nexign

Управление выполнением программы

nexign

nexign

Условные выражения

Условный оператор if

Шаблон

```
if (условие) оператор;
else оператор;
```

```
if (условие)
    оператор;
else if (условие)
    оператор;
else if (условие)
    оператор;
…
else
    оператор;
```

Фигурные скобки рекомендуется ставить, даже когда они необязательны

```
int a, b;
...
if (a < b)
    a = 0;
else
    b = 0;
```

```
int a, b;
...
if (a < b) {
    a = 0;
    ...
} else {
    b = 0;
    ...
}
```


nexign

switch-case

Шаблон

```
switch (выражение) {
    case значение:
        оператор;
        break;
    case значение:
        оператор;
        break;
                    …
    default:
      оператор;
}
```

- Можно писать произвольное число операторов для каждого варианта **case**
- Выход из выполнения последовательности операторов осуществляется с помощью оператора **break**
- Если **break** отсутствует, происходит переход в следующий блок операторов
- указанное выражение должно иметь тип **byte, short, int, char** или **перечислимый тип + String (Java 7+)**

Java 14+ Switch Expression

Expression	Statement

int numLetters = switch(day) {
case MONDAY, FRIDAY, SUNDAY -> 6;
case TUESDAY -> 7;
case THURSDAY, SATURDAY -> 8;
default -> 9;
}
	switch(day) {
	case MONDAY, FRIDAY, SUNDAY:
	numLetters = 6;
	break;
	case TUESDAY:
	logger.info("Tuesday");
	numLetters = 7;
	break;
	case THURSDAY:
	logger.info("Thursday");
	case SATURDAY:
	numLetters = 8;
	break;
	default:
	numLetters = 9;
	}

Тернарный оператор


```
public static void main(String[] args) {
        Man man = new Man(22);
        String securityAnswer = (man.getAge() >= 18) ? "Проходите!" : "Вход запрещён!";
        System.out.println(securityAnswer);
    }
}
```

nexign

Циклы

Циклы с условием

Шаблон цикла while

```
while (условие) {
    тело цикла
}
```

```
int i = 10;
while (i > 0){
    i--;
    System.out.println(i);
}
```

Шаблон цикла do…while

```
do {
    тело цикла
} while (условие);
```

```
int i = 10;
do {
    i--;
    System.out.println(i);
} while (i > 0);
```

Цикл for

Шаблон

```
for (инициализация; условие; итерация) {
      тело цикла
}
```

```java
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

- Каждый из блоков оператора **for** является необязательным, но при этом разделительные **“;”** требуется писать
- **for (;;)** — бесконечный цикл
- Наиболее употребительное использование оператора **for** для перебора значений некоторой переменной, увеличивающихся или уменьшающихся на 1

Цикл for each

Шаблон

```
for (тип итерационная_переменная : коллекция) {
      тело цикла
}
```

```
int nums[] = {1, 2, 3};
int y = 0;
for (int x : nums){
  y+= x;
  }
```

- Предназначен для циклической обработки коллекций (массивов, списков, деревьев и т. д.)
- В каждом проходе тела цикла извлекается очередной элемент коллекции и запоминается в переменной цикла
- Цикл выполняется до тех пор, пока не будут извлечены все элементы коллекции
- Итерационная переменная доступна "только для чтения".

Оператор break

- Передает управление
  на следующий за циклом оператор
- Применим ко всем видам циклов

```
for (String s: haystack) {
    if (needle.equals (s)) {
        found = true ;
        break;
    }
}
```

nexign

Оператор continue

- Прерывает текущую итерацию цикла и начинает следующую
- Перед новой итерацией проверяется условие цикла
- Применим ко всем видам циклов

