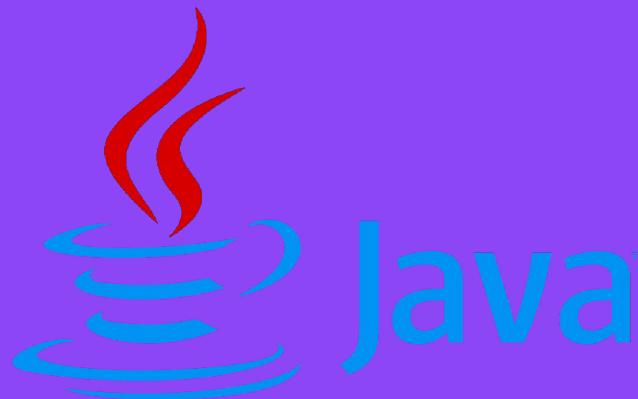


Специализация:  
ооп





# Специализация: ООП



## На прошлом уроке

- 📌 Данные
- 📌 Примитивные типы данных
- 📌 Ссылочные типы данных, массивы
- 📌 Базовый функционал языка
- 📌 Функции





## Что будет на уроке сегодня

- 📌 Классы;
- 📌 Объекты;
- 📌 Статика;
- 📌 Стек;
- 📌 Куча;
- 📌 Сборщик мусора;
- 📌 Конструкторы;
- 📌 Инкапсуляция;
- 📌 Наследование;
- 📌 Полиморфизм.





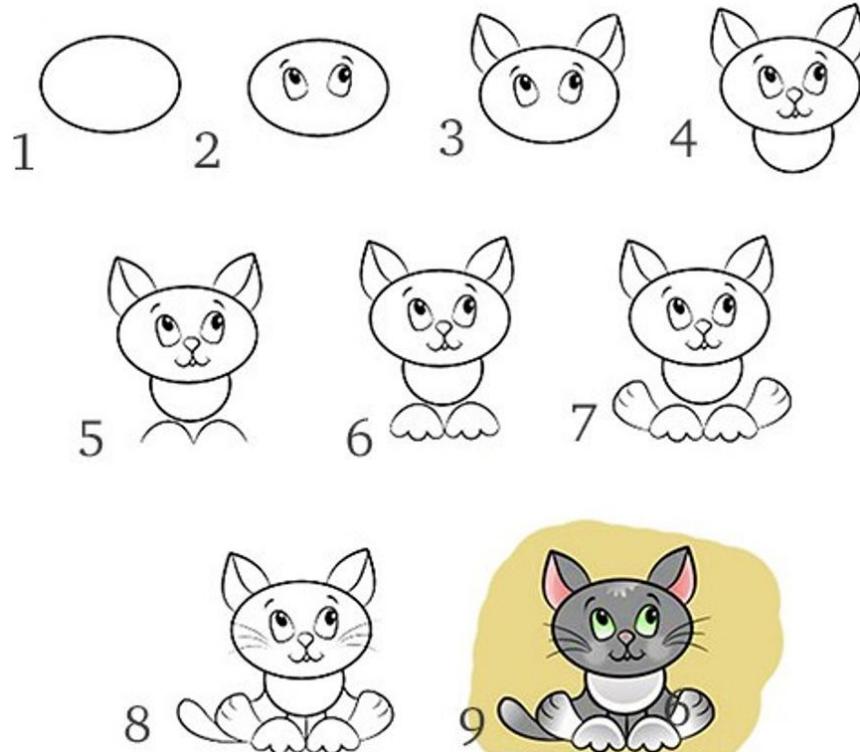
# Класс



## Понятие класса

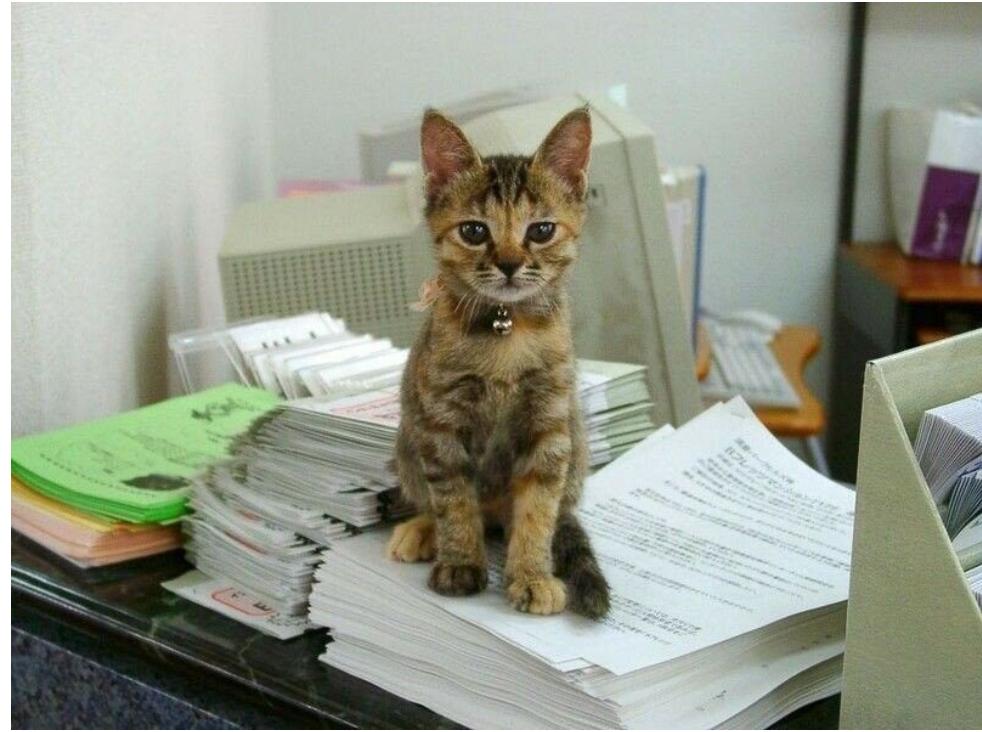
**Класс** - это новый тип данных;

**Класс** - это шаблон.





Например, класс





# Например, класс

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
HIJKLMNOP  
OPQRSTUVWXYZ  
VWXYZ Ø1  
23456789



Общество с ограниченной ответственностью «Весна»  
(ООО «Весна»)

г. Санкт-Петербург

«16» мая 2019 г.

ПРИКАЗ № 2-пр

Об утверждении штатного расписания

ПРИКАЗЫВАЮ:

- Утвердить штатное расписание № 1 от «13» мая 2019 г. с количеством 31 штатной единицы и месячным фондом заработной платы в размере 1 000 000 (одного миллиона) рублей.
- Ввести штатное расписание № 1 от «13» мая 2019 г. в действие с «03» июня 2019 г.
- Контроль за исполнением настоящего приказа возложить на руководителя кадровой службы И.И. Иванова

Приложение №1: штатное расписание № 1 от «13» мая 2019 г.

Генеральный директор

Субботин  
(подпись)

Субботин О.В.  
(расшифровка подписи)

С приказом ознакомлен:

Руководитель кадровой службы

Иванов  
(подпись)

Иванов И.И.  
(расшифровка подписи)

«17» мая 2019 г.

## ПРИКАЗ

«\_\_\_» 20\_\_ г.

№ 0190-П

## О назначении ответственного лица за осуществление строительного контроля

В целях организации строительного контроля на объекте «Строительство завода»

ПРИКАЗЫВАЮ:

- Назначить ответственным лицом за осуществление строительного контроля – ведущего инженера строительного контроля В.А. Крюкова.
- При осуществлении строительного контроля ответственному лицу руководствоваться проектной документацией, проектом производства работ, СНиП, СП, РД, ВСН и другой действующей нормативной документацией в строительстве.
- На время отсутствия Крюкова В.А. (отпуск, командировка, болезнь и т.д.), его обязанности возложить на заместителя главного инженера Иванова И.А.
- Контроль исполнения настоящего приказа оставляю за собой.

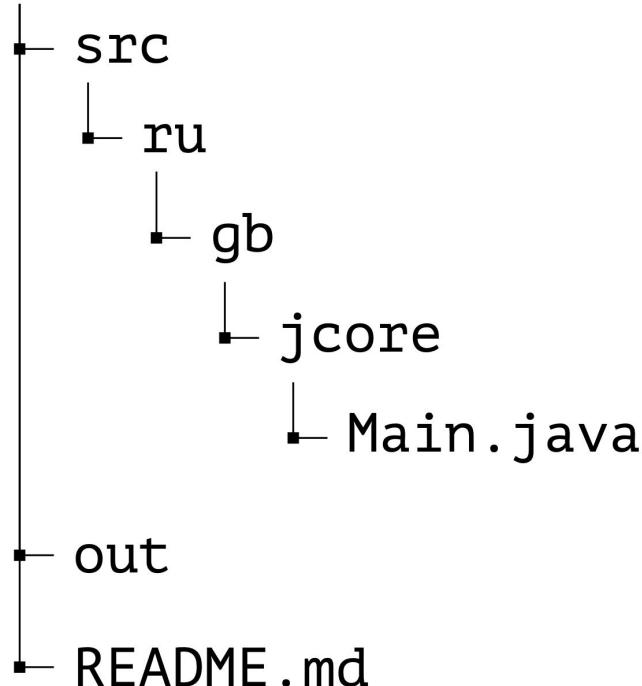
Главный инженер

А.И. Марков



## Структура проекта

Sample





## Структура

### New Java Class

C Cat

C Class

I Interface

R Record

E Enum

@ Annotation



## Поля класса

```
Main.java x Cat.java x
1 package ru.gb.jcore;
2
3 public class Cat {
4     String name;
5     String color;
6     int age;
7 }
```





## Создание экземпляра

```
6  
7     new Cat();  
8
```

```
7     Cat cat1;  
8     cat1 = new Cat();  
9
```





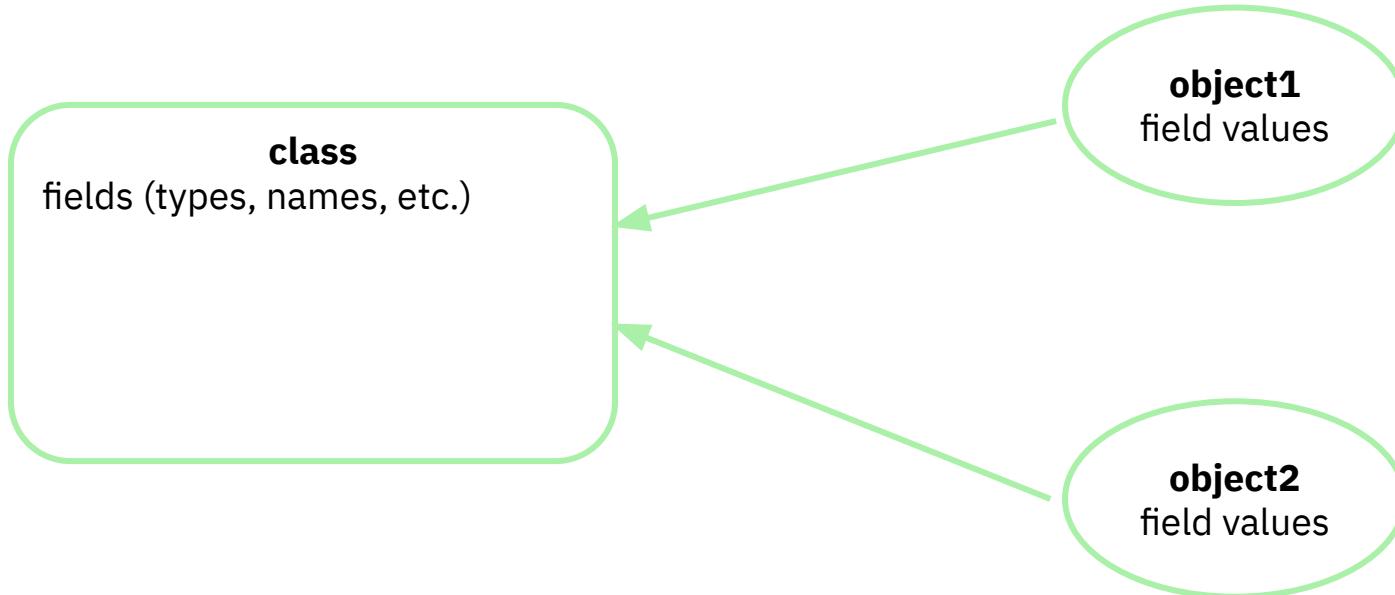
## Несколько объектов

```
4 ► ⌂ public static void main(String[] args) {  
5     Cat cat1 = new Cat(); Cat cat2 = new Cat();  
6  
7     cat1.name = "Барсик"; cat1.color = "Белый"; cat1.age = 4;  
8     cat2.name = "Мурзик"; cat2.color = "Черный"; cat2.age = 6;  
9  
10    System.out.println("Кот 1 имя: " + cat1.name +  
11        " цвет: " + cat1.color + "возраст: " + cat1.age);  
12  
13    System.out.println("Кот 2 имя: " + cat2.name +  
14        " цвет: " + cat2.color + "возраст: " + cat2.age);  
15 }  
16 }
```





## Отношения класса и объектов (поля)





# Объект



## Варианты создания экземпляра

```
6  
7     Cat cat1;  
8     cat1 = new Cat();  
9     Cat cat2 = new Cat();  
10    cat1.color = "White";  
11
```





