
ВСЕ
ОЧЕНЬ ПЛОХО
TURBOPORTAL.RU

Лекция 2

ООП простыми словами

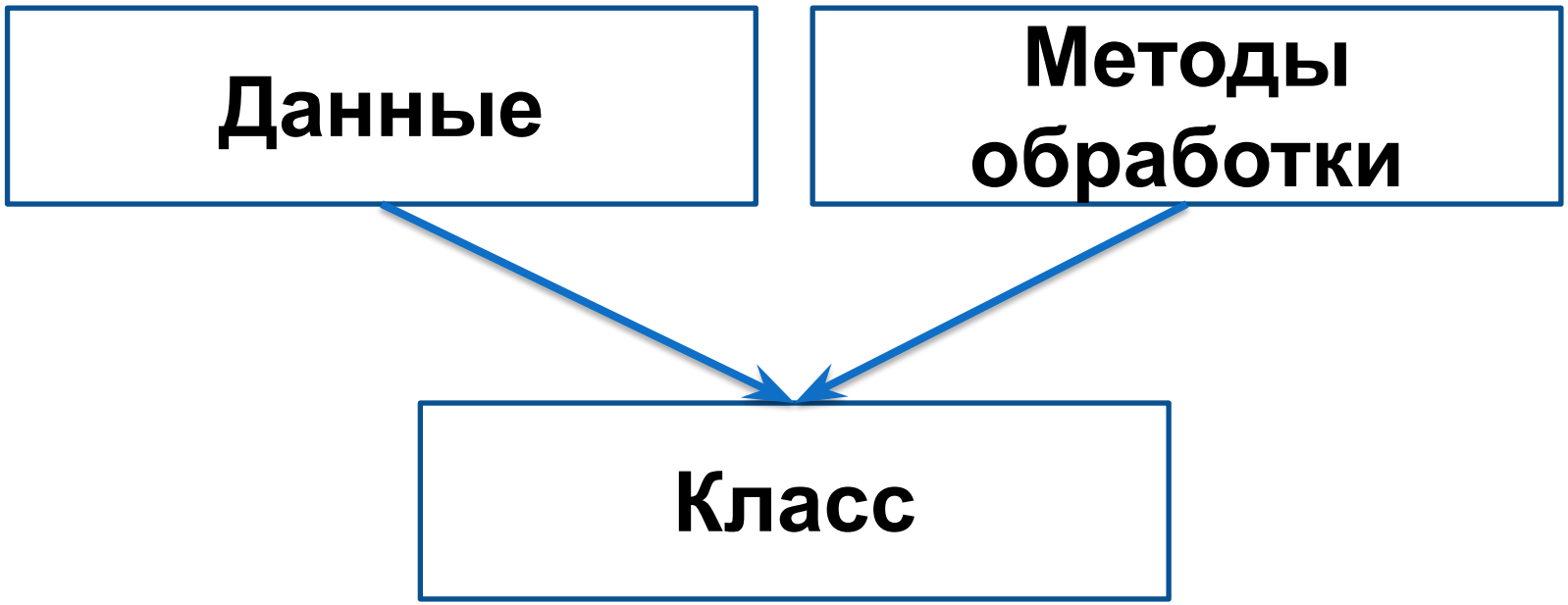
Опрос

1. Для какой цели изначально создавалась Java?
2. Как устроен механизм запуска программы на Java?
3. Три принципа ООП?

План занятия

1. Понятие класса
2. Понятие объекта
3. Понятие интерфейса
4. Наследование
5. Инкапсуляция
6. Полиморфизм
7. Лексика Java

Зачем вообще нужны классы



Простыми словами:

Класс это составной тип данных, в котором кроме данных, так же описано то, как с этими данными работать.

Объект

Объект – это экземпляр конкретного класса.

Класс

- **описывает множество объектов имеющих одинаковый набор данных**
- **описывает методы работы с этими данными**

Объект

- **состояние**
- **поведение**

Уникальность

Класс

Объект

Кот

Рабочий

Университет

Город

Интерфейс

Сущность, которая описывает, что могут делать объекты, которые реализуют этот интерфейс

Класс

- **Имеет методы с конкретной реализацией**
- **Эти методы уже не подлежат изменению**

