Базовый синтаксис Java

Алексей Владыкин

15 сентября 2014

1. Система типов Java
   - Примитивные типы
   - Преобразование типов
   - Ссылочные типы
2. Управляющие конструкции
   - Условные операторы
   - Циклы
3. Примеры

Примитивные типы

- `boolean`
- `char`
- `byte`, `short`, `int`, `long`
- `float`, `double`

Примитивные типы

- `boolean`
- `char`
- `byte`, `short`, `int`, `long`
- `float`, `double`

- Зарезервированные ключевые слова языка
- Не имеют полей и методов
- Передаются по значению

Ссылочные типы

- Все остальные
- Являются объектами (наследуют `java.lang.Object`)
- Имеют поля и методы
- Передаются по ссылке

Тип boolean

- Литералы: `false`, `true`
- Результат любого сравнения — `boolean`:

```
<    >    ==
<=   >=   !=
```

- Нет преобразования между `boolean` и другими примитивными типами

Логические операции

and	&&	&	&=
or	||	|	|=
xor	^		^=
not	!		

Логические операции

and	&&	&	&=
or	||	|	|=
xor	^		^=
not	!		

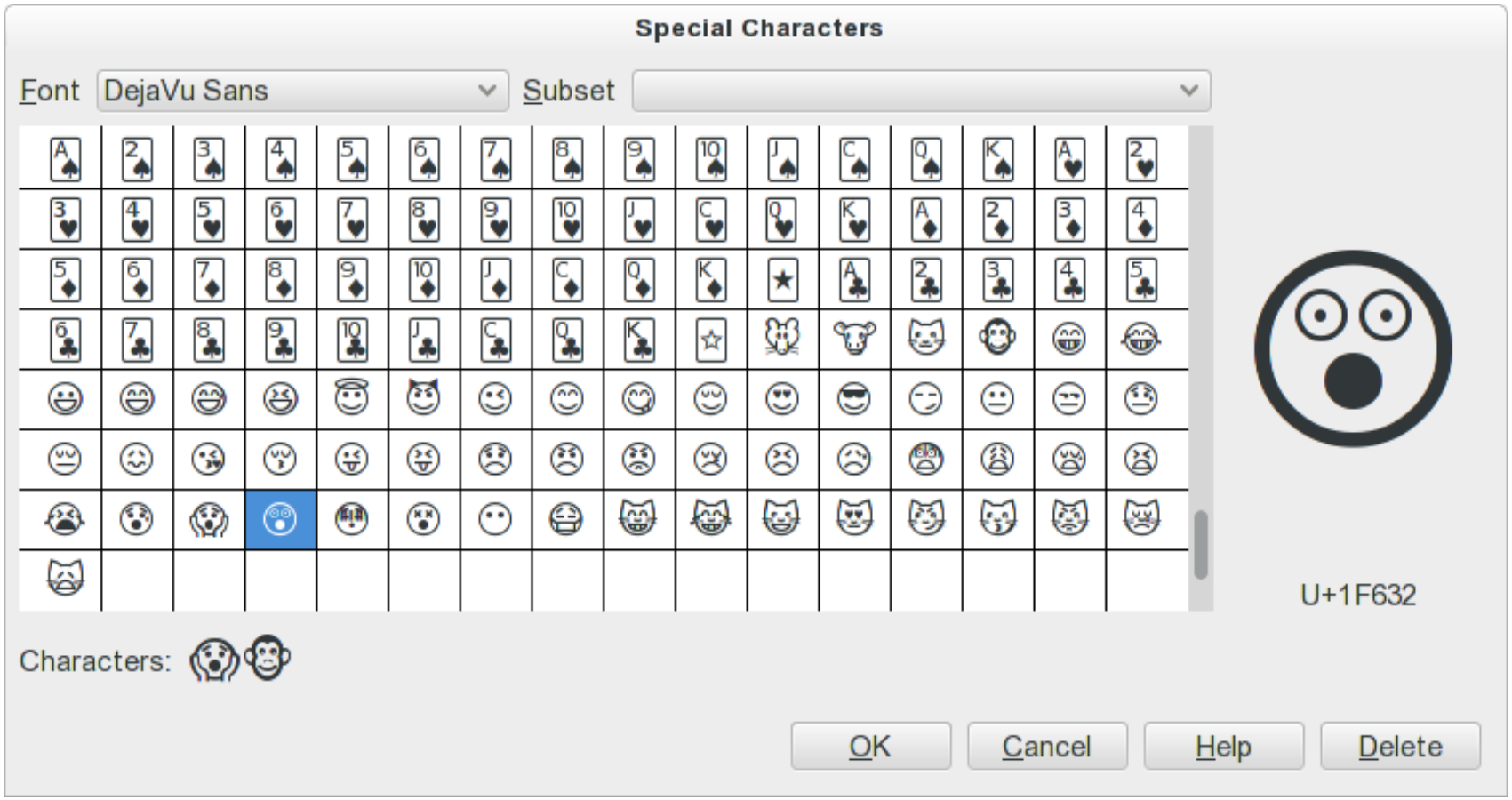
- && и || — вычисление по сокращенной схеме
- & и | — вычисление по полной схеме

