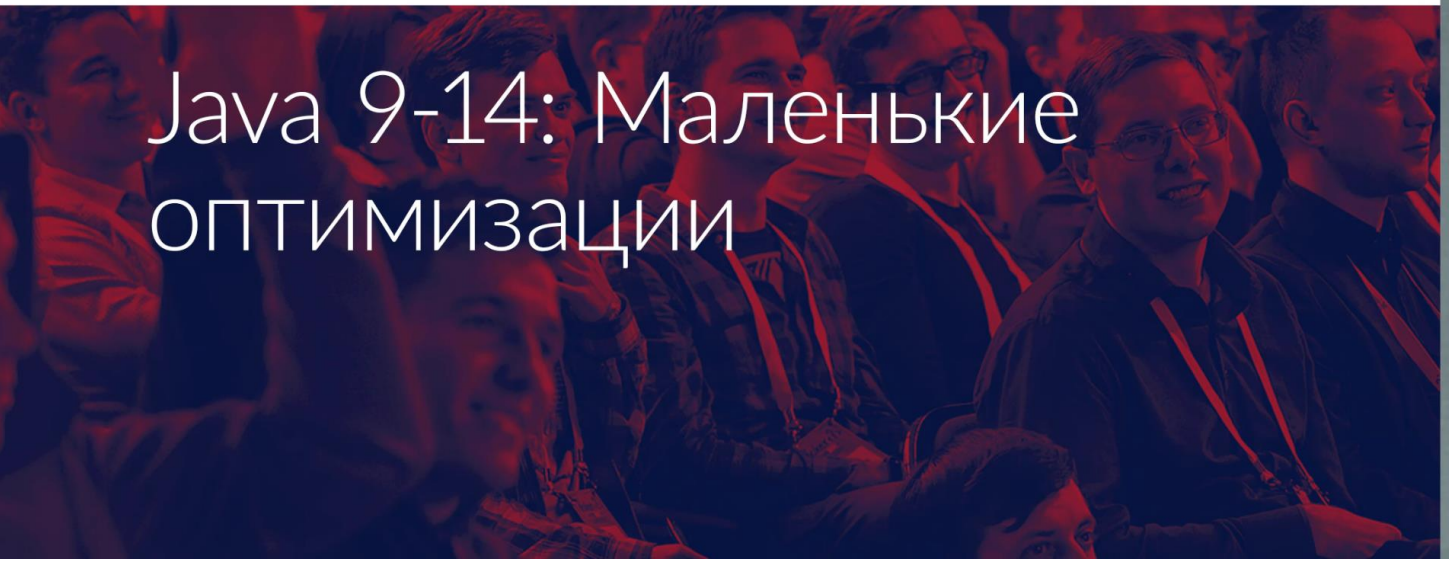
Ещё немного маленьких оптимизаций

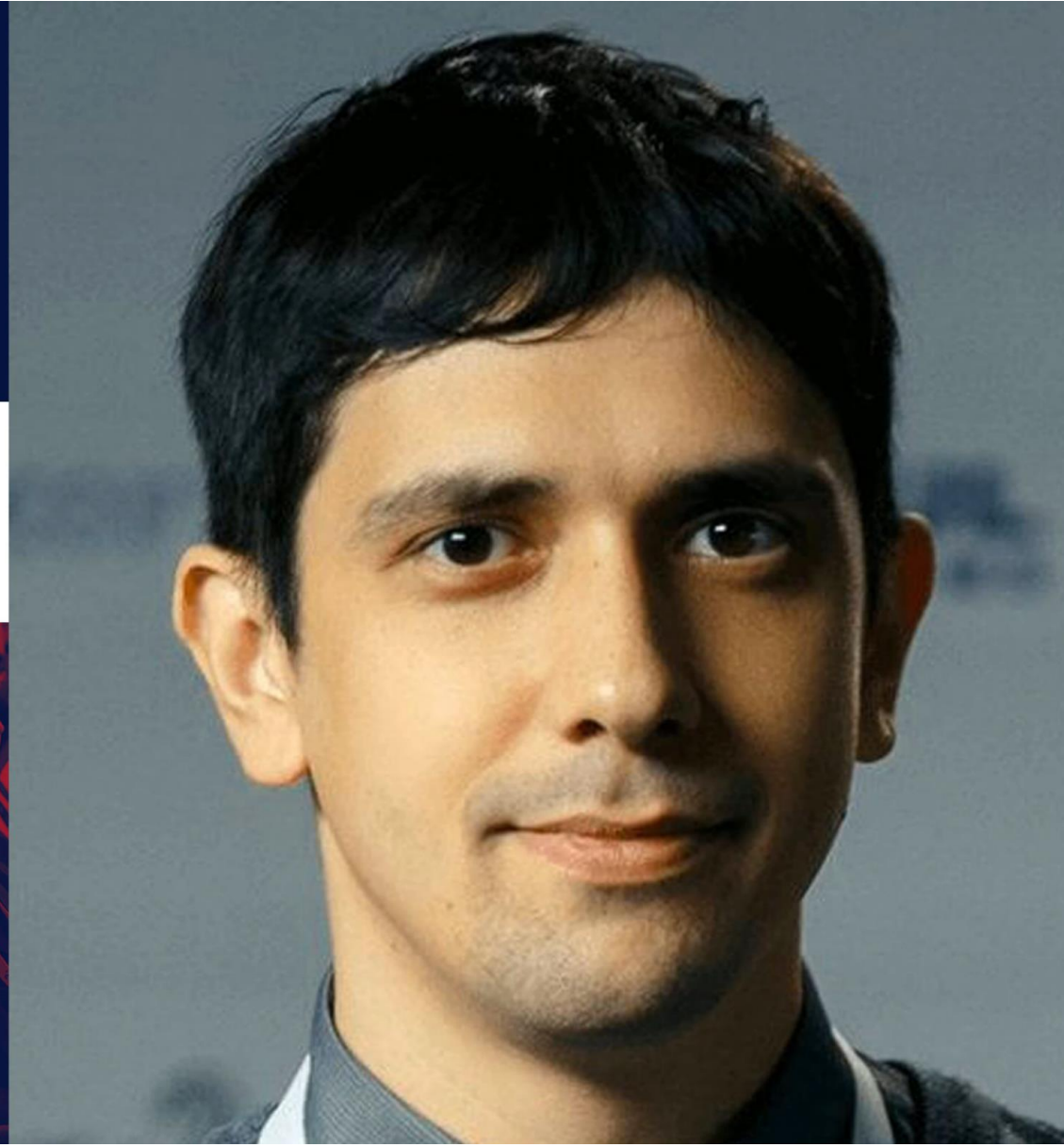
Тагир Валеев

Joker<?>
2019

Тагир Валеев
JetBrains



Java 9-14: Маленькие оптимизации



https://www.youtube.com/watch?v=5Y0Alqb9H_I

Фичи:

- ✓ Модули!
- ✓ AOT-компиляция
- ✓ Вывод типов локальных переменных (var)
- ✓ Switch expressions
- ✓ Records
- ✓ Collection factories
- ✓ HTTP client
- ✓ И т. д.

Большие оптимизации:

- ✓ GraalVM
- ✓ Shenandoah
- ✓ ZGC
- ✓ Compact strings
- ✓ Indified string concatenation
- ✓ AppCDS
- ✓ И т. д.

Ещё немного маленьких оптимизаций

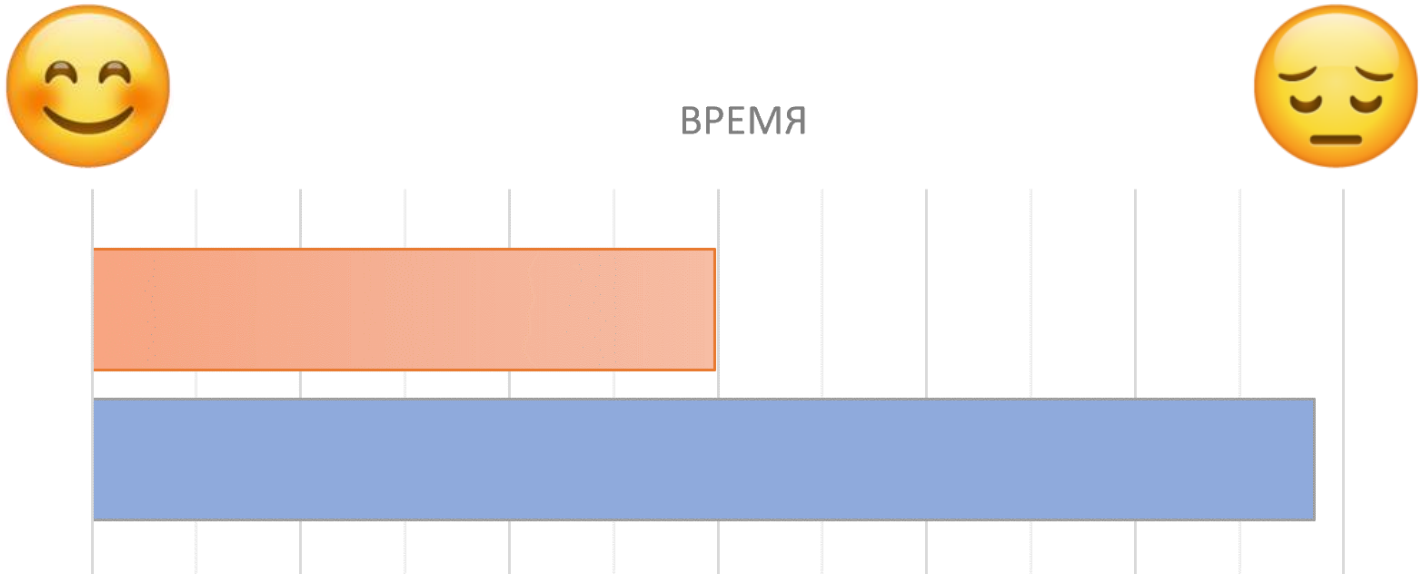
драма в трёх актах

Акт 1.
Строки

Акт 2.
Коллекции

Акт 3.
Рефлексия

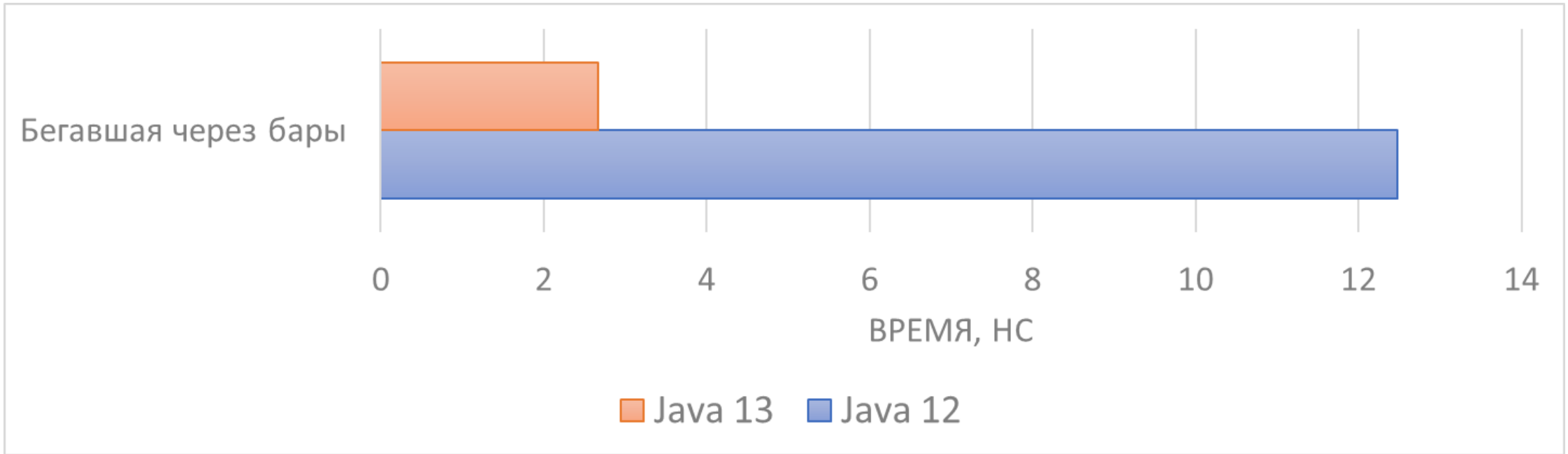
- ✓ Intel Core i7-6820HQ CPU @ 2.70GHz, 4 Core(s), HT
- ✓ Windows 10
- ✓ -XX:+UseParallelGC



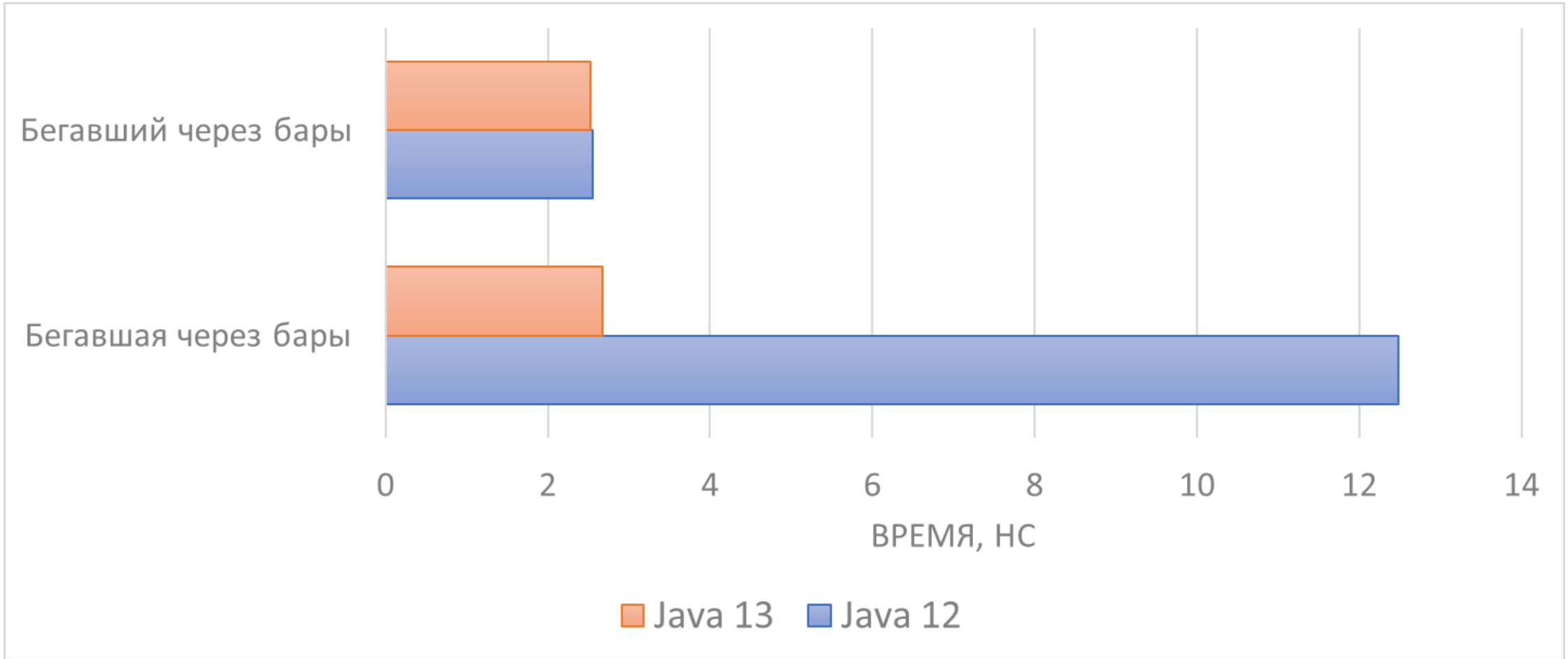
String.hashCode

JDK-8221836

```java
@Benchmark
public int calcHashCode() {
  return "Бегавшая через бары".hashCode();
}
```


Бегавшая через бары
0
2
4
6
8
10
12
14
ВРЕМЯ, НС
Java 13
Java 12

```java
@Benchmark
public int calcHashCode() {
  return "Бегавшая через бары".hashCode();
}

@Benchmark
public int calcHashCode2() {
  return "Бегавший через бары".hashCode();
}
```


Бегавший через бары
Бегавшая через бары
0
2
4
6
8
10
12
14
ВРЕМЯ, НС
Java 13
Java 12


```
/** Cache the hash code for the string */
private int hash; // Default to 0

public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        hash = h = isLatin1() ? StringLatin1.hashCode(value)
                              : StringUTF16.hashCode(value);
    }
    return h;
}
```

Алексей Шипилёв — Катехизис java.lang.String, JPoint-2015, Москва, 20.04.2015
https://www.youtube.com/watch?v=SZFe3m1DV1A

Хэшкоды: катехизис

«Штука про нули в хэшкодах некоторых перфекционистов беспокоит. Они считают, что 2^{-32} – это слишком большая вероятность, чтобы отдавать это на волю случая. Поэтому они говорят, давайте мы туда всё-таки засандалим булеву переменную. Но этого делать нельзя, потому что размер String вырастет для всех. В плохих случаях на 8 байт. Это будет очень плохо. Так что мы так делать не будем».

Deutsche Bank JetBRAINS T-Systems LUXOFT JUG.ru

Алексей Шипилёв — Катехизис java.lang.String, JPoint-2015, Москва, 20.04.2015
https://www.youtube.com/watch?v=SZFe3m1DV1A


```
// -XX:+UseCompressedOops
java.lang.String object internals:
 OFFSET  SIZE      TYPE DESCRIPTION
      0    12           (object header)
     12     4    char[] String.value
     16     4       int String.hash
     20     4           (loss due to the next object alignment)
Instance size: 24 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```


```
// -XX:+UseCompressedOops
java.lang.String object internals:
 OFFSET  SIZE     TYPE DESCRIPTION
      0    12          (object header)
     12     4   char[] String.value
     16     4      int String.hash
     20     4          (loss due to the next object alignment)
Instance size: 24 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total

// -XX:-UseCompressedOops
java.lang.String object internals:
 OFFSET  SIZE     TYPE DESCRIPTION
      0    16          (object header)
     16     8   char[] String.value
     24     4      int String.hash
     28     4          (loss due to the next object alignment)
Instance size: 32 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```


```
// -XX:+UseCompressedOops
java.lang.String object internals:
 OFFSET  SIZE      TYPE DESCRIPTION
      0    12           (object header)
     12     4    byte[] String.value
     16     4       int String.hash
     20     1      byte String.coder
     21     3           (loss due to the next object alignment)
Instance size: 24 bytes
Space losses: 0 bytes internal + 3 bytes external = 3 bytes total
```


```
// -XX:+UseCompressedOops
java.lang.String object internals:
 OFFSET  SIZE      TYPE DESCRIPTION
      0    12           (object header)
     12     4    byte[] String.value
     16     4       int String.hash
     20     1      byte String.coder
     21     1   boolean String.hashIsZero
     22     2           (loss due to the next object alignment)
Instance size: 24 bytes
Space losses: 0 bytes internal + 2 bytes external = 2 bytes total
```

```
private int hash; // Default to 0
private boolean hashIsZero; // Default to false;
public int hashCode() {
    int h = hash;
    if (h == 0 && !hashIsZero) {
        h = isLatin1() ? StringLatin1.hashCode(value)
                       : StringUTF16.hashCode(value);
        if (h == 0) {
            hashIsZero = true;
        } else {
            hash = h;
        }
    }
    return h;
}
```