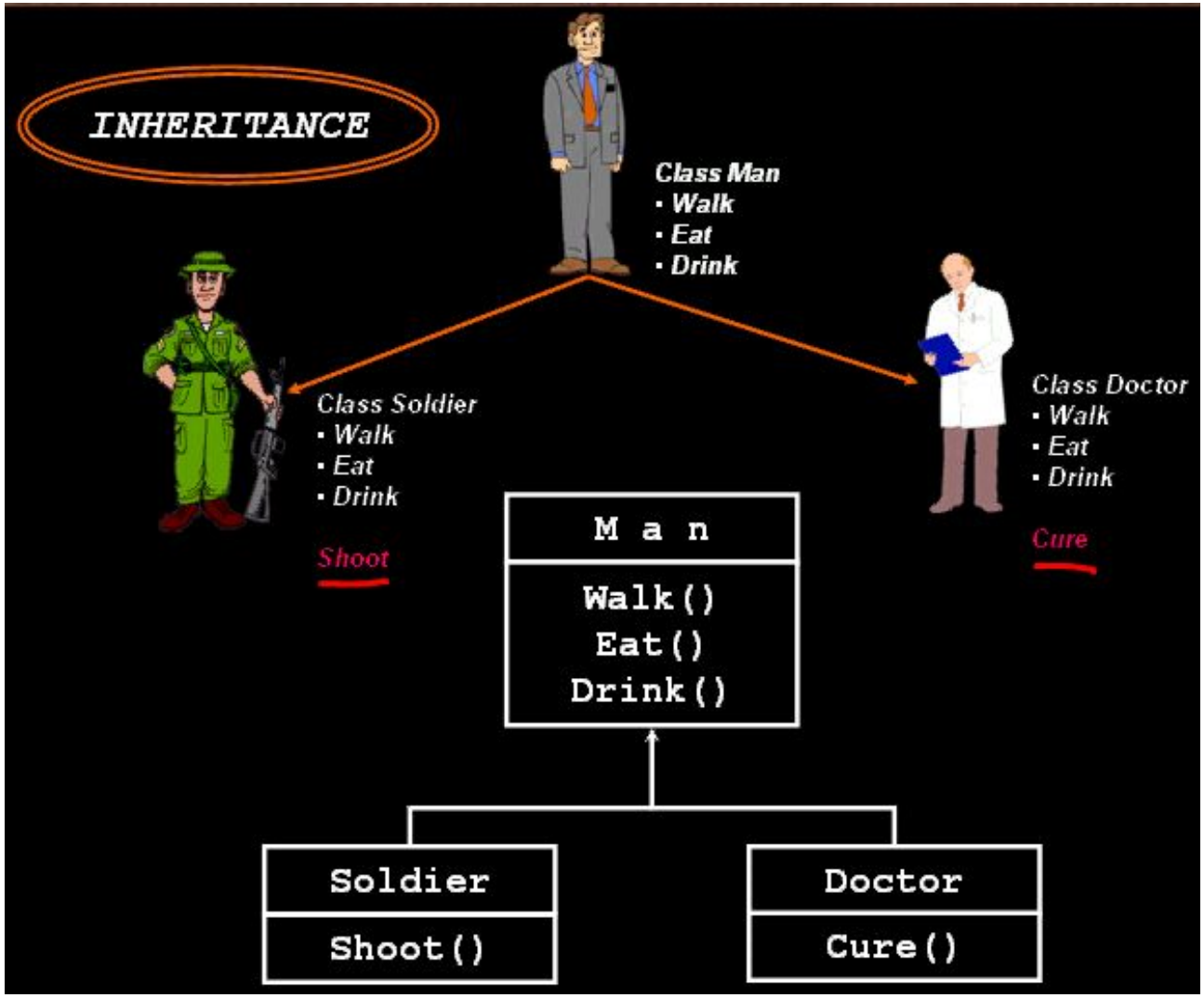
Интерфейс

- **Описывает лишь то, что мы можем произвести некоторые действия над объектом**

Коротко: интерфейс описывает только поведение объектов

Это твое ООП

Наследование

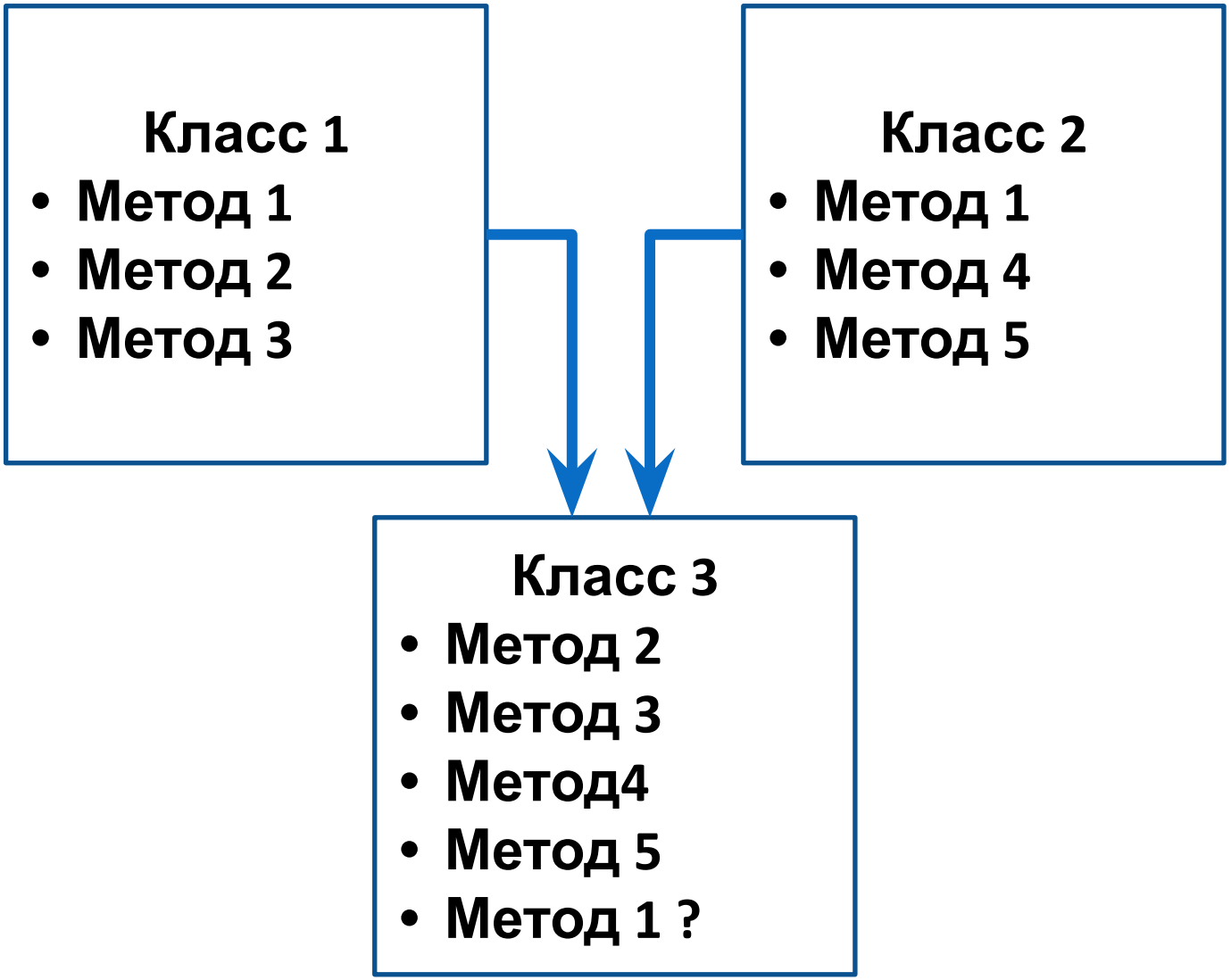


Наследование

Наследование – один из механизмов ООП, позволяющий классам-предкам использовать данные и методы класса-родителя, дополняя и расши

Проблема наследования

Пример


Класс 1
• Метод 1
• Метод 2
• Метод 3
Класс 2
• Метод 1
• Метод 4
• Метод 5
Класс 3
• Метод 2
• Метод 3
• Метод4
• Метод 5
• Метод 1 ?