## Оператор new

```
[квалификаторы] ИмяКласса имяПеременной = new ИмяКласса();
```



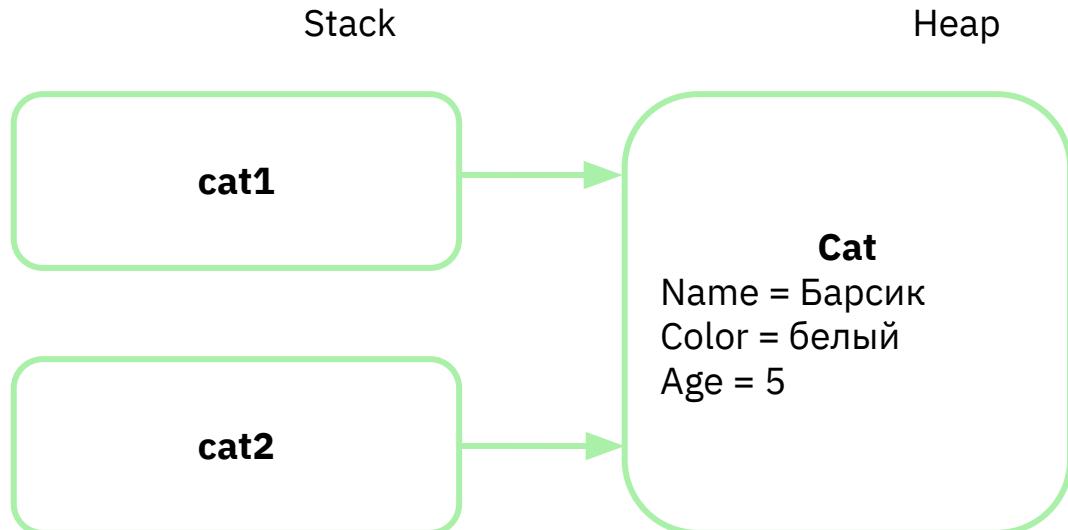


## Две ссылки на один объект

```
5     Cat cat1 = new Cat();
6     Cat cat2 = cat1;
7
8     cat1.name = "Barsik";
9     cat1.color = "White";
10    cat1.age = 4;
11
12    cat2.name = "Murzik";
13    cat2.color = "Black";
14    cat2.age = 6;
15
16    System.out.println("Cat1 named: " + cat1.name + " is " + cat1.color + " has age: " + cat1.age);
17    System.out.println("Cat2 named: " + cat2.name + " is " + cat2.color + " has age: " + cat2.age);
```



## Две ссылки на один объект





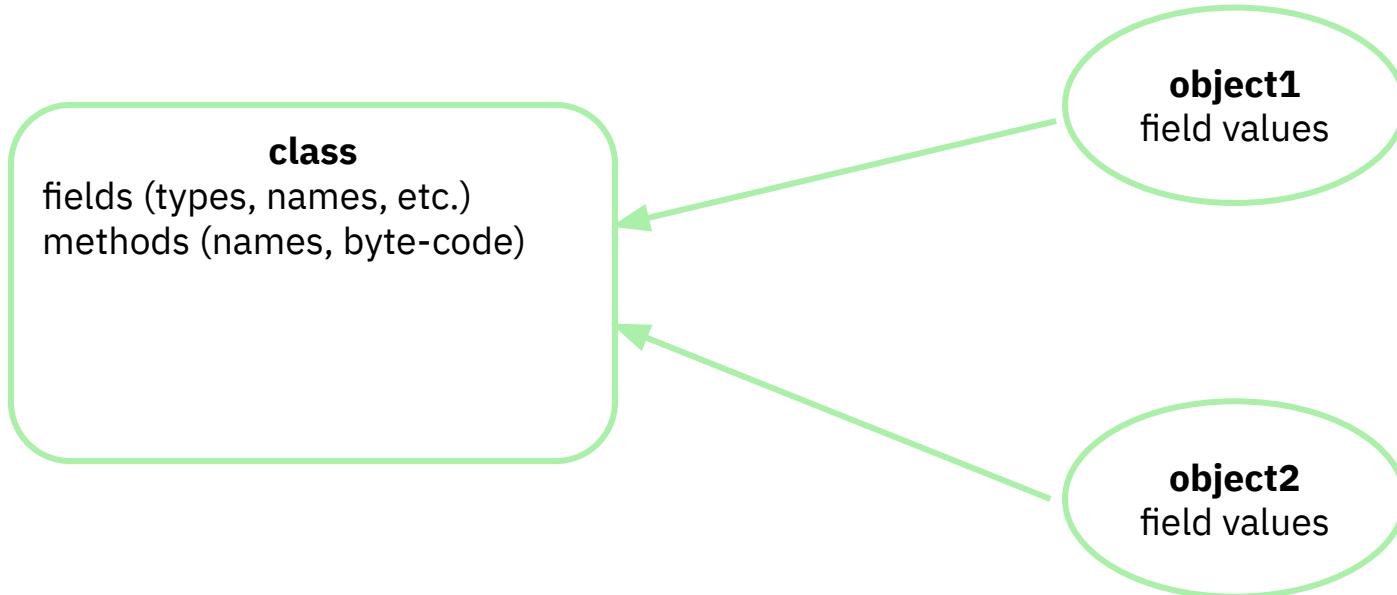
## Метод

**Метод** - это функция, принадлежащая классу





## Отношение класса и объектов (методы)





# Методы

```
>Main.java x Cat.java x
1 package ru.gb.jcore;
2
3 public class Cat {
4     String name;
5     String color;
6     int age;
7
8     void voice() {
9         System.out.println(name + " meows");
10    }
11
12    void jump() {
13        if (this.age < 5)
14            System.out.println(name + " jumps");
15    }
16 }
```



## Вызов методов

```
19         cat1.jump();
20         cat1.voice();
21         cat2.jump();
22         cat2.voice();
23
Run: Main ×
▶  /Library/Java/JavaVirtualMachines/liberica
  Cat1 named: Barsik is White has age: 4
  Cat2 named: Murzik is Black has age: 6
  Barsik jumps
  Barsik meows
  Murzik meows
```

Теперь ваша очередь!



## Ответьте на вопросы сообщением в чат

### Вопросы:

1. Что такое класс?
2. Что такое поле класса?
3. На какие три этапа делится создание объекта?





# Статика



## Ключевое слово static

static - (от греч. στατός, «неподвижный») — раздел механики, в котором изучаются условия равновесия механических систем под действием приложенных к ним сил и возникших моментов.



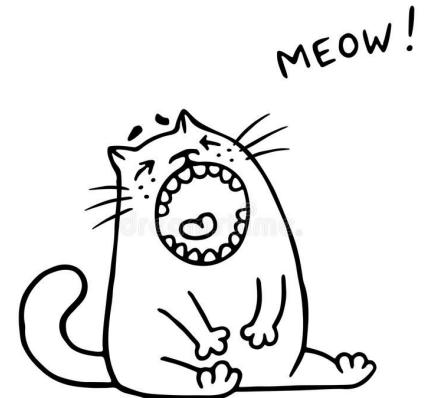
## Статические контексты

- 📌 Статические методы;
- 📌 Статические переменные;
- 📌 Статические вложенные классы;
- 📌 Статические блоки.



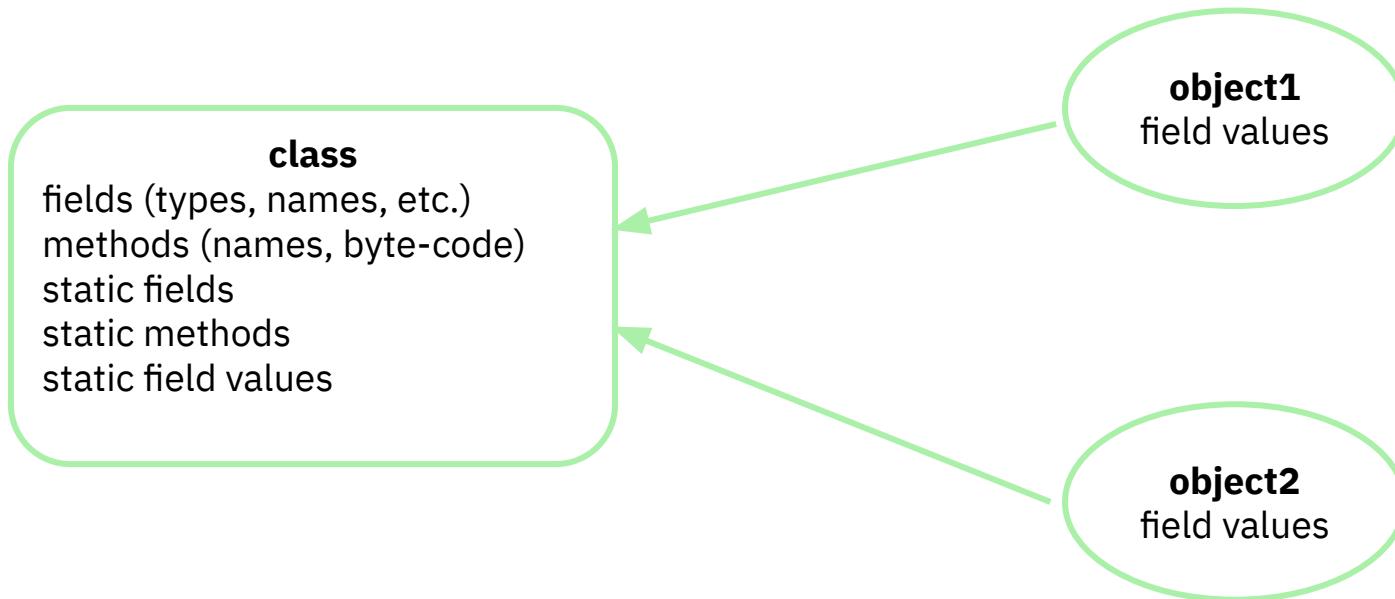
## Статические методы

**Статические методы** - это методы класса, а не объектов





## Отношение класса и объектов





## Статическое поле

```
24     System.out.println("A cat has " + Cat.pawsCount + " paws");
25 }
26 }
27
Run: Main
▶ A cat has 4 paws
```

```
3 public class Cat {
4     static int pawsCount = 4;
5
6     String name;
7     String color;
8     int age;
```



## Статическое поле (семантическая ошибка)

```
3 public class Cat {  
4     static int pawsCount = 4;  
5  
6     static String name;  
7     String color;  
8     int age;
```



## Статическое поле (признак семантической ошибки)

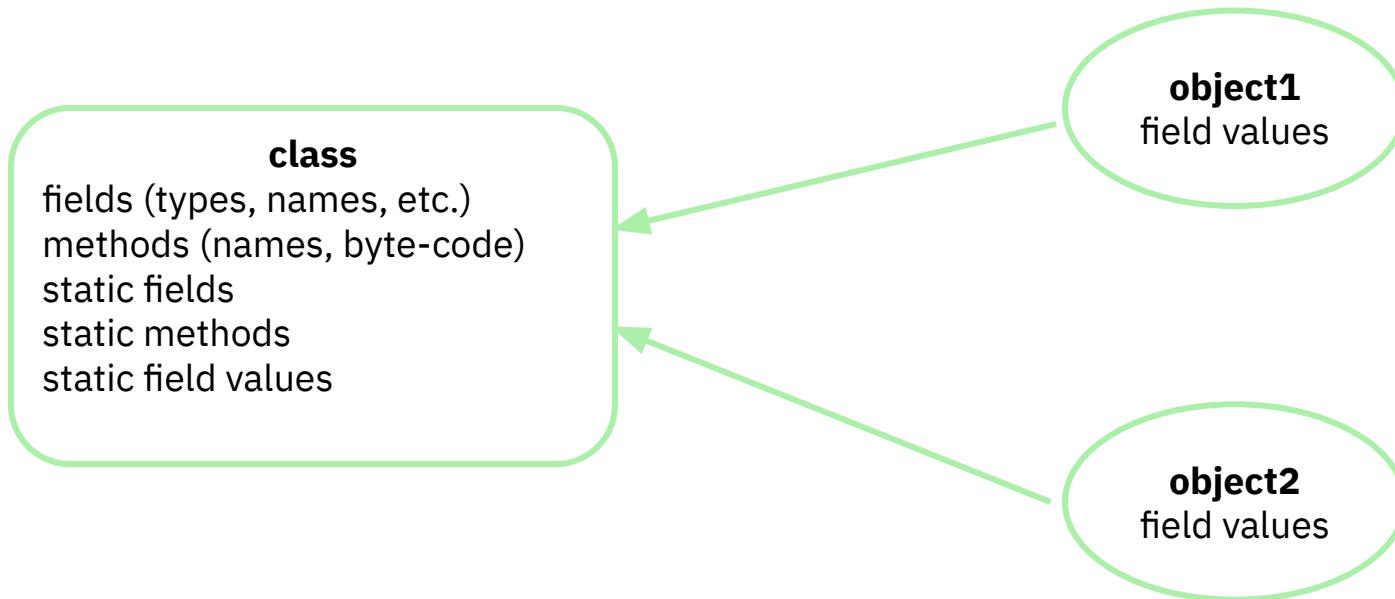
```
6     Cat cat1 = new Cat();
7     Cat cat2 = new Cat();
8
9     cat1.name = "Barsik";
10    cat1.color = "White";
11    cat1.age = 4;
12
13    cat2.name = "Murzik";
14    cat2.color = "Black";
15    cat2.age = 6;
16
17    System.out.println("Cat1 named: " + cat1.name + " is " + cat1.color + " has age:");
18    System.out.println("Cat2 named: " + cat2.name + " is " + cat2.color + " has age:");

Run: Main ×
```

```
/Library/Java/JavaVirtualMachines/liberica-jc
Cat1 named: Murzik is White has age: 4
Cat2 named: Murzik is Black has age: 6
```



## Отношение класса и объектов



Теперь ваша очередь!