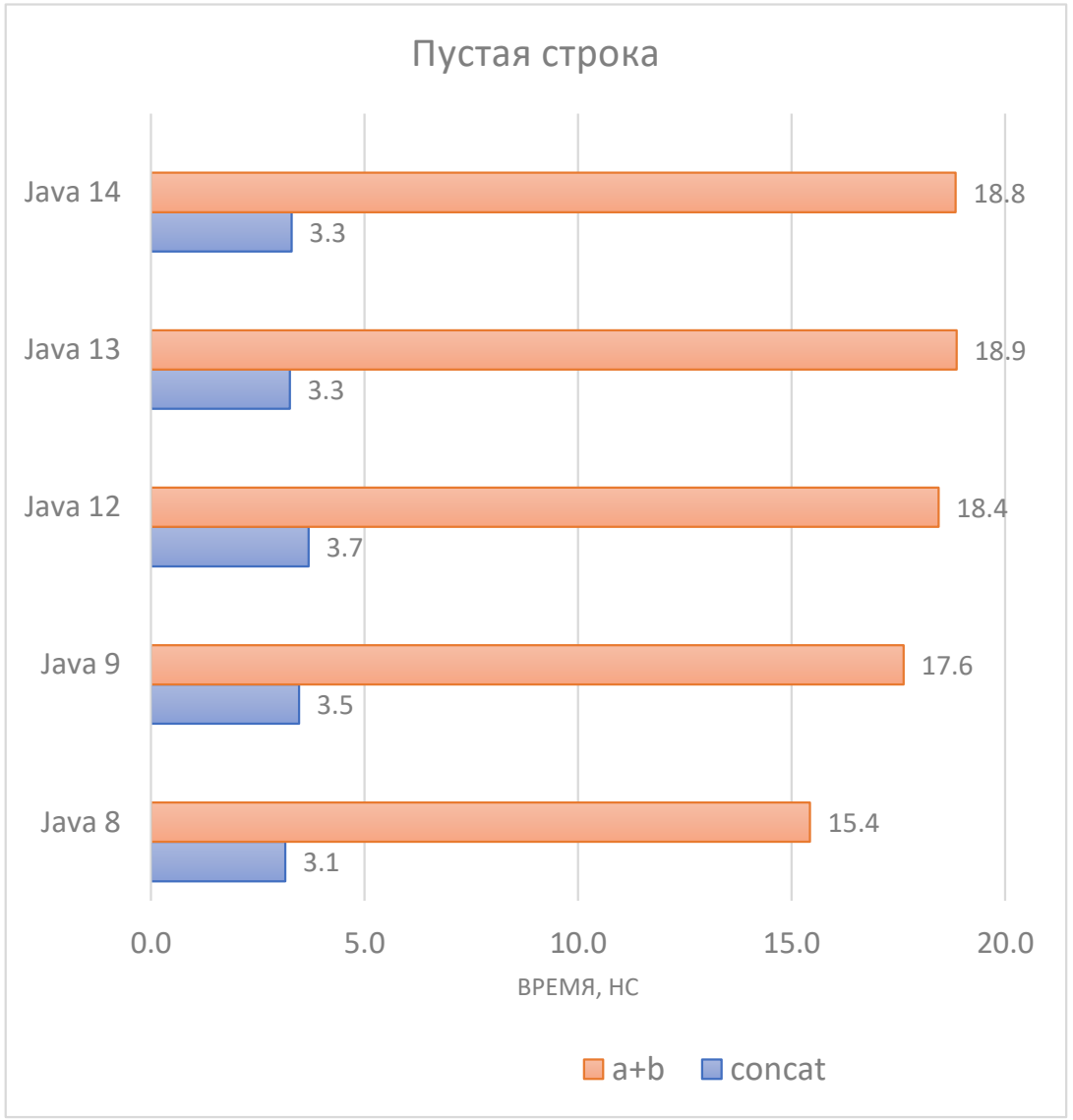
```java
private int hash; // Default to 0
private boolean hashIsZero; // Default to false;
public int hashCode() {
    // The hash or hashIsZero fields are subject to a benign data race,
    // making it crucial to ensure that any observable result of the
    // calculation in this method stays correct under any possible read of
    // these fields. Necessary restrictions to allow this to be correct
    // without explicit memory fences or similar concurrency primitives is
    // that we can ever only write to one of these two fields for a given
    // String instance, and that the computation is idempotent and derived
    // from immutable state
    int h = hash;
    if (h == 0 && !hashIsZero) {
        h = isLatin1() ? StringLatin1.hashCode(value)
                       : StringUTF16.hashCode(value);
        if (h == 0) {
            hashIsZero = true;
        } else {
            hash = h;
        }
    }
    return h;
}
```

- RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
  - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
    - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Andrew Dinn*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Peter Levart*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Peter Levart*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Andrew Dinn*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Andrew Dinn*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Peter Levart*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Andrew Dinn*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Andrew Dinn*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
    - RFR: 8221836: Avoid recalculating String.hash when zero *Peter Levart*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*
      - RFR: 8221836: Avoid recalculating String.hash when zero *John Rose*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Andrew Dinn*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Peter Levart*
      - RFR: 8221836: Avoid recalculating String.hash when zero *Aleksey Shipilev*
  - RFR: 8221836: Avoid recalculating String.hash when zero *Ivan Gerasimov*
    - RFR: 8221836: Avoid recalculating String.hash when zero *Claes Redestad*

String.concat

JDK-8222484

```java
@Param({"", "is a very very very very very very very very very cool conference!"})
String data;

@Benchmark
public String concat() {
  return "JPoint ".concat(data);
}

@Benchmark
public String plus() {
  return "JPoint " + data;
}
```

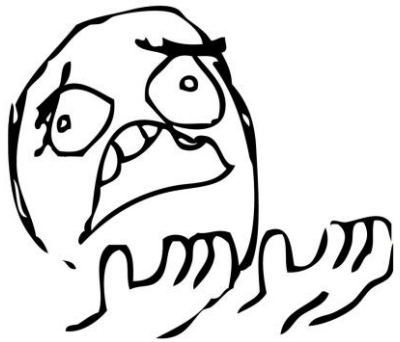

Пустая строка
Java 14
18.8
3.3
Java 13
18.9
3.3
Java 12
18.4
3.7
Java 9
17.6
3.5
Java 8
15.4
3.1
0.0
5.0
10.0
15.0
20.0
ВРЕМЯ, НС
a+b
concat

15.18.1. String Concatenation Operator +

If only one operand expression is of type `String`, then string conversion (§5.1.11) is performed on the other operand to produce a string at run time.

The result of string concatenation is a reference to a `String` object that is the concatenation of the two operand strings. The characters of the left-hand operand precede the characters of the right-hand operand in the newly created string.

The `String` object is newly created (§12.5) unless the expression is a constant expression (§15.29).



https://docs.oracle.com/javase/specs/jls/se14/html/jls-15.html#jls-15.18.1

Tagir Valeev
@tagir_valeev

Does anybody know why JLS 15.18.1 states that the result of the concatenation is *newly created* String object (unless it's a constant expression), so str+"" cannot just return str and must copy it? Was it just an oversight or there was a reason behind this?

12:16 AM · Jun 13, 2020 · Twitter Web App

View Tweet activity

25 Likes

concat

```
public String concat(String str)
```

Concatenates the specified string to the end of this string.

If the length of the argument string is `0`, then this `String` object is returned. Otherwise, a `String` object is returned that represents a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

Examples:

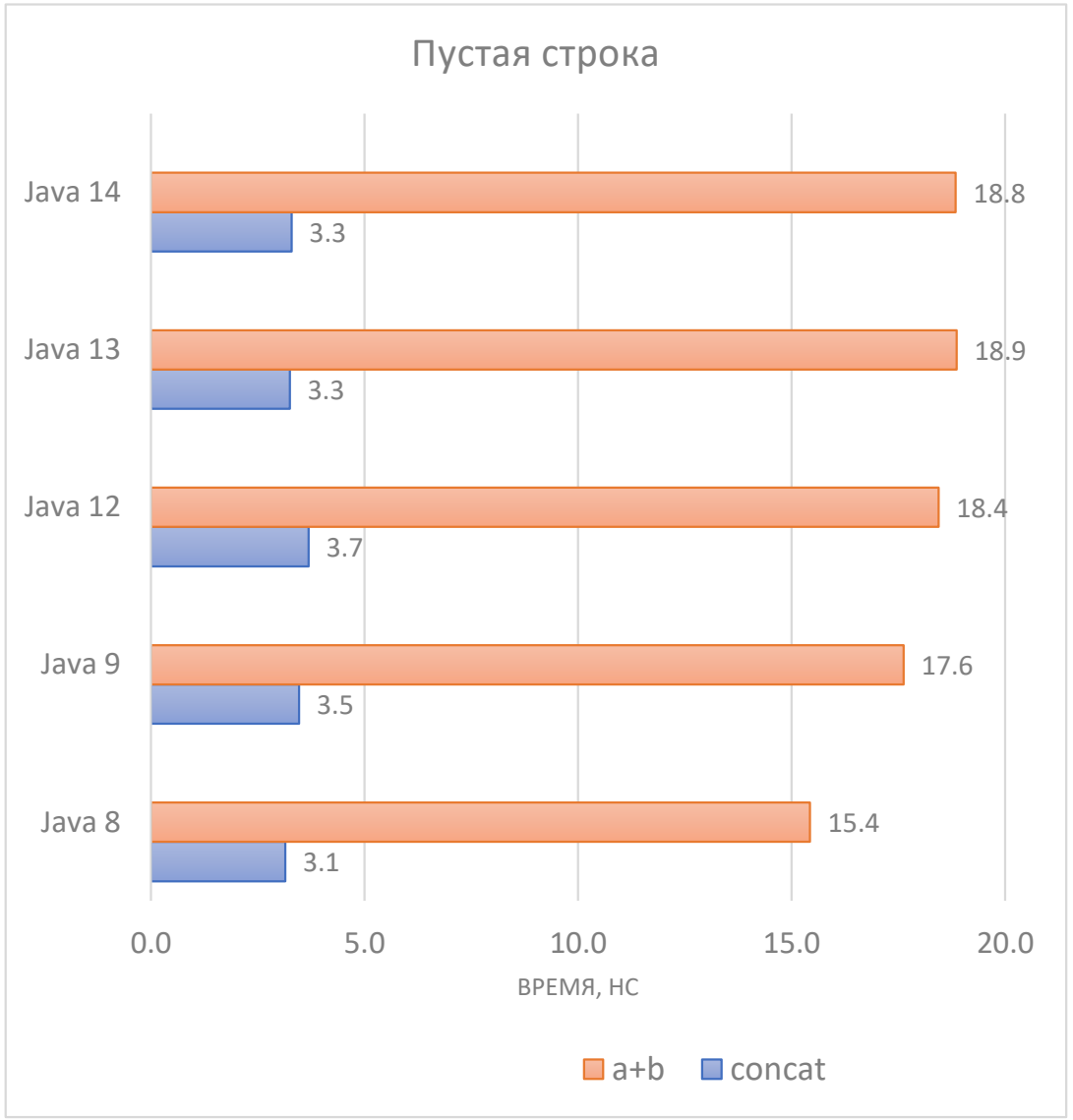
```
"cares".concat("s") returns "caress"
"to".concat("get").concat("her") returns "together"
```

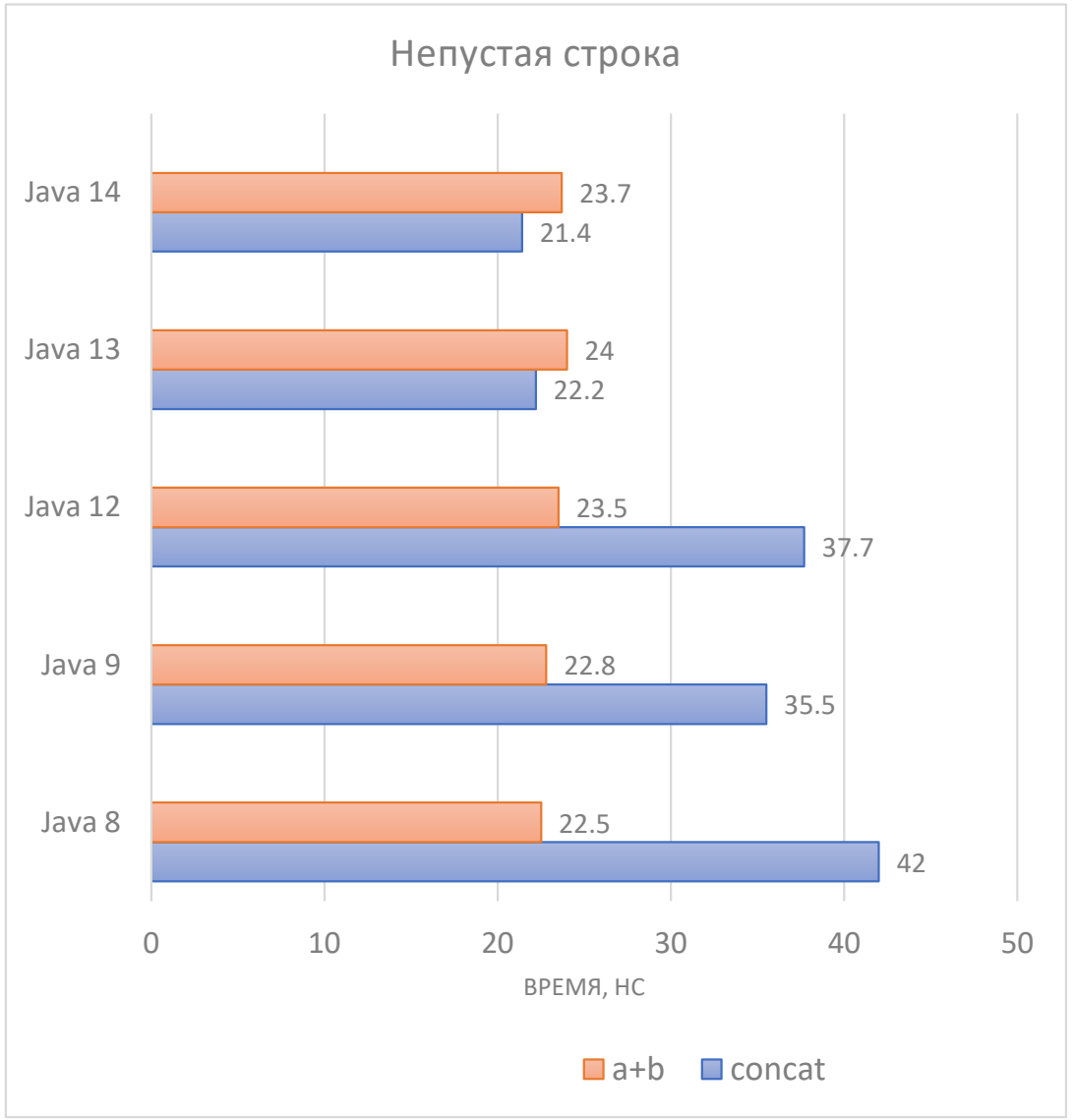
Parameters:

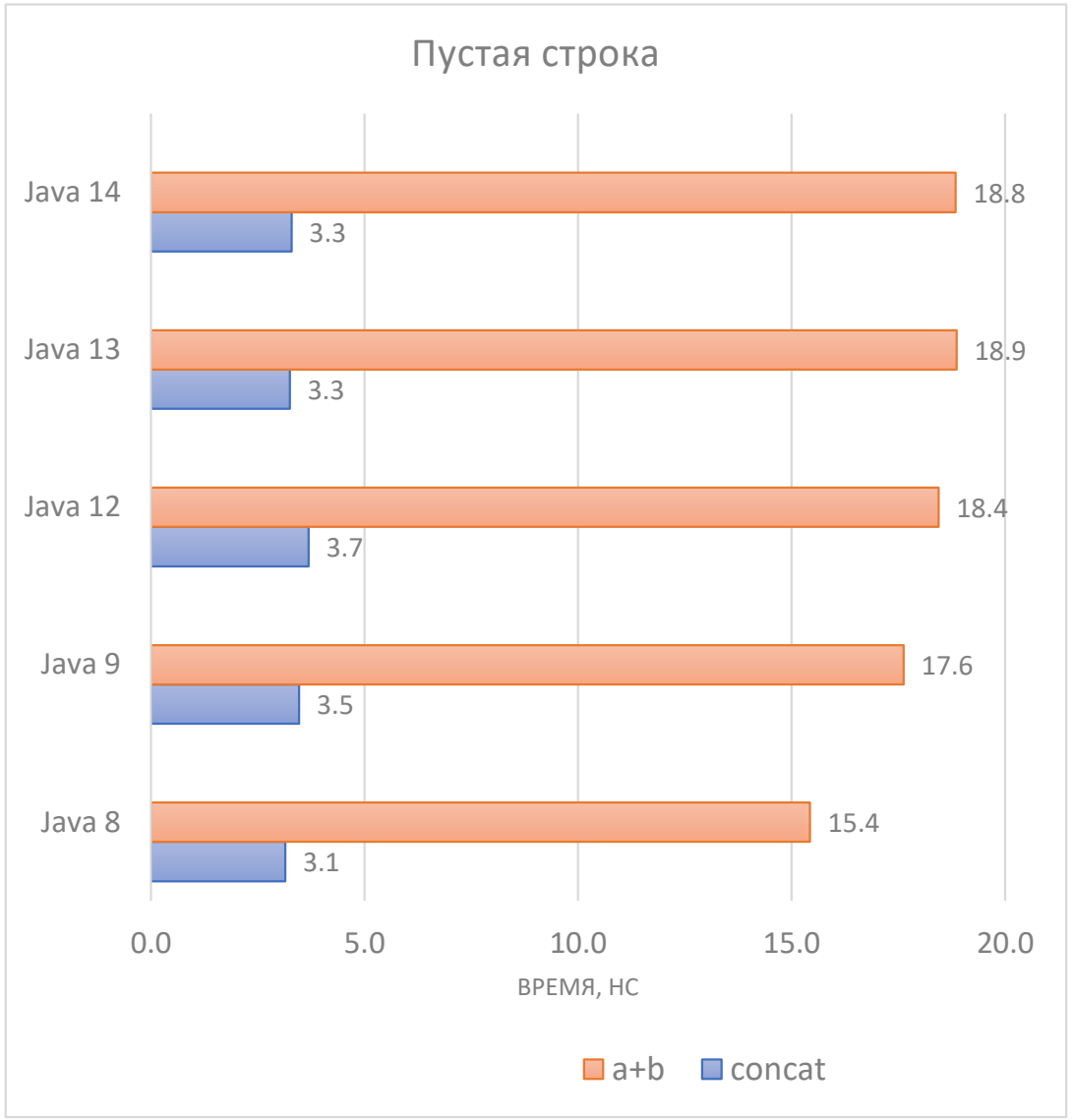
`str` - the `String` that is concatenated to the end of this `String`.

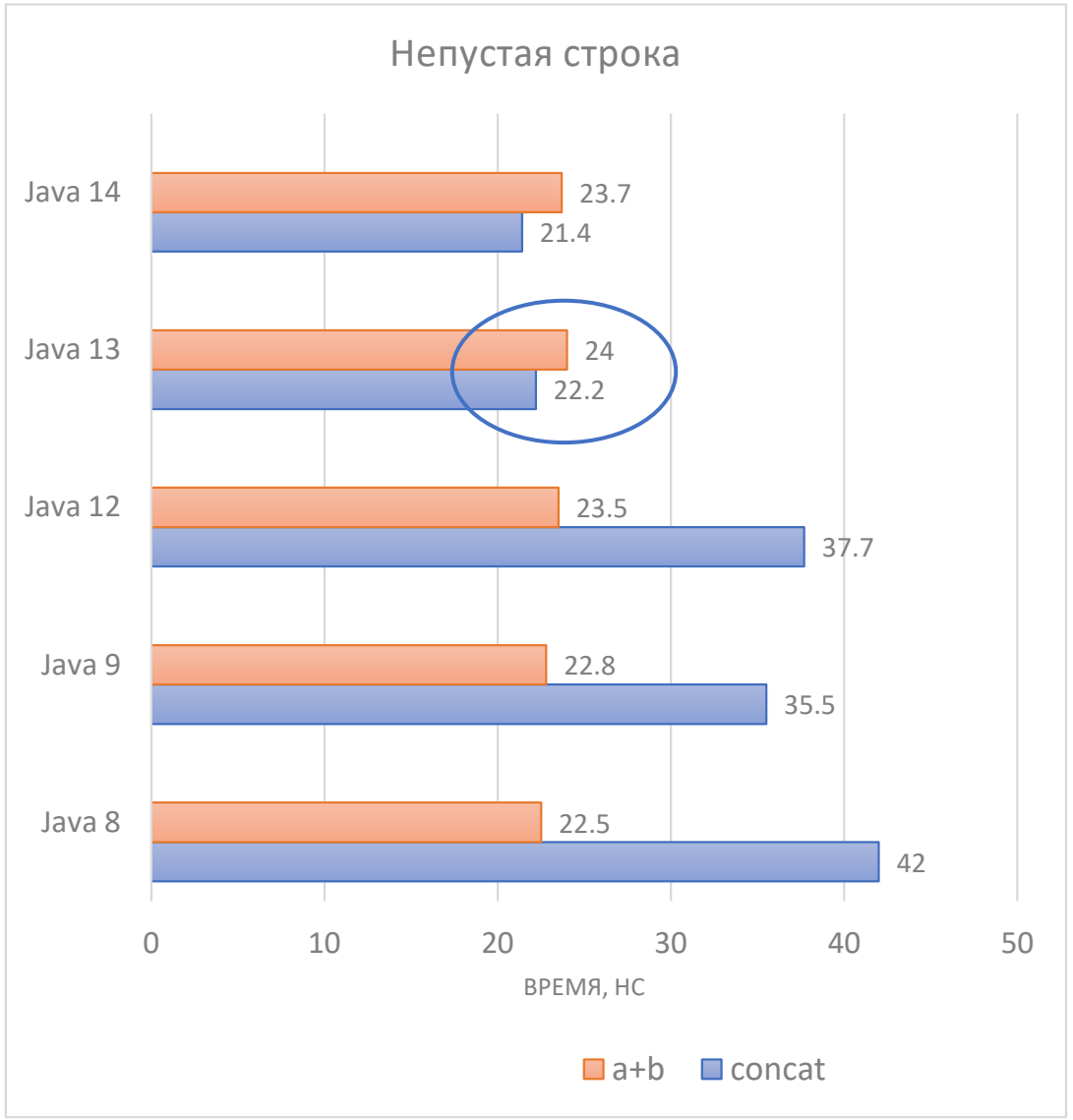
Returns:

a string that represents the concatenation of this object's characters followed by the string argument's characters.