java.lang.Boolean

- Класс-обертка для `boolean`

- `boolean parseBoolean(String)`
- `String toString(boolean)`

Тип char


Special Characters
Font
DejaVu Sans
Subset
U+1F632
Characters:
OK
Cancel
Help
Delete

Символьные значения

- `char` — 16 бит, беззнаковый

Символьные значения

- `char` — 16 бит, беззнаковый $(0 .. 2^{16} - 1)$
- Представляет номер символа в кодировке Unicode

Символьные значения

- `char` — 16 бит, беззнаковый $(0 .. 2^{16} - 1)$
- Представляет номер символа в кодировке Unicode
- Литералы:
  - символ в одинарных кавычках: `'a'`
  - шестнадцатеричный код символа: `'\u78bc'`
  - спецпоследовательности: `'\t'`, `'\n'`, `'\r'`, `'\''`, `'\\'`

Символьные значения

- `char` — 16 бит, беззнаковый $(0 .. 2^{16} - 1)$
- Представляет номер символа в кодировке Unicode
- Литералы:
  - символ в одинарных кавычках: `'a'`
  - шестнадцатеричный код символа: `'\u78bc'`
  - спецпоследовательности: `'\t'`, `'\n'`, `'\r'`, `'\''`, `'\\'`
- Свободно конвертируется в числовые типы и обратно

Суррогаты

- В современном Unicode больше символов, чем влезает в 16 бит
- Поэтому некоторые Unicode-сиволы в Java представляются двумя `char`'ами — суррогатной парой
- Такие символы можно хранить и в `int`

java.lang.Character

- Класс-обертка для char

- `char toLowerCase(char)`
- `char toUpperCase(char)`
- `boolean isLowerCase(char)`
- `boolean isUpperCase(char)`
- `boolean isDigit(char)`
- `boolean isLetter(char)`
- `boolean isHighSurrogate(char)`
- `boolean isLowSurrogate(char)`

Целочисленные типы

Диапазоны значений

Тип	Бит	Диапазон
`byte`	8	$-128 \,..\, +127$
`short`	16	$-2^{15} \,..\, +2^{15}-1$
`int`	32	$-2^{31} \,..\, +2^{31}-1$
`long`	64	$-2^{63} \,..\, +2^{63}-1$

Диапазоны значений

Тип	Бит	Диапазон
`byte`	8	$-128 \,..\, +127$
`short`	16	$-2^{15} \,..\, +2^{15}-1$
`int`	32	$-2^{31} \,..\, +2^{31}-1$
`long`	64	$-2^{63} \,..\, +2^{63}-1$

- Размер фиксирован, одинаков для всех платформ
- Все типы знаковые, беззнаковых вариантов нет

Литералы

- Десятичное число: 123
- Восьмеричное число: 0123
- Шестнадцатеричное число: 0x123
- Двоичное число: 0b101 (с Java 7)

Литералы

- Десятичное число: `123`
- Восьмеричное число: `0123`
- Шестнадцатеричное число: `0x123`
- Двоичное число: `0b101` (с Java 7)

- С подчеркиванием: `123_456_789` (с Java 7)
- С суффиксом `L` для `long`

Арифметические операции

сложение	+	+=
вычитание	-	-=
умножение	*	*=
деление	/	/=
остаток	%	%=
инкремент	++	
декремент	--	