## Ответьте на вопросы сообщением в чат

### Вопросы:

1. Какое свойство добавляет ключевое слово static полю или методу?
  - a. неизменяемость;
  - b. принадлежность классу;
  - c. принадлежность приложению.
2. Может ли статический метод получить доступ к полям объекта?
  - a. не может;
  - b. может только к константным;
  - c. может только к неинициализированным.

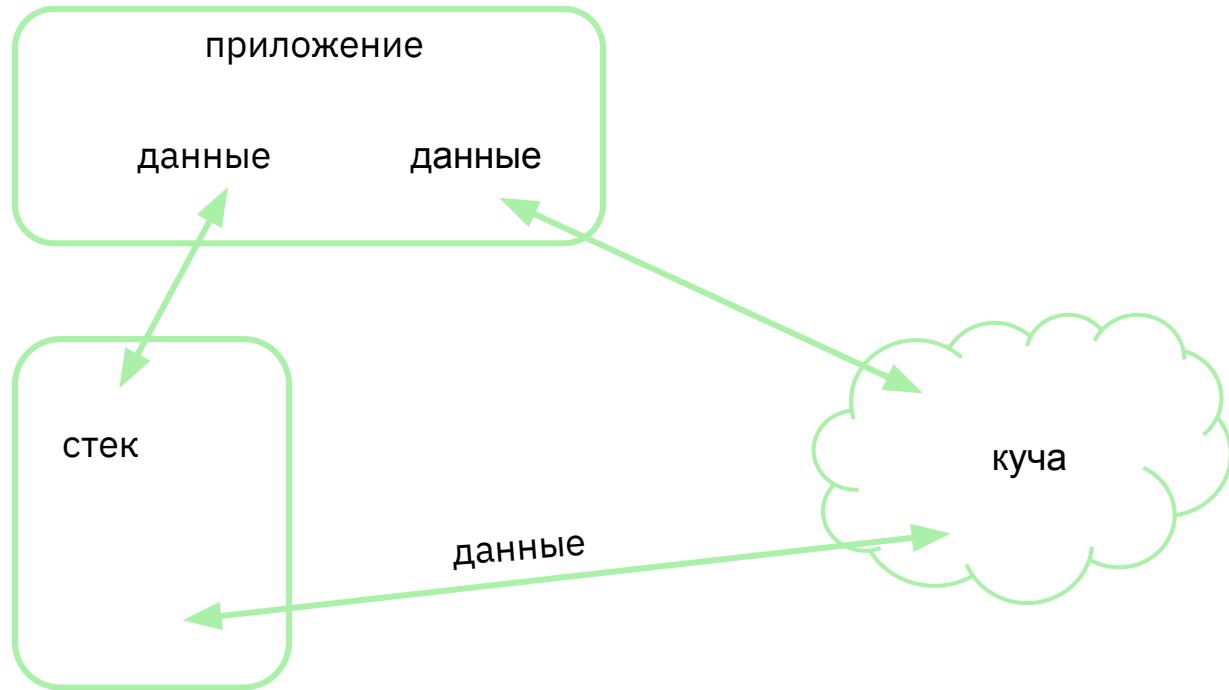




# Введение в управление памятью



## Память





## Стековая память



**LIFO**

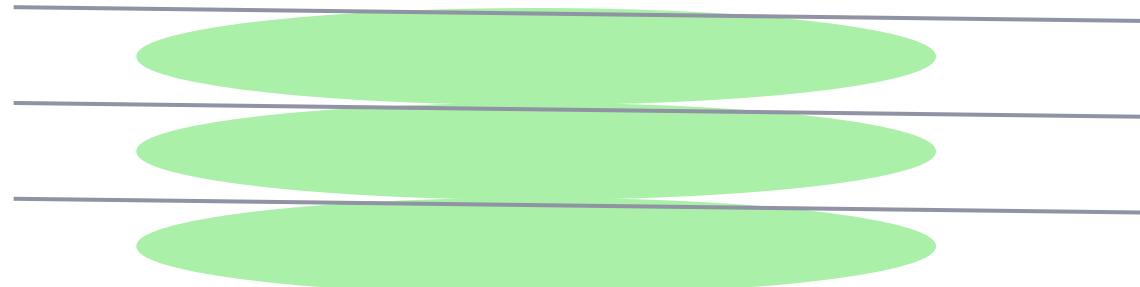
Last-In, First-Out

Последний зашёл, Первый вышел



## Стековая память

контекст метода



стек



# Многопоточность

поток 1



поток 2



поток 3



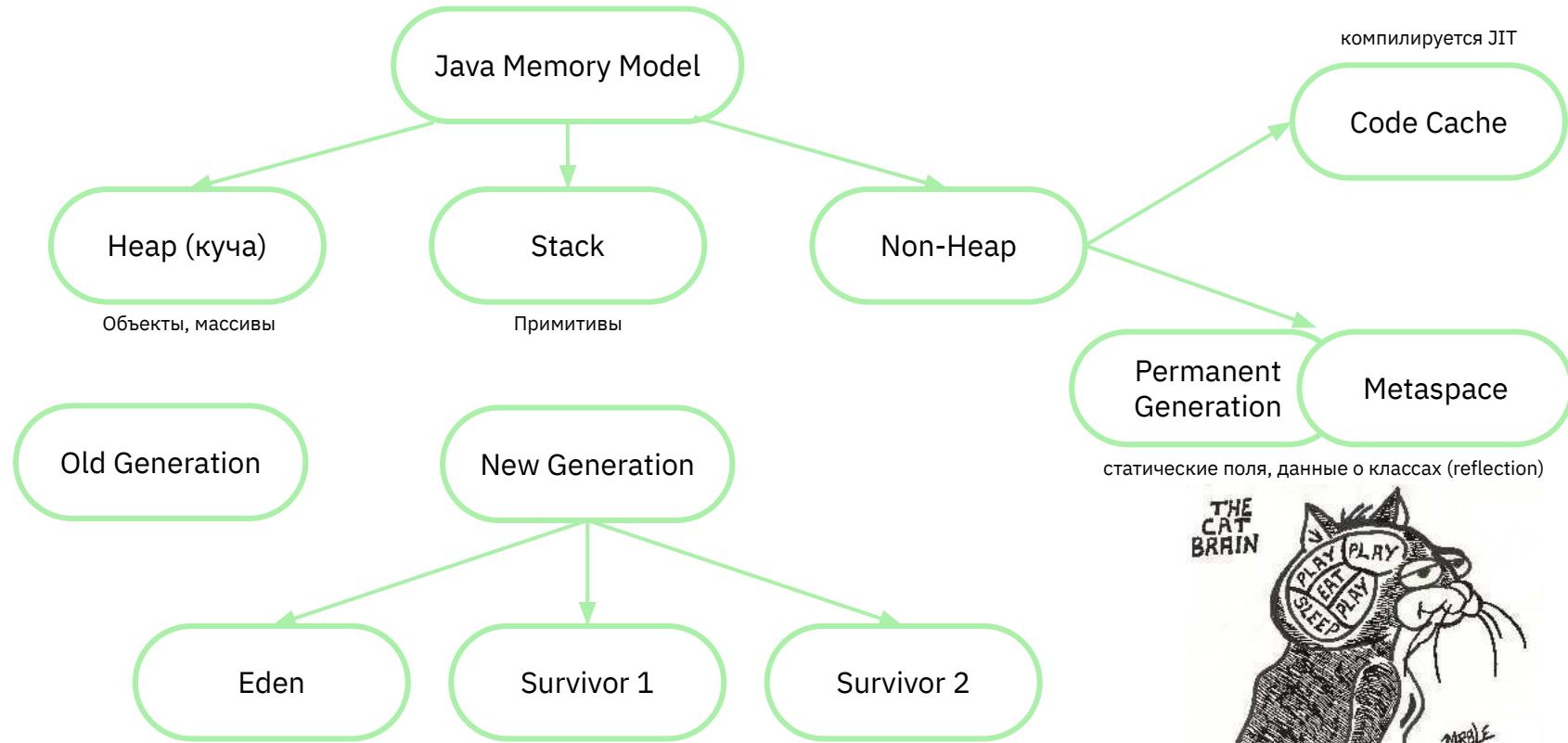


## Особенности стека

- 📌 Заполняется и освобождается по мере вызова и завершения новых методов;
- 📌 Переменные на стеке существуют до тех пор, пока выполняется метод в котором они были созданы;
- 📌 Если память стека будет заполнена, Java бросит исключение `java.lang.StackOverflowError`;
- 📌 Доступ к этой области памяти осуществляется быстрее, чем к куче;
- 📌 Является потокобезопасным, поскольку для каждого потока создаётся свой отдельный стек.

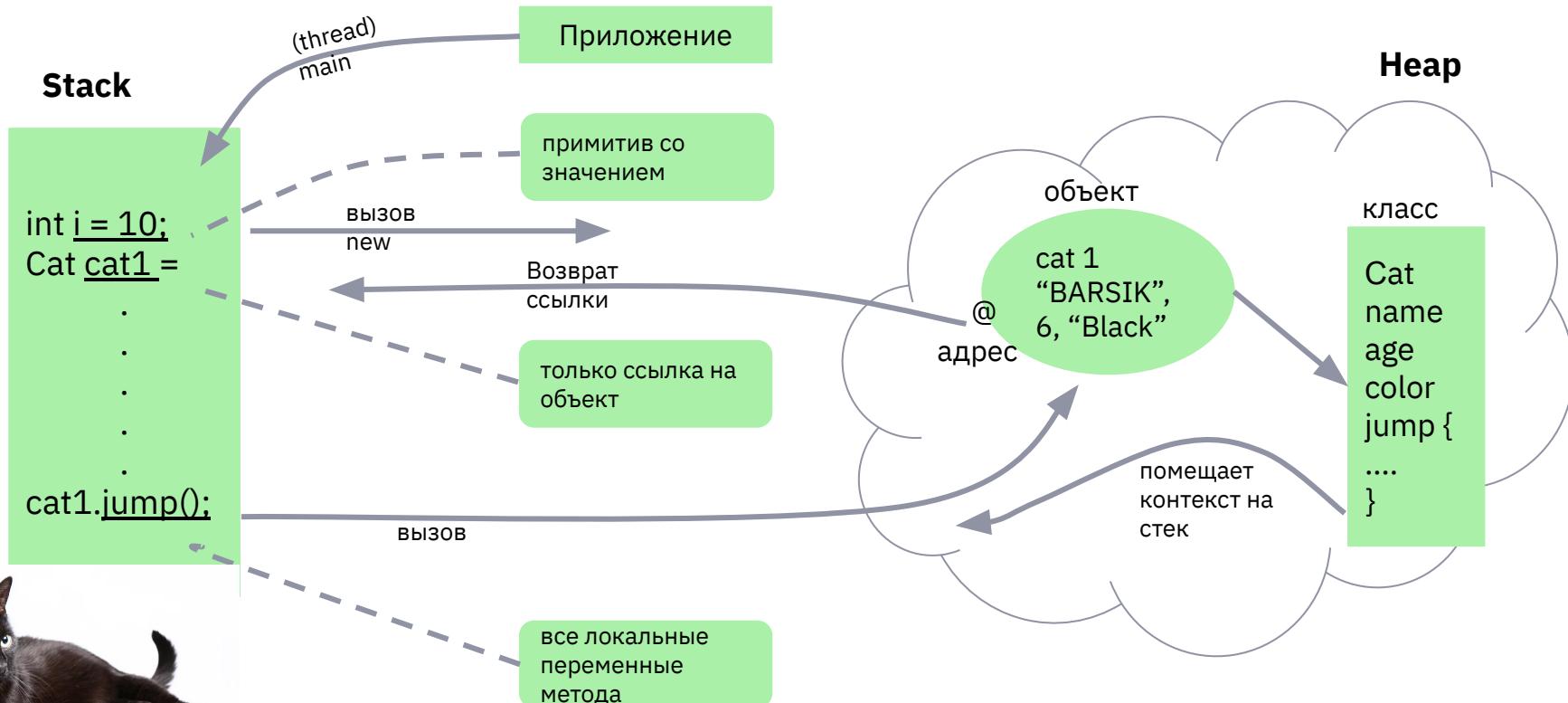


# Устройство памяти





## Класс и объект в памяти





## Поколения кучи

- 📌 Young Generation
- 📌 Old (Tenured) Generation
- 📌 Permanent Generation



## Особенности кучи

- 📌 В общем случае, размеры кучи на порядок больше размеров стека;
- 📌 Когда эта область памяти полностью заполняется, Java бросает `java.lang.OutOfMemoryError`;
- 📌 Доступ к ней медленнее, чем к стеку;
- 📌 Эта память, в отличие от стека, автоматически не освобождается. Для сбора неиспользуемых объектов используется сборщик мусора;
- 📌 В отличие от стека, который создаётся для каждого потока свой, куча не является потокобезопасной, поскольку для всех одна, и ее необходимо контролировать, правильно синхронизируя код.



# Сборщик мусора



## Процесс сборки мусора

- 📌 Этот процесс запускается автоматически Java, и Java решает, запускать или нет этот процесс;
- 📌 На самом деле это дорогостоящий процесс. При запуске сборщика мусора все потоки в вашем приложении приостанавливаются (в зависимости от типа GC);
- 📌 На самом деле это гораздо более сложный процесс, чем просто сбор мусора и освобождение памяти.