Пустая строка
Java 14
18.8
3.3
Java 13
18.9
3.3
Java 12
18.4
3.7
Java 9
17.6
3.5
Java 8
15.4
3.1
0.0
5.0
10.0
15.0
20.0
ВРЕМЯ, НС
a+b
concat

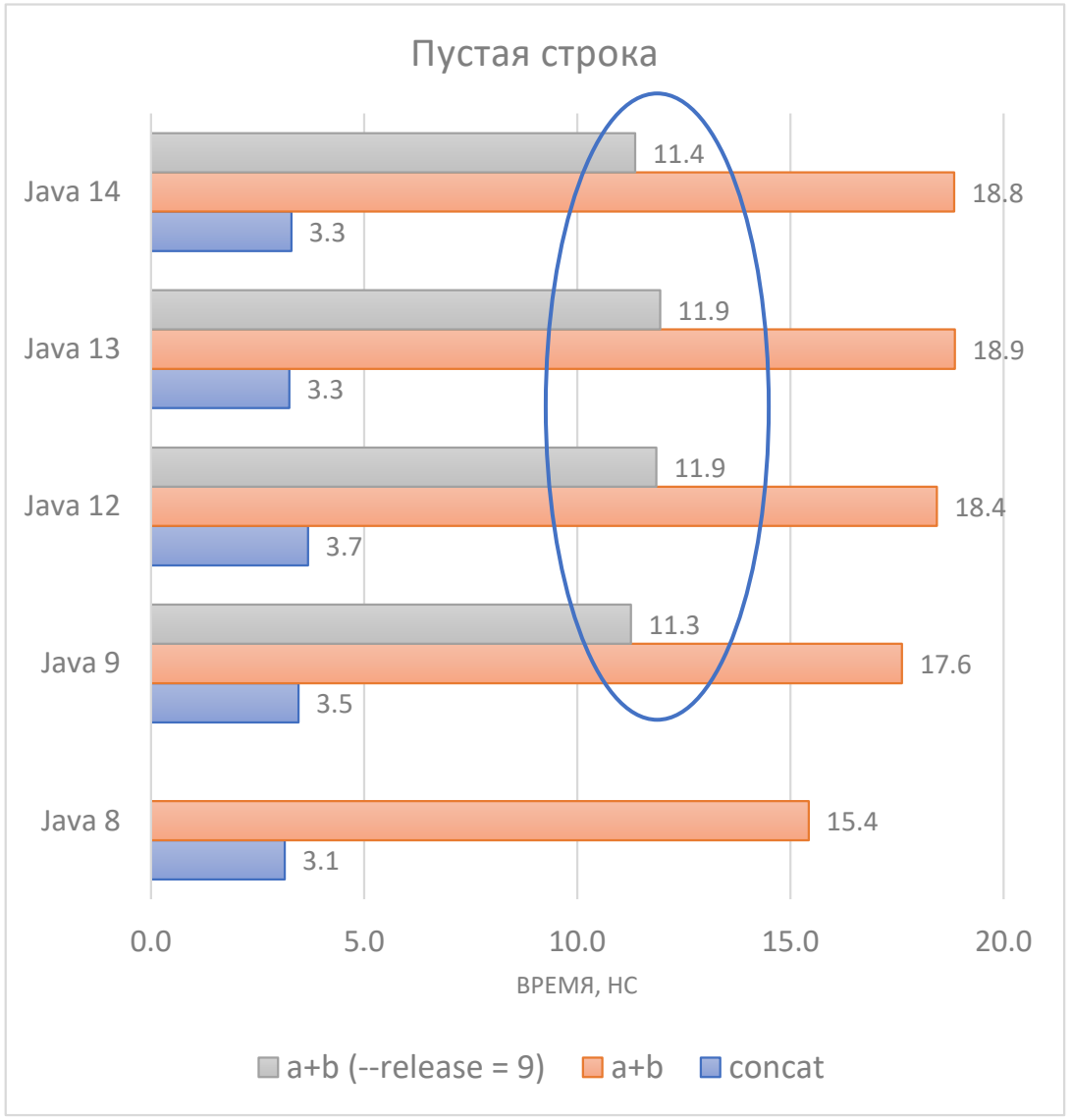

Непустая строка
Java 14
23.7
21.4
Java 13
24
22.2
Java 12
23.5
37.7
Java 9
22.8
35.5
Java 8
22.5
42
0
10
20
30
40
50
ВРЕМЯ, НС
a+b
concat

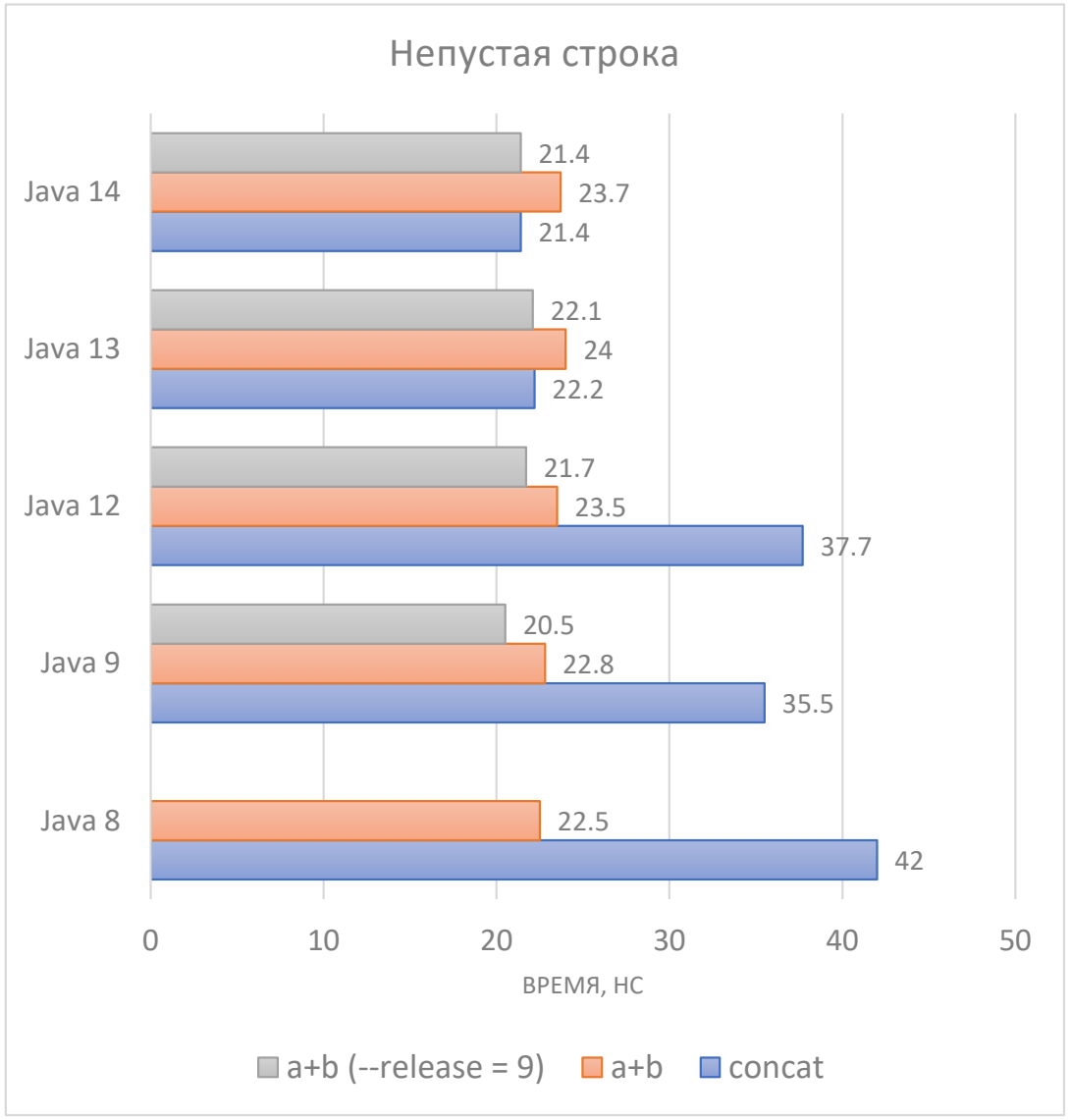

Пустая строка
Java 14
18.8
3.3
Java 13
18.9
3.3
Java 12
18.4
3.7
Java 9
17.6
3.5
Java 8
15.4
3.1
0.0
5.0
10.0
15.0
20.0
ВРЕМЯ, НС
a+b
concat

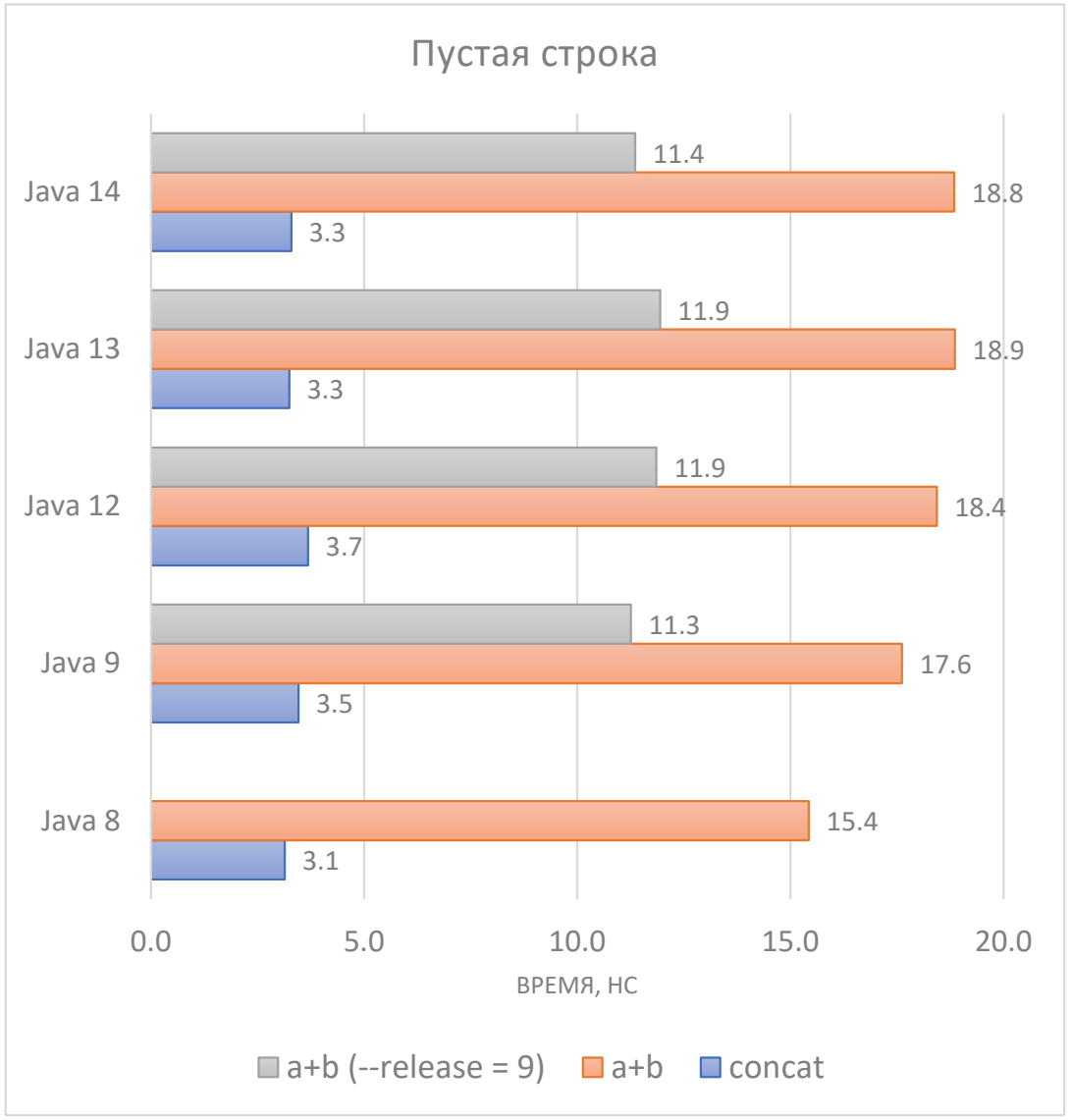

Непустая строка
Java 14
23.7
21.4
Java 13
24
22.2
Java 12
23.5
37.7
Java 9
22.8
35.5
Java 8
22.5
42
0
10
20
30
40
50
ВРЕМЯ, НС
a+b
concat

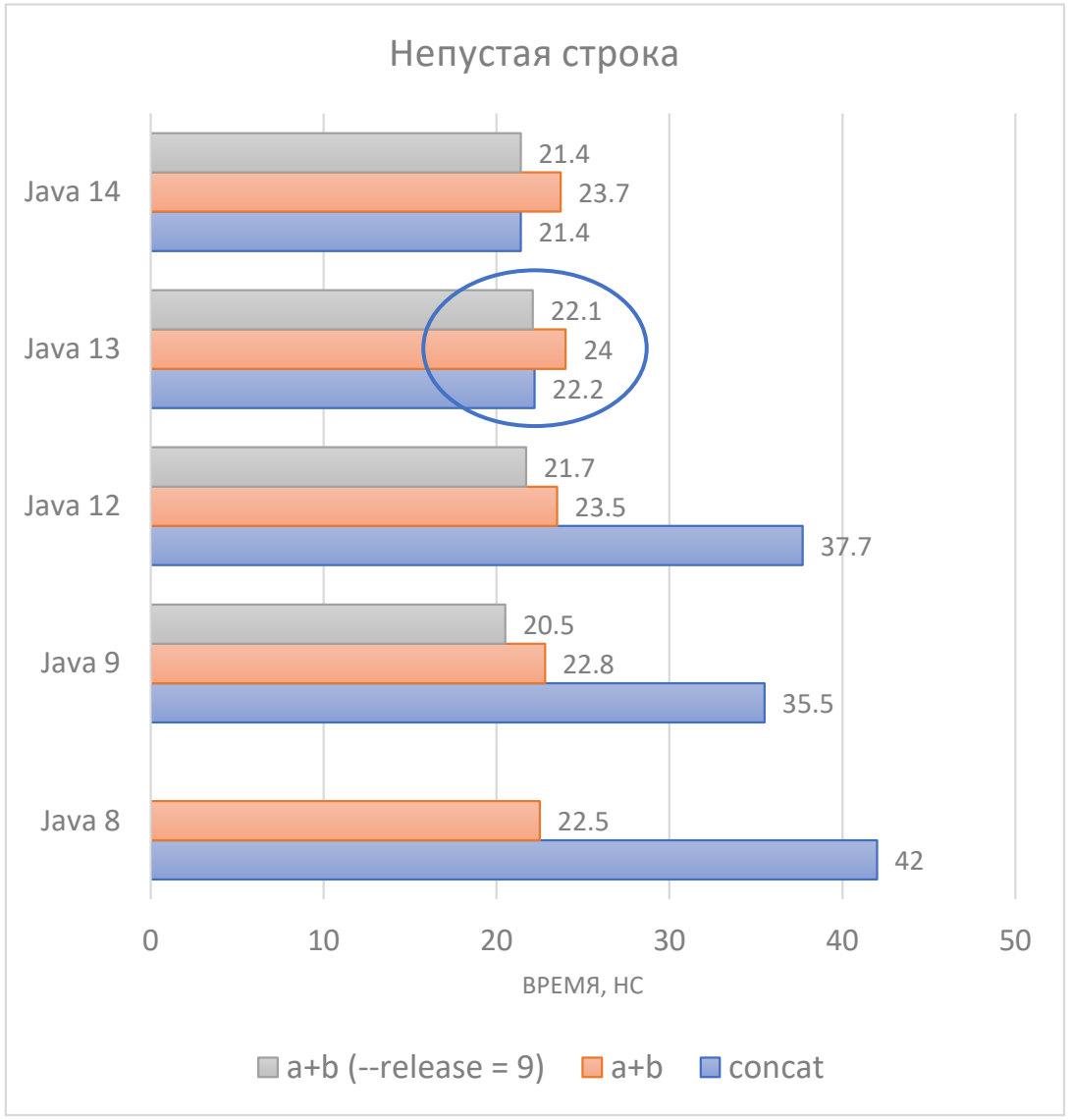
JEP 280: Indify String Concatenation

Owner	Aleksey Shipilev
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	9
Component	tools / javac
Discussion	core dash libs dash dev at openjdk dot java dot net, compiler dash dev at openjdk dot java dot net, hotspot dash dev at openjdk dot java dot net
Effort	M
Duration	M
Relates to	JEP 254: Compact Strings
Reviewed by	Michael Haupt, Paul Sandoz
Endorsed by	Brian Goetz
Created	2015/06/04 08:13
Updated	2017/05/17 01:02
Issue	8085796


Пустая строка
Java 14
11.4
18.8
3.3
Java 13
11.9
18.9
3.3
Java 12
11.9
18.4
3.7
Java 9
11.3
17.6
3.5
Java 8
15.4
3.1
0.0
5.0
10.0
15.0
20.0
ВРЕМЯ, НС
a+b (--release = 9)
a+b
concat


Непустая строка
Java 14
21.4
23.7
21.4
Java 13
22.1
24
22.2
Java 12
21.7
23.5
37.7
Java 9
20.5
22.8
35.5
Java 8
22.5
42
0
10
20
30
40
50
ВРЕМЯ, НС
a+b (--release = 9)
a+b
concat


Пустая строка
Java 14
11.4
18.8
3.3
Java 13
11.9
18.9
3.3
Java 12
11.9
18.4
3.7
Java 9
11.3
17.6
3.5
Java 8
15.4
3.1
0.0
5.0
10.0
15.0
20.0
ВРЕМЯ, НС
a+b (--release = 9)
a+b
concat


Непустая строка
Java 14
21.4
23.7
21.4
Java 13
22.1
24
22.2
Java 12
21.7
23.5
37.7
Java 9
20.5
22.8
35.5
Java 8
22.5
42
0
10
20
30
40
50
ВРЕМЯ, НС
a+b (--release = 9)
a+b
concat


```
public String concat(String str) {
    if (str.isEmpty()) {
        return this;
    }
    if (coder() == str.coder()) {
        byte[] val = this.value;
        byte[] oval = str.value;
        int len = val.length + oval.length;
        byte[] buf = Arrays.copyOf(val, len);
        System.arraycopy(oval, 0, buf, val.length, oval.length);
        return new String(buf, coder);
    }
    int len = length();
    int olen = str.length();
    byte[] buf = StringUTF16.newBytesFor(len + olen);
    getBytes(buf, 0, UTF16);
    str.getBytes(buf, len, UTF16);
    return new String(buf, UTF16);
}
```


```
public String concat(String str) {
    if (str.isEmpty()) {
        return this;
    }
    if (coder() == str.coder()) {
        byte[] val = this.value;
        byte[] oval = str.value;
        int len = val.length + oval.length;
        byte[] buf = Arrays.copyOf(val, len);
        System.arraycopy(oval, 0, buf, val.length, oval.length);
        return new String(buf, coder);
    }
    int len = length();
    int olen = str.length();
    byte[] buf = StringUTF16.newBytesFor(len + olen);
    getBytes(buf, 0, UTF16);
    str.getBytes(buf, len, UTF16);
    return new String(buf, UTF16);
}
```