```
for (String s : haystack) {
    if (!needle.equals (s)) {
        continue;
    }
    count++;
}
```

nexign

Q&A

Итоги

- Java - наследник C++
- Есть автоматическое управление памятью
- Код выполняется в JVM, что даёт простор для кастомизации
- Заточен на ООП и работу с бэкендом
- Есть рудименты процедурных языков в виде примитивов
- Для большинства действий уже есть классы/методы
- Есть обратная совместимость, мешающая частому внедрению новых фич

nexign

Классы
Объекты
Методы

nexign

План урока

- Общая информация об ООП
- Класс. Структура класса
- Поля
- Конструкторы
- Методы
- Наследование
- Методы класса Object

nexign

ООП. Общая информация

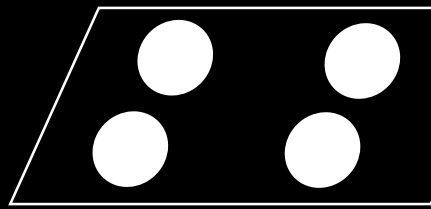
Парадигмы программирования

Императивная парадигма:

- исходный код программная инструкция;
- инструкции должны выполняться последовательно;
- данные результата выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;
- данные, полученные при выполнении инструкции, могут записываться в память.

nexign

Объектно-ориентированное программирование — парадигма программирования, в которой программа строится из взаимодействующих объектов



Принципы ООП

- **Инкапсуляция:**

  объединение данных и методов их обработки в одну сущность, приводящее к сокрытию реализации класса и отделению его внутреннего представления от внешних пользователей

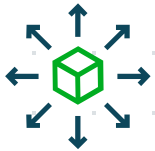
- **Наследование:**

  отношение между классами, при котором один класс использует структуру или поведение другого класса

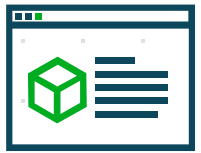
- **Полиморфизм:**

  способность объекта соответствовать во время выполнения двум или более возможным типам. С другой стороны, способность различных объектов предлагать свои реализации одного и того-же интерфейса

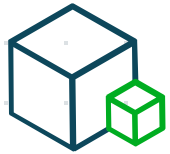
полиморфизм



инкапсуляция



наследование



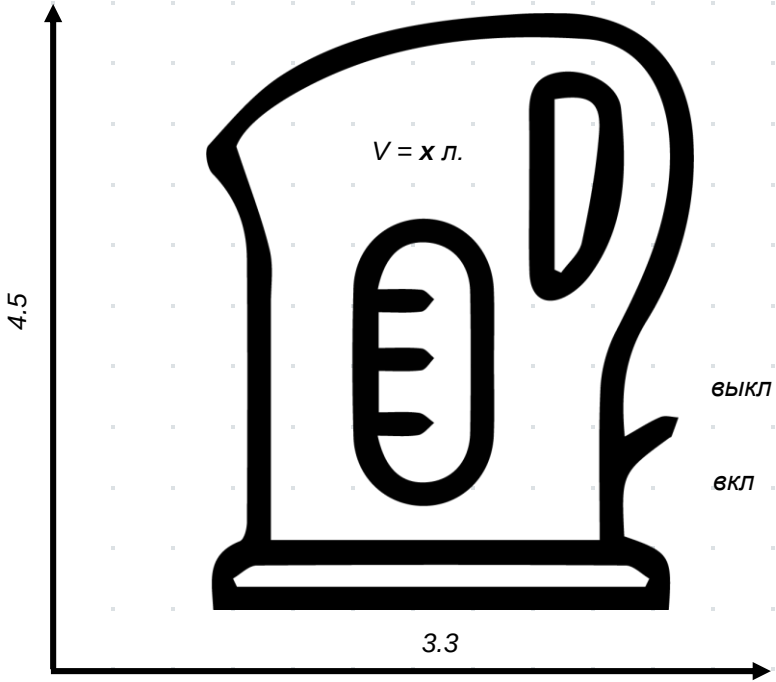
nexign

nexign

Класс. Структура класса

Класс

- Класс представляет собой абстрактное описание структуры реальных объектов
- Объекты создаются в оперативной памяти машины на основе такого описания, которое представляет собой перечисление свойств, имеющихся у объектов и методов, которые объектам доступны
- Объект представляет собой совокупность конкретных значений свойств и методов, работающих с этими свойствами
- Вся логика работы java-приложений построена на взаимоотношении классов и объектов



nexign

Структура класса

