Курс «QA Automation»

Основы Java 2

Николай Ивановский
Специалист по автоматизации CRM и интеграции

Массивы

Массив – нумерованный список объектов.

Хранит однотипные элементы.

Нумерация начинается с 0!

Массив

Объявление массива в java:

```
int[] i;
```

Простое объявление

```
int i[];
```

Обратное объявление

`int[] i` и `int i[]` – эквивалентны

Определение массива примитивов

Определение массива - это определение его размера.

```
i = new int[10];
```

← Простое определение

При простом определении элементы массива получат значение по умолчанию

```
int i[] = new int[10];
```

```
int i[] = {3, 5, 14, 23};
```

← Объявление переменной и определение массива

Объявление переменной и заполнении массива возможно только при объявлении переменной

~~`i = {5, 4, 18};`~~

Значения по умолчанию

При объявлении переменной простого типа, она заполняется значением по умолчанию

```
int i;
long l;
boolean b;
.....
```

Тип	Значение по умолчанию
Числа	0
boolean	false
char	\u0000

Доступ к элементам массива

Создадим массив:

```
int[] i = new int[6];
```

Присвоим 2^{му} элементу значение 12:

```
i[1]= 12;
```

Получим значение из 4го элемента:

```
int j = i[3];
```

Размер массива - это на всю жизнь

Изменить размер массива после его определения нельзя.

```
int[] i = new int[2];
i[0] = 1;
i[1] = 2;
i[3] = 3;
```

Текст ошибки при выполнении программы:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
    at com.company.Main.main(Main.java:11)
```

Определение массива ссылочных объектов

Есть ли у ссылочного типа данных значение по умолчанию?

```
String[] catName = new String[3];
System.out.println("иии в черном ящике - " + catName[1]);
```

Результат работы программы:

```
null
```

Определение массива ссылочных объектов

Все пустые ссылки в java – `null`. Работать с `null` нельзя. Это не объект.

```
catName[0].length();
```

Любое обращение к `null` как к объекту java не оценит.

Нет кота - нет имени - нельзя узнать длину имени.

Текст ошибки при выполнении программы:

Exception in thread "main" java.lang.NullPointerException

Что же делать? Инициализировать каждый элемент отдельно:

```
catName[0] = "Васька";
catName[1] = "Рыжик";
catName[2] = "Борис";
```

Копирование массивов или немного магии

```
int[] first = {2, 8, 11};
```
← Объявим и заполним массив

```
int[] second = first;
```
← Присвоим 2й переменной значение 1й

```
first[2] = 25;
```
← Изменим 3й элемент во втором массиве

Результат:

```
first => [2, 8, 25]
second => [2, 8, 25]
```

Массив - ссылочный тип данных.

Присваивая одной переменной массива значение второй, мы просто копируем ссылку на объект в памяти.

Копирование массивов и их увеличение

Создание массива:

```
int[] nums = {1, 2, 3};
```

Копирование массива. В памяти будет создана копия массива, и copiedNames будет ссылаться на нее.

```
int[] copiedNums = Arrays.copyOf(nums, nums.length());
```

Копирование массива с увеличением его размера. Первые элементы будут заполнены значениями из копирумого массива, а остальные будут выглядеть как при обычном определении массива (значения по умолчанию или `null`).

```
int[] growCopiedNums = Arrays.copyOf(nums, 2 * nums.length());
```

Двумерный массив

Двумерный массив это массив одномерных массивов

```
String[][] arr = new String[2][3]; //В 1й строке число строк, во 2 число столбов таблицы

arr[0][0] = "1";
arr[0][1] = "Васька";
arr[0][2] = "121987102";

arr[1][0] = "2";
arr[1][1] = "Рыжик";
arr[1][2] = «2819876107";
```

	0	1	2
0	1	Васька	**121987102**
1	2	Рыжик	**2819876107**

Один из вариантов инициализации:

```
int[][] a = {{1, 2, 3}, {4, 0, 0}};
```

Ряды в многомерном массиве могут содержать разное число столбцов

```
int[][] twoD = new int[3][];  // память под первое измерение
twoD[0] = new int[4];         // далее резервируем память под второе измерение
twoD[1] = new int[4];
twoD[2] = new int[4];
```

Условный оператор if

Самая простая форма. Выполнится, если логический оператор = true.

