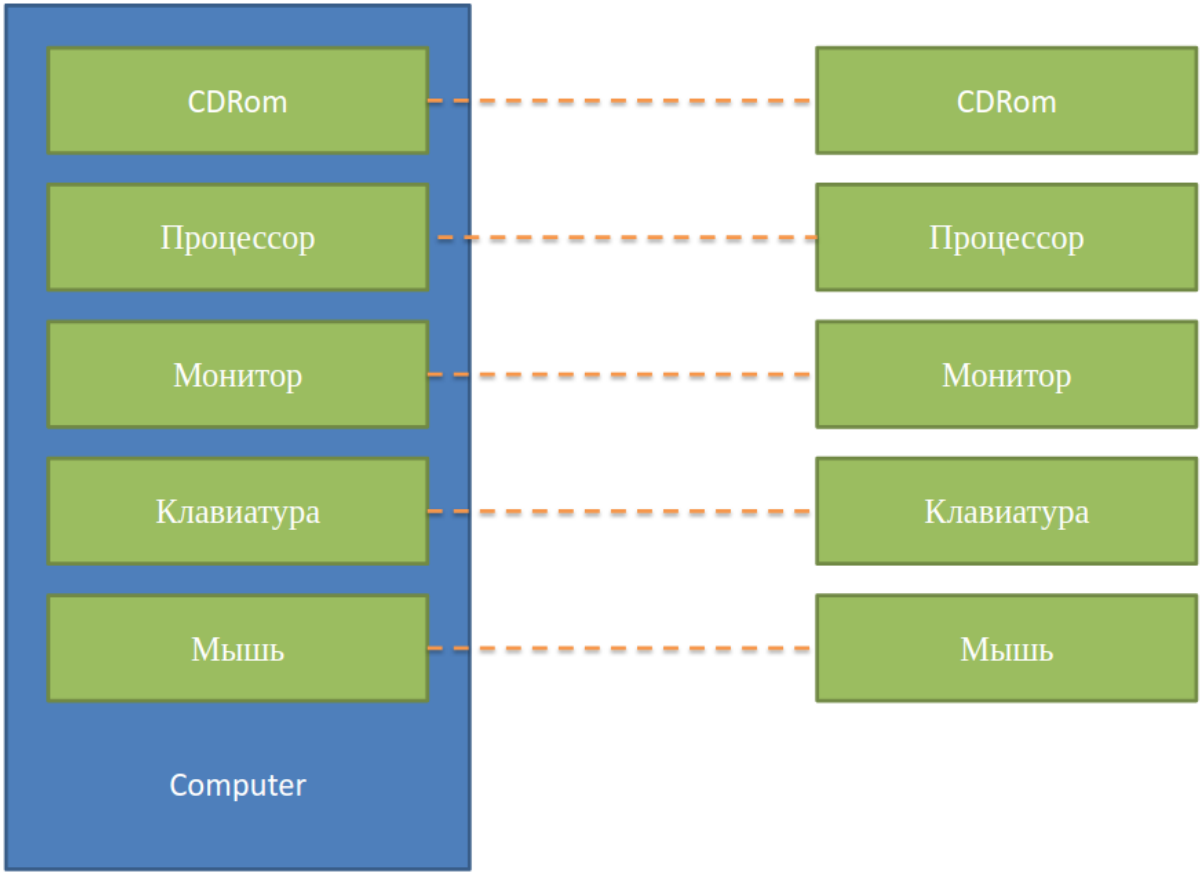

Class : HumanBeing
private
height
public
setHeight()
public
getHeight()
private
weight
public
setWeight()
public
getWeight()
private
bmi
public
setBmi()
public
getBmi()
Other Classes
HealthAnalyzer
BmiIndexer

Composition

Composition (Композиция)



Composition



Total

Нужно ли всегда создавать объекты?

- Даже если программа простейшая – всегда нужно создавать объекты и писать код в стиле ООП
- Это должно быть привычкой
- В программе не должно быть лишних объектов
- Никогда не давайте объекту чужие понятия и действия