**Еще об области действия переменных**

[Java Syntax](https://javarush.ru/quests/QUEST_JAVA_SYNTAX)

[Уровень 4](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_SYNTAX&level=4), Лекция 1

— Профессор всё свою линию гнёт. Все они, старые лекторишки, такие. Всё, что он может рассказать, ты можешь прочесть в книгах. Запомни, ещё ни один человек не научился плавать, прослушав с десяток лекций по этому поводу. Лекции помогают, когда ты 100% в теме и знаешь сам чуть меньше этого профессора.

— Ну, от лекций профессора же есть польза.

— Ага… вернее, мы надеемся, что она есть. Чем больше точек зрения на один и тот же материал ты услышишь, тем больше задумаешься, а как же дела обстоят на самом деле. Услышав только одну точку зрения, ты либо веришь ей, либо нет. Ладно, давай перейдём к делу.

— Давай рассмотрим картинку, которую я уже давал тебе ранее:

┏

┃public class Variables

┃┏

┃┃{

┃┃ private static String TEXT = "The end.";

┃┃ ┗━━━━━━━━━━━━━━┛

┃┃ public static void main (String[] args)

┃┃ ┏ ┗━━━━━━━┛

┃┃ ┃ {

┃┃ ┃ System.out.println("Hi");

┃┃ ┃ String s = "Hi!";

┃┃ ┃ ┏┗━━━━┛

┃┃ ┃ ┃ System.out.println(s);

┃┃ ┃ ┃ if (args != null)

┃┃ ┃ ┃ {

┃┃ ┃ ┃ String s2 = s;

┃┃ ┃ ┃ ┗━━━━┛

┃┃ ┃ ┃ ┏

┃┃ ┃ ┃ ┃ System.out.println(s2);

┃┃ ┃ ┃ ┗

┃┃ ┃ ┃ }

┃┃ ┃ ┃ Variables variables = new Variables();

┃┃ ┃ ┃ System.out.println(variables.classVariables);

┃┃ ┃ ┃ System.out.println(TEXT);

┃┃ ┃ ┗

┃┃ ┃ }

┃┃ ┗

┃┃ public String classVariables;

┃┃ ┗━━━━━━━━━━━━━━━┛

┃┃ public Variables()

┃┃ {

┃┃ classVariables = "Class Variables test.";

┃┃ }

┃┃}

┃┗

┗

1. Переменная, объявленная в методе, существует/видна с начала объявления до конца метода.

2. Переменная, объявленная в блоке кода, существует до конца этого блока кода.

3. Переменные — аргументы метода — существуют до конца выхода из метода.

4. Переменные класса/объекта существуют все время жизни содержащего их объекта. Их видимость регулируется специальными модификаторами доступа.

5. Статические переменные классов существуют все время работы программы. Их видимость также определяется модификаторами доступа.

— Да, помню эту картинку.

— Вот и отлично. Давай напомню тебе некоторые моменты.

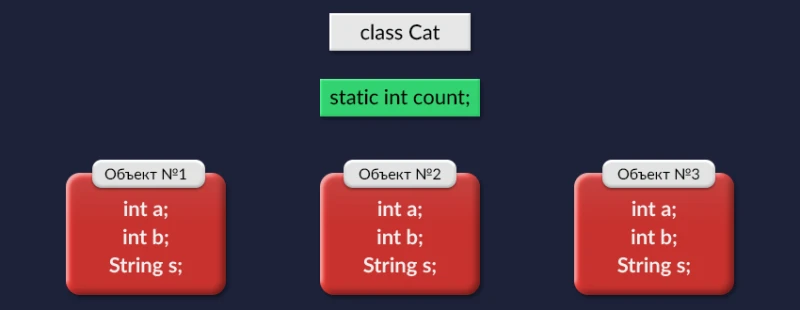
— Все переменные, объявленные внутри методов, существуют/видны с начала их объявления до конца метода (пример: 1).

— Если переменная определена/объявлена в каком-то блоке кода, то она существует до конца этого блока кода (пример: 2).

— Если переменная является аргументом функции, то она существует/видна во всем теле данного метода (пример: 3) .

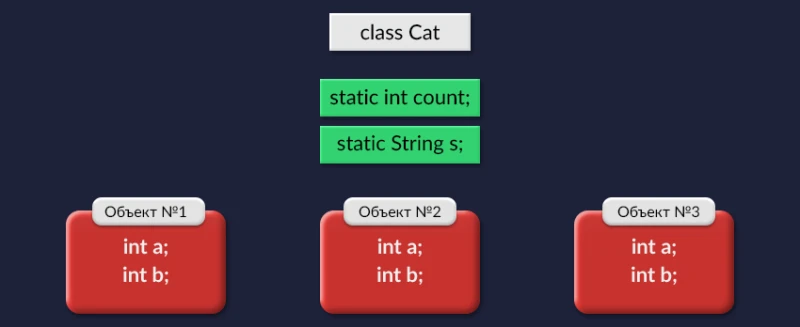
— Если переменная является переменной класса (пример: 4), то она привязана к конкретному объекту и существует все время, пока существует объект данного класса. **Если нет ни одного объекта, то нет и ни одной копии переменной.** К переменной можно обращаться (переменная видна) из всех методов класса, независимо от того, объявлены они до неё или после неё. Для каждого объекта создаётся своя, независимая от других объектов, переменная. Доступ к переменной невозможен из статических методов.