Вопрос

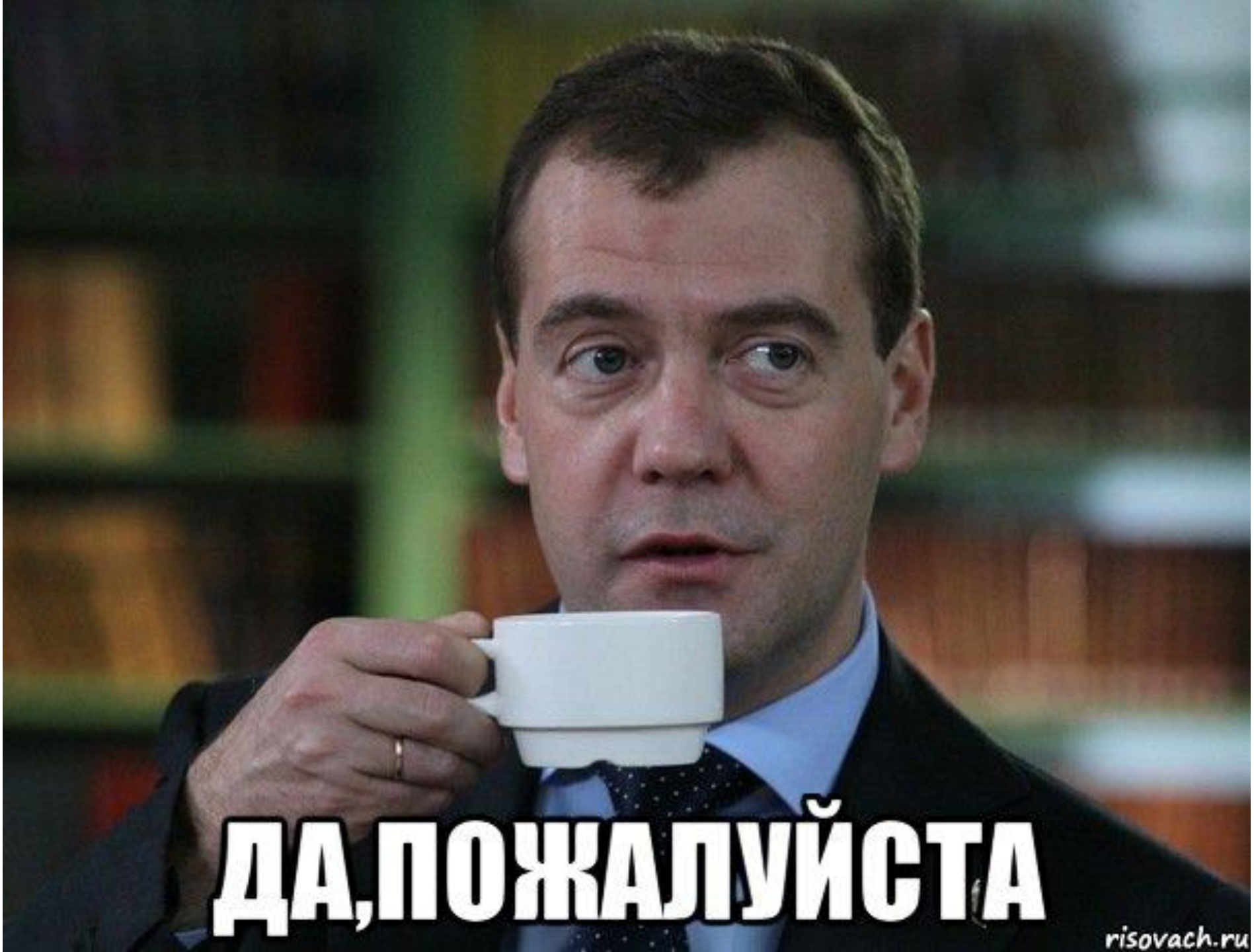
Что делать, когда у двух классов-родителей есть метод с одинаковым