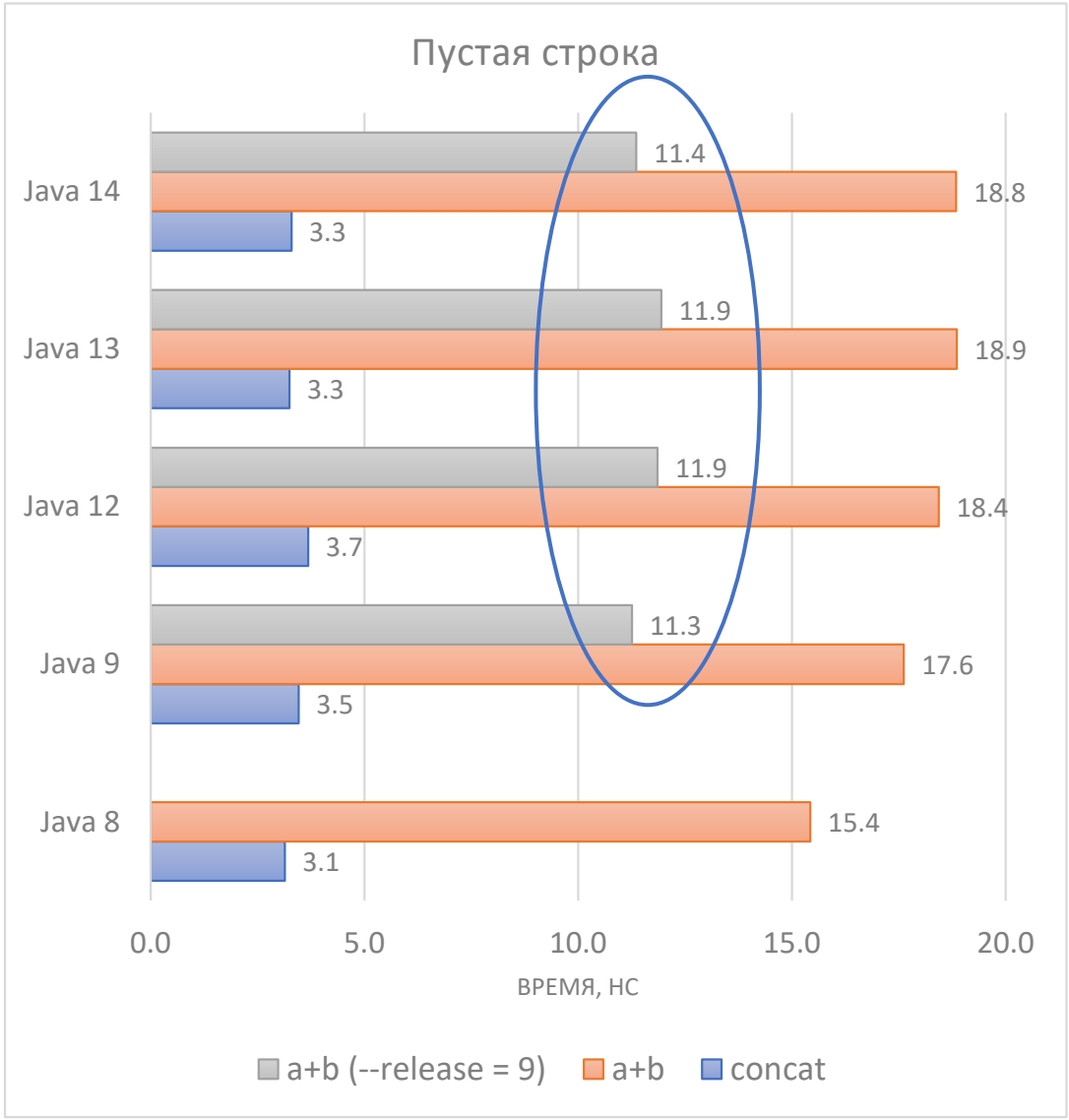

```
public String concat(String str) {
    if (str.isEmpty()) {
        return this;
    }
    if (coder() == str.coder()) {
        byte[] val = this.value;
        byte[] oval = str.value;
        int len = val.length + oval.length;
        byte[] buf = Arrays.copyOf(val, len);
        System.arraycopy(oval, 0, buf, val.length, oval.length);
        return new String(buf, coder);
    }
    int len = length();
    int olen = str.length();
    byte[] buf = StringUTF16.newBytesFor(len + olen);
    getBytes(buf, 0, UTF16);
    str.getBytes(buf, len, UTF16);
    return new String(buf, UTF16);
}
```


```
public String concat(String str) {
    if (str.isEmpty()) {
        return this;
    }
    if (coder() == str.coder()) {
        byte[] val = this.value;
        byte[] oval = str.value;
        int len = val.length + oval.length;
        byte[] buf = Arrays.copyOf(val, len);
        System.arraycopy(oval, 0, buf, val.length, oval.length);
        return new String(buf, coder);
    }
    int len = length();
    int olen = str.length();
    byte[] buf = StringUTF16.newBytesFor(len + olen);
    getBytes(buf, 0, UTF16);
    str.getBytes(buf, len, UTF16);
    return new String(buf, UTF16);
}
```


```
public String concat(String str) {
    if (str.isEmpty()) {
        return this;
    }
    return StringConcatHelper.simpleConcat(this, str);
}

static String simpleConcat(Object first, Object second) {
    String s1 = stringOf(first);
    String s2 = stringOf(second);
    // start "mixing" in length and coder or arguments, order is not
    // important
    long indexCoder = mix(initialCoder(), s2);
    indexCoder = mix(indexCoder, s1);
    byte[] buf = newArray(indexCoder);
    // prepend each argument in reverse order, since we prepending
    // from the end of the byte array
    indexCoder = prepend(indexCoder, buf, s2);
    indexCoder = prepend(indexCoder, buf, s1);
    return newString(buf, indexCoder);
}
```


```
static String simpleConcat(Object first, Object second) {
    …
    byte[] buf = newArray(indexCoder);
    …
}

static byte[] newArray(long indexCoder) {
    byte coder = (byte)(indexCoder >> 32);
    int index = (int)indexCoder;
    return (byte[]) UNSAFE.allocateUninitializedArray(byte.class, index << coder);
}
```


```
public String concat(String str) {
    …
    byte[] buf = Arrays.copyOf(val, len);
    System.arraycopy(oval, 0, buf, val.length, oval.length);
    …
}

public static byte[] copyOf(byte[] original, int newLength) {
    byte[] copy = new byte[newLength];
    System.arraycopy(original, 0, copy, 0,
                     Math.min(original.length, newLength));
    return copy;
}
```


Пустая строка
Java 14
Java 13
Java 12
Java 9
Java 8
11.4
18.8
3.3
11.9
18.9
3.3
11.9
18.4
3.7
11.3
17.6
3.5
15.4
3.1
0.0
5.0
10.0
15.0
20.0
ВРЕМЯ, НС
a+b (--release = 9)
a+b
concat

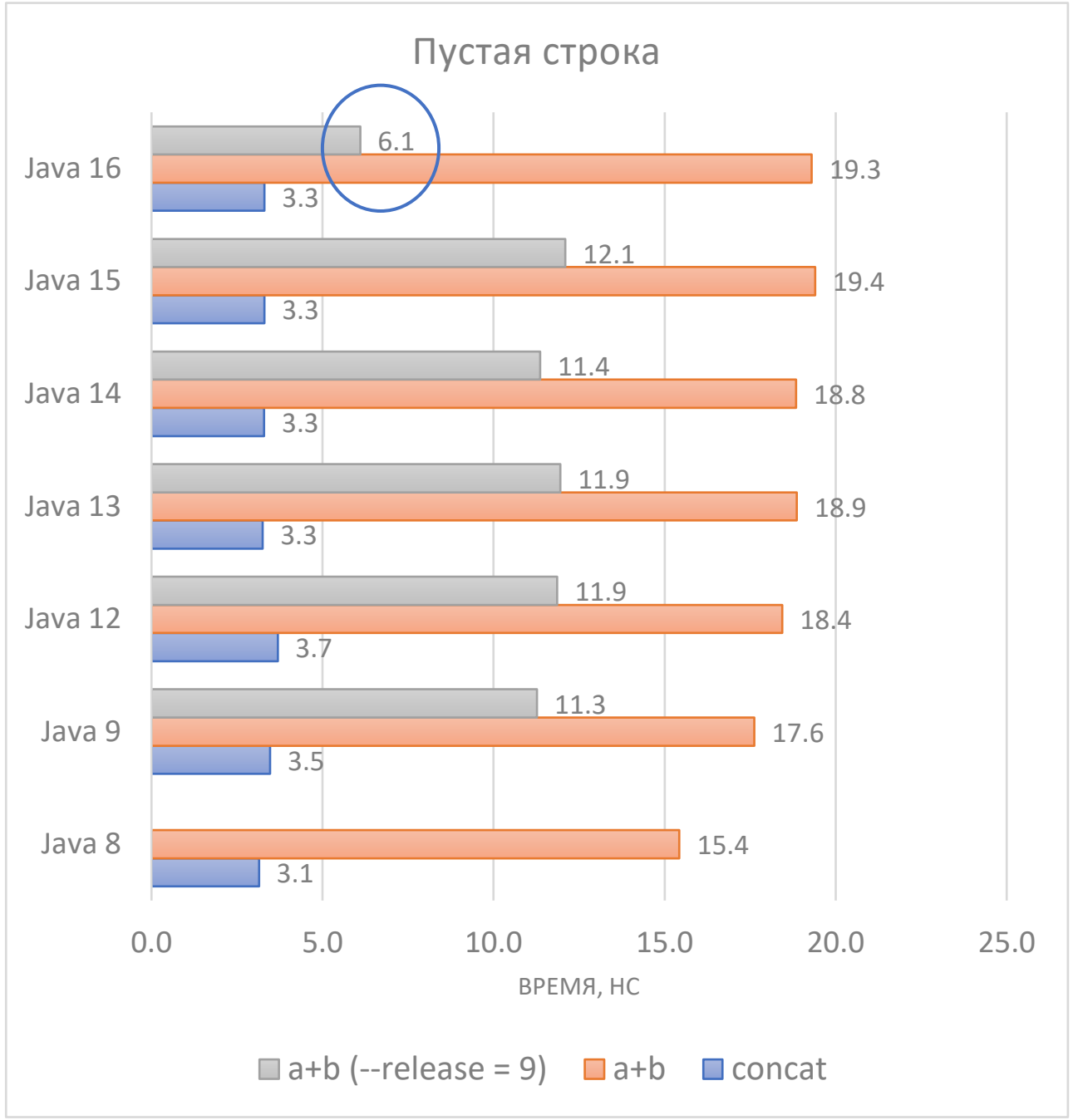
Concatenation with empty string

JDK-8247605

```
static String simpleConcat(Object first, Object second) {
    String s1 = stringOf(first);
    String s2 = stringOf(second);
    // start "mixing" in length and coder or arguments, order is not
    // important
    long indexCoder = mix(initialCoder(), s2);
    indexCoder = mix(indexCoder, s1);
    byte[] buf = newArray(indexCoder);
    // prepend each argument in reverse order, since we prepending
    // from the end of the byte array
    indexCoder = prepend(indexCoder, buf, s2);
    indexCoder = prepend(indexCoder, buf, s1);
    return newString(buf, indexCoder);
}
```

```
static String simpleConcat(Object first, Object second) {
    String s1 = stringOf(first);
    String s2 = stringOf(second);
    // start "mixing" in length and coder or arguments, order is not
    // important
    long indexCoder = mix(initialCoder(), s2);
    indexCoder = mix(indexCoder, s1);
    byte[] buf = newArray(indexCoder);
    // prepend each argument in reverse order, since we prepending
    // from the end of the byte array
    indexCoder = prepend(indexCoder, buf, s2);
    indexCoder = prepend(indexCoder, buf, s1);
    return newString(buf, indexCoder);
}
```


```
static String simpleConcat(Object first, Object second) {
    String s1 = stringOf(first);
    String s2 = stringOf(second);
    if (s1.isEmpty()) {
        // newly created string required, see JLS 15.18.1
        return new String(s2);
    }
    if (s2.isEmpty()) {
        // newly created string required, see JLS 15.18.1
        return new String(s1);
    }
    // start "mixing" in length and coder or arguments, order is not
    // important
    long indexCoder = mix(initialCoder(), s2);
    indexCoder = mix(indexCoder, s1);
    byte[] buf = newArray(indexCoder);
    // prepend each argument in reverse order, since we prepending
    // from the end of the byte array
    indexCoder = prepend(indexCoder, buf, s2);
    indexCoder = prepend(indexCoder, buf, s1);
    return newString(buf, indexCoder);
}
```

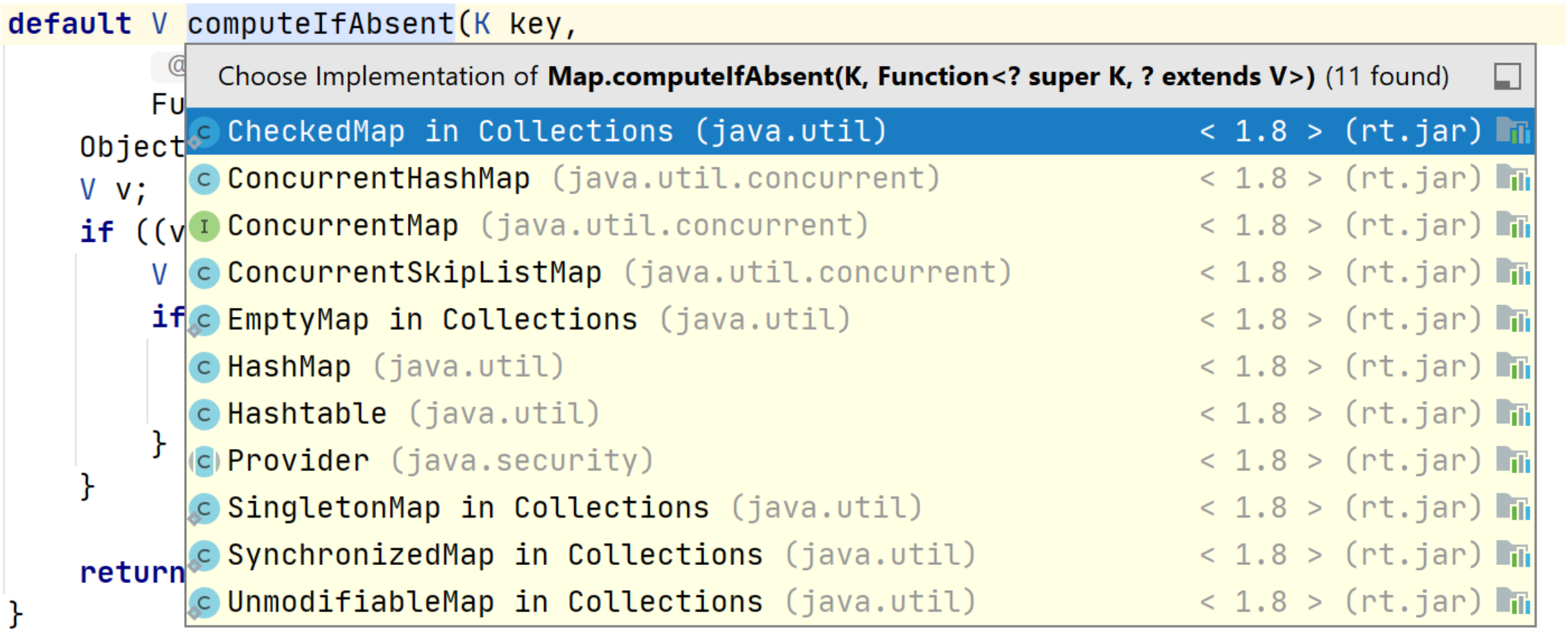

Пустая строка
Java 16
6.1
19.3
3.3
Java 15
12.1
19.4
3.3
Java 14
11.4
18.8
3.3
Java 13
11.9
18.9
3.3
Java 12
11.9
18.4
3.7
Java 9
11.3
17.6
3.5
Java 8
15.4
3.1
0.0
5.0
10.0
15.0
20.0
25.0
ВРЕМЯ, НС
a+b (--release = 9)
a+b
concat

TreeMap.computeIfAbsent and friends

JDK-8176894

- ✓ putIfAbsent
- ✓ computeIfAbsent
- ✓ computeIfPresent
- ✓ compute
- ✓ merge

```
default V computeIfAbsent(K key,
        Function<? super K, ? extends V> mappingFunction) {
    Objects.requireNonNull(mappingFunction);
    V v;
    if ((v = get(key)) == null) {
        V newValue;
        if ((newValue = mappingFunction.apply(key)) != null) {
            put(key, newValue);
            return newValue;
        }
    }

    return v;
}
```


default V computeIfAbsent(K key,
Choose Implementation of Map.computeIfAbsent(K, Function<? super K, ? extends V>) (11 found)
CheckedMap in Collections (java.util) < 1.8 > (rt.jar)
ConcurrentHashMap (java.util.concurrent) < 1.8 > (rt.jar)
ConcurrentMap (java.util.concurrent) < 1.8 > (rt.jar)
ConcurrentSkipListMap (java.util.concurrent) < 1.8 > (rt.jar)
EmptyMap in Collections (java.util) < 1.8 > (rt.jar)
HashMap (java.util) < 1.8 > (rt.jar)
Hashtable (java.util) < 1.8 > (rt.jar)
Provider (java.security) < 1.8 > (rt.jar)
SingletonMap in Collections (java.util) < 1.8 > (rt.jar)
SynchronizedMap in Collections (java.util) < 1.8 > (rt.jar)
UnmodifiableMap in Collections (java.util) < 1.8 > (rt.jar)

Code Review for jdk

Prepared by: Sergey Kuksenko on Thu Mar 16 11:34:24 PDT 2017
Workspace: /home/skuksenk/repj/jdk10/jdk
Compare against: http://hg.openjdk.java.net/jdk10/jdk10/jdk
Compare against version: 16827
Summary of changes: 174 lines changed: 159 ins; 4 del; 11 mod; 2997 unchg
Patch of changes: jdk.patch
Legend: **Modified file**
Deleted file
New file

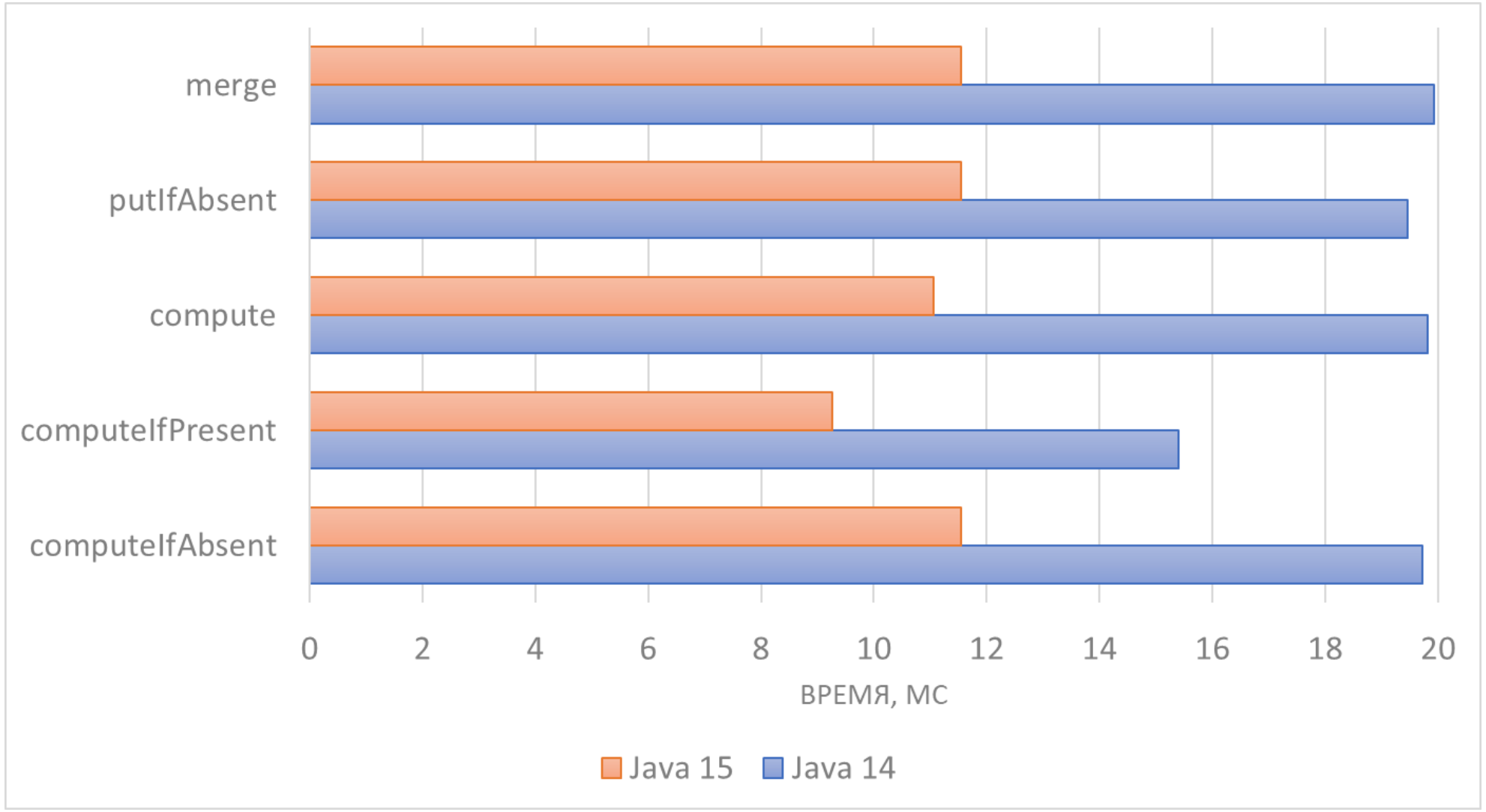
Cdiffs Udiffs Sdiffs Frames Old New Patch Raw **src/java.base/share/classes/java/util/TreeMap.java**

```
rev 16828 : implements default methods in TreeMap v0
rev 16829 : TreeMap
rev 16830 : TreeMap
```

174 lines changed: 159 ins; 4 del; 11 mod; 2997 unchg

This code review page was prepared using **/home/skuksenk/reps/webrev2/webrev.ksh** (vers 25.6-hg+openjdk.java.net).