```
if(логический оператор)
System.out.println("Какой то кот, или все таки код?»);
```

Самая простая форма с блоком кода

```
if(логический оператор) {
    System.out.println("Какой то кот, или все таки код?");
    System.out.println("Больше кода");
    System.out.println("Я написал эту строку потому, что три строки
нагляднее чем две");
}
```

Условный оператор if

Немного практики

```
if (2 * 2 == 5) {
    System.out.println("А как на счет математики?");
}
```

Чтобы все сошлось

```
if (2 * 2 == 5) {
    System.out.println("А как на счет математики?");
} else{
    System.out.println("Уфф, не все потеряно");
}
```

If внутри if, а это вообще законно?

Внутри блока кода `if` или `else` могут быть любые другие операторы

```
if(i == 10)
{
    if(j < 20) a = b;
    if(k > 100) c = d;
    else a = c;  // else относится к if(k > 100)
}
else a = d;
```

```
if (i == 10) {
    a = c;
} else
    if (j < 20) a = d;
        else c = d;
```

Else всегда относится к ближайшему возможному if

Switch

Switch – оператор выбора.

Switch - сравнивает то, что в него подано, с заданным списком вариантов.
Case - варианты выбора, с кодом, который будет выполнен в случае совпадения
Default - вариант, который будет выполнен по умолчанию (не обязателен)

```
int month = 3;
String monthString;
switch (month) {
    case 1:  monthString = "Январь";
        break;
    case 2:  monthString = "Февраль";
        break;
    case 3:  monthString = "Март";
        break;
   . . . . . .
        break;
    case 12: monthString = "Декабрь";
        break;
    default: monthString = "Не знаем такого";
        break;
}
```

Оператор `switch` проверяет варианты сверху вниз и исполняет, в случае совпадения.
`break` - прекращает дальнейшую проверку.
Нужен, чтобы не попасть в `default`, если совпадение найдено.

Switch

Если код в блоках `case` совпадает, их можно объединить:

```
switch (month) {
    case 12:
    case 1:
    case 2:
        season = "Зима";
        break;
    case 3:
    case 4:
    case 5:
        season = "Весна";
        break;
    case 6:
    case 7:
    case 8:
        season = "Лето";
        break;
    case 9:
    case 10:
    case 11:
        season = "Осень";
        break;
    default:
        season = "А что, так можно было?";
}
```

Еще немного важной информации

В качестве параметра в `switch` можно использовать:

- **byte**
- **short**
- **char**
- **int**
- **Enum**
- **String (JDK 7)**

Циклы

Цикл позволяет выполнить оператор или группу операторов несколько раз

- while
- do- while
- for
- «улучшенный for»

While

Самый распространенный вид цикла. Есть во всех языках, где вообще есть циклы.

Цикл выполняется до тех пор пока условие истинно

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

```
int counter = 10;
while (counter > 0) {
    System.out.println("Осталось " + counter + " сек.");
    counter--;
}
```

```
boolean isHungry; // голоден ли кот
isHungry = true; // где вы видали сытого кота?
while(!isHungry) {
    //Сюда мы никогда не зайдем
    System.out.println("Случилось чудо - кот не голоден");
}
```

```
boolean isHungry; // голоден ли кот
isHungry = true;
while(isHungry) {
    // бесконечный цикл
    System.out.println("Кот голодный...");
    //сколько кота ни корми... а он голодный
}
```

do - while

Аналогичен `while` во всем. Условие продолжение проверяется после выполнения блока кода. Один раз блок кода точно выполнится.

Используется ооооочень редко.

```
do {
    // код внутри цикла
}
while (условие);
```

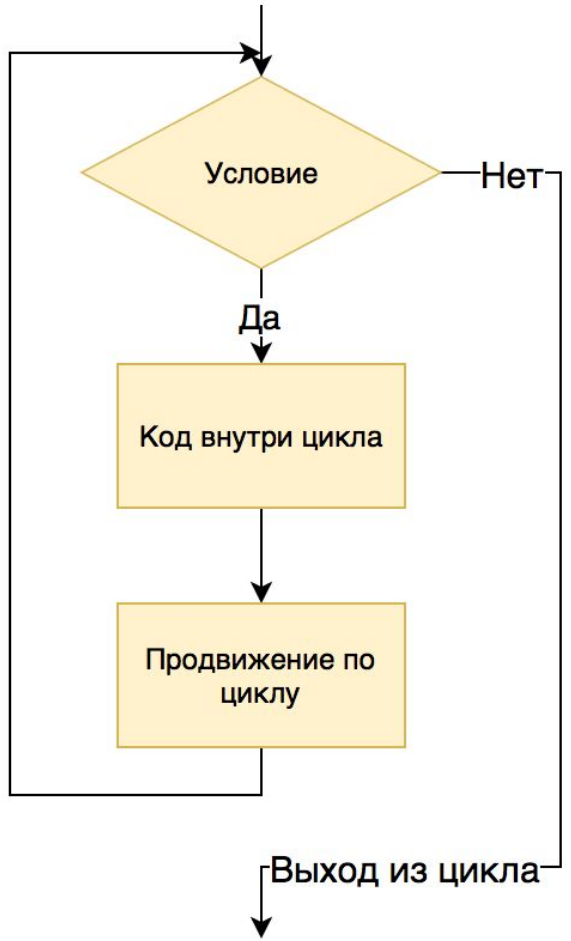
```
int counter = 10
do {
    System.out.println("Осталось " + counter + "
сек.");
} while (--counter > 0);
```

for

