ktu
1922

UŽ KIEKVIENOS TECHNOLOGIJOS – ŽMOGUS

Java kodo greitaveikos matavimas

© Giedrius Paulikas 2024

Benchmark

ktu

Informatikoje tai – kompiuterinių programų ar kitų operacijų vykdymas, siekiant įvertinti **santykinį** tam tikrų objektų **efektyvumą.**

Paprastas matavimas – kodas

ktu

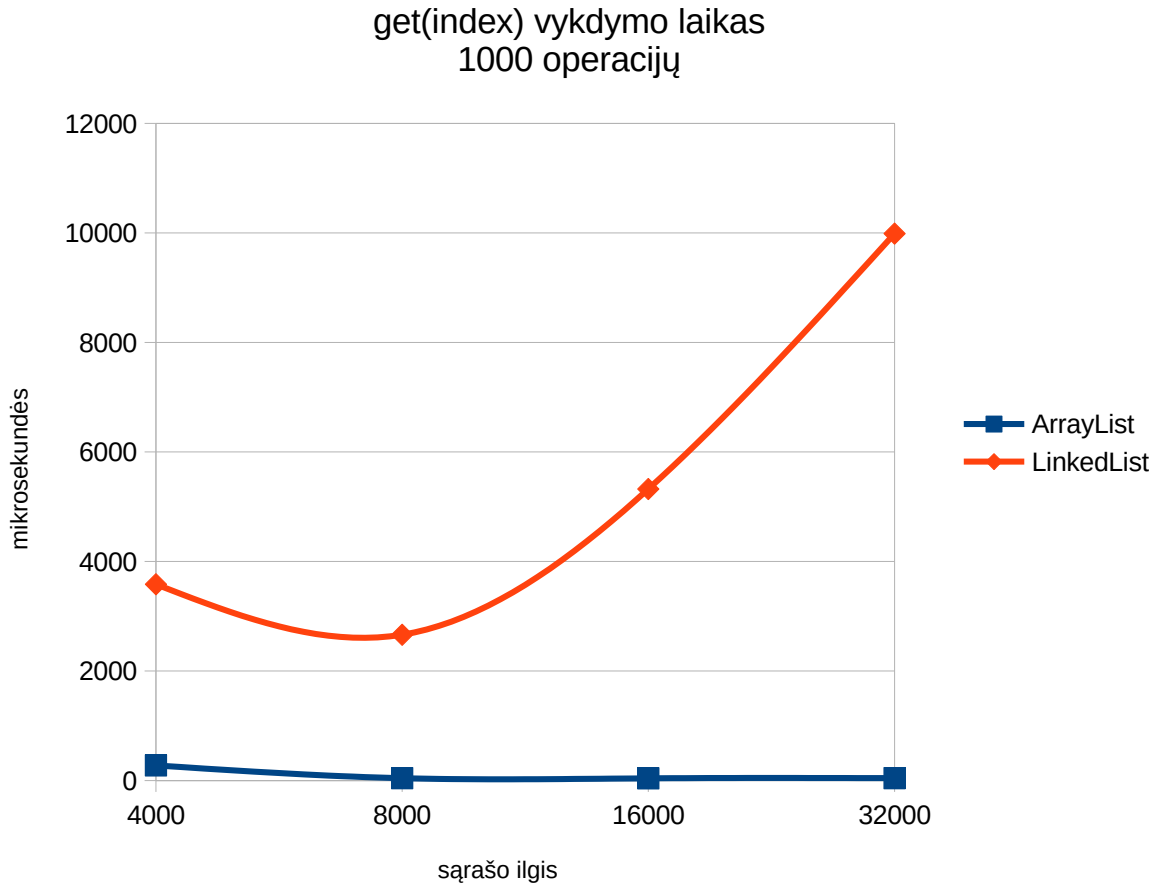
```
int[] indexes = new int[1_000];

for (int listSize : new int[]{4_000, 8_000,
16_000, 32_000}) {

    Util.generateIndexes(indexes, listSize);

    long arrayListTime = run(new ArrayList<>(),
listSize, indexes);

    long linkedListTime = run(new LinkedList<>(),
listSize, indexes);

    System.out.println(listSize + " " +
arrayListTime + " " + linkedListTime);

}
```

```
static long run(List<Float> list, int listSize,
int[] indexes) {

    Util.generateList(list, listSize);

    long start = System.nanoTime();

    for (int i : indexes) {

        list.get(i);

    };

    return System.nanoTime() - start;

}
```