```
class ClassName
{
   field1
   field2
   . . .
   constructor1
   constructor2
   . . .
   method1
   method2
   . . .
}
```

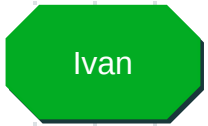
```
package org.megacompany.staff;
    class Employee {
        // instance fields
        private String name;
        private double salary;
        private LocalDate hireDay;
        // constructor
        public Employee(String n, double s, int year, int month, int day) {
            name = n;
            salary = s;
            hireDay = LocalDate.of(year, month, day);
        }
        // a method
        public String getName() {
            return name;
        }
        // more methods
         . . .
```


nexign

Рождение, жизнь и смерть объекта

Создание объекта

```
new Employee("Ivan");
```



nexign

Рождение, жизнь и смерть объекта

Присвоение ссылки

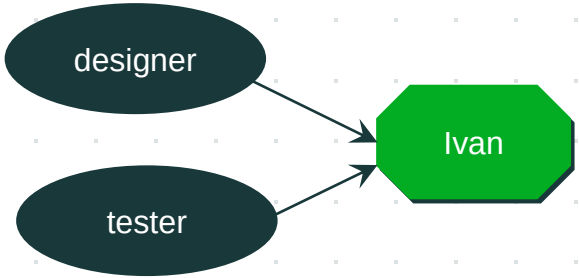
```
Employee designer = new Employee("Ivan");
```

nexign

Рождение, жизнь и смерть объекта

Присвоение ссылки

```
Employee tester = designer;
```

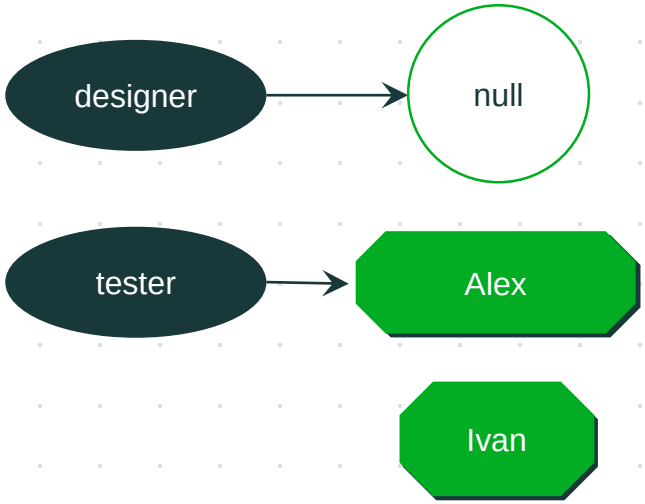


nexign

Рождение, жизнь и смерть объекта

Потеря ссылки

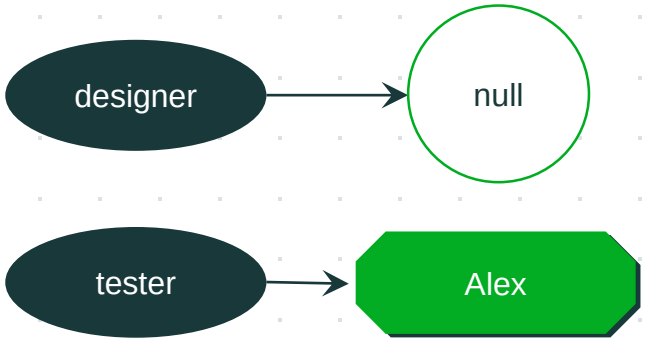
```
designer = null;
tester = new Employee("Alex");
```



nexign

Рождение, жизнь и смерть объекта

Сборка мусора



nexign

var Java 10+

local variable type inference