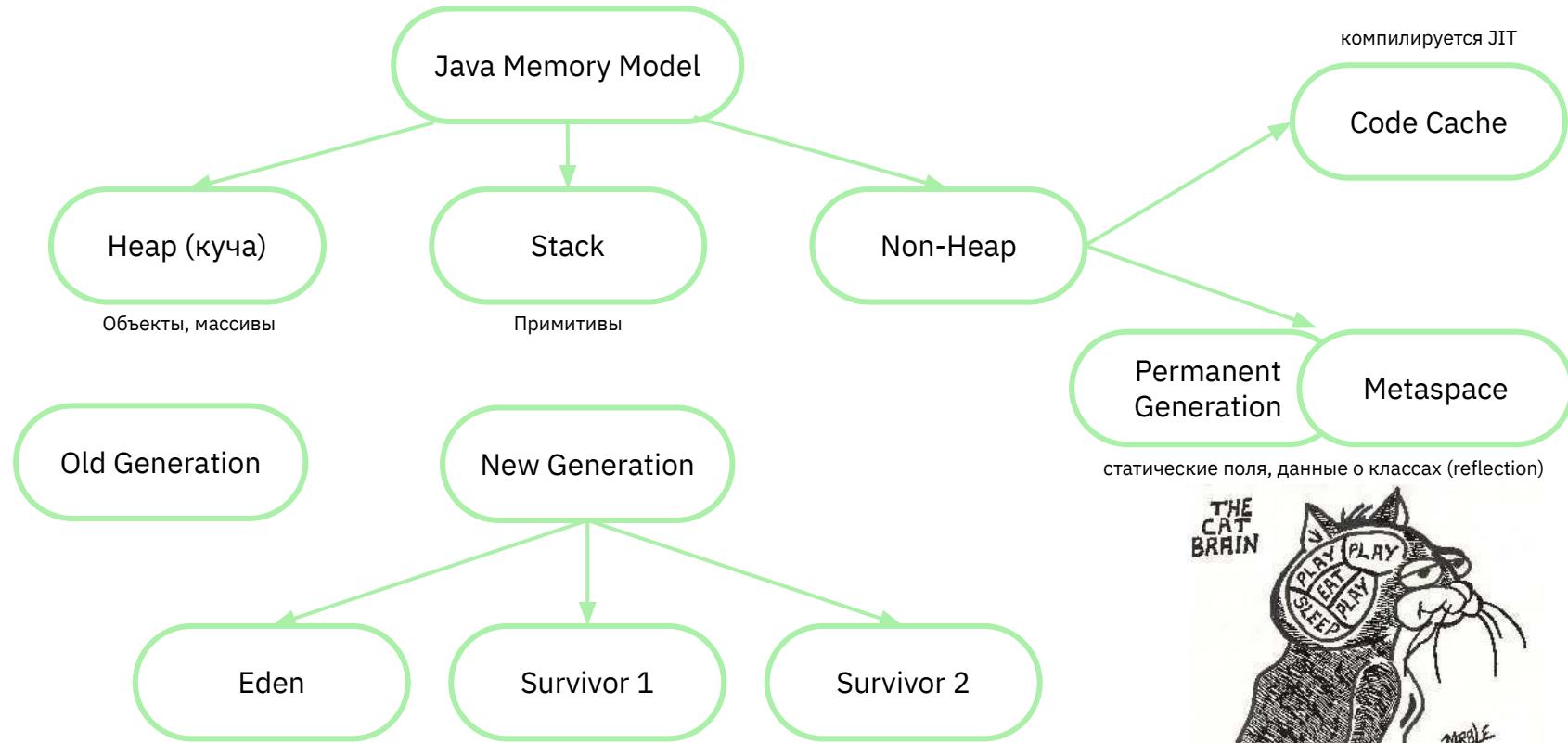
# Системный сборщик мусора

The screenshot shows a Java code editor interface with a dark theme. On the left, there's a 'Project' sidebar with a single folder icon. The main area displays the contents of the `System.java` file. The code is annotated with line numbers from 1755 to 1775. The code itself is as follows:

```
1755 /**
1756 * Runs the garbage collector.
1757 *
1758 * Calling the {@code gc} method suggests that the Java Virtual
1759 * Machine expend effort toward recycling unused objects in order to
1760 * make the memory they currently occupy available for quick reuse.
1761 * When control returns from the method call, the Java Virtual
1762 * Machine has made a best effort to reclaim space from all discarded
1763 * objects.
1764 *
1765 * The call {@code System.gc()} is effectively equivalent to the
1766 * call:
1767 * <blockquote><pre>
1768 * Runtime.getRuntime().gc()
1769 * </pre></blockquote>
1770 *
1771 * @see java.lang.Runtime#gc()
1772 */
1773 public static void gc() {
1774     Runtime.getRuntime().gc();
1775 }
```