Paprastas matavimas – rezultatai

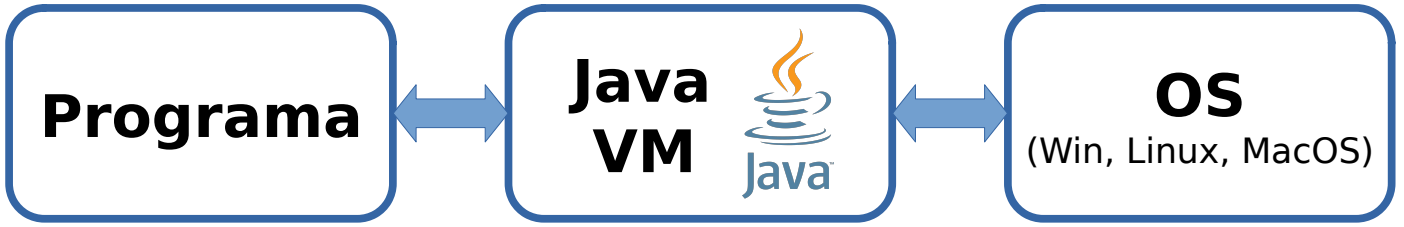
ktu

	ArrayList, µs	LinkedList, µs
4000	276.495	3582.875
8000	42.795	2661.929
16000	40.154	5321.878
32000	43.965	9986.685



Kas čia daros?..

ktu



- Javos virtuali mašina (VM) kodo vykdymo metu atlieka jo **optimizacijas**
- Tiesioginių priemonių optimizacijų valdymui programuotojas neturi

Patobulintas matavimas – kodas

ktu

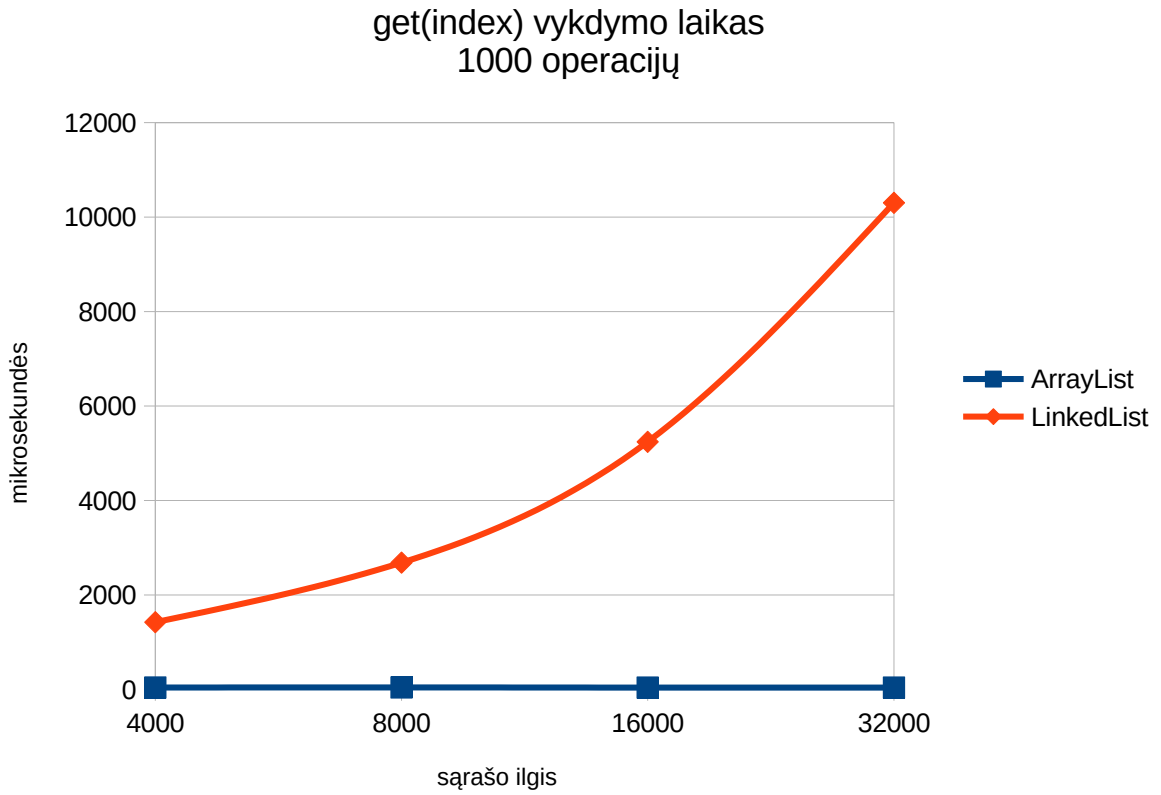
```
int[] indexes = new int[1_000];

for (int listSize : new int[]{64_000, 4_000,
8_000, 16_000, 32_000}) {

    Util.generateIndexes(indexes, listSize);

    long arrayListTime = run(new ArrayList<>(),
listSize, indexes);

    long linkedListTime = run(new LinkedList<>(),
listSize, indexes);

    System.out.println(listSize + " " +
arrayListTime + " " + linkedListTime);

}
```

```
static long run(List<Float> list, int listSize,
int[] indexes) {

    Util.generateList(list, listSize);

    long start = System.nanoTime();

    for (int i : indexes) {

        list.get(i);

    };

    return System.nanoTime() - start;

}
```