Арифметические операции

сложение	+	+=
вычитание	-	-=
умножение	*	*=
деление	/	/=
остаток	%	%=
инкремент	++	
декремент	--	

- Деление целочисленное

Особые случаи

- Деление на ноль — исключительная ситуация, бросается `ArithmeticException`
- Переполнение не является исключительной ситуацией, лишние старшие биты просто выкидываются

Побитовые операции

not	~	
and	&	&=
or	|	|=
xor	^	^=
shr	>>	>>=
shr'	>>>	>>>=
shl	<<	<<=

Побитовые операции

not	~	
and	&	&=
or	|	|=
xor	^	^=
shr	>>	>>=
shr'	>>>	>>>=
shl	<<	<<=

- >> — арифметический сдвиг
- >>> — логический сдвиг

Классы-обертки

- java.lang.Byte
- java.lang.Short
- java.lang.Integer
- java.lang.Long

- `MIN_VALUE`
- `MAX_VALUE`
- `toString(`*typename*`)`
- `parse`*Typename*`(String)`
- `bitCount(`*typename*`)`
- `reverse(`*typename*`)`
- `reverseBytes(`*typename*`)`

Вещественные типы

- Стандарт IEEE754
- Число представлено в виде $\pm m \cdot 2^e$

Тип	Бит	Знак	Мантисса	Экспонента
`float`	32	1	23	8
`double`	64	1	52	11

Литералы

- Обычная запись: `-1.234`
- Экспоненциальная запись: `-123.4e-2` ($-123.4 \cdot 10^{-2}$)
- Шестнадцатеричная запись: `0xFFFFpFF` ($FFFF \cdot 2^{FF}$)
- С суффиксом типа:
  - `38f`
  - `3e19d`
  - `123.4e-2f`
  - `444.444d`

Операции

сложение	+	+=
вычитание	-	-=
умножение	*	*=
деление	/	/=
остаток	%	%=
инкремент	++	
декремент	--	

- Побитовые операции не поддерживаются

Особые случаи

- Деление положительного числа на 0 дает $+\infty$
- Деление отрицательного числа на 0 дает $-\infty$
- Деление 0 на 0 дает NaN

Особые случаи

- Деление положительного числа на 0 дает $+\infty$
- Деление отрицательного числа на 0 дает $-\infty$
- Деление 0 на 0 дает NaN

- Переполнение дает $+\infty$ или $-\infty$,
  в зависимости от направления

Особые случаи

- Деление положительного числа на 0 дает $+\infty$
- Деление отрицательного числа на 0 дает $-\infty$
- Деление 0 на 0 дает NaN

- Переполнение дает $+\infty$ или $-\infty$,
  в зависимости от направления

- Любая арифметическая операция с NaN дает NaN
- NaN != NaN

strictfp

- Java использует FPU для вычислений с плавающей точкой
- Регистры FPU могут быть шире 64 бит
- Результаты вычислений могут отличаться

strictfp

- Java использует FPU для вычислений с плавающей точкой
- Регистры FPU могут быть шире 64 бит
- Результаты вычислений могут отличаться

- Модификатор `strictfp` включает режим строгой совместимости, результаты будут идентичны на любом процессоре

Классы-обертки

- java.lang.Float
- java.lang.Double

- `MIN_VALUE`
- `MAX_VALUE`
- `POSITIVE_INFINITY`
- `NEGATIVE_INFINITY`
- `NaN`
- `boolean isNaN(`*typename*`)`
- `toString(`*typename*`)`
- `parse`*Typename*`(String)`

java.lang.Math

- константы: `E`, `PI`
- тригонометрия: `sin`, `cos`
- степени: `sqrt`, `pow`, `exp`
- `min`, `max`
- ...


KEEP
CALM
AND
CALL
BATMAN

Неявное преобразование типов

- Преобразование целочисленных типов в более емкие (`byte` → `short` → `int` → `long`)

Неявное преобразование типов

- Преобразование целочисленных типов в более емкие (`byte` → `short` → `int` → `long`)
- Преобразование `char` в `int` и `long`

Неявное преобразование типов

- Преобразование целочисленных типов в более емкие (`byte` → `short` → `int` → `long`)
- Преобразование `char` в `int` и `long`
- Преобразование целочисленные типов в типы с плавающей точкой (возможна потеря точности)

Явное преобразование типов

- Оператор приведения типа: (*typename*)