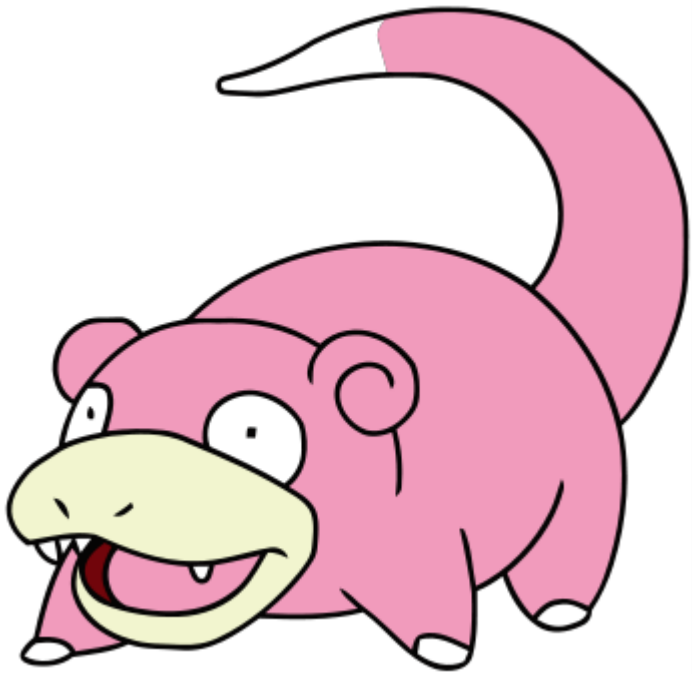
TreeMap size = 100 000



```java
public BigInteger fibo(int arg) {
    if (arg < 1) {
        throw new IllegalArgumentException();
    }
    if (arg <= 2) {
        return BigInteger.ONE;
    }
    return fibo(arg - 1).add(fibo(arg - 2));
}
```

```java
public BigInteger fibo(int arg) {
    if (arg < 1) {
        throw new IllegalArgumentException();
    }
    if (arg <= 2) {
        return BigInteger.ONE;
    }
    return fibo(arg - 1).add(fibo(arg - 2));
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
}
```

```java
public BigInteger fibo(int arg) {
    if (arg < 1) {
        throw new IllegalArgumentException();
    }
    if (arg <= 2) {
        return BigInteger.ONE;
    }
    return fibo(arg - 1).add(fibo(arg - 2));
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
}
```



```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {
    if (arg < 1) {
        throw new IllegalArgumentException();
    }
    if (arg <= 2) {
        return BigInteger.ONE;
    }
    return fibo(arg - 1).add(fibo(arg - 2));
}
```

```
public BigInteger fibo(int arg) {
    BigInteger value = map.get(arg);
    if (value == null) {
        value = calcFibo(arg);
        map.put(arg, value);
    }
    return value;
}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    BigInteger value = map.get(arg);
    if (value == null) {
        value = calcFibo(arg);
        map.put(arg, value);
    }
    return value;
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    BigInteger value = map.get(arg);
    if (value == null) {
        value = calcFibo(arg);
        map.put(arg, value);
    }
    return value;
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // 354224848179261915075
}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    BigInteger value = map.get(arg);
    if (value == null) {
        value = calcFibo(arg);
        map.put(arg, value);
    }
    return value;
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // 354224848179261915075
    System.out.println(fibo.map.size());
    // 100
}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    BigInteger value = map.get(arg);
    if (value == null) {
        value
        map.p
    }
    return va
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // 354224848179261915075
    System.out.println(fibo.map.size());
    // 100
}
```

Flip '=='

Invert 'if' condition

Negate '==' to '!='

Replace with 'computeIfAbsent' method call

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075




}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // null


}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // null
    System.out.println(fibo.map.size());
    // 185
}
```

```
public V computeIfAbsent(K key,
                         Function<? super K, ? extends V> mappingFunction) {
  1. Найти место в хэш-таблице
  2. Если там есть запись, вернуть значение из неё
  3. Иначе вызвать функцию mappingFunction
  4. Если функция вернула null, вернуть null
  5. Иначе создать запись и поместить её в ранее найденное место
  5*. При необходимости увеличить хэш-таблицу
  6. Увеличить size на 1
  7. Вернуть то что вернула функция на шаге 3.
}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
}
```

```
Map<Integer, BigInteger> map = new HashMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
}
```


```
Exception in thread "main" java.util.ConcurrentModificationException
 at java.base/java.util.HashMap.computeIfAbsent(HashMap.java:1226)
 at Fibo.fibo(Fibo.java:18)
 at Fibo.calcFibo(Fibo.java:14)
 at java.base/java.util.HashMap.computeIfAbsent(HashMap.java:1225)
 at Fibo.fibo(Fibo.java:18)
 at Fibo.calcFibo(Fibo.java:14)
 at java.base/java.util.HashMap.computeIfAbsent(HashMap.java:1225)
 at Fibo.fibo(Fibo.java:18)
 at Fibo.calcFibo(Fibo.java:14)
 …
```

```
Map<Integer, BigInteger> map = new TreeMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // 354224848179261915075
    System.out.println(fibo.map.size());
    // 100
}
```

```
Map<Integer, BigInteger> map = new TreeMap<>();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
}
```

Java 15

```
Exception in thread "main" java.util.ConcurrentModificationException
 at java.base/java.util.TreeMap.callMappingFunctionWithCheck(TreeMap.java:742)
 at java.base/java.util.TreeMap.computeIfAbsent(TreeMap.java:558)
 at Fibo.fibo(Fibo.java:18)
 at Fibo.calcFibo(Fibo.java:14)
 at java.base/java.util.TreeMap.callMappingFunctionWithCheck(TreeMap.java:740)
 at java.base/java.util.TreeMap.computeIfAbsent(TreeMap.java:558)
 at Fibo.fibo(Fibo.java:18)
 at Fibo.calcFibo(Fibo.java:14)
 at java.base/java.util.TreeMap.callMappingFunctionWithCheck(TreeMap.java:740)
 …
```

```
Map<Integer, BigInteger> map =
  new TreeMap<Integer, BigInteger>().descendingMap();

private BigInteger calcFibo(int arg) {…}

public BigInteger fibo(int arg) {
    return map.computeIfAbsent(arg, this::calcFibo);
}

public static void main(String[] args) {
    Fibo fibo = new Fibo();
    System.out.println(fibo.fibo(100));
    // 354224848179261915075
    System.out.println(fibo.map.get(100));
    // 354224848179261915075
    System.out.println(fibo.map.size());
    // 100
}
```

computeIfAbsent

```
default V computeIfAbsent(K key,
                          Function<? super K,? extends V> mappingFunction)
```

If the specified key is not already associated with a value (or is mapped to `null`), attempts to compute its value using the given mapping function and enters it into this map unless `null`.

If the mapping function returns `null`, no mapping is recorded. If the mapping function itself throws an (unchecked) exception, the exception is rethrown, and no mapping is recorded. The most common usage is to construct a new object serving as an initial mapped value or memoized result, as in:

```
map.computeIfAbsent(key, k -> new Value(f(k)));
```

Or to implement a multi-value map, `Map<K,Collection<V>>`, supporting multiple values per key:

```
map.computeIfAbsent(key, k -> new HashSet<V>()).add(v);
```

The mapping function should not modify this map during computation.

Implementation Requirements:

The default implementation is equivalent to the following steps for this `map`, then returning the current value or `null` if now absent:

```
if (map.get(key) == null) {
    V newValue = mappingFunction.apply(key);
    if (newValue != null)
        map.put(key, newValue);
}
```

ArrayList.removeIf

JDK-8071477

Collection.removeIf

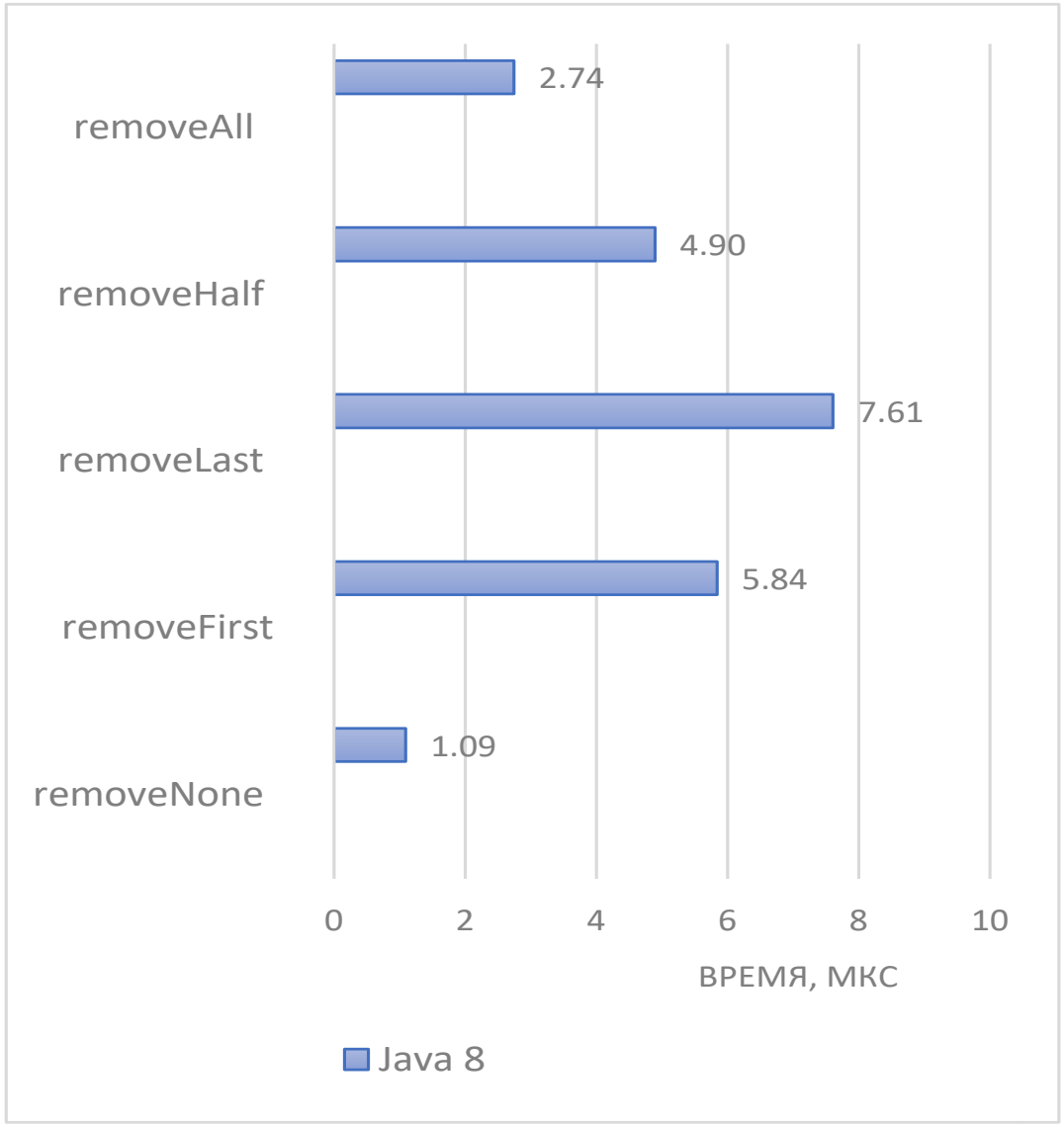
```
default boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    boolean removed = false;
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
        if (filter.test(each.next())) {
            each.remove();
            removed = true;
        }
    }
    return removed;
}
```

```
data = new ArrayList<>();
for (int i = 0; i < size; i++) {
  data.add(i);
}
```

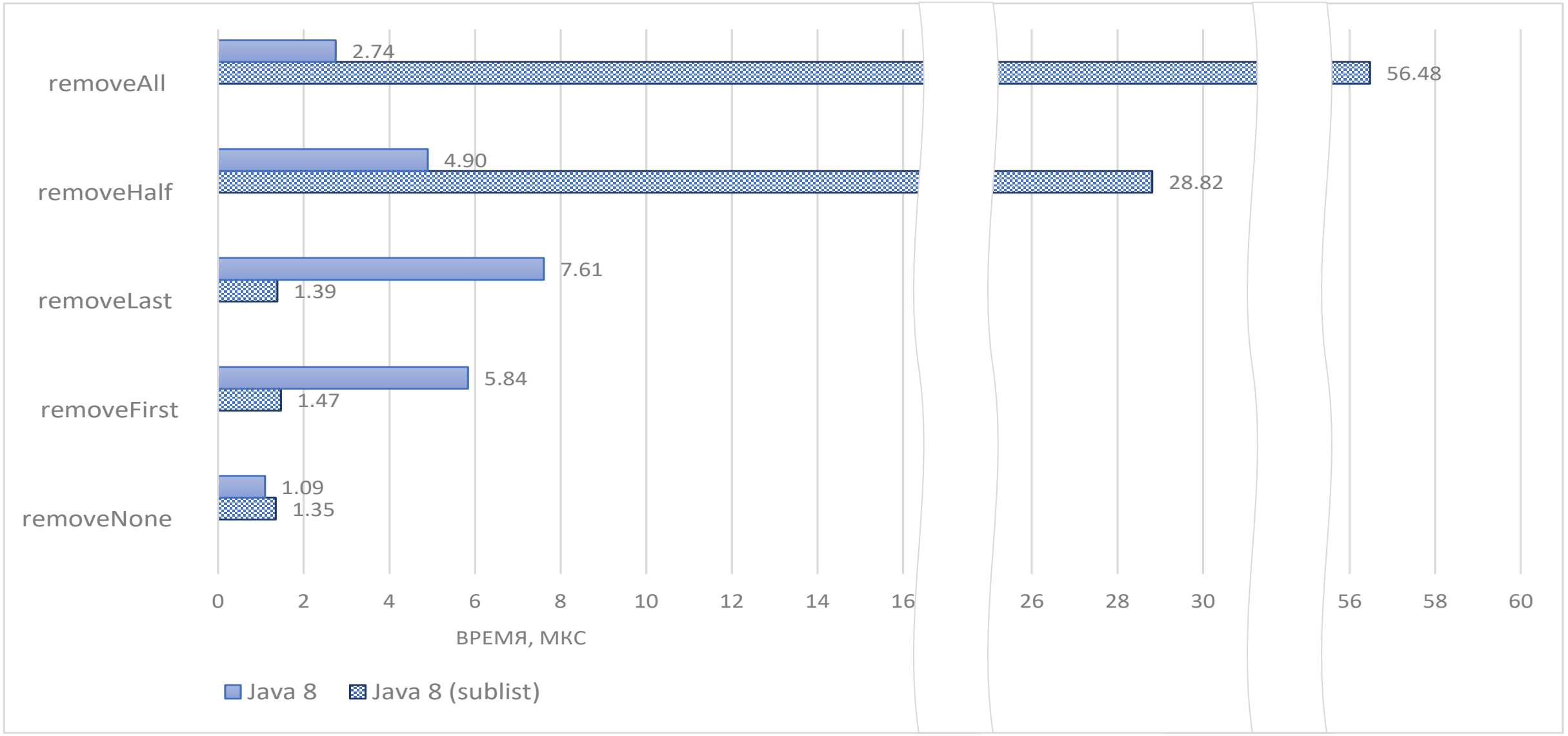
```
data = new ArrayList<>();
for (int i = 0; i < size; i++) {
  data.add(i);
}
```

```
removeAll:   list.removeIf(x -> true);

removeHalf:  list.removeIf(x -> x % 2 == 0);

removeLast:  list.removeIf(x -> x == size - 1);

removeFirst: list.removeIf(x -> x == 0);

removeNone:  list.removeIf(x -> x == size);
```

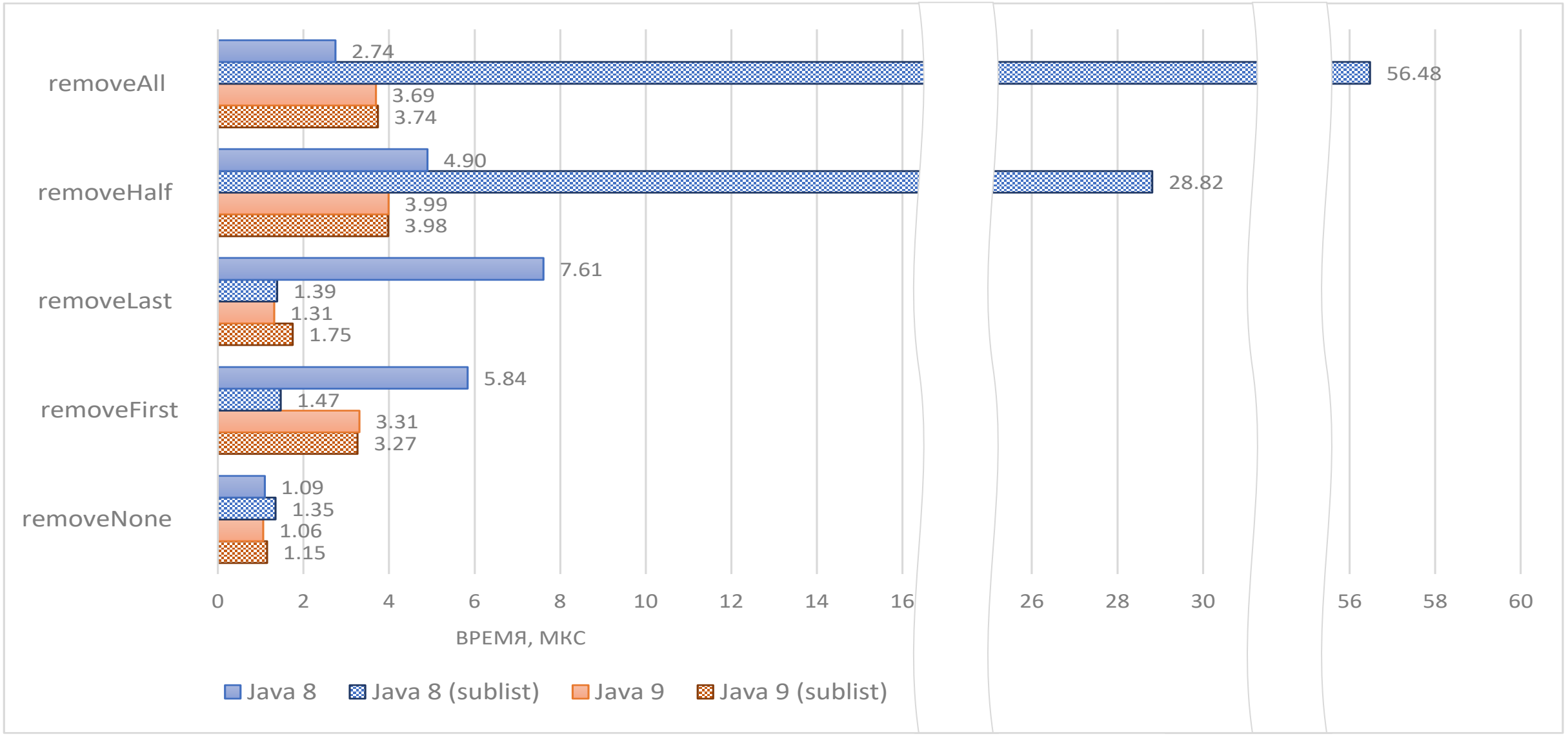
ArrayList.removeIf(), size = 1000



ArrayList.removeIf() & ArrayList.subList(0, size).removeIf(), size = 1000



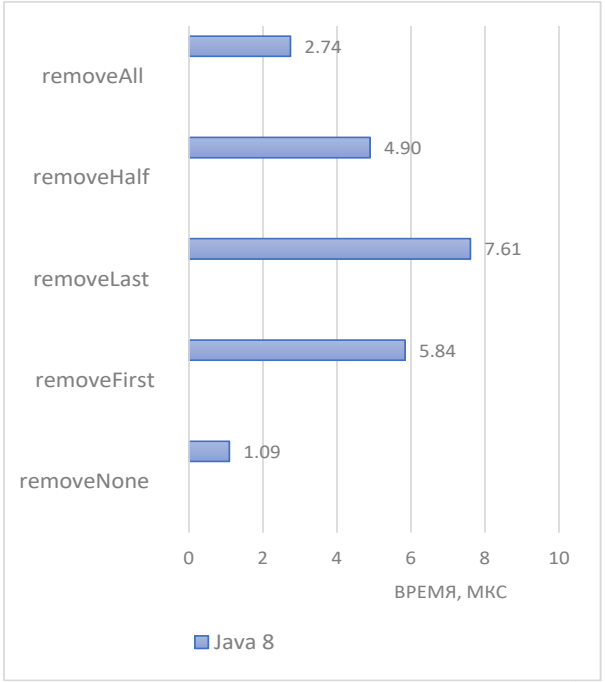
ArrayList.removeIf() & ArrayList.subList(0, size).removeIf(), size = 1000