TO-DOLIST
~~NOTHING~~
pikabu.ru

Множественное наследование

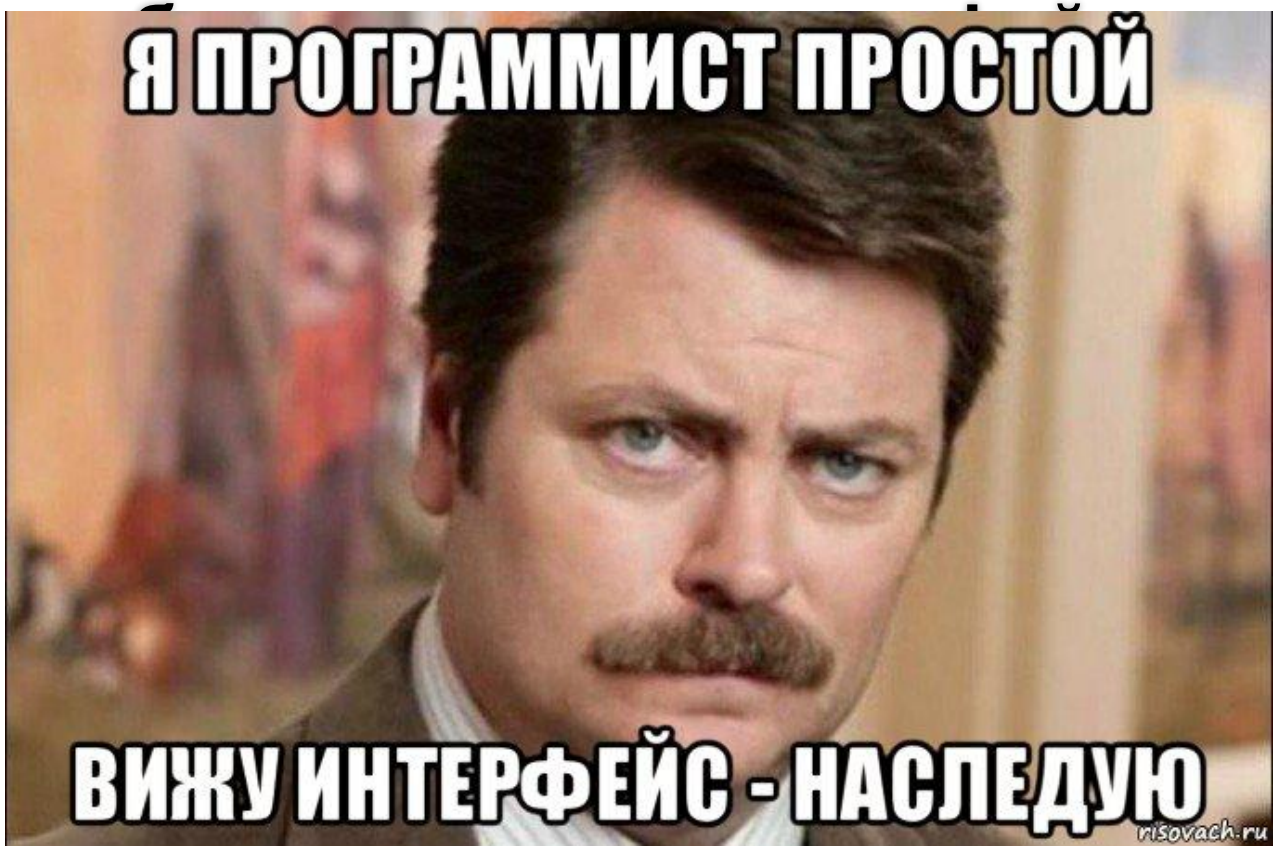
В Java нет и не может быть множественного наследования от классов.

А как же


ДА,ПОЖАЛУЙСТА
risovach.ru

Множественное наследование

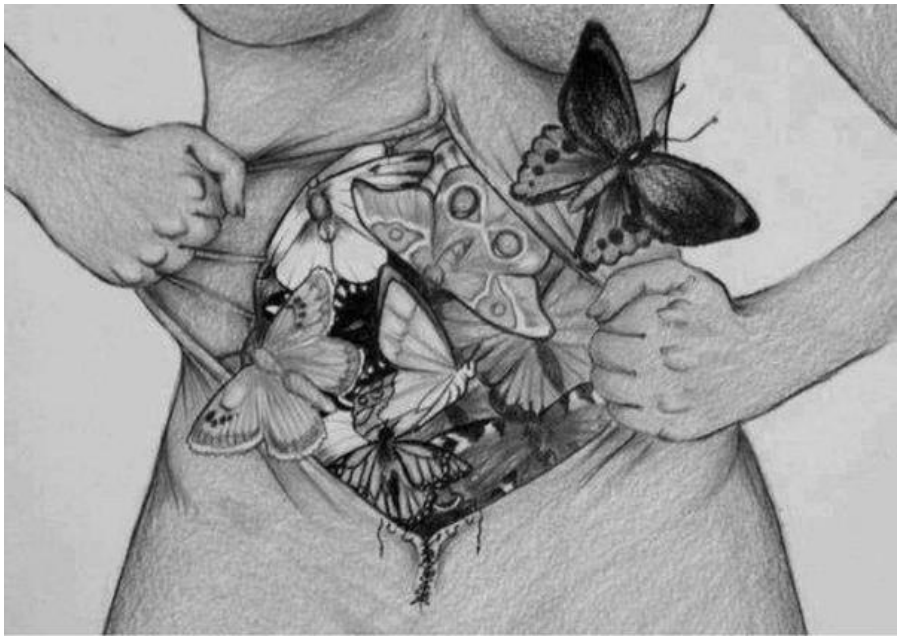
Так как интерфейсы описывают лишь поведение и не содержат какой-либо реализации, то мы можем наследовать