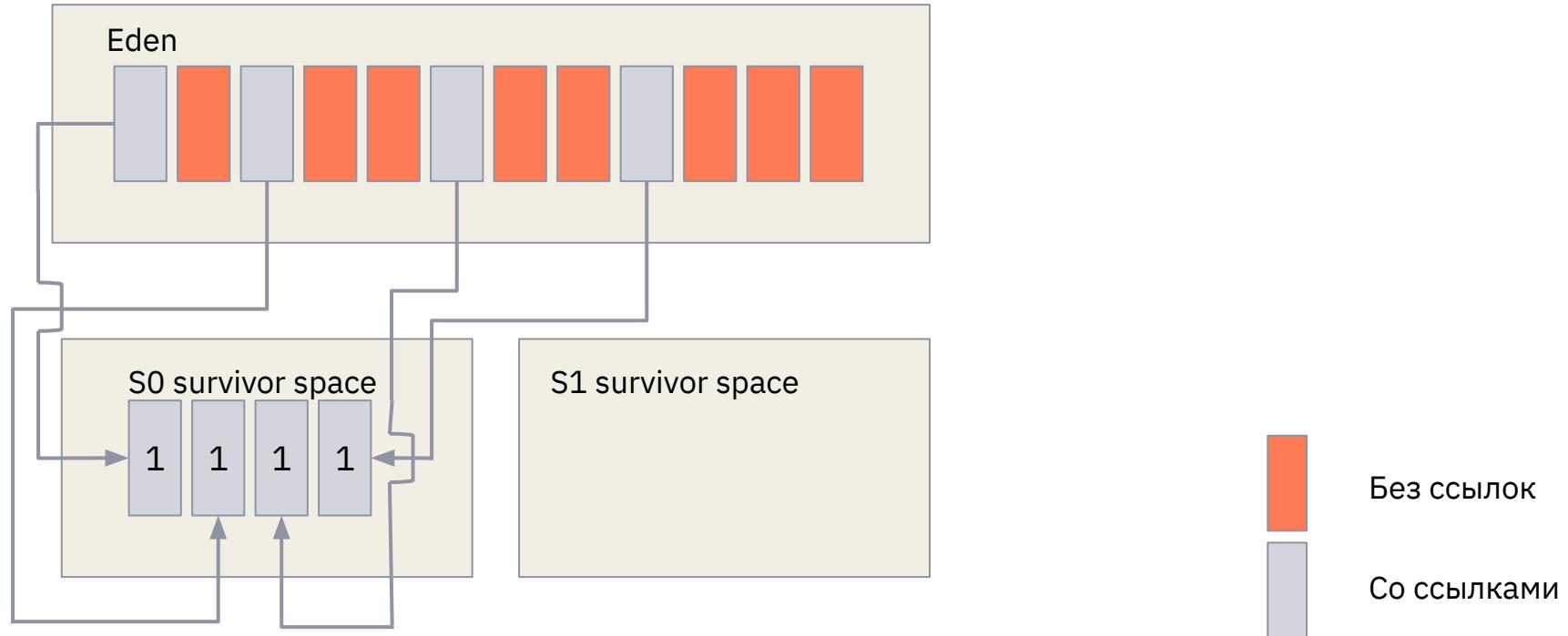


# Устройство памяти



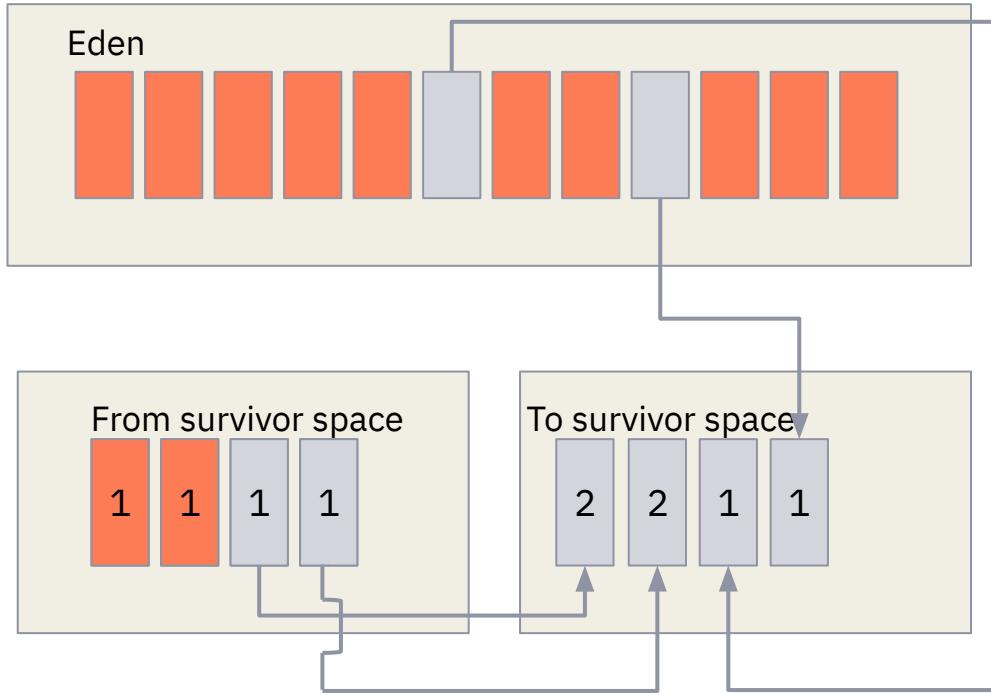


## Молодое поколение





## Выжившее поколение

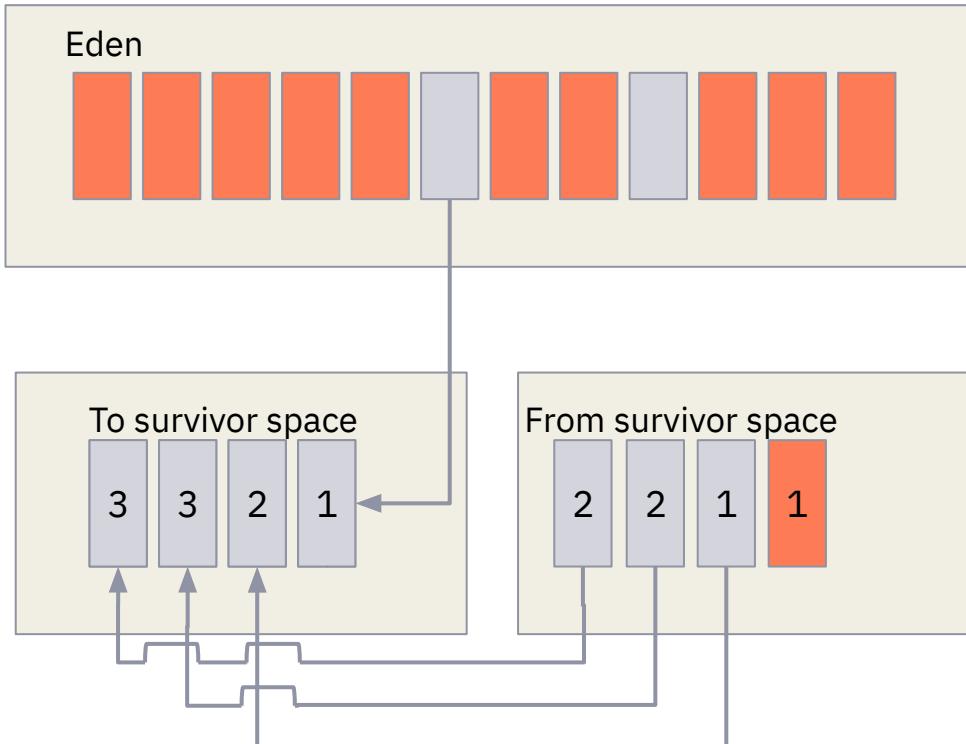


Без ссылок

Со ссылками



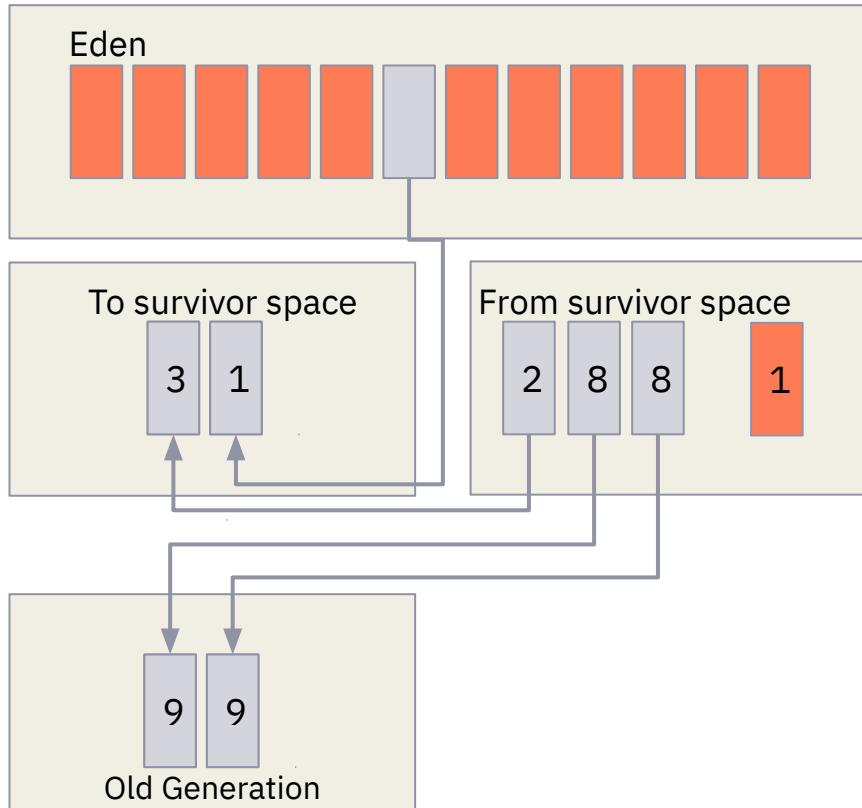
## Третье поколение



Без ссылок  
Со ссылками



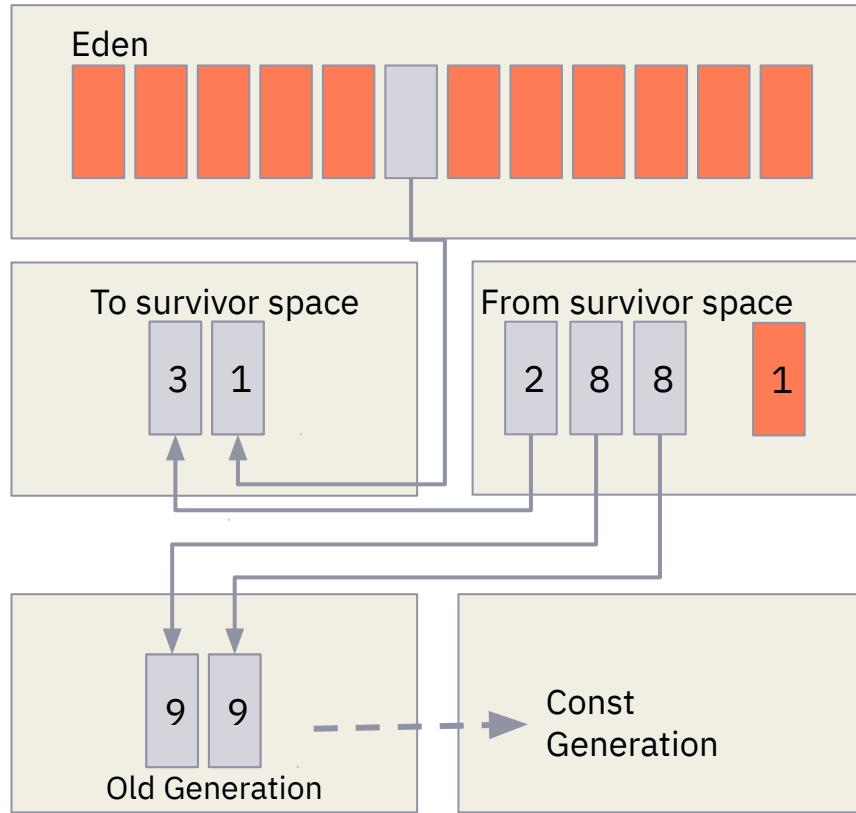
## Старое поколение



Без ссылок  
Со ссылками



## Постоянное поколение



Без ссылок  
Со ссылками



## Сборщик мусора. Реализации

- 📌 Последовательный;
- 📌 Параллельный;
- 📌 CMS;
- 📌 G1;
- 📌 ZGC



## Управление памятью. Итоги

- 📌 куча доступна везде, объекты доступны отовсюду;
- 📌 все объекты хранятся в куче, все локальные переменные хранятся на стеке;
- 📌 стек недолговечен;
- 📌 и стек и куча могут быть переполнены;
- 📌 куча много больше стека, но стек гораздо быстрее.

Теперь ваша очередь!



## Ответьте на вопросы сообщением в чат

### Вопросы:

1. По какому принципу работает стек?
2. Что быстрее, стек или куча?
3. Что больше, стек или куча?





# Конструктор



## Похожие объекты

```
4 ► ⌂ public static void main(String[] args) {  
5     Cat cat1 = new Cat(); Cat cat2 = new Cat();  
6  
7     cat1.name = "Барсик"; cat1.color = "Белый"; cat1.age = 4;  
8     cat2.name = "Мурзик"; cat2.color = "Черный"; cat2.age = 6;  
9  
10    System.out.println("Кот 1 имя: " + cat1.name +  
11        " цвет: " + cat1.color + "возраст: " + cat1.age);  
12  
13    System.out.println("Кот 2 имя: " + cat2.name +  
14        " цвет: " + cat2.color + "возраст: " + cat2.age);  
15 }  
16 }
```





## Очень плохой конструктор

The screenshot shows a Java code editor with two files open: Main.java and Cat.java. The Cat.java file is the active tab. The code defines a class Cat with static and instance variables, and a constructor that prints a message and initializes the variables.

```
1 package ru.gb.jcore;
2
3 public class Cat {
4     static int pawsCount = 4;
5
6     String name;
7     String color;
8     int age;
9
10    Cat() {
11        System.out.println("constructor is constructing...");
12        name = "Barsik";
13        color = "White";
14        age = 2;
15    }
16}
```



# Параметризованный конструктор

The screenshot shows a Java code editor interface with two tabs: 'Main.java' and 'Cat.java'. The 'Cat.java' tab is active, displaying the following code:

```
1 package ru.gb.jcore;
2
3 public class Cat {
4     static int pawsCount = 4;
5
6     String name;
7     String color;
8     int age;
9
10    Cat(String n, String c, int a) {
11        System.out.println("constructor is constructing...");
12        name = n;
13        color = c;
14        age = a;
15    }
16}
```

The code defines a class 'Cat' with static variable 'pawsCount' set to 4. It has instance variables 'name', 'color', and 'age'. A constructor is defined that takes three parameters: 'n', 'c', and 'a'. Inside the constructor, it prints a message to the console and initializes the instance variables. The code editor interface includes a 'Project' sidebar on the left.



## Вызов параметризированного конструктора

The screenshot shows an IDE interface with a dark theme. At the top, there are tabs for 'Project' and two files: 'Main.java' and 'Cat.java'. The code editor displays the following Java code:

```
1 package ru.gb.jcore;
2
3 public class Main{
4     public static void main(String[] args) {
5         System.gc();
6
7         Cat cat1 = new Cat( n: "Barsik", c: "White", a: 4);
8         Cat cat2 = new Cat( n: "Murzik", c: "Black", a: 6);
9
10        System.out.println("Cat1 named: " + cat1.name + " is " +
11        System.out.println("Cat2 named: " + cat2.name + " is " +
12
```

The 'Run' tab at the bottom is selected, showing the output of the program:

```
constructor is constructing...
constructor is constructing...
```



## Перегрузка конструкторов

```
10     public Cat(String n, String c, int a) {  
11         System.out.println("constructor is constructing...");  
12         name = n;  
13         color = c;  
14         age = a;  
15     }  
16  
17     public Cat(String n) {  
18         name = n;  
19         color = "unknown";  
20         age = 1;  
21     }
```



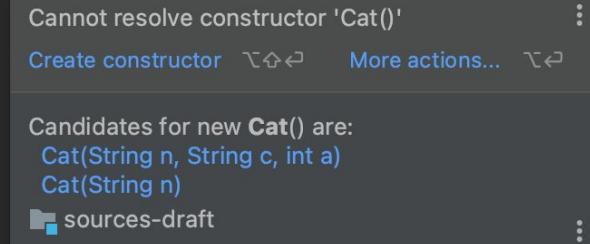
## Вызов перегруженных конструкторов

```
7     Cat cat1 = new Cat("Barsik", "White", 4);  
8     Cat cat2 = new Cat("Murzik");
```



## Конструктор по умолчанию

```
7     Cat cat1 = new Cat("Barsik", "White", 4);  
8     Cat cat2 = new Cat("Murzik");  
9     Cat cat3 = new Cat();
```





## Ключевое слово `this`

**this** - это указатель на текущий экземпляр класса.

нужен когда совпадают имена полей класса и параметров конструктора, а также, чтобы вызвать конструктор из конструктора.



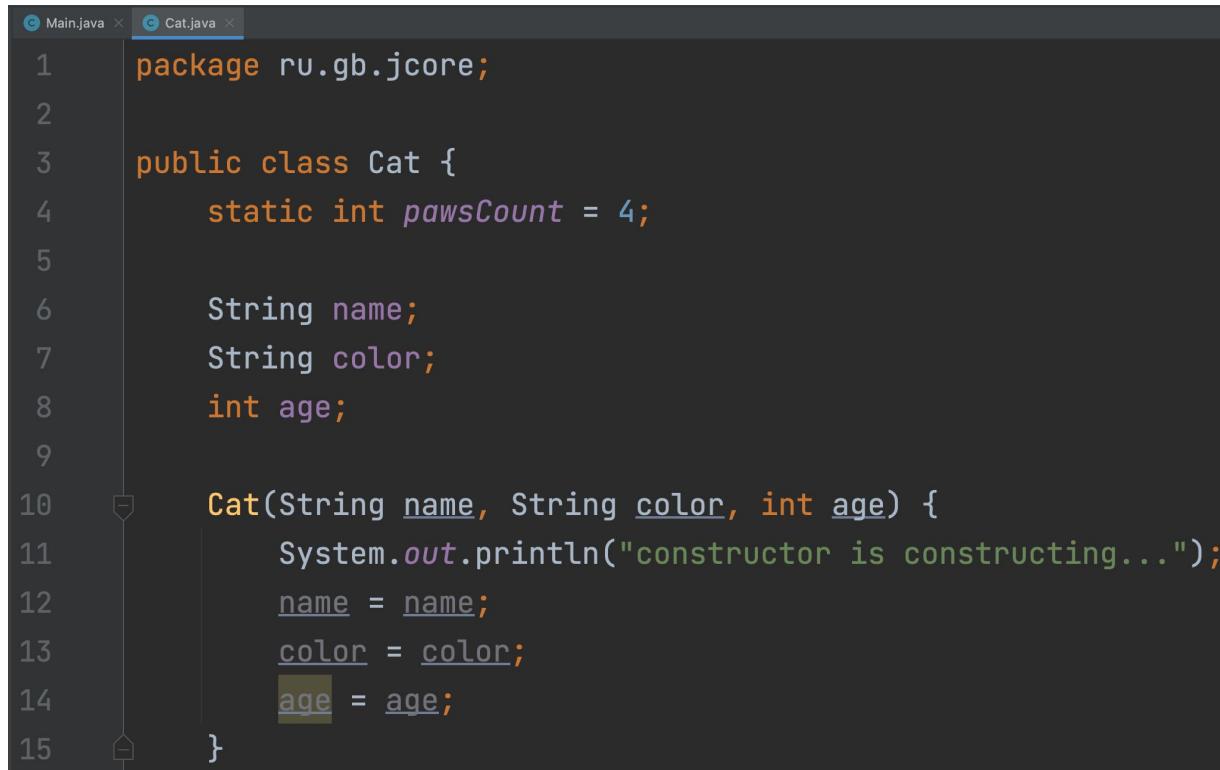
# Параметризованный конструктор

The screenshot shows a Java code editor interface. On the left, there's a 'Project' sidebar with a single folder icon. The main area has two tabs: 'Main.java' and 'Cat.java'. The 'Cat.java' tab is active, showing the following code:

```
1 package ru.gb.jcore;
2
3 public class Cat {
4     static int pawsCount = 4;
5
6     String name;
7     String color;
8     int age;
9
10    Cat(String n, String c, int a) {
11        System.out.println("constructor is constructing...");
12        name = n;
13        color = c;
14        age = a;
15    }
16}
```



## Сломанный параметризованный конструктор



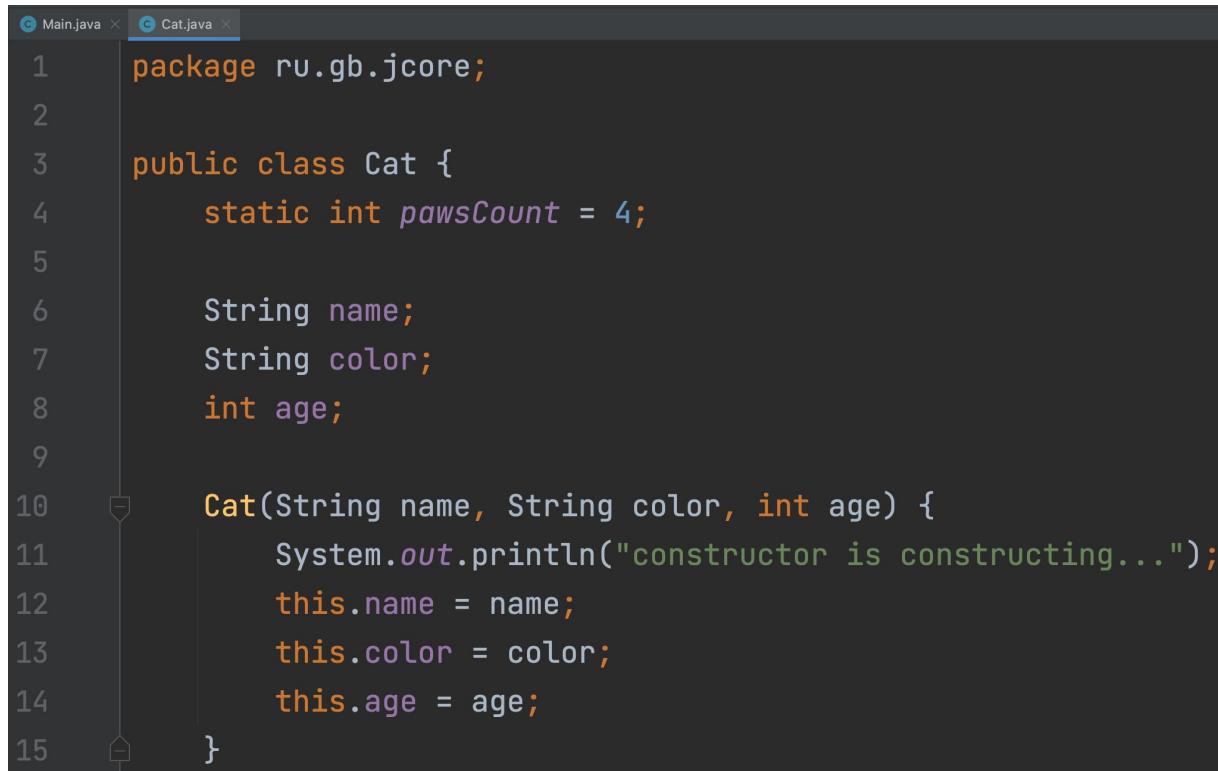
The screenshot shows a Java code editor with two tabs: "Main.java" and "Cat.java". The "Cat.java" tab is active, displaying the following code:

```
1 package ru.gb.jcore;
2
3 public class Cat {
4     static int pawsCount = 4;
5
6     String name;
7     String color;
8     int age;
9
10    Cat(String name, String color, int age) {
11        System.out.println("constructor is constructing...");
12        name = name;
13        color = color;
14        age = age;
15    }
16}
```

The code defines a class named "Cat" with static variable "pawsCount" set to 4. It has instance variables "name", "color", and "age". A constructor is defined that takes three parameters: "name", "color", and "age". Inside the constructor, each parameter is assigned back to itself. The "age" assignment is highlighted with a yellow background.