```
for(инициализация; логическое выражение (условие); шаг (итерация))
    команда
```

Инициализация - действие, выполняемое перед началом цикла
Логическое выражение - условие выхода из цикла, выполняется пока истина, проверяется перед итерацией
Шаг - действие, выполняемое после итерации цикла

```
for (int x = 0; x < 9; x = x + 1)
    System.out.println("Значение x: " + x);
```



Любое из этих действий можно пропустить

```
int i = 0;
for(; i < 10; i++){
    System.out.println(i);
}

for(;;){
    System.out.println("Здесь будет бесконечный цикл");
}
```

Еще немного про for

Цикл `for` может работать сразу с несколькими переменными в блоке инициализации и блоке изменения.
Они разделяются между собой запятой.

```
int a, b; //Инициализировать их нужно отдельно
for(a = 1, b = 4; a < b; a++, b--) {
    System.out.println("a = " + a);
    System.out.println("b = " + b);
}
```

Пример работы с массивом

```
String[] catName = {"Васька", "Рыжик", "Борис", "Мармелад"};
for (int i = 0; i < catName.length; i++){
    System.out.println(catName[i]);
}
```

«Улучшенный for»

Аналог `foreach` в других языках. Нужен для перебора всех элементов массива

```
String[] catName = {"Васька", "Рыжик", "Борис", "Мармелад"};
for (String name : catName){
    System.out.println(name);
}
```

`name` - переменная, в которую будет помещен элемент массива
`catName` - массив

Операторы управления циклами

`break` - завершает выполнение цикла

```
for(int i = 0; i < 100; i++) {
    if(i == 5) break; // выходим из цикла, если i равно 5
    System.out.println("i: " + i);
}
System.out.println("Цикл завершён”);
```

`continue` - прерывает выполнение текущей итерации цикла

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) //Остаток от деления на 2 равен 0
        continue;
    System.out.println(i); //Выведет только нечетные числа
}
```

Операторы управления циклами

Важно помнить, эти операторы влияют только на цикл, непосредственно внутри которого находятся

```
for (int i = 1; i < 4; i++) {
    System.out.println("Проход " + i + ": ");
    for (int j = 0; j < 100; j++) {
        if (j == 5)
            break; // выходим из цикла, если j равно 5
        System.out.print(j + " ");
    }
}
```

Проход 1: 0 1 2 3 4
Проход 2: 0 1 2 3 4
Проход 3: 0 1 2 3 4

String - особые массивы

Массив char

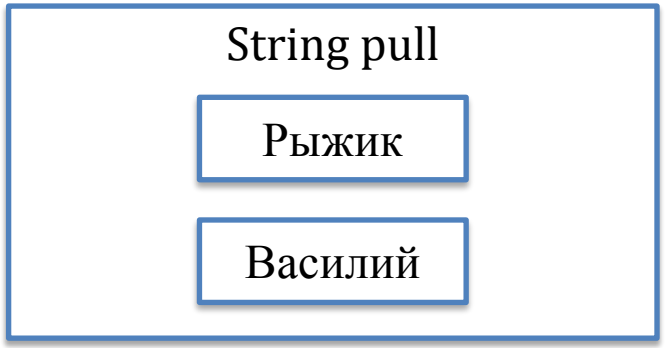
```
char[] word = {'П', 'р', 'и', 'в', 'е', 'т'};
String hello = new String(word);
```

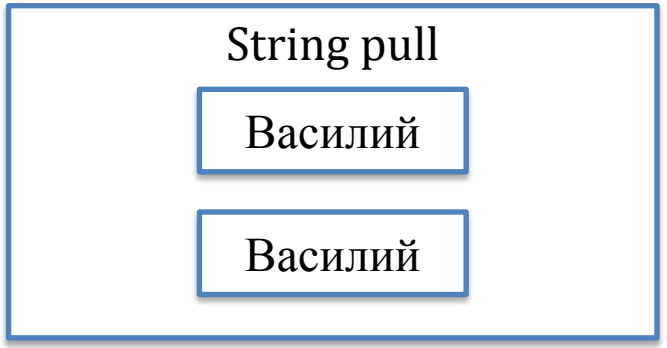
- Неизменяемый (immutable)
- Однажды созданную в памяти строку невозможно изменить
- Совсем никак