Явное преобразование типов

- Оператор приведения типа: (*typename*)
- При приведении более емкого целого типа к менее емкому старшие биты просто отбрасываются

Явное преобразование типов

- Оператор приведения типа: `(`*`typename`*`)`
- При приведении более емкого целого типа к менее емкому старшие биты просто отбрасываются
- При приведении типа с плавающей точкой к целому типу дробная часть отбрасывается (никакого округления)

Явное преобразование типов

- Оператор приведения типа: `(`*`typename`*`)`
- При приведении более емкого целого типа к менее емкому старшие биты просто отбрасываются
- При приведении типа с плавающей точкой к целому типу дробная часть отбрасывается (никакого округления)
- Слишком большое дробное число при приведении к целому превращается в `MAX_VALUE` или `MIN_VALUE`

Явное преобразование типов

- Оператор приведения типа: `(`*`typename`*`)`
- При приведении более емкого целого типа к менее емкому старшие биты просто отбрасываются
- При приведении типа с плавающей точкой к целому типу дробная часть отбрасывается (никакого округления)
- Слишком большое дробное число при приведении к целому превращается в `MAX_VALUE` или `MIN_VALUE`
- Слишком большой `double` при приведении к `float` превращается в `Float.POSITIVE_INFINITY` или `Float.NEGATIVE_INFINITY`

Автоматическое расширение

- При вычислении выражения (`a @ b`) аргументы `a` и `b` преобразовываются в числа, имеющие одинаковый тип:
  - если одно из чисел double, то в `double`;
  - иначе, если одно из чисел `float`, то в `float`;
  - иначе, если одно из чисел `long`, то в `long`;
  - иначе оба числа преобразуются в `int`.

```
byte b = 1;
byte c = b + 1; // compilation error
```

Неявное приведение с потерей данных

- Сокращенная запись `var @= expr`
  раскрывается в `var = (`*typename*`) (var @ (expr))`
- Неявно срабатывает приведение типа, в том числе с потерей данных

```
short n = -1;
while (n != 0) {
    n >>>= 1;
}
```

Boxing/unboxing

- Autoboxing: примитивное значение $\rightarrow$ объект-обертка
- Autounboxing: объект-обертка $\rightarrow$ примитивное значение

```
Integer i = 1;
Integer j = i + 1;
int k = i + j;
```

Ссылочные типы

- Все остальные, кроме примитивных
- Передаются по ссылке
- Являются объектами (наследуют `java.lang.Object`)
- Имеют поля и методы
- Ссылка может принимать значение `null`

Массивы

- Массив обозначается квадратными скобками

```
int[] numbers;
String[] args;
boolean bits[];
char[] letters, digits;
float rates[], maxRate;
```

Создание

- Массив создается оператором `new`
- Все элементы массива инициализируются нулями
- Размер массива фиксируется в момент создания

```
int[] numbers = new int[100];
String[] args = new String[1];
boolean[] bits = new boolean[0];
```

Инициализация

- Можно перечислить значения всех элементов при создании массива

```
int[] numbers = new int[] {1, 2, 3, 4, 5};
boolean[] bits = new boolean[] {true, true, false};

// this works only in variable declaration
char[] digits = {
        '0', '1', '2', '3', '4',
        '5', '6', '7', '8', '9'};
```

Индексация

- Элементы индексируются с нуля
- Длина массива доступна как `array.length`
- При выходе за границы массива бросается исключение

```
int[] numbers = {1, 2, 3, 4, 5};
// numbers.length -> 5
// numbers[0] -> 1
// numbers[1] -> 2
// numbers[4] -> 5
// numbers[5] -> ArrayIndexOutOfBoundsException
```

Многомерные массивы

- Многомерный массив — это массив массивов

```
int[][] matrix0;
int[][] matrix1 = new int[2][2];
int[][] matrix2 = {{1, 2}, {3, 4}};
int[] row = matrix2[0]

// matrix2[1][1] -> 4
// row[0] -> 1
```

Многомерные массивы

- Разрешены ступенчатые массивы

```
int[][] triangle = {
        {1, 2, 3, 4, 5},
        {6, 7, 8, 9},
        {10, 11, 12},
        {13, 14},
        {15}};
// triangle.length -> 5
// triangle[0].length -> 5
// triangle[4].length -> 1
```