## Правильный конструктор



The screenshot shows a Java code editor with two tabs: "Main.java" and "Cat.java". The "Cat.java" tab is active, displaying the following code:

```
1 package ru.gb.jcore;
2
3 public class Cat {
4     static int pawsCount = 4;
5
6     String name;
7     String color;
8     int age;
9
10    Cat(String name, String color, int age) {
11        System.out.println("constructor is constructing...");
12        this.name = name;
13        this.color = color;
14        this.age = age;
15    }
}
```



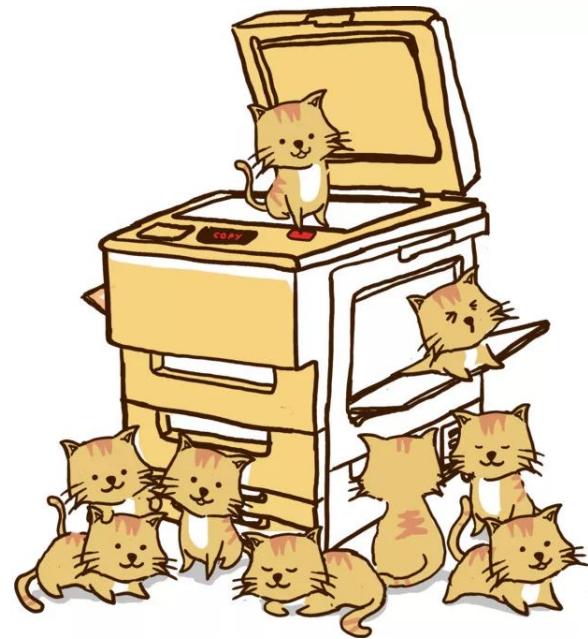
## Вызов одного конструктора из другого

```
10     public Cat(String name, String color, int age) {  
11         this(name, color);  
12         System.out.println("constructor is constructing...");  
13         this.age = age;  
14     }  
15  
16     public Cat(String name, String color) {  
17         this.name = name;  
18         this.color = color;  
19     }
```



## Конструктор копирования

```
public Cat (Cat cat) {  
    this(cat.name, cat.color, cat.age);  
}
```



Теперь ваша очередь!



## Ответьте на вопросы сообщением в чат

### Вопросы:

1. Для инициализации нового объекта с абсолютно идентичными значениями свойств переданного объекта используется
  - a. пустой конструктор;
  - b. конструктор по-умолчанию;
  - c. конструктор копирования.
2. Что означает ключевое слово this?





# Инкапсуляция



## Инкапсуляция

**Инкапсуляция** (англ. *encapsulation*, от лат. *in capsula*) — в информатике, процесс разделения элементов абстракций, определяющих ее структуру (данные) и поведение (методы); инкапсуляция предназначена для изоляции контрактных обязательств абстракции (протокол/интерфейс) от их реализации. На практике это означает, что класс должен состоять из двух частей: интерфейса и реализации.



## Инкапсуляция как чёрный ящик



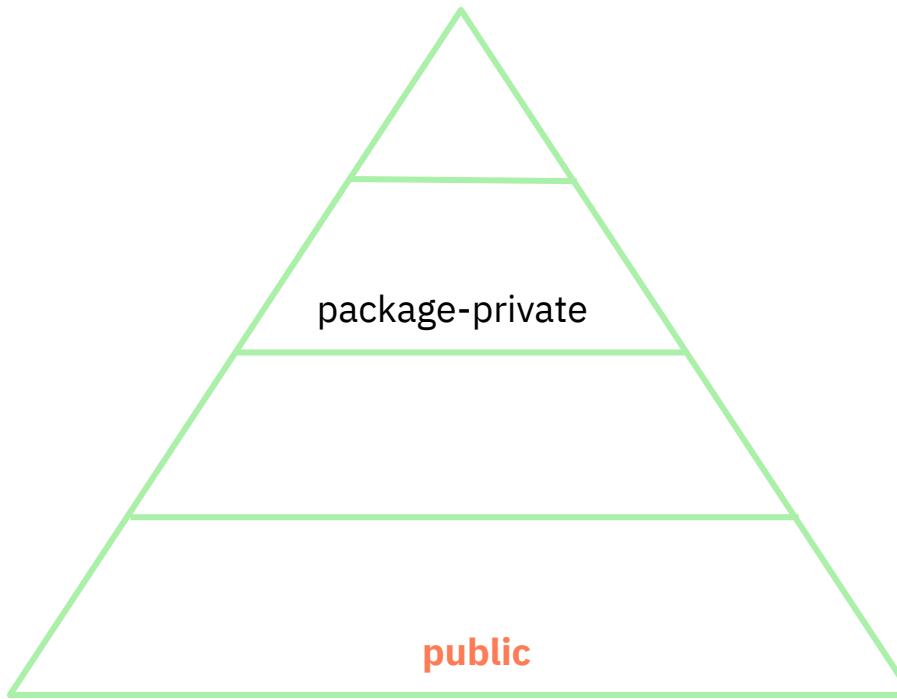


## Инкапсуляция как чёрный ящик



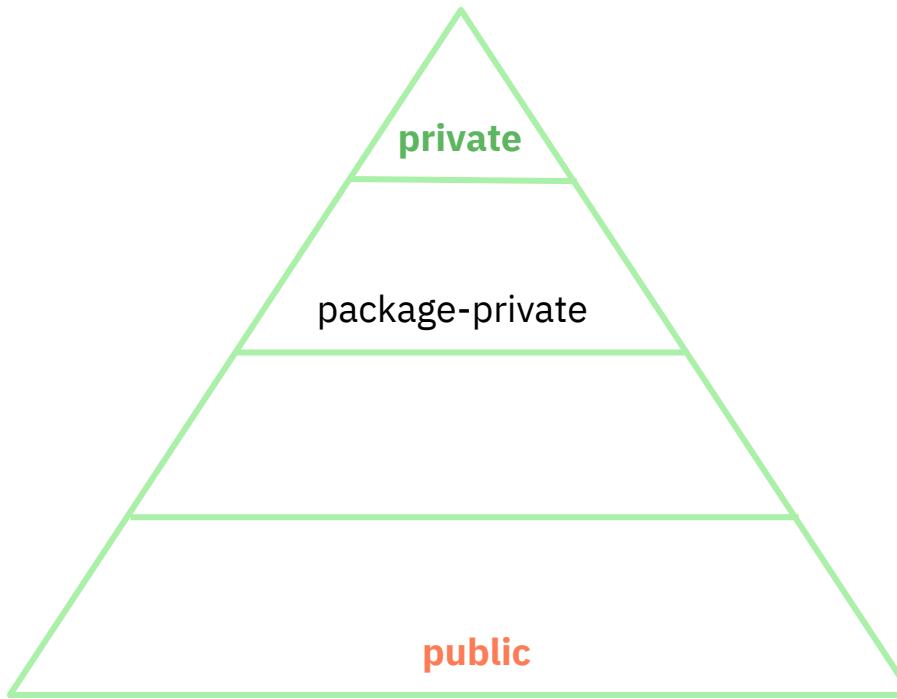


## Модификаторы доступа





## Модификаторы доступа





## Нарушение инкапсуляции

```
17  
18     cat1.name = "";  
19     cat1.color = "GREEEEEN";  
20     cat1.age = -12345;  
21
```



## Не инкапсулированный кот

```
3  public class Cat {  
4      static int pawsCount = 4;  
5  
6      String name;  
7      String color;  
8      int age;  
9  
10     public Cat(String name, String color, int age) {
```



## Инкапсулированный кот

```
3  public class Cat {  
4      static int pawsCount = 4;  
5  
6      private String name;  
7      private String color;  
8      private int age;  
9  
10     public Cat(String name, String color, int age) {
```



## Геттеры и сеттеры

```
29
30     public String getName() { return name; }
31     public void setName(String name) { this.name = name; }
32     public String getColor() { return color; }
33     public void setColor(String color) { this.color = color; }
34     public int getAge() { return age; }
35     public void setAge(int age) { this.age = age; }
36 }
```



## Доступ через геттеры и сеттеры

```
8     System.out.println("Cat1 named: " +
9             cat1.getName() + " is " +
10            cat1.getColor() + " has age: " +
11            cat1.getAge());
12    System.out.println("Cat2 named: " +
13            cat2.getName() + " is " +
14            cat2.getColor() + " has age: " +
15            cat2.getAge());
```



## Запрещаем недопустимое

```
29
30     public String getName() { return name; }
31     public void setName(String name) { this.name = name; }
32     public String getColor() { return color; }
33     public int getAge() { return age; }
34 }
```



## Хранение возраста

```
3  public class Cat {  
4      static int pawsCount = 4;  
5  
6      private String name;  
7      private String color;  
8      private int age;  
9  
10     public Cat(String name, String color, int age) {
```



## Хранение года рождения и вычисление возраста

```
6     private String name;  
7     private final String color;  
8     private int birthYear;  
9  
10    public Cat(String name, String color, int birthYear) {  
11        this(name, color);  
12        this.birthYear = birthYear;  
13    }
```

```
32    public int getAge() {  
33        return 2022 - birthYear;  
34    }
```



Теперь ваша очередь!

## Ответьте на вопросы сообщением в чат

### Вопросы:

1. Перечислите модификаторы доступа
2. Инкапсуляция - это
  - a. архивирование проекта;
  - b. сокрытие информации о классе;
  - c. создание микросервисной архитектуры.





# Наследование



## Идентичный коту класс собаки

```
1 package ru.gb.jcore;

2
3 public class Dog {
4     static int pawsCount = 4;
5     private String name;
6     private String color;
7     private int birthYear;
8
9     public Dog(String name, String color, int birthYear) {
10         this(name, color);
11         this.birthYear = birthYear;
12     }
13
14     public Dog(String name, String color) {
15         this.name = name;
16         this.color = color;
17     }
18
19     void voice() { System.out.println(name + " barks"); }
20
21     void jump() {
22         if (this.getAge() < 5)
23             System.out.println(name + " jumps");
24     }
25 }
```





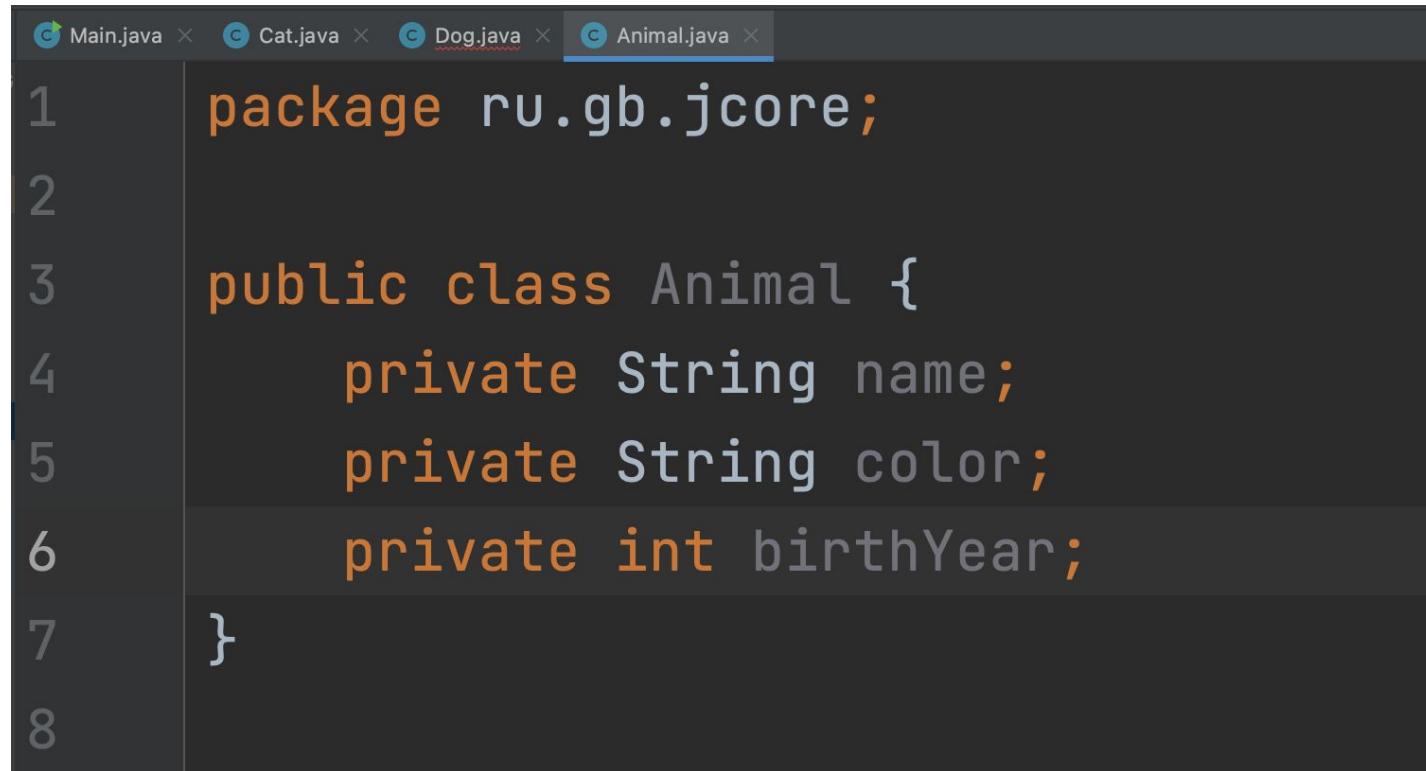
## Наследование

**Наследование** (англ. inheritance) – концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

Наследование в Java реализуется ключевым словом `extends` (англ. - расширять)