String - в чем же отличие

```
String catName1 = "Рыжик";
String catName2 = "Василий";
String catName3 = "Василий";
```

При такой инициализации java проверяет наличие строки в пуле строк, и если она уже есть то ссылается на существующую.
Пулл строк находится в heap (куче), там же, где и все другие объекты, но работает по другим принципам





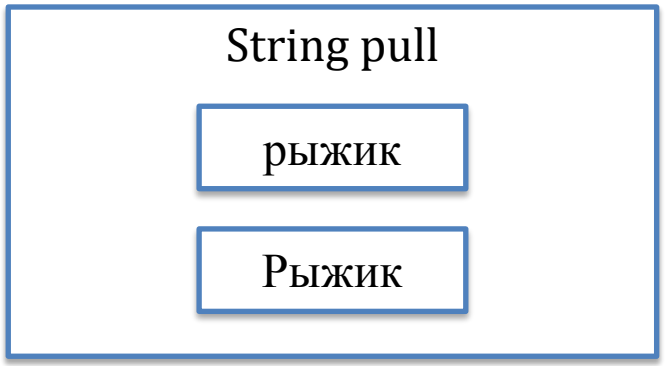
Хочу новый объект

```
String catName2 = "Василий";
String catName3 = new String("Василий");
```

String immutable

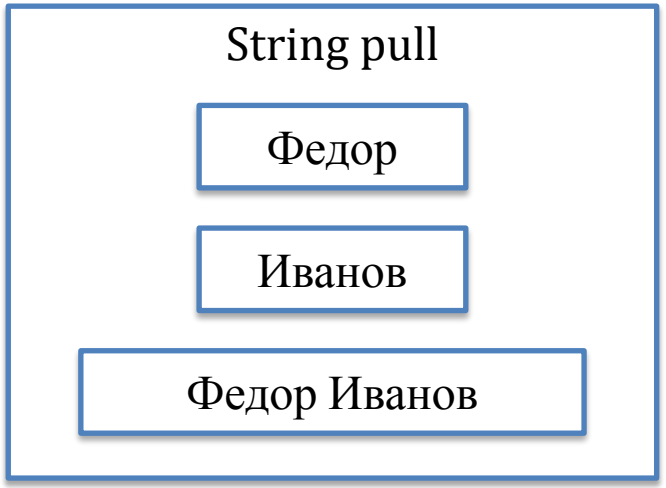
Изменяя строку, мы создаем еще один экземпляр строки

```
String catName = "рыжик";
catName = "Рыжик";
```



Соединяя строки мы тоже создаем еще один экземпляр

```
String fullName = "Федор ";
String lastName = "Иванов";
fullName = fullName + lastName;
```



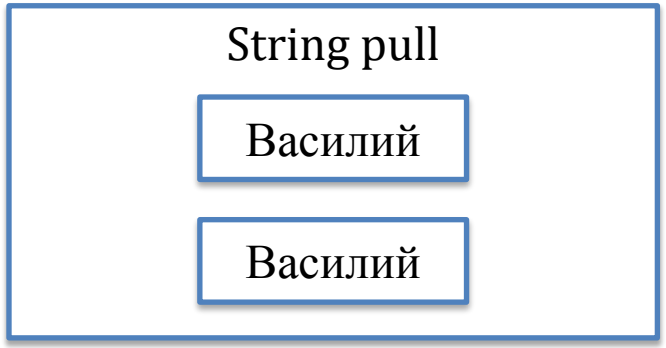
Сравнение строк == или equals?

== – сравнение по ссылке

equals – сравнение по значению

Ссылки на разные объекты не равны

```
String catName2 = "Василий";
    String catName3 = new String("Василий");
    if (catName2 == catName3)
        System.out.print("Ссылки равны");
    else
        System.out.print("Ссылки не равны");
}
```



Значения равны

```
String catName2 = "Василий";
    String catName3 = new String("Василий");
    if (catName2.equals(catName3))
        System.out.print("Значения равны");
    else
        System.out.print("Значения не равны");
}
```

Домашнее задание

Реализовать алгоритм сортировки массива int из представленных на выбор (Selection sort, Gnome sort, Cocktail sort)

Если хочется посложнее (Quick sort, Comb sort), будет в плюс

Алгоритмы можно посмотреть на сайте http://algolab.valemak.com.Там же есть реализации на java.