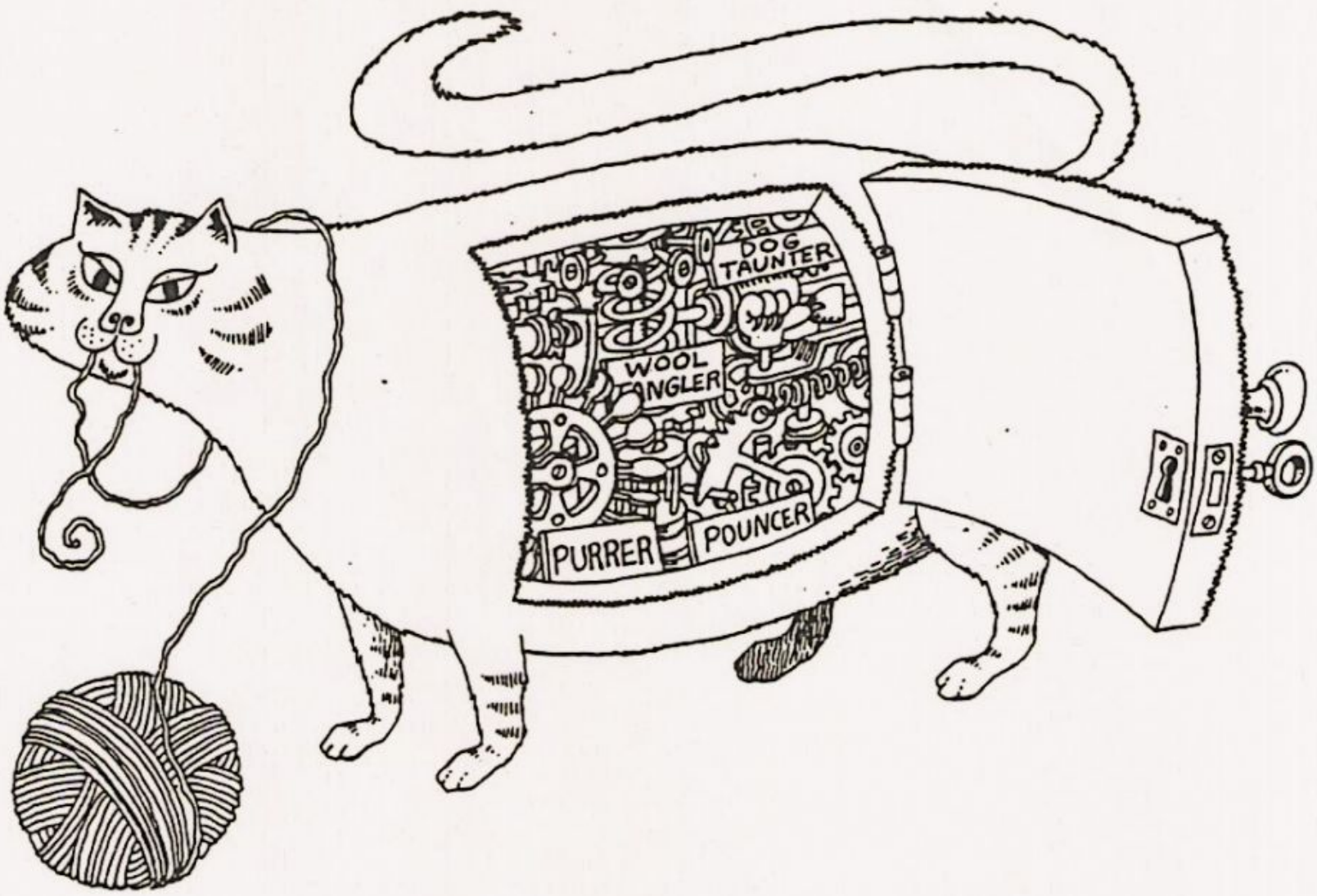


Инкапсуляция

Объединение данных и методов их обработки в одном классе.

Это приводит к сокрытию реализации класса и отделению внутреннего представления от внешнего.