Представление в памяти

- Одномерный массив занимает непрерывный участок памяти
- Двумерный массив занимает $n + 1$ участок в памяти, где n — первая размерность

```
int[][] a = new int[10][1000];
int[][] b = new int[1000][10];
```

Varargs

- Специальный синтаксис для массива аргументов
- Поддерживается с Java 5

```
int max(int[] numbers);
// usage: max(new int[] {1, 2, 3, 4});

int max(int... numbers);
// usage: max(1, 2, 3, 4);
```

Как сравнить два массива

- `a == b`
  сравнивает ссылки

Как сравнить два массива

- `a == b`
  сравнивает ссылки
- `a.equals(b)`
  сравнивает ссылки

Как сравнить два массива

- `a == b`
  сравнивает ссылки
- `a.equals(b)`
  сравнивает ссылки
- `Arrays.equals(a, b)`
  сравнивает содержимое

Как сравнить два массива

- `a == b`
  сравнивает ссылки
- `a.equals(b)`
  сравнивает ссылки
- `Arrays.equals(a, b)`
  сравнивает содержимое
- `Arrays.deepEquals(a, b)`
  сравнивает содержимое многомерных массивов

Как распечатать массив

- `System.out.println(a)`
  выводит «абракадабру» `[I@2ce83912`

Как распечатать массив

- `System.out.println(a)`
  выводит «абракадабру» `[I@2ce83912`
- `System.out.println(Arrays.toString(a))`
  выводит содержимое

Как распечатать массив

- `System.out.println(a)`
  выводит «абракадабру» `[I@2ce83912`
- `System.out.println(Arrays.toString(a))`
  выводит содержимое
- `System.out.println(Arrays.deepToString(a))`
  выводит содержимое многомерных массивов

java.util.Arrays

- copyOf, copyOfRange
- fill
- sort
- binarySearch

- java.lang.System.arraycopy

Строки

- Класс `java.lang.String`
- Последовательность символов произвольной длины в кодировке UTF-16
- Строка — это не `char[]`, хотя есть способы конвертации
- Никаких нулевых символов в конце, длина хранится отдельно

Создание строк

- Строковые литералы

```
String zeros = "\u0000\u0000";
String hello = "Hello";
String specialChars = "\r\n\t\"\\";
String unicodeEscapes = "\u0101\u2134\u03ff";
```

Создание строк

- Строковые литералы

```
String zeros = "\u0000\u0000";
String hello = "Hello";
String specialChars = "\r\n\t\"\\";
String unicodeEscapes = "\u0101\u2134\u03ff";
```

- Создание из массива символов

```
char[] charArray = {'a', 'b', 'c'};
String string = new String(charArray);
```

Доступ к содержимому строки

- Строки неизменяемы
- `int length()`
- `char charAt(int index)`
- `char[] toCharArray()`
- `String substring(int beginIndex)`
  `String substring(int beginIndex, int endIndex)`