## Класс животного



The screenshot shows a dark-themed code editor with a tab bar at the top containing four tabs: Main.java, Cat.java, Dog.java, and Animal.java. The Animal.java tab is currently active, indicated by a blue underline. The code itself is displayed in a monospaced font. It defines a class named Animal with private attributes for name, color, and birthYear, and concludes with a closing brace. The code is numbered from 1 to 8 on the left side.

```
1 package ru.gb.jcore;
2
3 public class Animal {
4     private String name;
5     private String color;
6     private int birthYear;
7 }
8
```

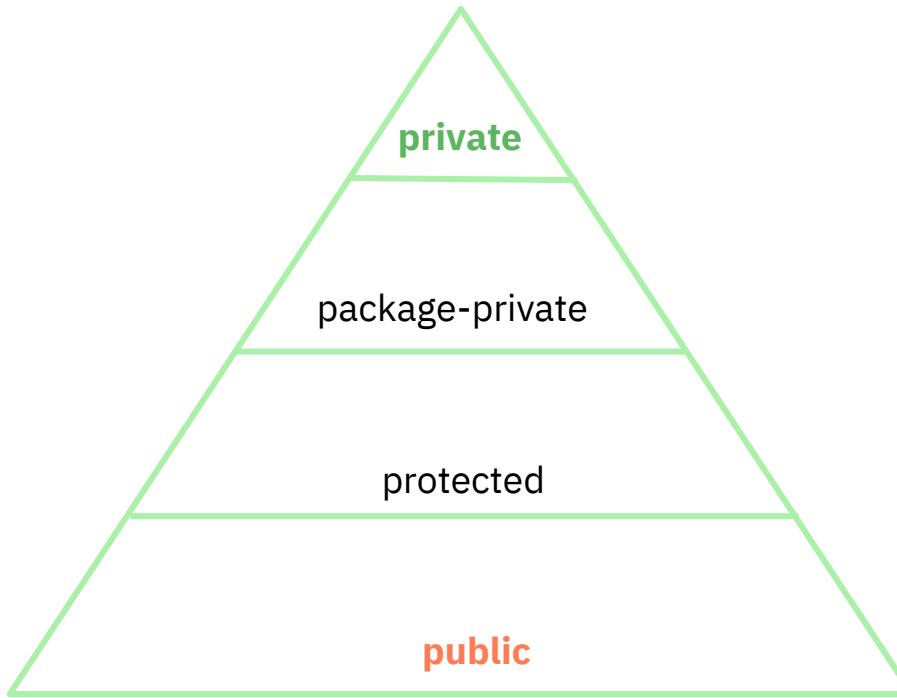


## Класс животного

```
>Main.java x Cat.java x Dog.java x Animal.java x
1 package ru.gb.jcore;
2
3 public class Animal {
4     private String name;
5     private String color;
6     private int birthYear;
7
8     void jump() {
9         if (this.getAge() < 5)
10            System.out.println(name + " jumps");
11    }
12
13    public String getName() { return name; }
14    public void setName(String name) { this.name = name; }
15    public String getColor() { return color; }
16    public int getAge() { return 2022 - birthYear; }
17
18 }
```



## Модификаторы доступа





## Конструктор животного

```
8      public Animal(String name, String color, int birthYear) {  
9          this.name = name;  
10         this.color = color;  
11         this.birthYear = birthYear;  
12     }
```



## Порядок вызова конструкторов

```
3 public class Animal {  
4     |  
5     Animal() {  
6         System.out.println("Animal ctor");  
7     }  
8 }
```

```
3 public class Cat extends Animal {  
4     static int pawsCount = 4;  
5  
6     Cat() {  
7         System.out.println("Cat ctor");  
8     }  
9 }
```

```
22 Cat cat3 = new Cat();
```

Run: Main ×

```
Animal ctor  
Cat ctor
```





## Вызов родительского конструктора из наследника

```
6     public Dog(String name, String color, int birthYear) {  
7         super(name, color, birthYear);  
8     }
```



## Ещё один наследник

```
3  public class Bird extends Animal {  
4      static int pawsCount = 2;  
5  
6      public Bird(String name, String color, int birthYear) {  
7          super(name, color, birthYear);  
8      }  
9  
10     void voice() { System.out.println(name + " tweets"); }  
13 }
```

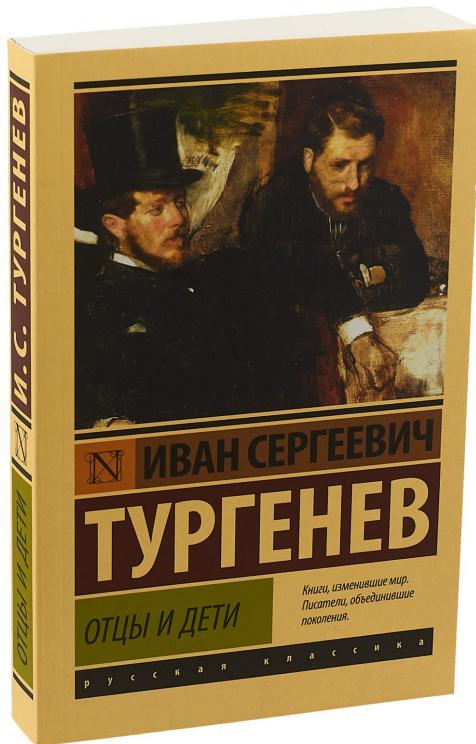


## Дополнительные свойства

```
3  public class Bird extends Animal {  
4      static int pawsCount = 2;  
5      int flyHeight;  
6  
7      public Bird(String name, String color, int birthYear, int flyHeight) {  
8          super(name, color, birthYear);  
9          this.flyHeight = flyHeight;  
10     }  
11  
12     void voice() { System.out.println(name + " tweets"); }  
13  
14 }
```

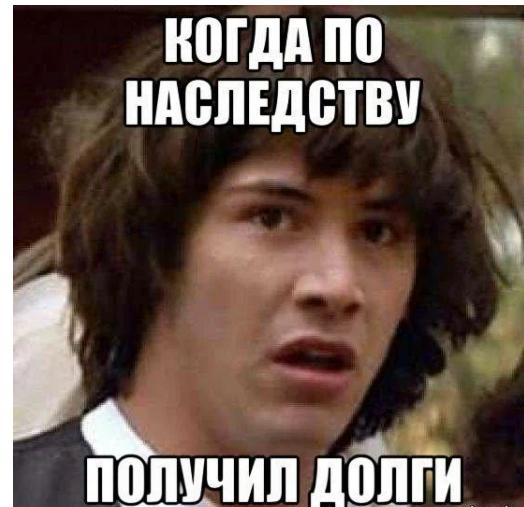
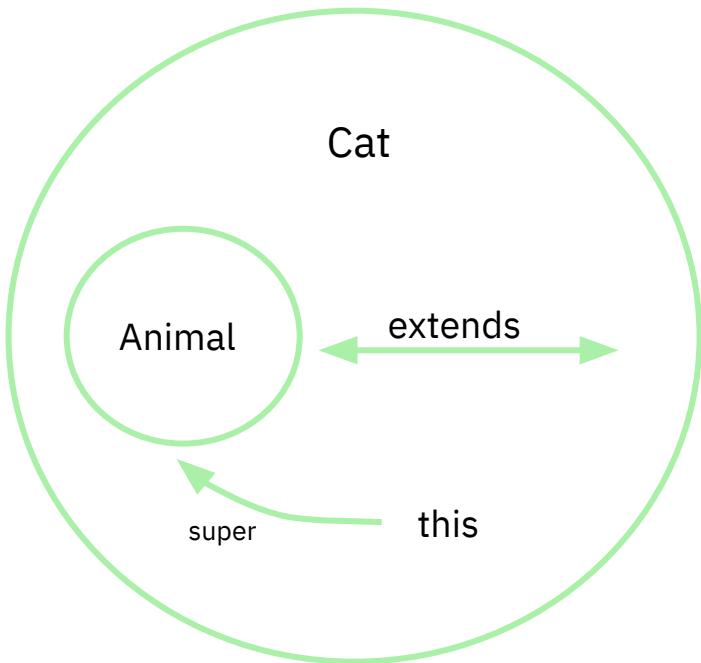


this vs. super





## this() vs. super()





## this.method() this.field vs. super.method() super.field



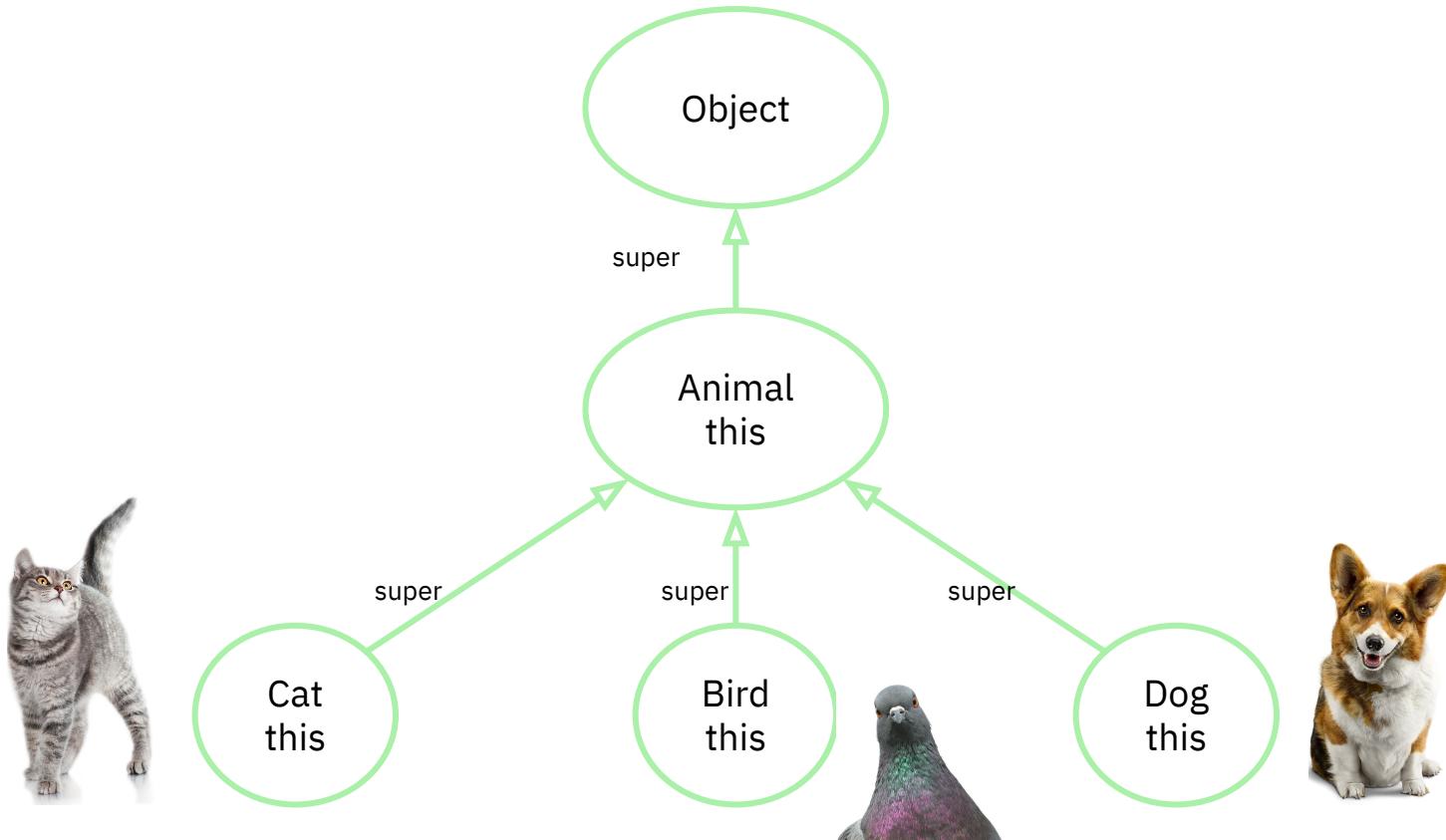


Множественное наследование запрещено





## Object. Каскадное наследование





## Общие ссылки

```
4 ►   public static void main(String[] args) {  
5       Cat cat1 = new Cat("Barsik", "White", 4);  
6       Cat cat2 = new Cat("Murzik", "Black", 6);  
7       Object animal = new Animal("Cat", "Black", 3);  
8       Object cat = new Cat("Murka", "Black", 4);  
9       Object dog = new Dog("Bobik", "White", 2);  
10      Animal birdAnimal = new Bird("Chijik", "Grey", 3, 10);  
11      Animal catAnimal = new Cat("Marusya", "Orange", 1);  
12
```