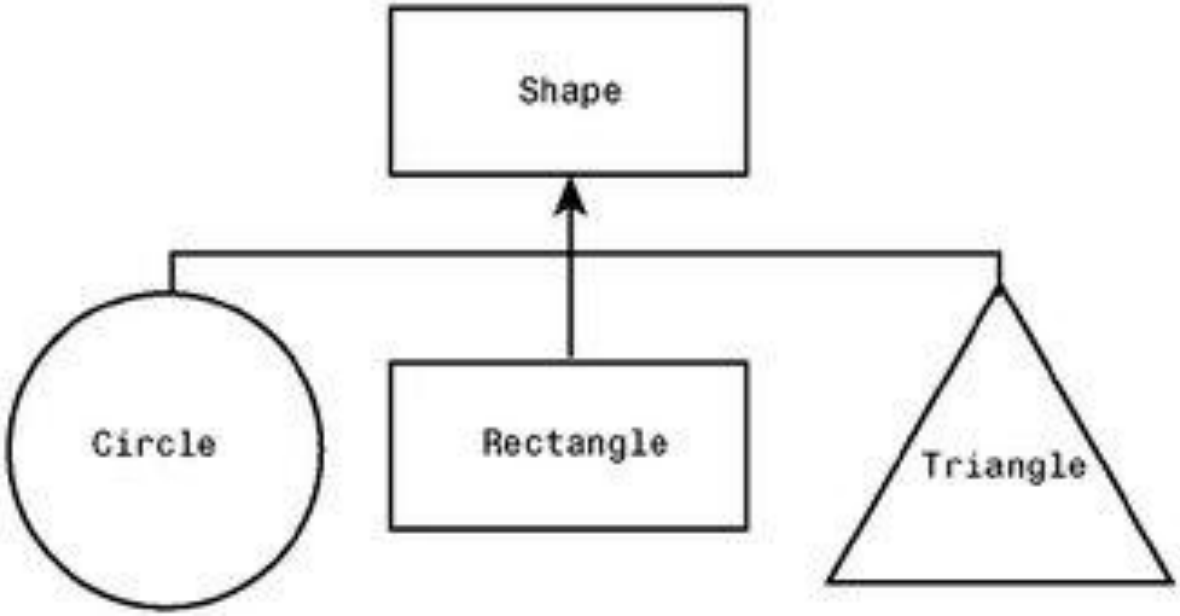




Encapsulation hides the details of the implementation of an object.

Полиморфизм

Способность объекта соответствовать во время выполнения двум или более возможным типам (классам).



Возвращаемся в Java

- Все сущности в Java являются объектами, классами либо интерфейсами
- Строгая реализация инкапсуляции
- Реализовано одиночное наследование от класса и множественное от интерфейсов

Пакетик

Java package :

- **механизм логического объединения классов**
- **«библиотека», которая может распространяться независимо от других пакетов и может применяться в сочетании с другими пакетами.**

В пакет входят:

- **Классы**
- **Интерфейсы**
- **Пакеты**

А зачем они нужны

- Объединяют логически связанные классы и интерфейсы в единое целое
- Создают пространство имен, необходимое для устранения конфликтов идентификаторов, относящихся к различным классам

А как сделать пакет

- В виде иерархии каталогов, в которых содержаться файлы с классами
- В виде jar-архива

Имена

Имена – это названия переменных.

Пространства имен

- **Пакеты**
- **Классы**
- **Поля**
- **Методы**
- **Локальные переменные**

Бывают составные (java.lang.Double) и простые (Double) имена.

Пример

```
package Newly;

class Newly {
   Newly Newly(Newly Newly) {
        Newly:
           for(;;) {
              if (Newly.Newly(Newly) == Newly)
                  break Newly;
           }
           return Newly;
      }
}
```

Работа с пакетами

Объявление находится в самом начале файла

```
package <имя_пакета>
```

Чтобы получить доступ к типам в другом пакете, необходимо произвести операцию импорта.

Например:

```
import java.net.URL
import java.net.*
```

Объявления

```
package first;
class MyFirstClass {
}
interface MyFirstInterface {
}
```

- Область видимости класса и интерфейса – пакет
- Доступ к типу извне его пакета
  - по составному имени
  - через выражения импорта
- Разграничение (модификаторы) доступа

Объявления

- В одном файле может быть максимум один `public` тип
- Имя публичного типа и имя файла должны совпадать
- Другие не-`public` типы файла должны использоваться только внутри текущего пакета
- Как правило, один файл содержит один тип

Правила именования

- **Пакеты**
  `java.lang, javax.swing, ru.ssau.fit`
- **Типы**
  `Student, ArrayIndexOutOfBoundException`
  `Cloneable, Runnable, Serializable`
- **Поля**
  `value, enabled, distanceFromShop`
- **Методы**
  `getValue, setValue, isEnabled, length, toString`
- **Поля-константы**
  `PI, SIZE_MIN, SIZE_MAX, SIZE_DEF`

Лексика Java

Кодировка

- Java ориентирован на Unicode
- Символы Unicode задаются в следующем формате

  `\u0046, \u00C6, \u01A9`

- Java чувствителен к регистру!