```
public boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    // figure out which elements are to be removed
    // any exception thrown from the filter predicate at this stage
    // will leave the collection unmodified
    int removeCount = 0;
    final BitSet removeSet = new BitSet(size);
    final int expectedModCount = modCount;
    final int size = this.size;
    for (int i=0; modCount == expectedModCount && i < size; i++) {
        @SuppressWarnings("unchecked")
        final E element = (E) elementData[i];
        if (filter.test(element)) {
            removeSet.set(i);
            removeCount++;
        }
    }
    if (modCount != expectedModCount) {
        throw new ConcurrentModificationException();
    }
    …
```


```
    // shift surviving elements left over the spaces
    // left by removed elements
    final boolean anyToRemove = removeCount > 0;
    if (anyToRemove) {
        final int newSize = size - removeCount;
        for (int i=0, j=0; (i < size) && (j < newSize); i++, j++) {
            i = removeSet.nextClearBit(i);
            elementData[j] = elementData[i];
        }
        for (int k=newSize; k < size; k++) {
            elementData[k] = null;  // Let gc do its work
        }
        this.size = newSize;
        if (modCount != expectedModCount) {
            throw new ConcurrentModificationException();
        }
        modCount++;
    }

    return anyToRemove;
}
```

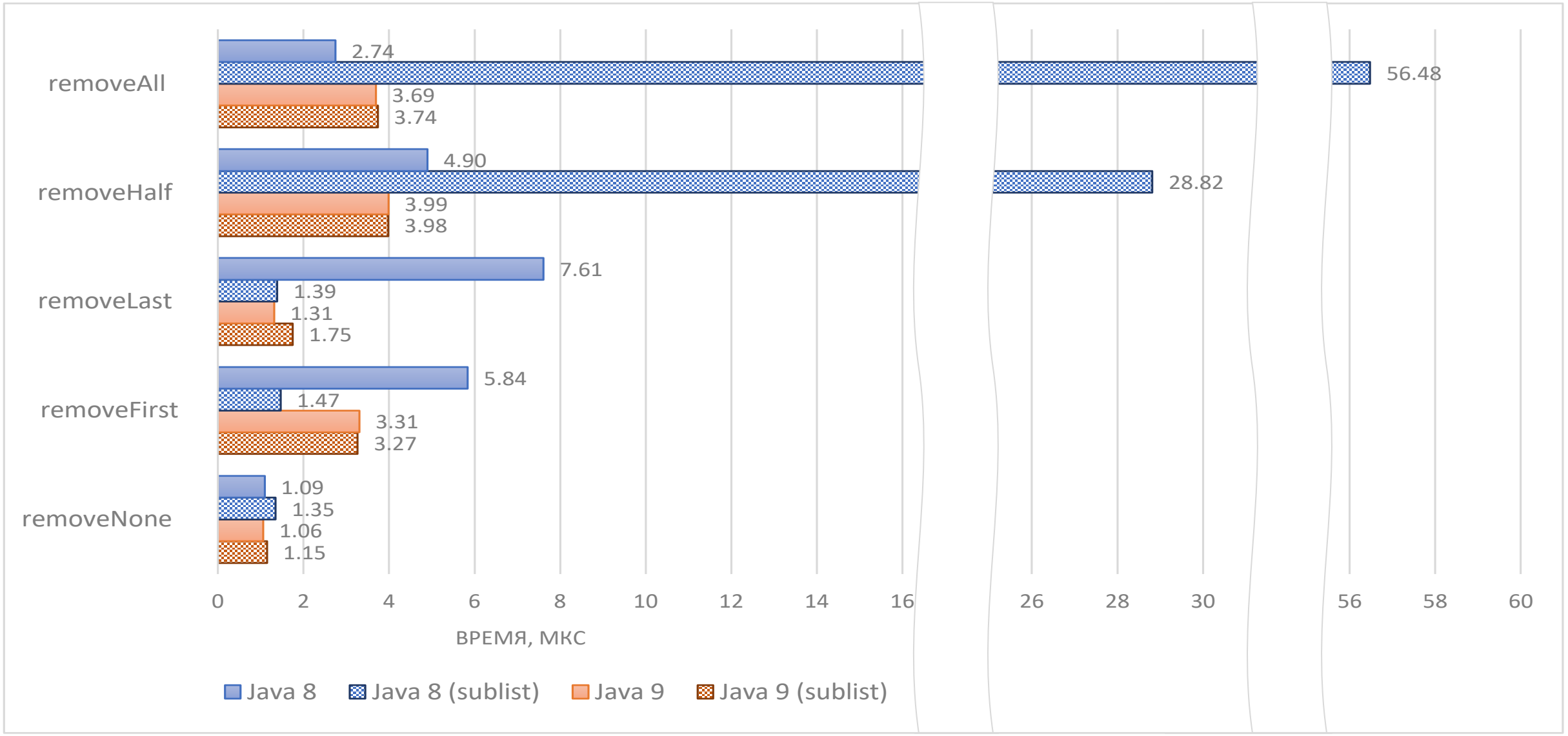

```
    // shift surviving elements left over the spaces
    // left by removed elements
    final boolean anyToRemove = removeCount > 0;
    if (anyToRemove) {
        final int newSize = size - removeCount;
        for (int i=0, j=0; (i < size) && (j < newSize); i++, j++) {
            i = removeSet.nextClearBit(i);
            elementData[j] = elementData[i];
        }
        for (int k=newSize; k < size; k++) {
            elementData[k] = null;  // Let gc do its work
        }
        this.size = newSize;
        if (modCount != expectedModCount) {
            throw new ConcurrentModificationException();
        }
        modCount++;
    }

    return anyToRemove;
}
```




```
boolean removeIf(Predicate<? super E> filter, int i, final int end) {
    Objects.requireNonNull(filter);
    int expectedModCount = modCount;
    final Object[] es = elementData;
    // Optimize for initial run of survivors
    for (; i < end && !filter.test(elementAt(es, i)); i++)
        ;
    // Tolerate predicates that reentrantly access the collection for
    // read (but writers still get CME), so traverse once to find
    // elements to delete, a second pass to physically expunge.
    if (i < end) {
        …
    } else {
        if (modCount != expectedModCount)
            throw new ConcurrentModificationException();
        return false;
    }
}
```


```
if (i < end) {
    final int beg = i;
    final long[] deathRow = nBits(end - beg); // new long[((n - 1) >> 6) + 1];
    deathRow[0] = 1L;   // set bit 0
    for (i = beg + 1; i < end; i++)
        if (filter.test(elementAt(es, i)))
            setBit(deathRow, i - beg); // bits[i >> 6] |= 1L << i;
    if (modCount != expectedModCount)
        throw new ConcurrentModificationException();
    modCount++;
    int w = beg;
    for (i = beg; i < end; i++)
        if (isClear(deathRow, i - beg)) // (bits[i >> 6] & (1L << i)) == 0;
            es[w++] = es[i];
    shiftTailOverGap(es, w, end);
    return true;
} else { … }
```

ArrayList.removeIf() & ArrayList.subList(0, size).removeIf(), size = 1000



HashSet.removeIf

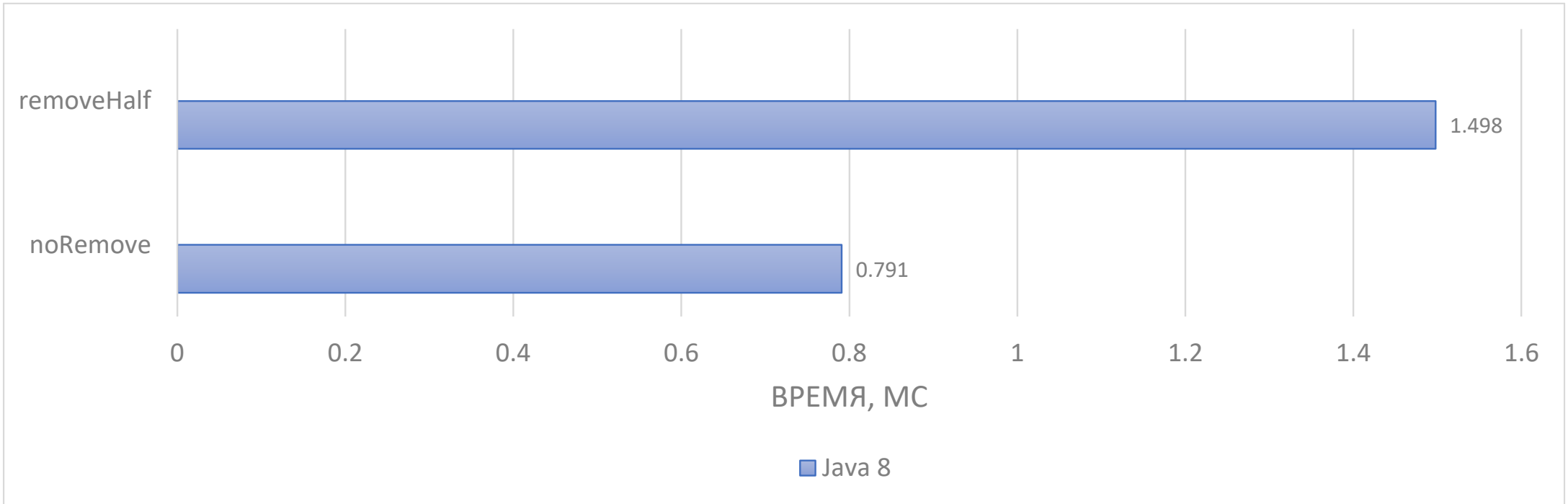
JDK-8170733

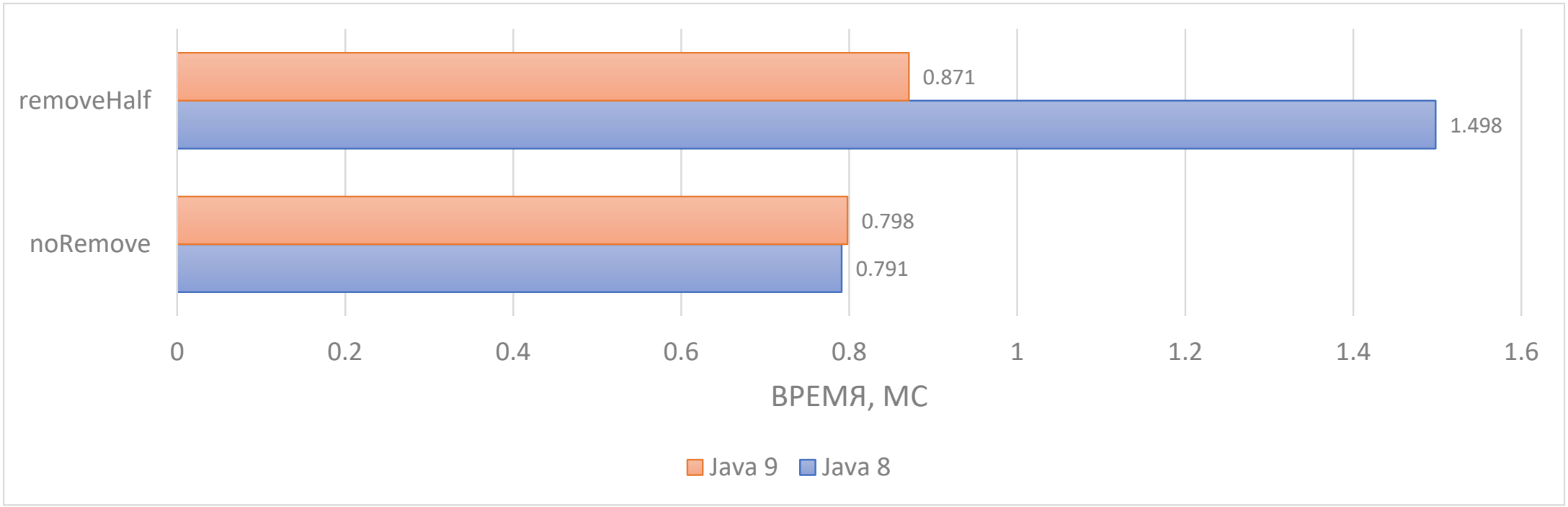
```
HashSet<List<Integer>> set;

@Setup
public void setup() {
  set = IntStream.range(0, 1000)
      .mapToObj(i -> IntStream.range(0, i).boxed().collect(Collectors.toList()))
      .collect(Collectors.toCollection(HashSet::new));
}
```

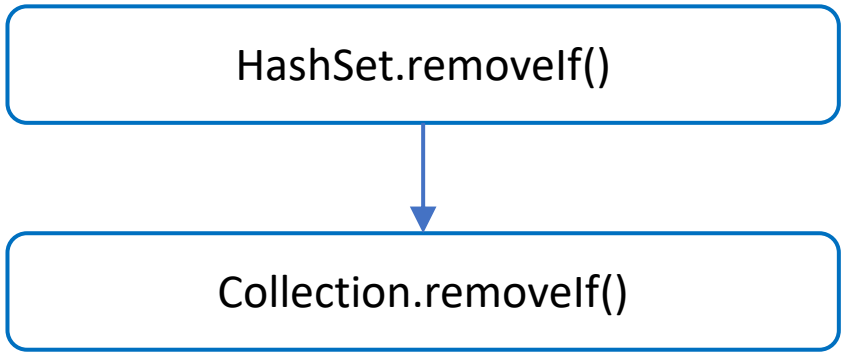
```
[]
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
…
```

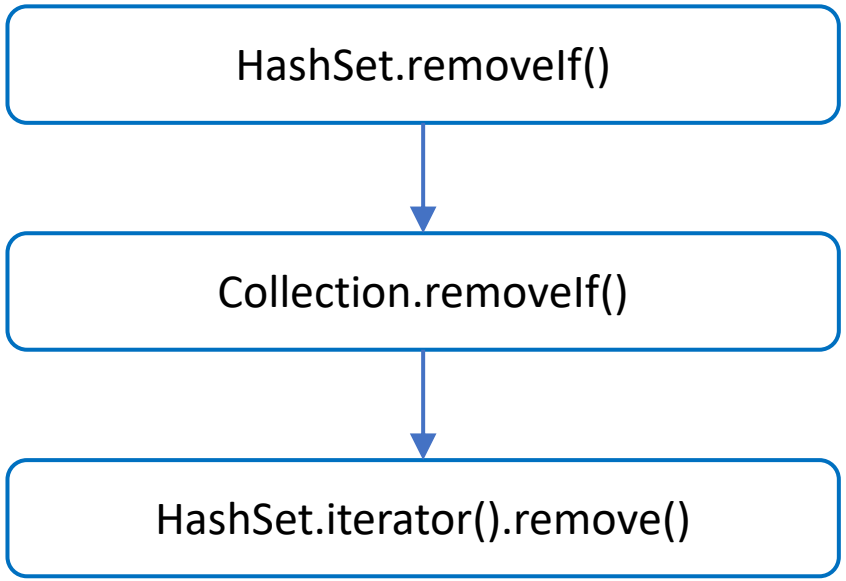
```java
@Benchmark
public Set<List<Integer>> removeHalf() {
  Set<List<Integer>> copy = new HashSet<>(set);
  copy.removeIf(list -> list.size() > 500);
  return copy;
}
```

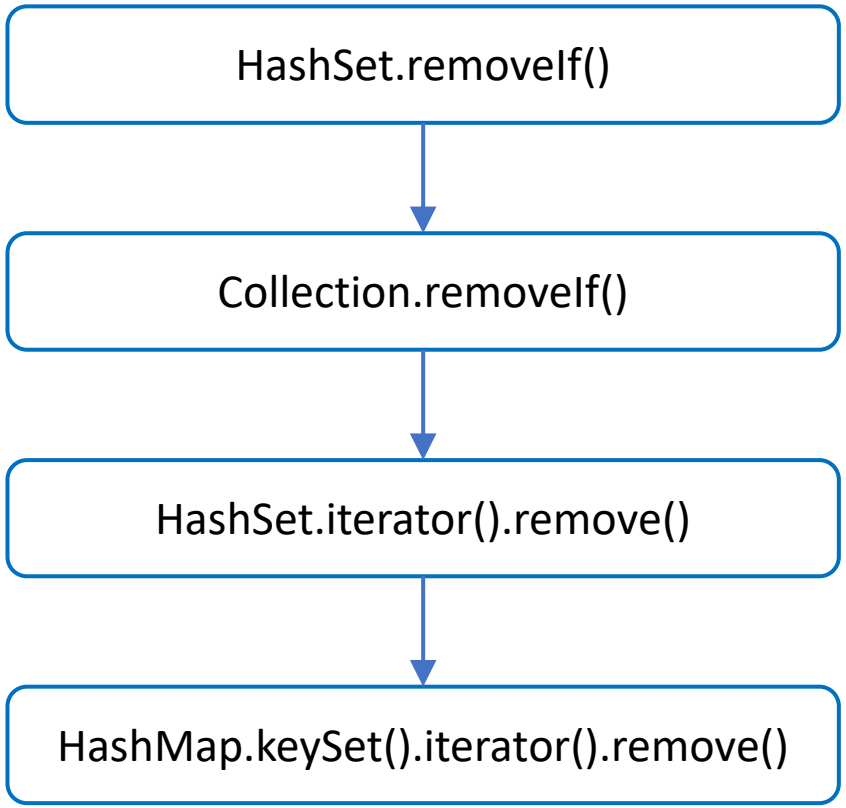
```java
@Benchmark
public Set<List<Integer>> removeHalf() {
  Set<List<Integer>> copy = new HashSet<>(set);
  copy.removeIf(list -> list.size() > 500);
  return copy;
}