Конкатенация строк

- Оператор +

```
String helloWorld = "Hello" + " World!";
```

Конкатенация строк

- Оператор +

```
String helloWorld = "Hello" + " World!";
```

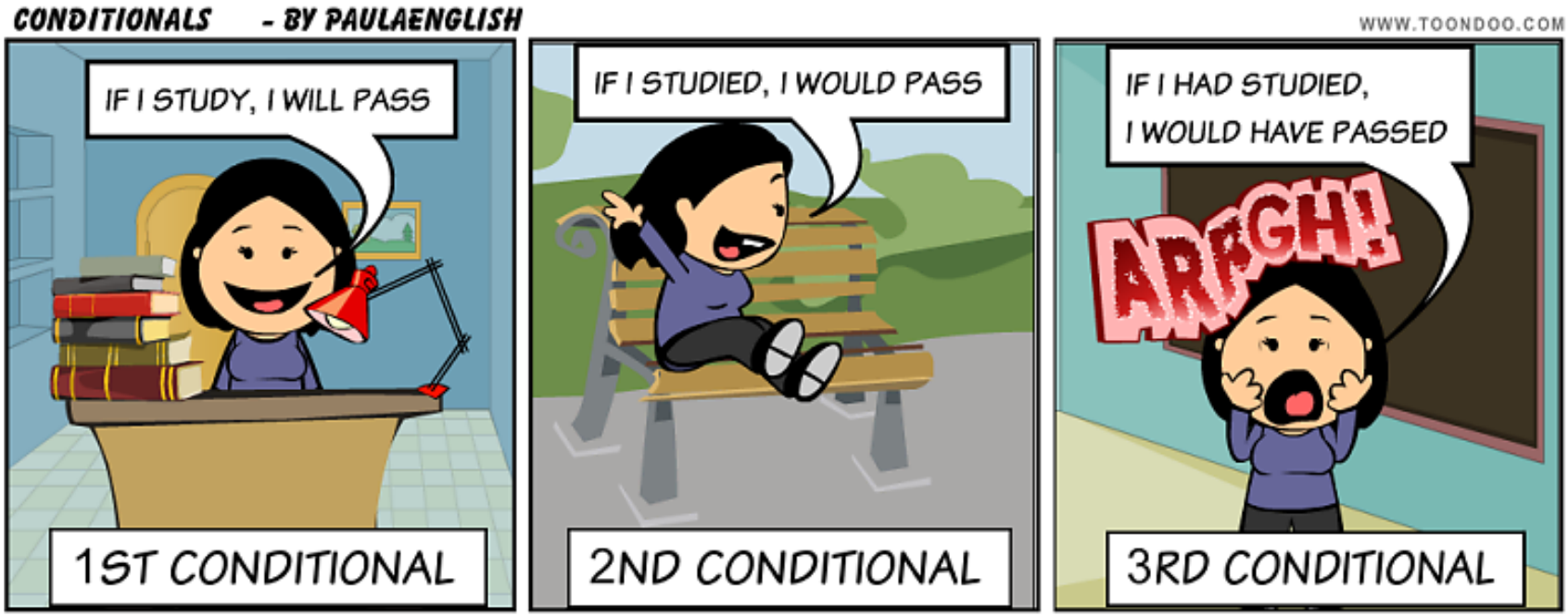
- java.lang.StringBuilder

```
StringBuilder buf = new StringBuilder();
buf.append("Hello");
buf.append(" World!");
String result = buf.toString();
```

- Компилятор преобразует + в операции с StringBuilder

Сравнение строк

- Оператор == сравнивает ссылки, а не содержимое строки
- `boolean equals(Object anObject)`
  `boolean equalsIgnoreCase(String anotherString)`
- `int compareTo(String anotherString)`
  `int compareToIgnoreCase(String anotherString)`


CONDITIONALS - BY PAULAENGLISH
WWW.TOONDOO.COM
IF I STUDY, I WILL PASS
1ST CONDITIONAL
IF I STUDIED, I WOULD PASS
2ND CONDITIONAL
IF I HAD STUDIED,
I WOULD HAVE PASSED
ARRGH!
3RD CONDITIONAL

Оператор if

```
if (weatherIsGood) {
    walkInThePark();
} else {
    readBooksAtHome();
}
```

- Есть вариант без `else`
- Условие типа `boolean`, нельзя написать `if (i = 3) { ... }`
- Фигурные скобки рекомендуется ставить, даже когда они необязательны

Оператор ?:

```
if (weatherIsGood) {
    System.out.println("Weather is good");
} else {
    System.out.println("Weather is bad");
}

// same effect, but much shorter
System.out.println("Weather is "
        + (weatherIsGood ? "good" : "bad"));
```

Оператор switch

```
switch (digit) {
    case 0:
        text = "zero";
        break;

    case 1:
        text = "one";
        break;

    // case 2 - case 9

    default:
        text = "???";
}
```

- Без `break` исполнение продолжается
- Работает для примитивных типов `byte`, `short`, `char`, `int`, а также для `enum`
- В Java 7 добавлен `switch` для `String`

```
if (digit == 0) {
    text = "zero";
} else if (digit == 1) {
    text = "one";
} else if (digit == 2) {
    text = "two";
} /* 3 - 9 */
else {
    text = "???";
}
```


I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk
I will not waste chalk

Цикл while

```
while (haveTime() && haveMoney()) {
    goShopping();
}
```

- Цикл с предусловием
- Условие типа boolean
- Фигурные скобки рекомендуется ставить, даже когда они необязательны

Цикл do while

```
do {
    goShopping();
} while (haveTime() && haveMoney());
```