Patobulintas matavimas – rezultatai

ktu

	ArrayList, µs	LinkedList, µs
~~**64000**~~	~~329.833~~	~~23623.892~~
4000	40.493	1423.931
8000	42.838	2683.397
16000	39.306	5241.339
32000	39.228	10302.222



Java Microbenchmark Harness (JMH)

ktu

- Java kūrėjų įrankis, padidinantis *benchmark*'ų rezultatų **patikimumą**
- Pagal Javos kodo **anotacijas** sugeneruoja pagalbinį *benchmark*'ų kodą
- Prieš greitaveikos matavimus atlieka Javos VM **"apšildymą"**

JMH matavimas – Mode, State(s)

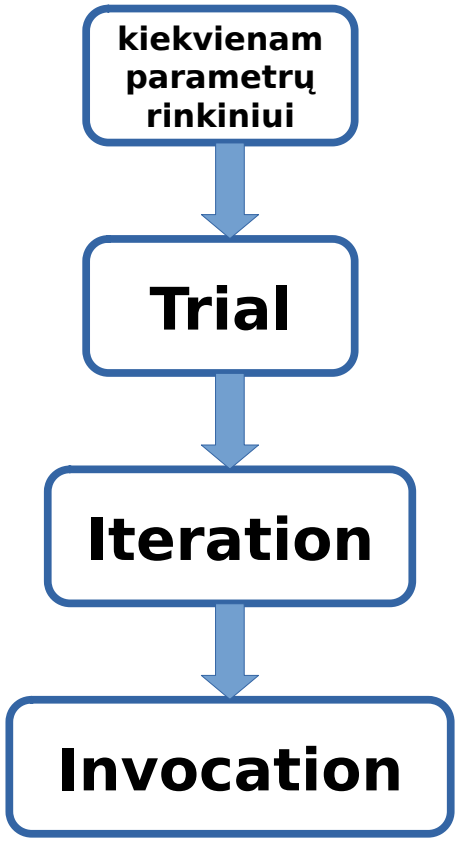
ktu

```
@BenchmarkMode(Mode.AverageTime)
@State(Scope.Benchmark)
@OutputTimeUnit(TimeUnit.MICROSECONDS)
@Warmup(
    Time = 1,
    timeUnit = TimeUnit.SECONDS
)
@Measurement(
    Time = 1,
    timeUnit = TimeUnit.SECONDS
)
public class JmhBenchmark {
}
```