@Benchmark
public Set<List<Integer>> noRemove() {
  return new HashSet<>(set);
}
```

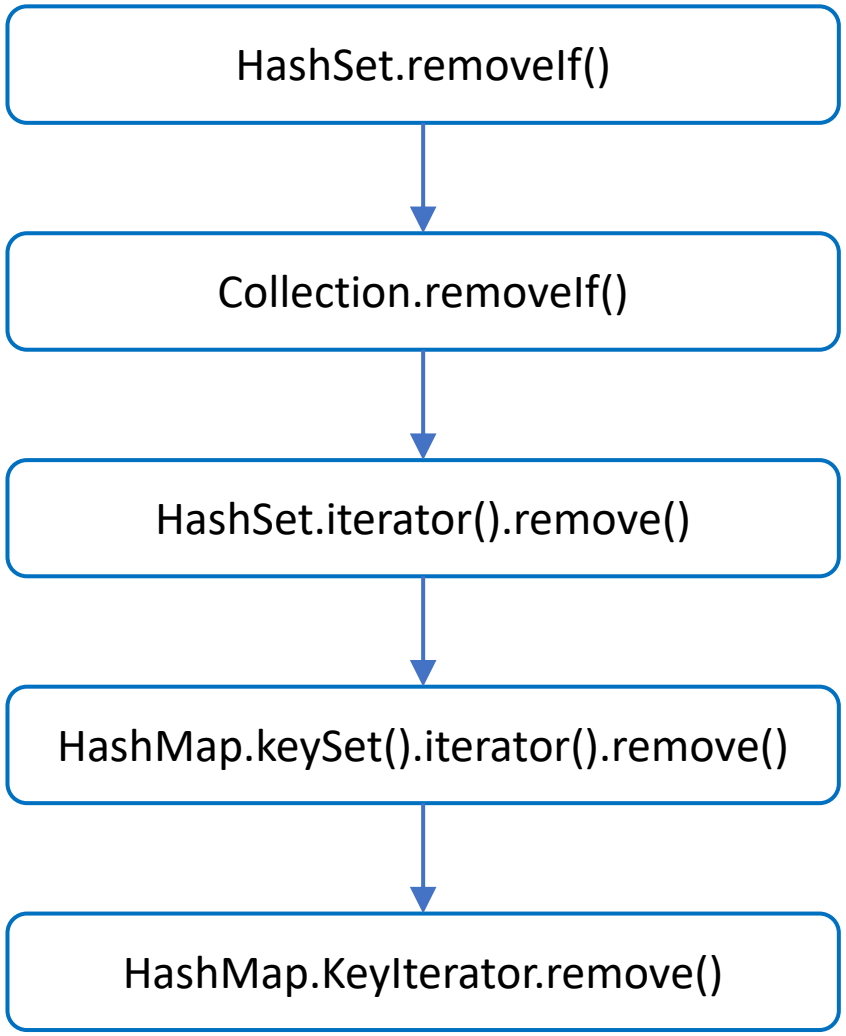

removeHalf
noRemove
1.498
0.791
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
ВРЕМЯ, МС
Java 8

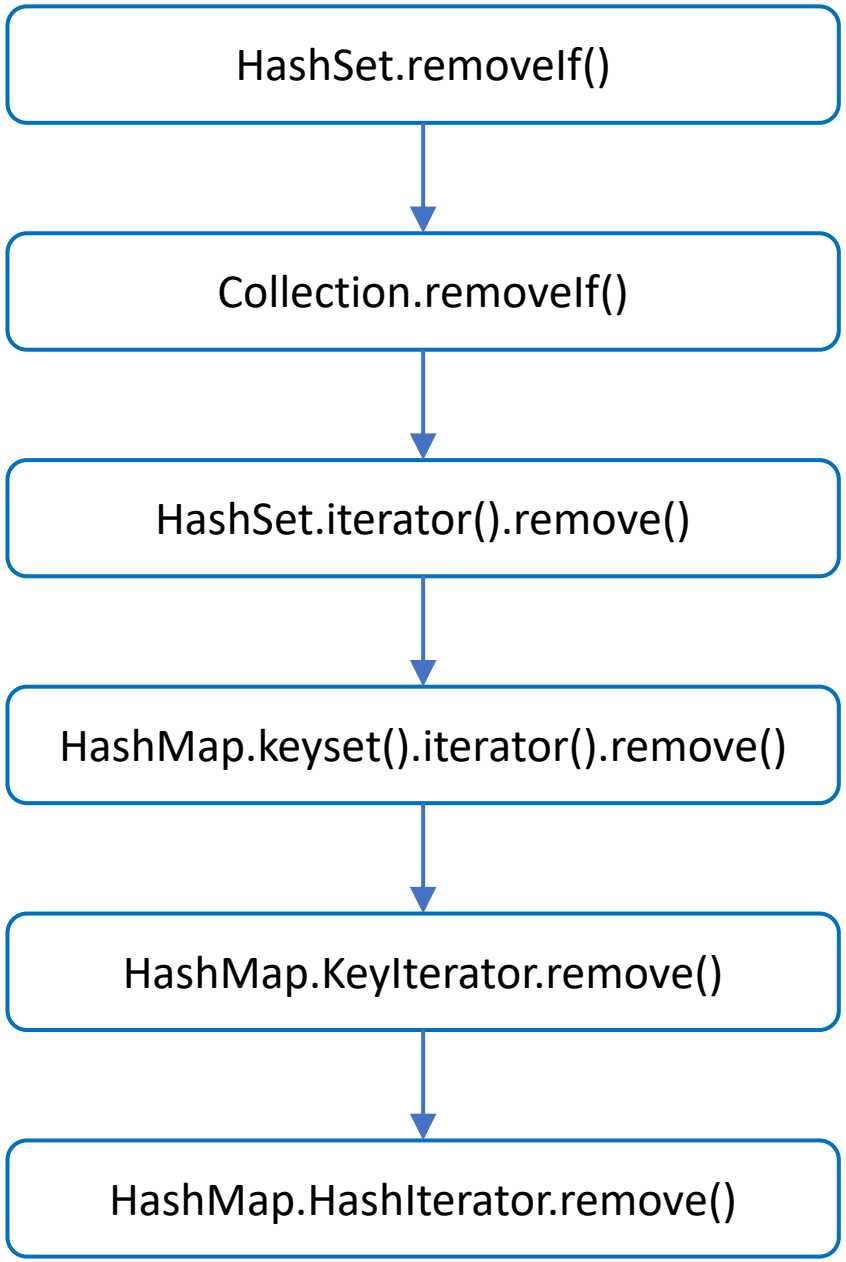

removeHalf
noRemove
0.871
1.498
0.798
0.791
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
ВРЕМЯ, МС
Java 9
Java 8


HashSet.removeIf()
Collection.removeIf()


HashSet.removeIf()
Collection.removeIf()
HashSet.iterator().remove()


HashSet.removeIf()
Collection.removeIf()
HashSet.iterator().remove()
HashMap.keySet().iterator().remove()


HashSet.removeIf()
Collection.removeIf()
HashSet.iterator().remove()
HashMap.keySet().iterator().remove()
HashMap.KeyIterator.remove()


HashSet.removeIf()
Collection.removeIf()
HashSet.iterator().remove()
HashMap.keyset().iterator().remove()
HashMap.KeyIterator.remove()
HashMap.HashIterator.remove()

```
public final void remove() {
    Node<K,V> p = current;
    if (p == null)
        throw new IllegalStateException();
    if (modCount != expectedModCount)
        throw new ConcurrentModificationException();
    current = null;
    K key = p.key;
    removeNode(hash(key), key, null, false, false);
    removeNode(p.hash, p.key, null, false, false);
    expectedModCount = modCount;
}
```

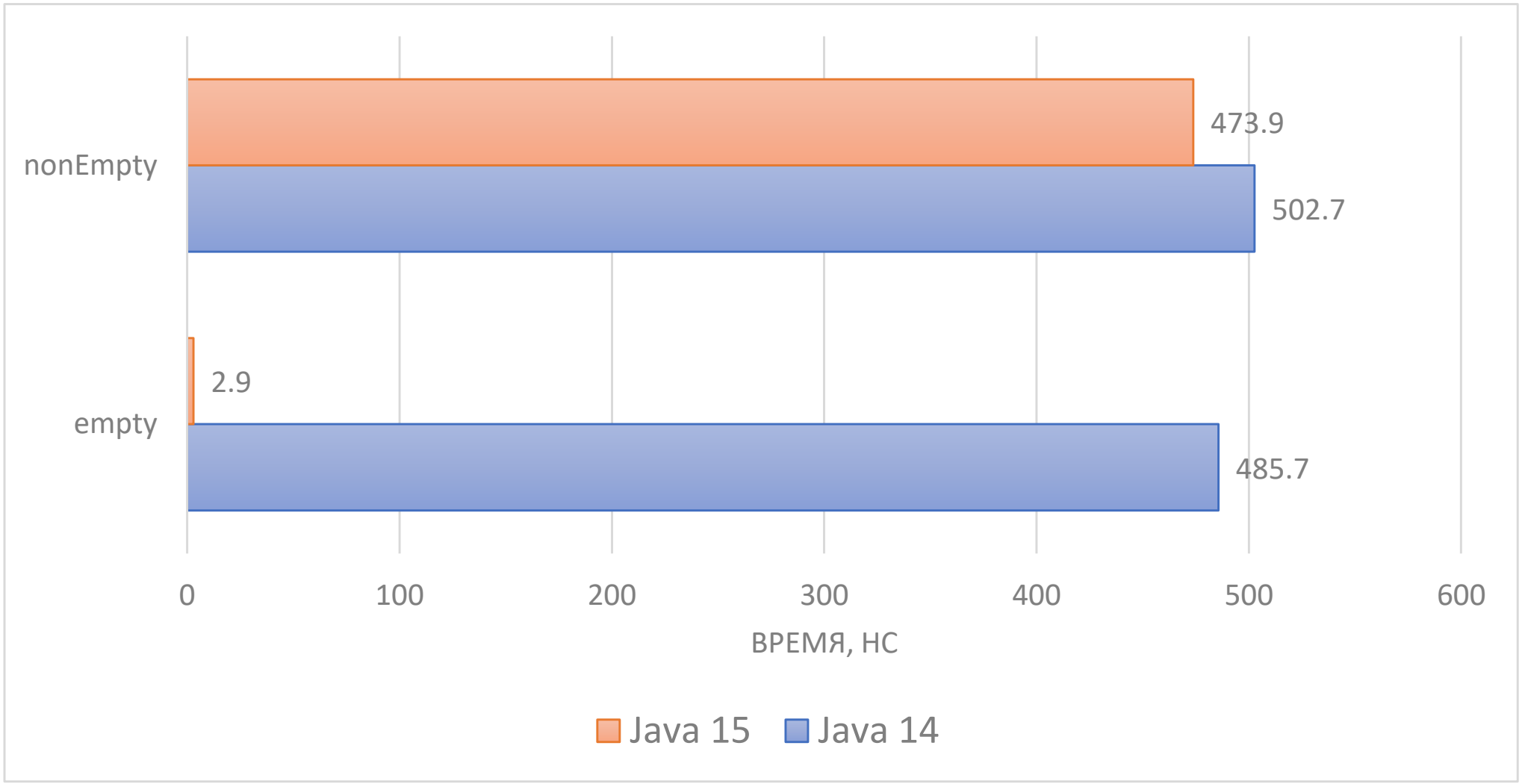
HashSet
HashMap.keySet
HashMap.values
HashMap.entrySet
LinkedHashSet
LinkedHashMap.keySet
LinkedHashMap.values
LinkedHashMap.entrySet

iterator().remove()
removeIf()
removeAll()
retainAll()

HashMap.containsKey

JDK-8245677

```
HashMap<List<Integer>, String> emptyMap;
HashMap<List<Integer>, String> nonEmptyMap;
List<Integer> key;

@Setup
public void setup() {
  emptyMap = new HashMap<>();
  nonEmptyMap = new HashMap<>();
  nonEmptyMap.put(Collections.emptyList(), "");
  key = IntStream.range(0, 500).boxed().collect(Collectors.toList());
}

@Benchmark
public boolean containsInEmpty() {
  return emptyMap.containsKey(key);
}

@Benchmark
public boolean containsInNonEmpty() {
  return nonEmptyMap.containsKey(key);
}
```


nonEmpty
473.9
502.7
empty
2.9
485.7
0
100
200
300
400
500
600
ВРЕМЯ, НС
Java 15
Java 14

```
final Node<K,V> getNode(int hash, Object key) {
    Node<K,V>[] tab; Node<K,V> first, e; int n; K k;
    if ((tab = table) != null && (n = tab.length) > 0 &&
        (first = tab[(n - 1) & hash]) != null) {
        …
    }
    return null;
}
```

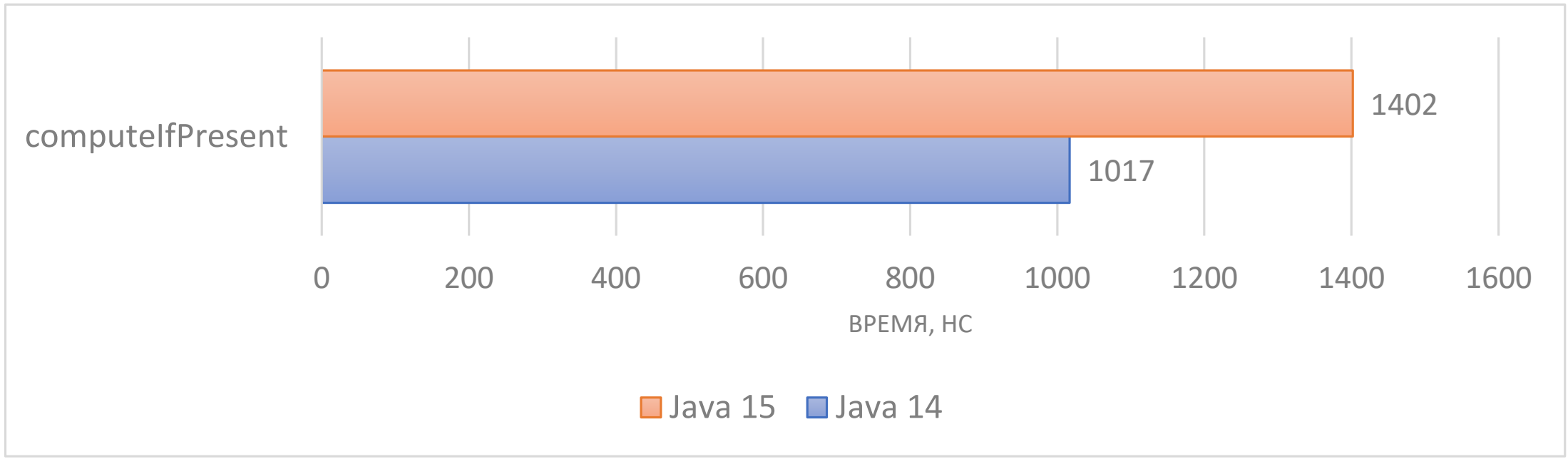

```java
final Node<K,V> getNode(int hash, Object key) {
    Node<K,V>[] tab; Node<K,V> first, e; int n; K k;
    if ((tab = table) != null && (n = tab.length) > 0 &&
        (first = tab[(n - 1) & hash]) != null) {
        …
    }
    return null;
}
```


```java
final Node<K,V> getNode(Object key) {
    Node<K,V>[] tab; Node<K,V> first, e; int n, hash; K k;
    if ((tab = table) != null && (n = tab.length) > 0 &&
        (first = tab[(n - 1) & (hash = hash(key))]) != null) {
        …
    }
    return null;
}
```

```java
public boolean containsKey(Object key) {
    return getNode(hash(key), key) ≠ null;
    return getNode(key) ≠ null;
}

public V get(Object key) {
    Node<K,V> e;
    return (e = getNode(hash(key), key)) == null ? null : e.value;
    return (e = getNode(key)) == null ? null : e.value;
}

public V getOrDefault(Object key, V defaultValue) {
    Node<K,V> e;
    return (e = getNode(hash(key), key)) == null ? defaultValue : e.value;
    return (e = getNode(key)) == null ? defaultValue : e.value;
}
```

```
public V computeIfPresent(K key,
                          BiFunction<? super K, ? super V, ? extends V> remappingFunction) {
    if (remappingFunction == null)
        throw new NullPointerException();
    Node<K,V> e; V oldValue;
    int hash = hash(key);
    if ((e = getNode(hash, key)) != null &&
    if ((e = getNode(key)) != null &&
        (oldValue = e.value) != null) {
        int mc = modCount;
        V v = remappingFunction.apply(key, oldValue);
        if (mc != modCount) { throw new ConcurrentModificationException(); }
        if (v != null) {
            e.value = v;
            afterNodeAccess(e);
            return v;
        }
        else
        else {
            int hash = hash(key);
            removeNode(hash, key, null, false, true);
        }
    }
    return null;
}
```

```java
@Benchmark
public String addRemove() {
  emptyMap.put(key, "");
  return emptyMap.computeIfPresent(key, (k, v) -> null);
}
```



Class.getSimpleName

JDK-8187123

```java
System.out.println(Map.Entry.class.getName());
// -> java.util.Map$Entry

System.out.println(Map.Entry.class.getCanonicalName());
// -> java.util.Map.Entry

System.out.println(Map.Entry.class.getSimpleName());
// -> Entry
```

```
                                        Java 8

Map.Entry.class.getName()                4.2 ns
  Integer.class.getName()                4.2 ns

Map.Entry.class.getCanonicalName()     506.7 ns
  Integer.class.getCanonicalName()      93.8 ns

Map.Entry.class.getSimpleName()        164.0 ns
  Integer.class.getSimpleName()         83.7 ns
```

	Java 8	Java 11
`Map.Entry.class.getName()`	4.2 ns	4.4 ns
`Integer.class.getName()`	4.2 ns	4.4 ns
`Map.Entry.class.getCanonicalName()`	506.7 ns	5.7 ns
`Integer.class.getCanonicalName()`	93.8 ns	5.7 ns
`Map.Entry.class.getSimpleName()`	164.0 ns	5.7 ns
`Integer.class.getSimpleName()`	83.7 ns	5.7 ns

```
public String getName() {
    String name = this.name;
    if (name == null)
        this.name = name = getName0();
    return name;
}

// cache the name to reduce the number of calls into the VM
private transient String name;
private native String getName0();
```


```
public String getCanonicalName() {
    if (isArray()) {
        String canonicalName = getComponentType().getCanonicalName();
        if (canonicalName != null)
            return canonicalName + "[]";
        else
            return null;
    }
    if (isLocalOrAnonymousClass())
        return null;
    Class<?> enclosingClass = getEnclosingClass();
    if (enclosingClass == null) { // top level class
        return getName();
    } else {
        String enclosingName = enclosingClass.getCanonicalName();
        if (enclosingName == null)
            return null;
        return enclosingName + "." + getSimpleName();
    }
}
```

JDK / JDK-8005232

(JEP-149) Class Instance size reduction

Details

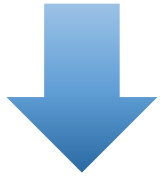
Type:	Enhancement	Status:	CLOSED
Priority:	P3	Resolution:	Fixed
Affects Version/s:	8	Fix Version/s:	8
Component/s:	core-libs		
Labels:	noreg-cleanup		
Subcomponent:	java.lang		
Resolved In Build:	b75		
CPU:	generic		
OS:	generic		
Verification:	Not verified		

This CR covers the "Class Instance Size Reduction" project of that JEP.

In Java 8, using a 32-bit example, a java.lang.Class instance is 112 bytes consisting of:

- 8 byte object header
- 20 declared fields (mostly references, some int)
- 5 injected fields (3 references, 2 ints)

That gives: 8 + (20*4) +(5*4) = 108 bytes. But as we need 8-byte alignment that increases to 112 bytes.



This is a saving of 7 reference fields ie. 28 bytes, resulting in a new Class instance size of 80 bytes. This saves a further 4 bytes due to the fields being 8-byte aligned without any need for padding. So overall we save 32 bytes per class instance.

The ReflectionData instance itself consumes 48 bytes, while a SoftReference consumes 32 bytes.

For classes that don't use reflection this is an obvious win of 32 bytes per class. For classes that use all 8 reflection caches this is also a win as we save 7 SoftReferences ie 224 bytes.

```
// reflection data that might get invalidated
// when JVM TI RedefineClasses() is called
private static class ReflectionData<T> {
    volatile Field[] declaredFields;
    volatile Field[] publicFields;
    volatile Method[] declaredMethods;
    volatile Method[] publicMethods;
    volatile Constructor<T>[] declaredConstructors;
    volatile Constructor<T>[] publicConstructors;
    // Intermediate results for getFields and getMethods
    volatile Field[] declaredPublicFields;
    volatile Method[] declaredPublicMethods;
    volatile Class<?>[] interfaces;

    // Value of classRedefinedCount when we created this ReflectionData instance
    final int redefinedCount;

    ReflectionData(int redefinedCount) {
        this.redefinedCount = redefinedCount;
    }
}
```

```
private transient volatile SoftReference<ReflectionData<T>> reflectionData;

// Incremented by the VM on each call to JVM TI RedefineClasses()
// that redefines this class or a superclass.
private transient volatile int classRedefinedCount;

// Lazily create and cache ReflectionData
private ReflectionData<T> reflectionData() {
    SoftReference<ReflectionData<T>> reflectionData = this.reflectionData;
    int classRedefinedCount = this.classRedefinedCount;
    ReflectionData<T> rd;
    if (reflectionData != null &&
        (rd = reflectionData.get()) != null &&
        rd.redefinedCount == classRedefinedCount) {
        return rd;
    }
    // else no SoftReference or cleared SoftReference or stale ReflectionData
    // -> create and replace new instance
    return newReflectionData(reflectionData, classRedefinedCount);
}
```

```
private ReflectionData<T> newReflectionData(
                                SoftReference<ReflectionData<T>> oldReflectionData,
                                                    int classRedefinedCount) {
    while (true) {
        ReflectionData<T> rd = new ReflectionData<>(classRedefinedCount);
        // try to CAS it...
        if (Atomic.casReflectionData(
                        this, oldReflectionData, new SoftReference<>(rd))) {
            return rd;
        }
        // else retry
        oldReflectionData = this.reflectionData;
        classRedefinedCount = this.classRedefinedCount;
        if (oldReflectionData != null &&
            (rd = oldReflectionData.get()) != null &&
            rd.redefinedCount == classRedefinedCount) {
            return rd;
        }
    }
}
```


```
 private static class ReflectionData<T> {
     volatile Field[] declaredFields;
     …
     volatile Class<?>[] interfaces;

+    // Cached names
+    String simpleName;
+    String canonicalName;
+    static final String NULL_SENTINEL = new String();