- Цикл с постусловием
- Нужна точка с запятой в конце
- Фигурные скобки рекомендуется ставить, даже когда они необязательны

Цикл for

```
for (int i = 0; i < args.length; i++) {
    System.out.println(args[i]);
}
```

- Все части заголовка не обязательны
- `for (;;)` — бесконечный цикл
- Фигурные скобки рекомендуется ставить, даже когда они необязательны

Цикл foreach

```
for (String arg : args) {
    System.out.println(arg);
}
```

- Добавлен в Java 5
- Применим к массивам и классам, реализующим интерфейс `java.lang.Iterable`

Оператор break

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

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

Оператор continue

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

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

Метки

- Операторы `break` и `continue` действуют на ближайший цикл
- Можно указать другой цикл при помощи метки

```
outer:
for (int[] row : matrix) {
    for (int x : row) {
        if (x > 100) {
            found = true;
            break outer;
        }
    }
}
```

```
public class Fibonacci {

    public static long getFibonacciNumber(int n) {
        if (n <= 0) {
            return 0;
        }
        long prev = 0;
        long curr = 1;
        for (int i = 1; i < n; ++i) {
            long next = prev + curr;
            prev = curr;
            curr = next;
        }
        return curr;
    }

    public static void main(String[] args) {
        for (int i = 0; i <= Integer.parseInt(args[0]); ++i) {
            System.out.printf("fib(%d) = %d\n",
                    i, getFibonacciNumber(i));
        }
    }
}
```

```
import java.math.BigInteger;

public class FibonacciBigInteger {

    public static BigInteger getFibonacciNumber(int n) {
        if (n <= 0) {
            return BigInteger.ZERO;
        }
        BigInteger prev = BigInteger.ZERO;
        BigInteger curr = BigInteger.ONE;
        for (int i = 1; i < n; ++i) {
            BigInteger next = prev.add(curr);
            prev = curr;
            curr = next;
        }
        return curr;
    }

    public static void main(String[] args) {
        for (int i = 0; i <= Integer.parseInt(args[0]); ++i) {
            System.out.printf("fib(%d) = %d\n",
                    i, getFibonacciNumber(i));
        }
    }
}
```

```
import java.util.Arrays;

public class Anagrams {

    public static boolean areAnagrams(String a, String b) {
        char[] charsFromA = getSortedChars(a);
        char[] charsFromB = getSortedChars(b);
        return Arrays.equals(charsFromA, charsFromB);
    }

    private static char[] getSortedChars(String s) {
        char[] chars = s.toCharArray();
        Arrays.sort(chars);
        return chars;
    }

    public static void main(String[] args) {
        System.out.println(areAnagrams("silent", "listen")
                ? "anagrams"
                : "not anagrams");
    }
}
```

```
public class Palindromes {

    public static boolean isPalindrome(String s) {
        String normalizedText = normalize(s);
        return normalizedText.equals(reverse(normalizedText));
    }

    private static String normalize(String s) {
        return s.toLowerCase().replaceAll("\\W+", "");
    }

    private static String reverse(String s) {
        return new StringBuilder(s).reverse().toString();
    }

    public static void main(String[] args) {
        System.out.println(isPalindrome("Madam, I'm Adam")
                ? "palindrome" : "not palindrome");
    }
}
```

```
public class Polygons {

    public static double getArea(double[][] polygon) {
        int size = polygon.length;
        double sum = 0;
        for (int i = 0; i < size; ++i) {
            int j = (i + 1) % size;
            sum += det(polygon[i][0], polygon[i][1],
                       polygon[j][0], polygon[j][1]);
        }
        return Math.abs(sum / 2);
    }

    private static double det(double x1, double y1,
                              double x2, double y2) {
        return x1 * y2 - x2 * y1;
    }

    public static void main(String[] args) {
        double[][] polygon = new double[][] {
                {1, 1}, {1, 2}, {2, 2}, {2, 1}};
        System.out.printf(
                "Polygon area = %1.3f\n", getArea(polygon));
    }
}
```

Что сегодня узнали

- Что такое «примитивные» и «ссылочные типы»
- Как на Java производить логические и арифметические вычисления
- Какие есть подводные камни
- Как создавать и использовать массивы
- Как создавать и использовать строки
- Как в Java написать ветвление и цикл, какие есть разновидности