JMH matavimas – Params, Fixtures

ktu

```
public class JmhBenchmark {

    @Param({"4000", "8000", "16000", "32000"})
    public int listSize;


    ArrayList<Float> arrayList = new
ArrayList<>();
    LinkedList<Float> linkedList = new
LinkedList<>();
    int[] indexes = new int[1_000];

    @Setup(Level.Trial)
    public void generateLists() {
        Util.generateList(arrayList, listSize);
        Util.generateList(linkedList, listSize);
    }

    @Setup(Level.Iteration)
    public void generateIndexes() {
        Util.generateIndexes(indexes, listSize);
    }
}
```



JMH matavimas – Benchmarks

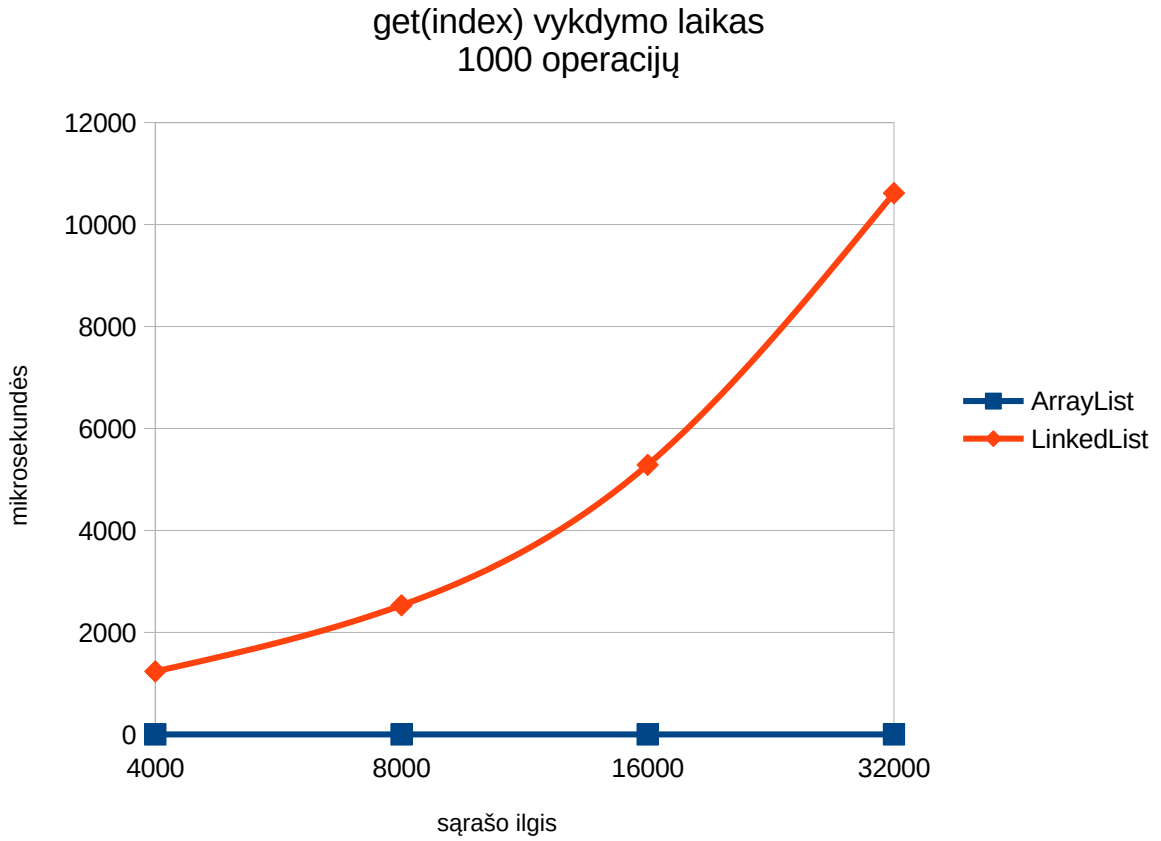
ktu

```
public class JmhBenchmark {

    …

    @Benchmark
    public void arrayListGet() {
        listGet(arrayList);
    }

    @Benchmark
    public void linkedListGet() {
        listGet(linkedList);
    }

    private void listGet(List<Float> list) {
            for (int i : indexes) {
                list.get(i);
            }
        }
    }
```