Исходный код

Ваш код делится на:

- **Пробелы**
- **Комментарии**
- **Лексемы**

Комментарии

- `// Комментарий`
  Все что идет после `//` и до конца текущей строки игнорируются

- `/* Комментарий */`
  Все символы, заключенные между `/*` и `*/`, игнорируются

- `/** Комментарий */`
  Комментарии javadoc

Составляющие кода

- Идентификаторы
- Служебные слова

  `class, public, const, goto`
- Литералы
- Разделители

  `{ } [ ] ( ) ; . ,`
- Операторы

  `= > < ! ? : == && ||`

Идентификаторы

- Имена, задаваемые элементам языка для доступа к ним
- Можно записывать символами Unicode
- Состоят из букв и цифр, знаков `_` и `$`
- Не допускают совпадения со служебными словами, литералами `true`, `false`, `null`
- Длина имени не ограничена

Служебные слова

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

Типы данных

- **Ссылочные**
  - Предназначены для работы с объектами
  - Переменные содержат ссылки на объекты
  - Тип переменной определяет то, какой объект будет доступен
- **Примитивные (простые)**
  - Предназначены для работы со значениями простых типов
  - Переменные содержат непосредственно значения

Ссылочные типы

- К ссылочным типам относятся типы классов (в т.ч. массивов) и интерфейсов
- Переменная ссылочного типа способна содержать ссылку на объект, относящийся к этому типу
- Ссылочный литерал `null`

Примитивные типы

- **Булевский (логический) тип**
  - `boolean` – допускает хранение значений true или false
- **Целочисленные типы**
  - `char` – 16-битовый символ Unicode
  - `byte` – 8-битовое целое число со знаком
  - `short` – 16-битовое целое число со знаком
  - `int` – 32-битовое целое число со знаком
  - `long` – 64-битовое целое число со знаком
- **Вещественные типы**
  - `float` – 32-битовое число с плавающей точкой
  - `double` – 64-битовое число с плавающей точкой

Литералы

- **Булевы**
  `true false`
- **Символьные**
  `'a' '\n' '\\' '\377' '\u0064'`
- **Целочисленные**
  `29  035`
  - По умолчанию имеют тип `int`
- **Числовые с плавающей запятой**
  `1.  .1  1e1 1e-4D 1e+5f`
  - По умолчанию имеют тип `double`
- **Строковые**
  `“Я литерал”  ""`

Описание класса

Класс может содержать:

- **поля**
- **методы**
- **вложенные классы и интерфейсы**

```
public class CurrentEquipment implements Equipment {
    private String model;

    public CurrentEquipment(String model) {this.model = model;}

    public String getModel() {return model;}

    private class SampleClass{}
}
```

Модификаторы объявления класса

- `public`
  Признак общедоступности класса
- `abstract`
  Признак абстрактности класса
- `final`
  Завершенность класса (класс не допускает наследования)
- `strictfp`
  Повышенные требования к операциям с плавающей точкой

Поля класса

- **Объявление поля:**

```
[модификаторы] <тип> {<имя> [=
<инициализирующее выражение>]};
```

```
double sum = 2.5 + 3.7;
```

```
public double val = sum + 2 *
Math.sqrt(2);
```

- **Если поле явно не инициализируются, ему присваивается значение по умолчанию его типа (`0`, `false` или `null`)**

Поля класса

- **Модификаторы полей:**
  - **модификаторы доступа**
  - **`static`**
    поле статично (принадлежит контексту класса)
  - **`final`**
    поле не может изменять свое значение после инициализации
  - **`transient`**
    поле не сериализуется (влияет только на механизмы сериализации)
  - **`volatile`**
    усиливает требования к работе с полем в многопоточных программах

Методы

- **Объявление метода:**

```
[модификаторы] <тип> <сигнатура>
[throws исключения] {<тело>}
```

```
class Primes {
    static int nextPrime(int current) {
        <Вычисление простого числа в теле метода>
    }
}
```

Модификаторы методов

- Модификаторы доступа
- `abstract`
  абстрактность метода (тело при этом не описывается)
- `static`
  статичность метода (метод принадлежит контексту класса)
- `final`
  завершенность метода (метод не может быть переопределен при наследовании)

Модификаторы методов

- `synchronized`
  синхронизированность метода (особенности вызова метода в многопоточных приложениях)
- `native`
  «нативность» метода (тело метода не описывается, при вызове вызывается метод из native-библиотеки)
- `strictfp`
  повышенные требования к операциям с плавающей точкой

Особенности методов

- Для нестатических методов вызов через ссылку на объект или в контексте объекта

```
reference.method();
methodReturningReference().method();
```

- Для статических методов вызов через имя типа, через ссылку на объект или в контексте класса

```
ClassName.staticMethod();
reference.staticMethod();
staticMethodReturningReference().method();
```

- Наличие круглых скобок при вызове **обязательно**, т.к. они являются оператором вызова метода

Особенности методов

- Возвращается **одно** значение простого или объектного типа

```
return someValue;
```

- Аргументы передаются **по значению**
  - для примитивных типов копируются сами значения
  - для ссылочных типов копируется значение ссылки
- Перегруженными являются методы с одинаковыми именами и различными **сигнатурами**

Спасибо за внимание. Ваши вопросы