     …
 }
```


```
public String getCanonicalName() {
    ReflectionData<T> rd = reflectionData();
    String canonicalName = rd.canonicalName;
    if (canonicalName == null) {
        rd.canonicalName = canonicalName = getCanonicalName0();
    }
    return canonicalName == ReflectionData.NULL_SENTINEL ?
                            null : canonicalName;
}
```

	Java 8	Java 11
Map.Entry.class.getName()	4.2 ns	4.4 ns
Integer.class.getName()	4.2 ns	4.4 ns
Map.Entry.class.getCanonicalName()	506.7 ns	5.7 ns
Integer.class.getCanonicalName()	93.8 ns	5.7 ns

List of Patches for Amazon Corretto 8

PDF | Kindle | RSS

This section lists all the patches applied to OpenJDK for Amazon Corretto 8. We also provide links to the issues in the OpenJDK project.

...

[C8-10] Speed up Class.getSimpleName() and Class.getCanonicalName().

Memorization greatly speeds up these functions. This patch includes correctness unit tests. See JDK-8187123.

...

https://docs.aws.amazon.com/corretto/latest/corretto-8-ug/patches.html

[C8-10] Speed up Class.getSimpleName() and Class.getCanonicalName().

Browse files

Memoization greatly speeds up these functions. This patch includes regression unit tests.

develop 8.252.09.2 ... 8.202.08.2

Eric Edens authored and **bernd-aws** committed on 13 Mar 2018 1 parent 3a908ed commit 5916600bcfa6bd3ccd5a5b1bf4c2b40ad7bb1893

Showing 2 changed files with **100 additions** and **0 deletions**. Unified Split

28 src/jdk/src/share/classes/java/lang/Class.java

```
@@ -1303,6 +1303,20 @@ boolean isPartial() {
1303 1303          * @since 1.5
1304 1304          */
1305 1305         public String getSimpleName() {
1306      +          String simpleName = this.simpleName;
1307      +          if (simpleName == null) {
1308      +              this.simpleName = simpleName = getSimpleNameImpl();
1309      +          }
1310      +          return simpleName;
1311      +      }
1312      +
1313      +      // cache the simple name to reduce the latency of calling getSimpleName()
1314      +      private transient String simpleName;
1315      +
1316      +      /**
1317      +       * Non-cached implementation of getSimpleName
1318      +       */
1319      +      private String getSimpleNameImpl() {
```

	Java 8	Java 11	Java 8 (Corretto)
`Map.Entry.class.getName()`	4.2 ns	4.4 ns	4.2 ns
`  Integer.class.getName()`	4.2 ns	4.4 ns	4.2 ns
`Map.Entry.class.getCanonicalName()`	506.7 ns	5.7 ns	4.2 ns
`  Integer.class.getCanonicalName()`	93.8 ns	5.7 ns	4.1 ns
`Map.Entry.class.getSimpleName()`	164.0 ns	5.7 ns	4.2 ns
`  Integer.class.getSimpleName()`	83.7 ns	5.7 ns	4.2 ns

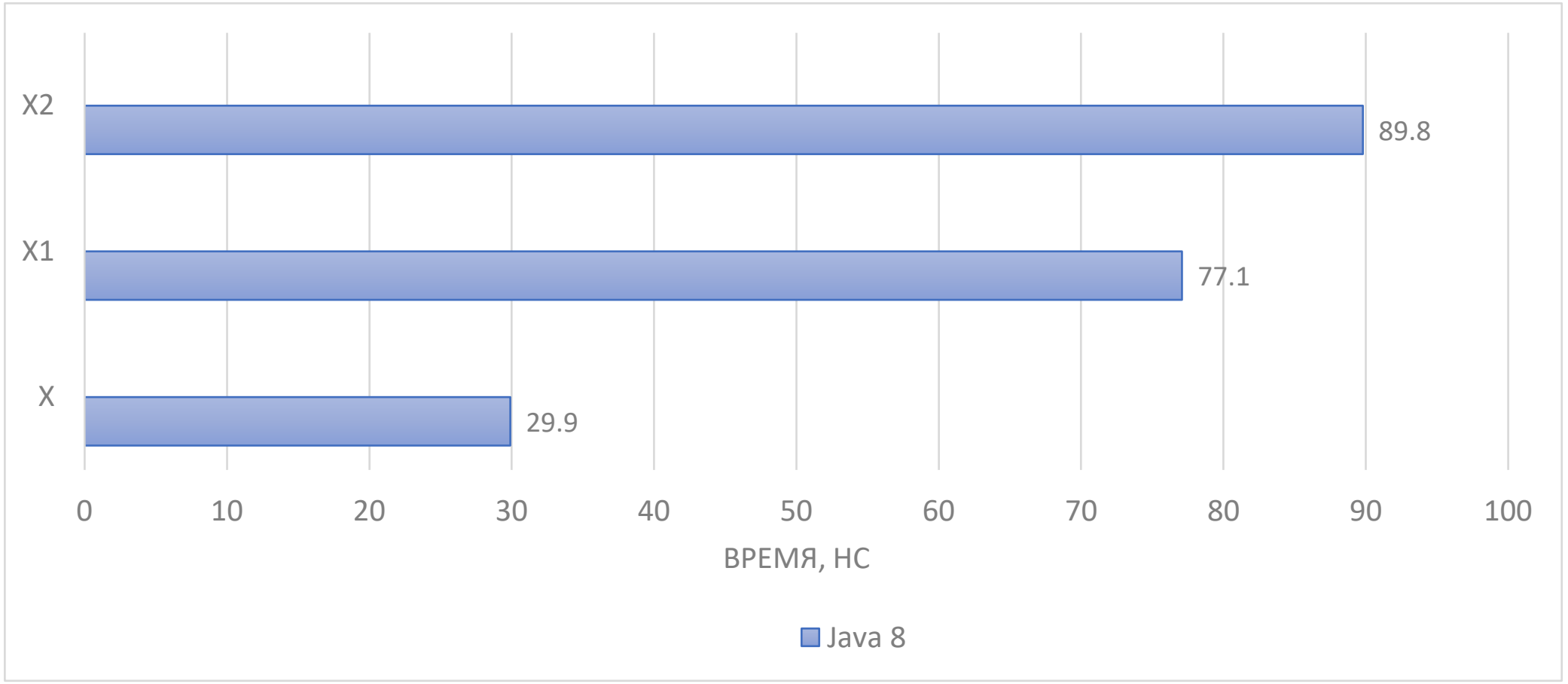
Class.getConstructor

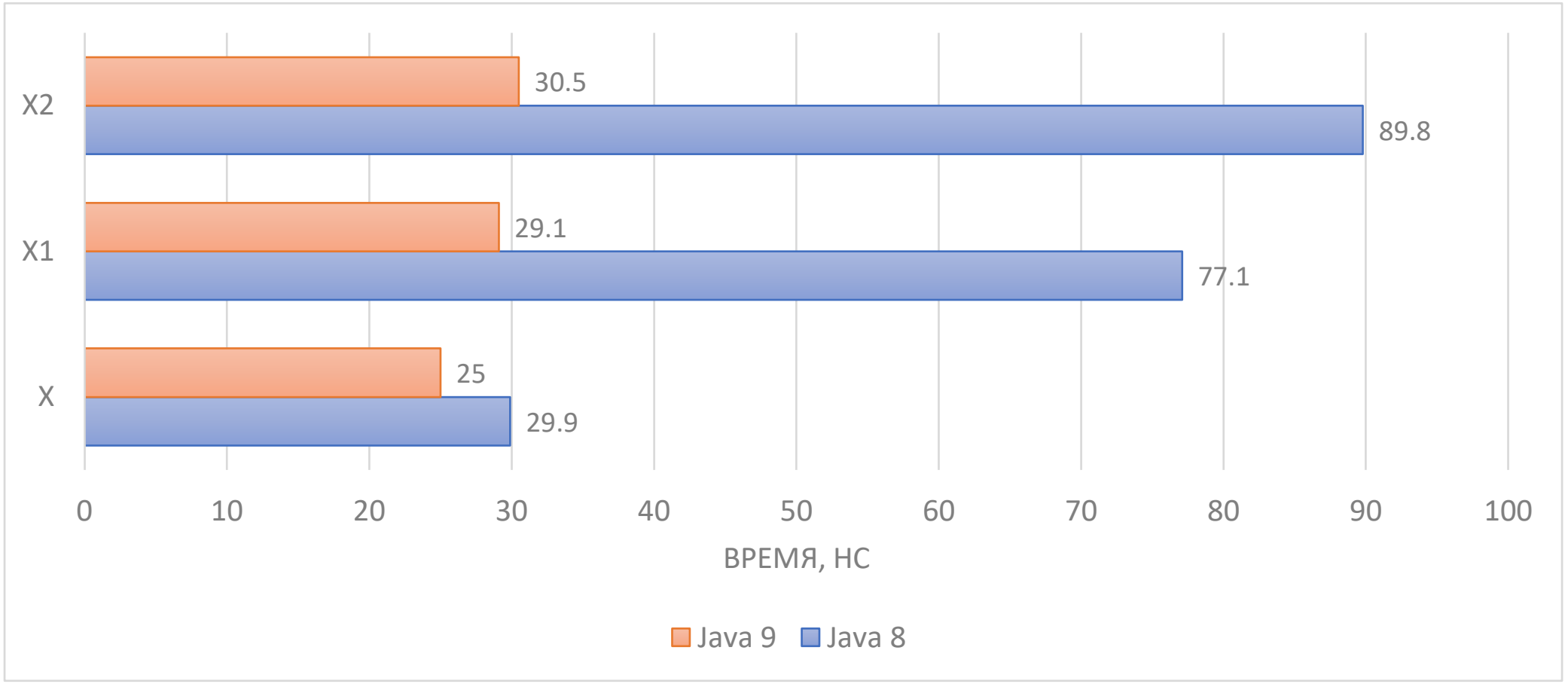
JDK-8172190

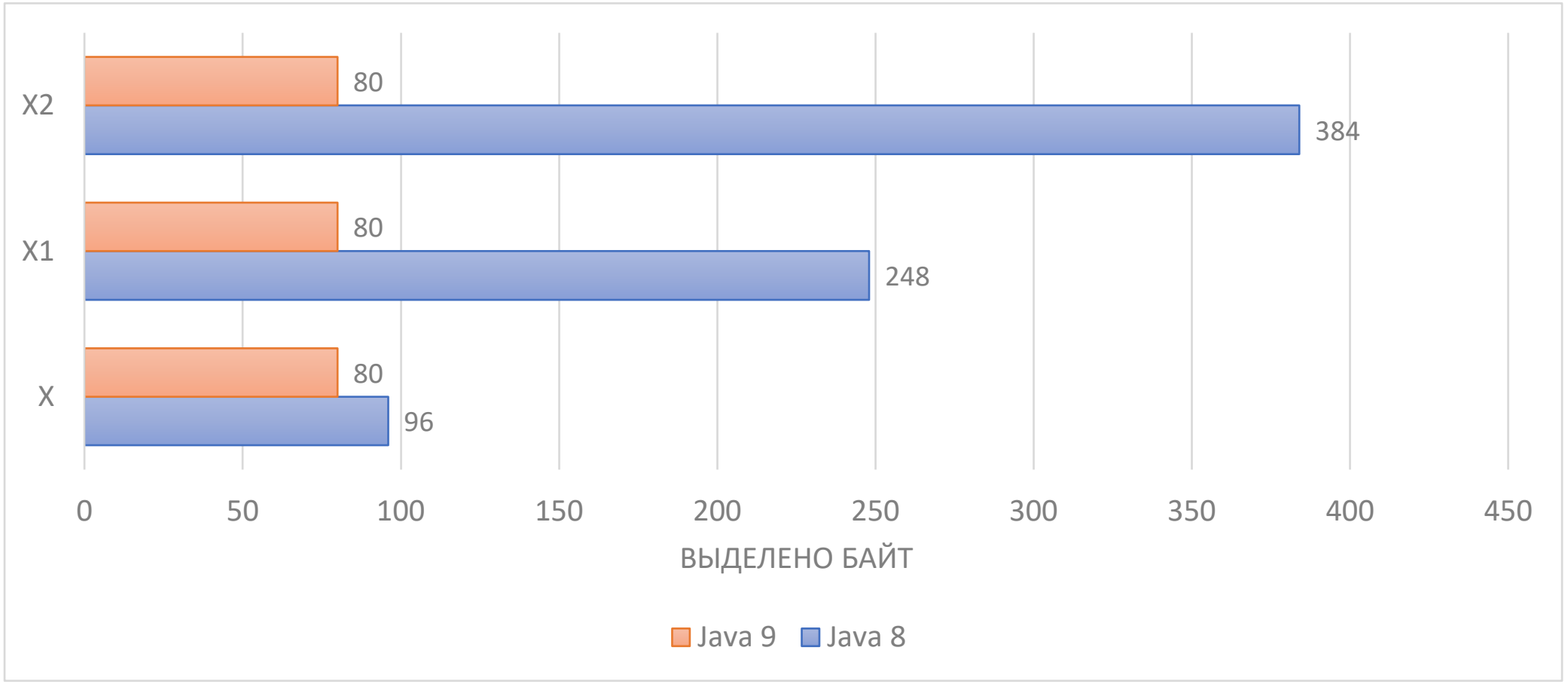
```
public static class X {
  public X() {}
}

public static class X1 {
  public X1() {}
  public X1(int p1) {}
  public X1(int p1, int p2) {}
  public X1(int p1, int p2, int p3) {}
  public X1(int p1, int p2, int p3, int p4) {}
  public X1(int p1, int p2, int p3, int p4, int p5) {}
}
```

```
public static class X {…}

public static class X1 {…}

public static class X2 {
  public X2() {}
  public X2(int p1) {}
  public X2(int p1, int p2) {}
  public X2(int p1, int p2, int p3) {}
  public X2(int p1, int p2, int p3, int p4) {}
  public X2(int p1, int p2, int p3, int p4, int p5) {}
  public X2(int p1, int p2, int p3, int p4, int p5, int p6) {}
  public X2(int p1, int p2, int p3, int p4, int p5, int p6, int p7) {}
  public X2(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8) {}
  public X2(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8,
            int p9) {}
  public X2(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8,
            int p9, int p10) {}
}
```

```java
@Benchmark
public Constructor<?> getConstructorX() throws NoSuchMethodException {
  return X.class.getConstructor();
}

@Benchmark
public Constructor<?> getConstructorX1() throws NoSuchMethodException {
  return X1.class.getConstructor();
}

@Benchmark
public Constructor<?> getConstructorX2() throws NoSuchMethodException {
  return X2.class.getConstructor();
}
```


X2
89.8
X1
77.1
X
29.9
0
10
20
30
40
50
60
70
80
90
100
ВРЕМЯ, НС
Java 8


X2
30.5
89.8
X1
29.1
77.1
X
25
29.9
0
10
20
30
40
50
60
70
80
90
100
ВРЕМЯ, НС
Java 9
Java 8


X2
80
384
X1
80
248
X
80
96
0
50
100
150
200
250
300
350
400
450
ВЫДЕЛЕНО БАЙТ
Java 9
Java 8


```
private Constructor<T> getConstructor0(Class<?>[] parameterTypes,
                                       int which) throws NoSuchMethodException
{
    Constructor<T>[] constructors = privateGetDeclaredConstructors(
                                        (which == Member.PUBLIC));
    for (Constructor<T> constructor : constructors) {
        if (arrayContentsEq(parameterTypes,
                            constructor.getParameterTypes())) {
            return getReflectionFactory().copyConstructor(constructor);
        }
    }
    throw new NoSuchMethodException(
              getName() + ".<init>" + argumentTypesToString(parameterTypes));
}
```


```
private Constructor<T> getConstructor0(Class<?>[] parameterTypes,
                                       int which) throws NoSuchMethodException
{
    Constructor<T>[] constructors = privateGetDeclaredConstructors(
                                        (which == Member.PUBLIC));
    for (Constructor<T> constructor : constructors) {
        if (arrayContentsEq(parameterTypes,
                            constructor.getParameterTypes())) {
            return getReflectionFactory().copyConstructor(constructor);
        }
    }
    throw new NoSuchMethodException(
              getName() + ".<init>" + argumentTypesToString(parameterTypes));
}
```

```
public final class Constructor<T> extends Executable {
    …

    @Override
    public Class<?>[] getParameterTypes() {
        return parameterTypes.clone();
    }

    …
}
```


```
public final class Constructor<T> extends Executable {
    …

    @Override
    public Class<?>[] getParameterTypes() {
        return parameterTypes.clone();
    }

    @Override
    Class<?>[] getSharedParameterTypes() {
        return parameterTypes;
    }

    …
}
```


```
public final class Constructor<T> extends Executable {
    …

    @Override
    public Class<?>[] getParameterTypes() {
        return parameterTypes.clone();
    }

    @Override
    Class<?>[] getSharedParameterTypes() {
        return parameterTypes;
    }

    …
}
```

java.lang.Class
java.lang.reflect.Constructor


```
private Constructor<T> getConstructor0(Class<?>[] parameterTypes,
                                       int which) throws NoSuchMethodException
{
    ReflectionFactory fact = getReflectionFactory();
    Constructor<T>[] constructors = privateGetDeclaredConstructors(
                                           (which == Member.PUBLIC));
    for (Constructor<T> constructor : constructors) {
        if (arrayContentsEq(parameterTypes,
                            fact.getExecutableSharedParameterTypes(constructor))) {
            return constructor;
        }
    }
    throw new NoSuchMethodException(methodToString("<init>", parameterTypes));
}
```

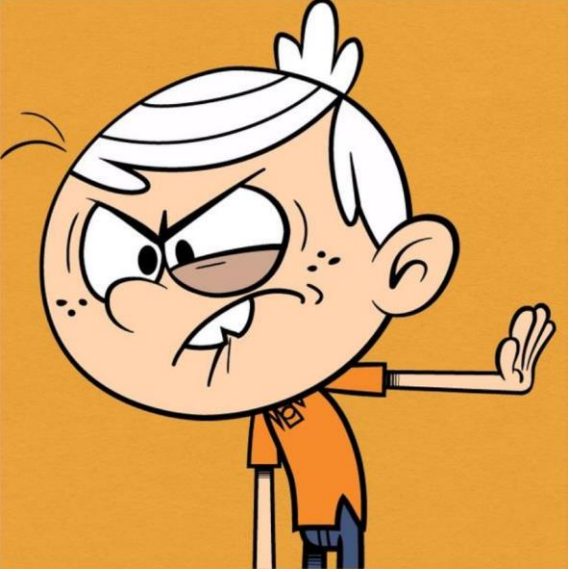

```
package jdk.internal.reflect;

public class ReflectionFactory {
    …
    private final JavaLangReflectAccess langReflectAccess;
    private ReflectionFactory() {
        this.langReflectAccess = SharedSecrets.getJavaLangReflectAccess();
    }
    …

    public Class<?>[] getExecutableSharedParameterTypes(Executable ex) {
        return langReflectAccess.getExecutableSharedParameterTypes(ex);
    }
    …
}
```

```
package jdk.internal.access;

public class SharedSecrets {
    private static JavaLangReflectAccess javaLangReflectAccess;

    public static void setJavaLangReflectAccess(JavaLangReflectAccess jlra) {
        javaLangReflectAccess = jlra;
    }

    public static JavaLangReflectAccess getJavaLangReflectAccess() {
        return javaLangReflectAccess;
    }

    …
}
```


```
package jdk.internal.access;

/** An interface which gives privileged packages Java-level access to
    internals of java.lang.reflect. */

public interface JavaLangReflectAccess {
    /** Gets the shared array of parameter types of an Executable. */
    public Class<?>[] getExecutableSharedParameterTypes(Executable ex);

    …
}
```

```
package java.lang.reflect;

public class AccessibleObject implements AnnotatedElement {
    static {
        // AccessibleObject is initialized early in initPhase1
        SharedSecrets.setJavaLangReflectAccess(new ReflectAccess());
    }
    …
}
```

```
package java.lang.reflect;

/** Package-private class implementing the
    jdk.internal.access.JavaLangReflectAccess interface, allowing the java.lang
    package to instantiate objects in this package. */

class ReflectAccess implements jdk.internal.access.JavaLangReflectAccess {
   …

   public Class<?>[] getExecutableSharedParameterTypes(Executable ex) {
       return ex.getSharedParameterTypes();
   }

   …
}
```



`getConstructor0` ← `Constructor.getParameterTypes()` ← `Constructor.parameterTypes.clone()`

```
new ReflectAccess()
```

```
SharedSecrets.
setJavaLangReflectAccess()
```

```
SharedSecrets.
javaLangReflectAccess
```

```
SharedSecrets.
getJavaLangReflectAccess()
```

```
new ReflectionFactory()
```

```
ReflectionFactory.
langReflectAccess
```

```
Constructor.
parameterTypes
```

```
getConstructor0
```

```
ReflectionFactory.
getExecutableSharedParameterTypes()
```

```
Constructor.
getSharedParameterTypes()
```

```
ReflectAccess.
getExecutableSharedParameterTypes()
```

Итого

Issue ID	Method	Fix version
JDK-8221836	String.hashCode	13
JDK-8222484	String.concat	13
JDK-8247605	str+emptyStr	16
JDK-8176894	TreeMap.compute	15
JDK-8071477	ArrayList.removeIf	9
JDK-8170733	HashSet.removeIf	9
JDK-8245677	HashMap.containsKey	15
JDK-8187123	Class.getSimpleName	11
JDK-8172190	Class.getConstructor	9

Спасибо
за внимание

https://twitter.com/tagir_valeev
https://habrahabr.ru/users/lany
https://github.com/amaembo
tagir.valeev@jetbrains.com