## Общие ссылки

```
11  
12     Object cat = new Cat("Murka", "Black", 4);  
13     Cat cat3 = (Cat) cat;  
14
```



# instanceof



## Оператор instanceof

оператор instanceof возвращает истину, если объект принадлежит классу или его суперклассам и ложь в противном случае





## Проверка на возможность преобразования

```
12     Object cat = new Cat("Murka", "Black", 4);
13     if (cat instanceof Dog) {
14         Dog dogIsCat = (Dog) cat;
15         dogIsCat.jump();
16     } else {
17         System.out.println("Conversion is invalid");
18     }
```

Run: Main



Conversion is invalid



# Ключевое слово final



## Константность

**final** - переменная с конечным значением.  
Класс с финальной реализацией.





## Запрет наследования

```
3 public class Parrot extends Bird {  
4     public Parrot(String name, int year, int flyHeight) {}  
5     super(name, color, bir  
6 }  
7  
8
```

Cannot inherit from final 'ru.gb.jcore.Bird'  
Make 'Bird' not final ⌂ More actions... ⌂

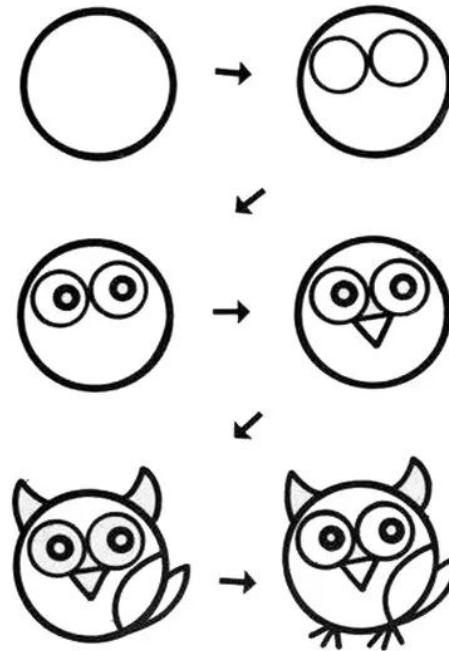
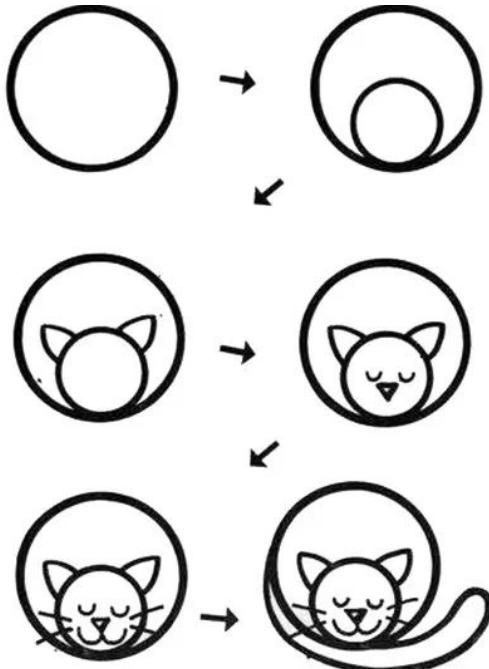
ru.gb.jcore  
public final class Bird  
extends Animal  
sources-draft



# Абстракция



## Абстракция





## Абстрактный метод

18

19      ⬇

abstract void voice();

20



## Абстрактный класс животного

```
3 ⏵ public abstract class Animal {  
4     protected String name;  
5     protected String color;  
6     protected int birthYear;  
7  
8     public Animal(String name, String color, int birthYear) {  
9         this.name = name;  
10        this.color = color;  
11        this.birthYear = birthYear;  
12    }  
13  
14    void jump() {  
15        if (this.getAge() < 5)  
16            System.out.println(name + " jumps");  
17    }  
18  
19 ⏵     abstract void voice();
```



## Абстракции в языке Java

**Абстрактный метод** - это метод не содержащий реализации (только объявление метода)

**Абстрактный класс** - класс содержащий хотя бы один абстрактный метод.

Абстрактный класс нельзя инстанцировать

Теперь ваша очередь!



## Ответьте на вопросы сообщением в чат

### Вопросы:

1. Какое ключевое слово используется при наследовании?
  - a. parent;
  - b. extends;
  - c. как в C++, используется двоеточие.
2. super - это
  - a. ссылка на улучшенный класс;
  - b. ссылка на расширенный класс;
  - c. ссылка на родительский класс.
3. Не наследуются от Object
  - a. строки;
  - b. потоки ввода-вывода;
  - c. ни один вариант не верный.





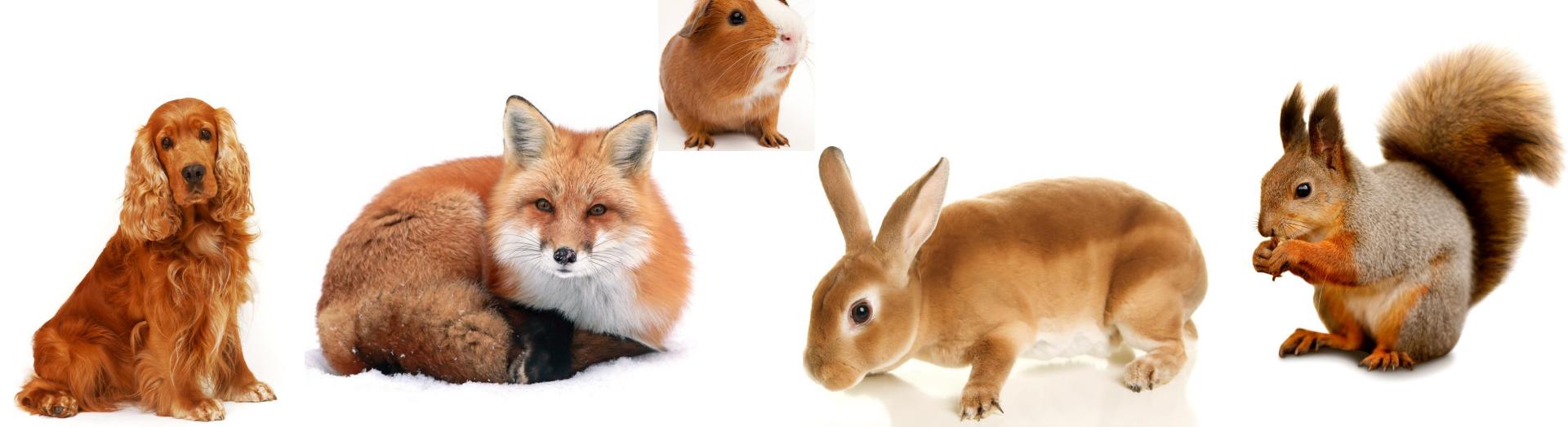
# Полиморфизм



## Полиморфизм



**Полиморфизм** – это возможность объектов с одинаковой спецификацией иметь различную реализацию (Overriding)





# Абстрактный класс животного

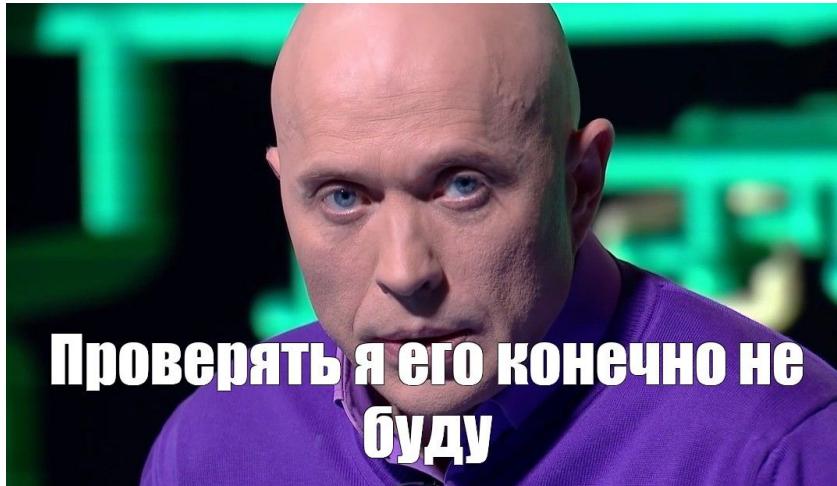
Project

Cat.java	Dog.java	Bird.java
1 package ru.gb.jcore;	1 package ru.gb.jcore;	1 package ru.gb.jcore;
2	2	2
3 public class Cat extends Animal {	3 public class Dog extends Animal {	3 public final class Bird extends Animal {
4     static int pawsCount = 4;	4     static int pawsCount = 4;	4     static int pawsCount = 2;
5	5	5
6     public Cat(String name, String color, int birthYear) {	6     public Dog(String name, String color, int birthYear) {	6     public Bird(String name, String color, int birthYear) {
7         super(name, color, birthYear);	7     }	7         super(name, color, birthYear);
8     }	8 }	8     this.flyHeight = flyHeight;
9	9	9 }
10    @Override	10    @Override	10    @Override
11    void voice() {	11    void voice() {	11    void voice() {
12        System.out.println(name + " meows");	12        System.out.println(name + " barks");	12        System.out.println(name + " tweets");
13    }	13    }	13    }
14}	14}	14}
15}	15}	15}
16}	16}	16}
17}	17}	17}



## Аннотация @Override

Аннотации реализуют вспомогательные интерфейсы. Аннотация @Override проверяет, действительно ли метод переопределяется, а не перегружается.





# Абстрактный класс животного

The image shows a Java code editor with two tabs open: `Cat.java` and `Animal.java`. The `Animal.java` file contains the following code:

```
2
3 public abstract class Animal {
4     protected String name;
5     protected String color;
6     protected int birthYear;
7
8     public Animal(String name, String color, int birthYear) {
9         this.name = name;
10        this.color = color;
11        this.birthYear = birthYear;
12    }
13
14    void move() {
15        System.out.println(name + " walks on paws");
16    }
}
```

The `Snake.java` file contains the following code:

```
2
3 public class Snake extends Animal {
4     public Snake(String name, String color, int birthYear) {
5         super(name, color, birthYear);
6     }
7
8     @Override
9     void move() {
10        System.out.println(name + " crawls");
11    }
12
13     @Override
14     void voice() {
15        System.out.println(name + " hisses");
16    }
17}
```



## Перекрытие (hiding) методов

```
37     Animal animal = new Cat("Barsik", "White", 4);  
38     Cat cat3 = new Cat("Murzik", "Black", 6);  
39     animal.self();  
40     cat3.self();
```

Run: Main ×



I'm an animal

I'm a cat



## Научный подход к полиморфизму

Полиморфизм в языках программирования и теории типов — способность функции обрабатывать данные разных типов

Существуют параметрический полиморфизм и ad-hoc-полиморфизм

Широко распространено определение полиморфизма, приписываемое Бьёрну Страуструпу (автор языка C++): **«один интерфейс — много реализаций»**



## Параметрический полиморфизм. Начало

К полиморфизму также относится перегрузка методов  
(Overloading)





## Перегрузка метода прыжка

```
22     void jump() {  
23         if (this.getAge() < 5)  
24             System.out.println(name + " jumps");  
25     }  
26  
27     void jump(String place) {  
28         if (this.getAge() < 5)  
29             System.out.println(name + " jumps to " + place);  
30     }  
31     void jump(int count) {  
32         if (this.getAge() < 5)  
33             System.out.println(name + " jumps " + count + " times");  
34     }
```



## Как следовать ООП?

- 📌 абстрагируйте
- 📌 классифицируйте
- 📌 снова абстрагируйте
- 📌 инкапсулируйте



Теперь ваша очередь!

## Ответьте на вопросы сообщением в чат

### Вопросы:

1. Является ли перегрузка полиморфизмом
  - a. да, это истинный полиморфизм
  - b. да, это часть истинного полиморфизма
  - c. нет, это не полиморфизм
2. Что обязательно для переопределения?
  - a. полное повторение сигнатуры метода
  - b. полное повторение тела метода
  - c. аннотация Override





## На этом уроке

- 📌 Классы;
- 📌 Объекты;
- 📌 Статика;
- 📌 Стек и куча;
- 📌 Сборщик мусора;
- 📌 Конструкторы;
- 📌 Инкапсуляция;
- 📌 Наследование;
- 📌 Полиморфизм.





## Практическое задание

- 📌 Написать класс кота (из лекции) так, чтобы каждому объекту кота присваивался личный порядковый целочисленный номер.
- 📌 Написать классы кота, собаки, птицы, наследники животного. У всех есть три действия: бежать, плыть, прыгать. Действия принимают размер препятствия и возвращают булев результат. Три ограничения: высота прыжка, расстояние, которое животное может пробежать, расстояние, которое животное может проплыть. Следует учесть, что коты не любят воду.
- 📌 \* Добавить механизм, создающий 25% разброс значений каждого ограничения для каждого объекта животного





Надо много учиться,  
чтобы знать хоть немного.

Шарль Луи де Монтескьё



## Вложенные классы

