Библиотека классов Java

Croc Java School

Библиотека классов Java

Преимущественно реализована на Java.

Классы упакованы в монолитный архив rt.jar (до jigsaw).

Java 9+ разбита на модули:

```
java --list-modules
```

В Java 11 4K+ классов

https://docs.oracle.com/en/java/javase/11/docs/api/

Основные пакеты

```
java.lang
java.io, java.nio
java.net
java.math
java.util, java.util.concurrent, java.util.function
java.time
java.sql
java.text
java.security, javax.crypto
java.awt, java.swing, javax.imageio
...
```

System: стандартные потоки ввода-вывода

Стандартные потоки процесса

`System.in`

`System.out`

`System.err`

System: завершение процесса

Завершение процесса без ошибки

```
System.exit(0); // EXIT_SUCCESS
```

Завершение с кодом отличным от нуля сигнализирует об ошибке

```
System.exit(1);
```

System: время

Текущее время в формате POSIX time в миллисекундах (с 01.01.1970)

```
long millis = System.currentTimeMillis();
```

Время в наносекундах относительно фиксированной в рамках инстанса JVM точки отсчета (возможно, фиксированной в будущем)

```
long t = System.nanoTime();
```

System.arrayCopy()

```
int[] a = new int[] { 1, 2, 3, 4, 5, 6, 7 };
System.arraycopy(
        a, 1,
        a, 3,
        4
);
System.out.println(Arrays.toString(a));
```

```
[1, 2, 3, 2, 3, 4, 5]
```

Arrays

```
int[] array = new int[] { 5, 6, 7, 1, 2, 3 };
System.out.println(array); // [I@129a8472
System.out.println(Arrays.toString(array)); // [5, 6, 7, 1, 2, 3]
```

Бинарный поиск

```
int[] array = new int[] { 5, 6, 7, 1, 2, 3 };

int index = Arrays.binarySearch(array, 4); // index = -1
Arrays.sort(array);
int index = Arrays.binarySearch(array, 4); // index = -4

-4 = -insertIndex - 1
insertIndex = 4 - 1 = 3

[1, 2, 3, 5, 6, 7]
```

Копирование массива

```
int[] array = new int[] { 5, 6, 7, 1, 2, 3 };
array = Arrays.copyOf(array, array.length + 1);
```

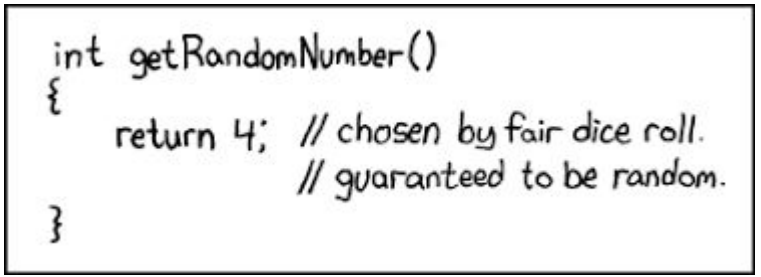
```
[5, 6, 7, 1, 2, 3, 0]
```

Вставка элемента с сохранением сортировки

```
int[] array = new int[] { 1, 2, 3, 5, 6, 7 };
int key = 4;

int i = Arrays.binarySearch(array, key);
if (i < 0) {
   i = -i - 1;
   array = Arrays.copyOf(array, array.length + 1);
   System.arraycopy(array, i, array, i + 1, array.length - i - 1);
   array[i] = key;
}

[1, 2, 3, 4, 5, 6, 7]
```

Random

```
Random rnd = new Random(System.currentTimeMillis());

int a = rnd.nextInt();
```



Типы-обертки

```
byte    - Byte
short   - Short
int     - Integer
long    - Long
float   - Float
double  - Double
char    - Character
boolean - Boolean
          +
          Number
          Void
```

Неизменяемые объекты

```
public class Immutable {

  private final int value;

  public Immutable(int value) {
     this.value = value;
  }

  public int getValue() {
     return value;
  }

  // no setter
}
```

Кэширование значений

Все типы-обертки неизменяемые.

Целочисленные обертки кэшируются.

Диапазон кэширования по умолчанию: [-128, 127] для `Short`, `Integer`, `Long`

Диапазон можно изменить параметром JVM:

`-XX:AutoBoxCacheMax=<size>`

У `Float` и `Double` кэша нет.

valueOf

Разные объекты:

```
Integer a = new Integer(5); // deprecated
Integer b = new Integer(5);
```

Один объект:

```
Integer a = Integer.valueOf(5);
Integer b = Integer.valueOf(5);
```

parseX

Integer:

```
public static int parseInt(String s, int radix)
    throws NumberFormatException;

int year = Integer.parseInt(“2021”); // radix 10
int year2 = Integer.parseInt("11111100101", 2);
```

equals()

Оператор == сравнивает значения переменных

- Примитивы: сами значения
- Ссылочные типы: значения ссылок (адреса)

Метод `equals()` позволяет определить сравнение по значению полей.