```
TheLongestNameYouCanEverImagine theLongestNameYouCanEverImagine = new TheLongestNameYouCanEverImagine();
```

```
var theLongestNameYouCanEverImagine = new TheLongestNameYouCanEverImagine();
```


nexign

var Java 10+

- Ключевое слово var можно использовать при объявлении локальных переменных в конструкторах, блоках инициализации, методах.
- var нужно инициализировать сразу после именования — в одном операторе. При этом можно переносить такое объявление переменной на разные строки.
- Объявлять сразу несколько переменных с помощью var в одном операторе нельзя.
- Инициализировать var-переменную значением null без явного указания типа тоже нельзя.
- Значение var-переменной в дальнейшем меняться может, а вот тип — нет.
- var допустимо использовать в качестве названия переменной, но нельзя так именовать тип: класс, интерфейс или перечисление.

nexign

Структура класса. Поля

Поля

- В отличие от локальных переменных, поля можно не инициализировать явно.
- В этом случае примитивные типы получают значение по умолчанию **(0, false),** а поля со ссылками — значение **null**.
- Проинициализировать поле по месту его определения не возбраняется: **int a = 42** или даже **int a = getValue().**

```
class Employee {
        // instance fields
        private String name;
        private double salary;
        private LocalDate hireDay;
}
```

Модификаторы доступа

private

доступен только методам данного класса

public

доступен всем желающим

protected

доступен наследникам и всем классам в пакете

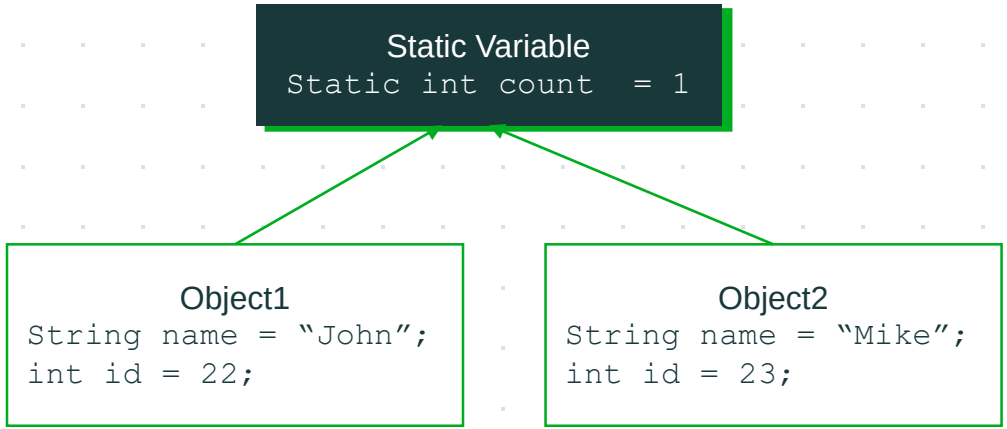
default

доступен всем классам в том же пакете

Статические поля в JVM

Модификатор **static** означает, что поле принадлежит классу, а не создаваемому от него экземпляру.

Иными словами, одна переменная связана со всеми объектами, созданными по этому классу.



nexign

Статические поля

- Статические переменные могут быть созданы только, как переменные класса. Они не могут быть локальными переменными

- К статическим полям класса можно получить доступ без создания объекта, используя имя класса (ссылка на объект не нужна)

```
public class Main {
    public static void main(String[] args) {
        static String wrongVar =
        "Modifier 'static' not allowed here";
        System.out.println(wrongVar);
    }
}

public static int counts = 100;
System.out.println("Результат: " + Car.counts);
```

Неизменяемые (inmutable) поля

```
static final double GRAVITATIONAL_CONSTANT = 9.81;

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

Константы

статические финальные поля, содержимое которых неизменно. Это относится к примитивам, String, неизменяемым типам и неизменяемым коллекциям неизменяемых типов. Если состояние объекта может измениться, он не является константой.

nexign

Структура класса. Конструкторы

Конструкторы

Конструктор класса

именованный блок инструкций в составе класса, вызываемый каждый раз при создании нового объекта