— Если переменная объявлена статической – помечена ключевым словом **static**, то она существует все время, когда существует её класс. Обычно JVM загружает класс в память при первом его использовании, тогда же и инициализируются статические переменные.

[](https://cdn.javarush.ru/images/article/57a7fd71-e8f7-4f38-898f-538fcd86e544/original.jpeg)

— В примере выше у нас объявлен класс Cat, у которого есть 4 переменные: a,b,s – обычные, а count – статическая. Если создать несколько объектов такого класса (например, 3 шт.), то каждый из них будет содержать свою собственную копию обычных переменных класса. Статическая же переменная – общая на всех. Формально она даже не находится внутри этих объектов, т.к. существует даже тогда, когда ни одного объекта класса Cat создано не было.

— Вот что произойдет, если мы объявим переменную s статической:

[](https://cdn.javarush.ru/images/article/0cb8bf91-8ce5-4b81-a6cf-021fa4f7dcd4/original.jpeg)

— Ага. Вроде понятно.

— А можно объявить одинаковые переменные?

— В методе — нет. Все переменные, объявленные внутри метода, должны иметь уникальные имена. Аргументы метода также считаются его переменными.

— А как насчёт переменных класса?

— Переменные класса тоже должны быть уникальными в рамках каждого конкретного класса.

Но есть исключение – имена переменных метода и переменных класса могут совпадать.

— А если мы в коде изменяем переменную, а таких переменных две, какая будет изменяться?

— Если в методе видны (доступны) несколько переменных, например, переменная класса и переменная метода, то тогда обращение произойдет к переменной метода.

Пример с двумя переменными count

class Main

{

public int count = 0; //объявили переменную класса

public void run()

{

count = 15; //обращение к переменной класса

int count = 10; //объявили локальную переменную метода

count++; //обращение к переменной метода

}

}

— В этом коде объявлены две переменных count. В строке 3 объявлена переменная класса, а в строке 8 – переменная метода.

— Когда метод run будет исполняться, то произойдет вот что:

— В строке 7 мы обращаемся к переменной класса и присваиваем ей значение 15

— В строке 8 объявляется (создается) новая переменная метода – count. Она закрывает собой переменную класса. Любой дальнейший код в методе будет видеть (обращаться) именно к переменной метода.

— Понятно.

— Переменная метода закрывает собой переменную класса. Т.е. обращение будет происходить к переменной метода. Но к переменной класса тоже можно обратиться, только более сложным способом.

|  |  |
| --- | --- |
| Статическая переменная класса  ClassName.variableName  *Примеры:*  Cat.catsCount | Обычная переменная класса  this.variableName  *Примеры:*  this.catsCount |

— А что ещё за статические методы и статические переменные?

— Статические методы и переменные привязаны не к объектам класса, а непосредственно к самому классу. Т.е. если мы создадим 10 объектов класса **Variables** из примера в самом начале уровня, у нас будет 10 переменных classVariable (по одной на каждый объект) и всего одна общая переменная **TEXT**.

— У меня тут вопрос.

— А чем отличаются статические и нестатические методы?

— Давай рассмотрим, как работает обычный нестатический метод:

|  |  |
| --- | --- |
| Как выглядит код  Cat cat = new Cat();  String name = cat.getName();  cat.setAge(17);  cat.setChildren(cat1, cat2, cat3); | Что происходит на самом деле  Cat cat = new Cat();  String name = Cat.getName(cat);  Cat.setAge(cat,17);  Cat.setChildren(cat, cat1, cat2, cat3); |

— При вызове метода в виде **«объект»** точка «имя метода», на самом деле вызывается метод класса, в который первым аргументом передаётся **тот самый объект**. Внутри метода он получает имя **this**. Именно с ним и его данными происходят все действия.

— Нифига себе. Так вот как, оказывается, оно все работает!

— А вот как работает статический метод:

|  |  |
| --- | --- |
| Как выглядит код  Cat cat1 = new Cat();  Cat cat2 = new Cat();  int catCount = Cat**.getAllCatsCount**(); | Что происходит на самом деле  Cat cat1 = new Cat();  Cat cat2 = new Cat();  int catCount = Cat**.getAllCatsCount**(**null**); |

— При вызове статического метода, никакого объекта внутрь не передаётся. Т.е. this равен null, поэтому статический метод не имеет доступа к нестатическим переменным и методам (ему нечего передать в такие методы в качестве this).

— Хм. Вроде понятно. Хотя как-то поверхностно.

— А вот и дядя Диего… С его задачками.

4

Задача

Java Syntax,  4 уровень,  1 лекция

Мне не подходит этот возраст…

Подумайте, что делает программа. Исправьте ошибку в программе чтобы переменная age объекта person изменила свое значение. Подсказка: тщательно просмотрите метод adjustAge

4

Задача

Java Syntax,  4 уровень,  1 лекция

Цена яблок

На аграрной планете Эплсиид главный заработок подавляющего большинства обитателей — продажа лучших в этом уголке Галактики яблок торговым кораблям Федерации и заезжим туристам. Давайте вступим в товарно-денежные отношения с абстрактным покупателем с учётом, что сейчас конец сезона и надо как-то пережить зиму: напишем метод, повышающий цену на яблоки!