Сравнение типов-оберток

```
Integer a = new Integer(5);
Integer b = new Integer(5);
a == b; // false
a.equals(b); // true

Integer a = Integer.valueOf(5);
Integer b = Integer.valueOf(5);
a == b; // true
a.equals(b); // true
```

BigInteger

Неизменяемый тип для *больших* целых чисел

`[-2^Integer.MAX_VALUE, 2^Integer.MAX_VALUE)`

`Все методы-операции возвращают новые объекты.`

```
BigInteger x = new BigInteger("2021");
x.pow(2021); // x не изменился (вспоминаем про final)
x = x.pow(2021); // x изменился
```

BigDecimal

Целочисленное представление + scale (число знаков после запятой).

```
BigDecimal unitPrice = new BigDecimal("0.01");
BigDecimal total = BigDecimal.ZERO
       .setScale(2, RoundingMode.UP);
for (int i = 0; i < 10; i++) {
   total = total.add(unitPrice);
}

System.out.println("Total: " + total);
```

```
Total: 0.10
```

~~double для денег~~

```
double unitPrice = 0.01;
double total = 0.0;
for (int i = 0; i < 10; i++) {
   total += unitPrice;
}

System.out.println("Total: " + total);
```

```
Total: 0.09999999999999999
```

Форматирование цен

```
BigDecimal prize = new BigDecimal("45600000000");
NumberFormat format = NumberFormat.getCurrencyInstance(Locale.KOREA);

System.out.println(format.format(prize));
```

```
₩45,600,000,000
```

Форматирование чисел

```
NumberFormat format = NumberFormat.getNumberInstance();
format.setMaximumFractionDigits(2);
format.setRoundingMode(RoundingMode.UP);

System.out.println(format.format(2.269)); // 2.27
```

Строки

Интернирование

Строки — неизменяемые объекты.

Интернирование строк — использование общего пула при совпадении значения.

```
String vegetable = new String("Carrot"); // not interned
vegetable == "Carrot"; // false
vegetable.equals("Carrot"); // true

String vegetable = "Carrot"; // interned
vegetable == "Carrot"; // true
vegetable.equals("Carrot"); // true
```

StringBuilder

```
String vegetable = "Carrot";
StringBuilder reversed = new StringBuilder();
for (int i = vegetable.length() - 1; i>= 0; i--) {
   reversed.append(vegetable.charAt(i));
}
vegetable = reversed.toString(); // torraC
```

Выполнять конкатенацию в цикле неоптимально.

Но статические конкатенации компилятор умеет оптимизировать самостоятельно.

split/join

```
String[] tokens = "one,  two; three".split("[\\s,;]+");
String str = String.join("-", tokens); // one-two-three
```

Регулярные выражения

```
String str = "Вы потратили 50 рублей. Текущий баланс: -3566.00.";
StringBuilder blurred = new StringBuilder();

Pattern pattern = Pattern.compile("[+-]?\\d+([\\.,]\\d+)?");
Matcher matcher = pattern.matcher(str);
while (matcher.find()) {
   matcher.appendReplacement(blurred, "X");
}
matcher.appendTail(blurred);
```

Вы потратили X рублей. Текущий баланс: X.

Даты

New date/time API (java.time)

LocalDate, LocalTime, LocalDateTime

ZonedDateTime

Instant

Duration

LocalDate, LocalTime, LocalDateTime

```
LocalDate theoryDay = LocalDate.of(2021, 10, 21);
LocalDateTime homeworkDeadline = theoryDay
        .plusWeeks(1)
        .atTime(LocalTime.of(18, 0));

LocalDateTime now = LocalDateTime.now();
boolean amIOk = !now.isAfter(homeworkDeadline);
int hoursLeft = (int)Duration.between(now, homeworkDeadline)
        .toHours();
```

DateTimeFormatter

https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

```
DateTimeFormatter formatter = DateTimeFormatter
    .ofPattern("dd.MMM.yyyy");
LocalDate date = LocalDate.parse("21.Oct.2021", formatter);
while (date.getDayOfWeek() != DayOfWeek.SUNDAY) {
   date = date.plusDays(1);
}
System.out.println("Next sunday: " + date.format(formatter));
```

Таймзоны

```
LocalDate theoryDay = LocalDate.of(2021, 10, 21);
LocalDateTime homeworkDeadline = theoryDay
        .plusWeeks(1)
        .atTime(LocalTime.of(18, 0));

ZonedDateTime atMoscow  = homeworkDeadline.atZone(ZoneOffset.ofHours(3));
Object atNovosibirsk = atMoscow
        .withZoneSameInstant(ZoneId.of("Asia/Novosibirsk"));

atMoscow: 2021-10-28T18:00+03:00
atNovosibirsk: 2021-10-28T22:00+07:00[Asia/Novosibirsk]
```

Instant vs LocalDateTime vs ZonedDateTime

`Instant` соответствует конкретному моменту во времени. Технический timestamp

`LocalDateTime` определяет дату+время, но не момент (для одного экземпляра момент различается в зависимости от контекста — временной зоны).

`ZonedDateTime` как и `Instant` соответствует конкретному моменту времени и содержит информацию о временной зоне (в отличие от `Instant`).

Old date/time API

```
Date date = new Date();

Instant instant = date.toInstant();
date = Date.from(instant);
```

Сколько времени займет сортировка миллиарда чисел на вашем компьютере?*

** в одном потоке*

Сортировка массива случайных чисел

```
int[] array = new int[1_000_000_000];
Random rnd = new Random(System.currentTimeMillis());
for (int i = 0; i < array.length; i++) {
   array[i] = rnd.nextInt();
}

Arrays.sort(array);
```

Измерение времени исполнения

```
int[] array = new int[1_000_000_000];
Random rnd = new Random(System.currentTimeMillis());
for (int i = 0; i < array.length; i++) {
   array[i] = rnd.nextInt();
}
long t0 = System.nanoTime();
Arrays.sort(array);
long t = System.nanoTime() - t0;
System.out.println(t / 1e9); // выводим интервал в секундах
```

Результат измерения

```
171.714630615 (секунд)
```

Можно ли оценить результат без измерений?

- сложность сортировки O(N logN)
- каждая итерация 10-30 атомарных инструкций за итерацию
- тактовая частота 2.3 ГГц

T = [10 .. 30] × 10^9 × log(10^9) ÷ 2.3 × 10^9 = [90 .. 270]