- Имя конструктора всегда совпадает с именем класса
- Класс может иметь несколько конструкторов
- Конструктор не имеет возвращаемого значения

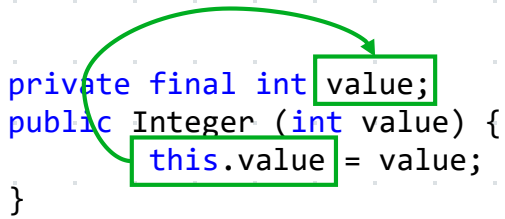
```
public class Person {
//public-конструктор без аргументов
    public Person() {
    ....
    }
//package-private конструктор с
аргументом
    Person(String  name) {
    ....
    }
}
```

Конструкторы

- Если в класс явным образом не включен конструктор, он снабжается конструктором по умолчанию, не имеющим параметров
- Если конструктор явным образом включен в класс, то конструктор по умолчанию в него не добавляется
- Если в родительском классе нет конструктора без аргументов,конструктор по умолчанию не создаётся
- Если класс имеет несколько конструкторов, они должны явно отличаться по количеству и типу передаваемых параметров
- Конструктор не обязан быть публичным.

Поле this()

- Ключевое слово **this** применяется тогда, когда из метода экземпляра класса необходимо сослаться на поле или метод того же самого экземпляра
- Как правило, это слово используется для того, чтобы избежать конфликта при совпадении имен параметров метода и полей объекта




```
public class Integer {
    private final int value ;
    public Integer (int value) {
        this.value = value;
    }
    registerMe(Registrator r) {
        //нужна ссылка на себя
        r.register(this);
    }
}
```


nexign

Пример

```
public class Employee {
        public String name;
        public int age;
        public Employee(String name, int age) {
            this.name = name;
            this.age = age;
        }
            public Employee() {
            }
            public void sayMyName() {
                System.out.println(this.name);
            }
        public static void main(String[] args) {
            Employee e1 = new Employee();
            Employee e2 = new Employee("Heisenberg", 22);
            e1.sayMyName();
            e2.sayMyName();
        }
    }
```

Перегрузка конструкторов

```
class Person {
    Person(String  name){
    ...
    }

    Person(){
        this("unknown");
    }
}
```


nexign

Методы

Методы – это алгоритмы, обрабатывающие данные(состояния)

Шаблон

Модификаторы Тип Имя(*список параметров*){
Тело // return *выражение;*
}

- Если в методе в какой-либо из ветвей не использовать оператор return, будет выдана ошибка компиляции
- Если тип возвращаемого значения void, оператор return не нужен

```
public class Hello {
    private String s;

    public void getName(String s) {
        System.out.println("Hello" + s);
    }
}
```

nexign

Структура класса. Методы

Параметры методов

- Метод может иметь ноль или более параметров
- Каждый параметр имеет свой тип и имя
- Внутри тела метода параметры используются как локальные переменные
- Тип параметра может быть как примитивным, так и ссылочным

```
void increment(int tmpInt) {
    tmpInt++;
}

public static void main(String[] args)
{
    int myInt = 42;
    Hello obj = new Hello();
    obj.increment(myInt);
    System.out.println(myInt);
}
```

Параметры переменной длины

```
static void vaTest(int ... v){
    System.out.println("Number of args:" +v.length);
    System.out.println("Contents:");
    for (int x : v) {
        System.out.println(x + " ");
    }
}
```

Варианты вызова метода vaTest

```
vaTest(10);  vaTest(1,2,3);  vaTest();
```

Параметры переменной длины

- В список параметров наряду с параметрами переменной могут быть включены «обычные» параметры
- Такие параметры обязательно надо указывать при вызове метода
- Параметр, содержащий переменное число аргументов, должен быть последним в списке параметров метода

  (*int doIt(int a, int b, double c, int… vals) { … })*
- В списке параметров метода может быть только один параметр переменной длины

  (*int doIt(int a, int b, double c,* ***int… vals, double… moreVals****){ … }*)

Перегрузка методов (Overloading)

Сигнатура метода

Имя метода в сочетании с числом, типом и порядком следования его параметров

- Тип возвращаемого значения и имена параметров в сигнатуру не входят
- Методы, имеющие разные сигнатуры, считаются различными
- Если сигнатуры методов совпадают, перегрузка запрещена

```
static int addNumbers(int i1, int i2){
    return i1 + i2;
}

static int addNumbers(double d1, int i2){
    return (int)d1 + i2;
}

static int addNumbers(int i1,double d2){
    return i1 + (int)d2;
}

static int addNumbers(int i1, int i2, int i3){
    return i1 + i2 + i3;
}
```

Статические методы

Статическим методам доступны только статические переменные и вызовы других статических методов

```
class Employee {

    private static int nextId = 1;
    private int id;
    . . .
    public static int getNextId() {
        return nextId; // returns static field
    }
}

        . . .
   Employee.nextId() //имя класса вместо объекта
```

Причины использовать статические методы

- Для доступа / управления статическими переменными и другими статическими методами, которые не зависят от объектов
- Для служебных, вспомогательных классов и интерфейсов, поскольку не требуют создания объектов и соответственно, обеспечивают большую производительность
- Когда методу требуется доступ лишь к статическим полям класса

nexign

Методы класса Object

Методы класса Object

В Java все классы являются потомками класса Object. Из него любой java-класс наследует ряд полезных на практике методов:

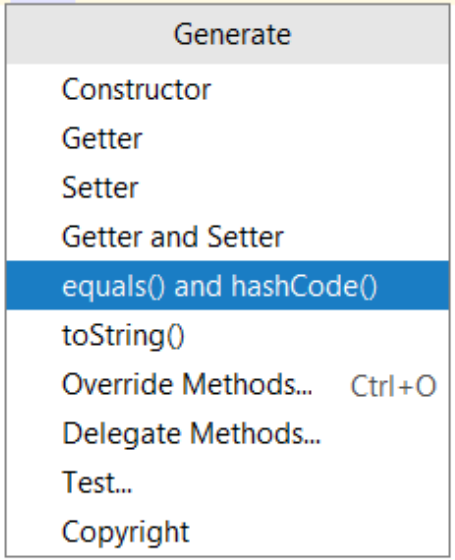
- **public boolean equals(Object obj)** – возвращает true в случае, когда равны значения объекта, из которого вызывается метод, и объекта, передаваемого через ссылку obj в списке параметров. В классе **Object** равенство рассматривается как равенство ссылок и эквивалентно оператору сравнения “==”.
- **public int hashCode()** – выдаёт *хэш-код* объекта. В **Java** хэш-код заменяет адрес объекта в тех случаях, когда для каких-либо целей надо хранить таблицы адресов объектов.
- **protected Object clone() throws CloneNotSupportedException** – метод занимается копированием объекта и возвращает ссылку на созданный клон (дубликат) объекта.
- **public String toString() -** Возвращает строковое представление объекта

Генерация hashCode и equals

IntelliJ Idea:

Alt + Insert

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



nexign

“Джентельменский набор” для класса

```
public class Point {
    private final int x;
    private final int y;

    /*Подождите! Нам нужны:
     * конструктор
     * getX() и getY()
     * equals и hashCode
     * toString
     * 40 строчек кода из ничего!
     */

    public double distance(Point other) {
    ...
    }
}
```

Records (Java 14+)

```
record Person(String name, int age){

static int minAge;
static{
        minAge = 18;
        System.out.println("Static initializer");
      }
   }
}
```

- private final поля
- конструктор
- доступ через x() и y()
- equals, hashCode и toString
- невозможно наследование от класса и от рекорда
- возможна реализация интерфейсов

Q&A

Спасибо!


Aleksandr.Mityasov


@mityasov


Aleksandr.Mityasov@nexign.com