JMH matavimas – rezultatai

ktu

	ArrayList, µs	LinkedList, µs
4000	0.441	1236.023
8000	0.442	2532.494
16000	0.441	5286.967
32000	0.442	10616.272



JMH – Dead code

ktu

```
@Benchmark
public void linkedListGet() {
    listGet(linkedList);
}


private void listGet(List<Float> list) {
    for (int i : indexes) {
        list.get(i);
    }
}
```

JMH – Dead code – Returns & Blackholes

ktu

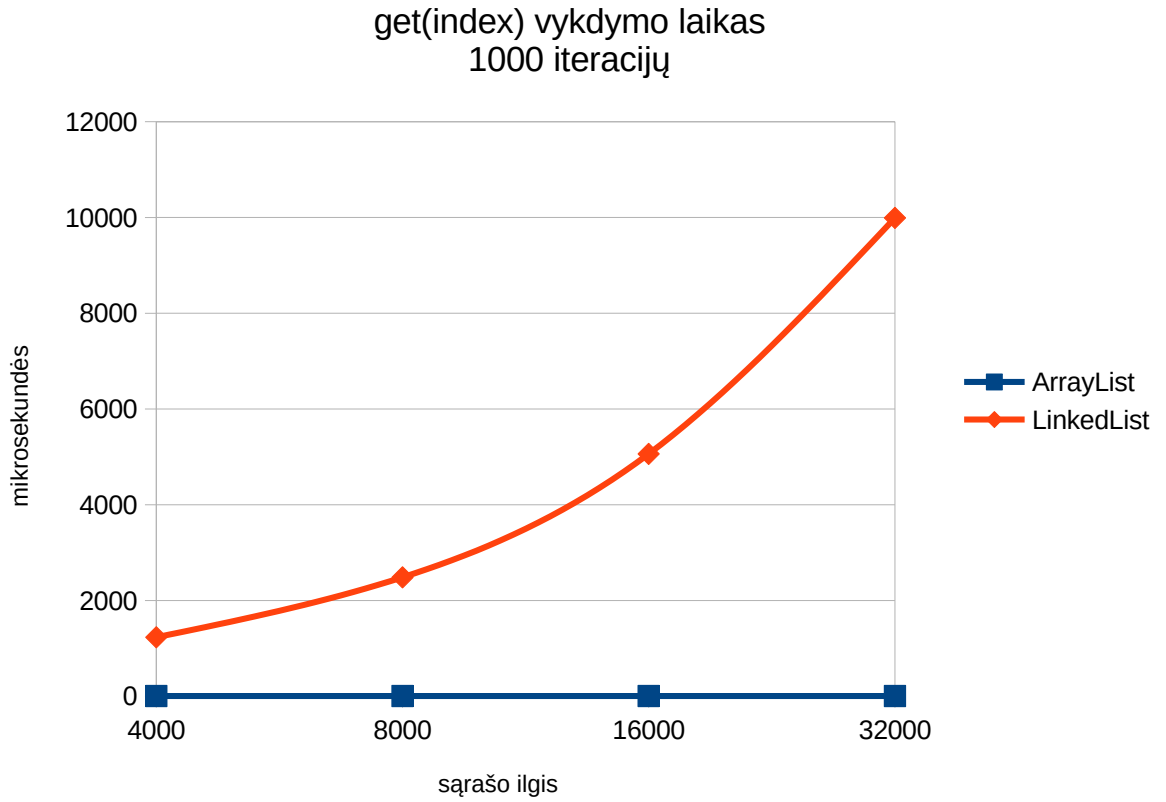
```
@Benchmark
public Float linkedListGet() {
    return listGet(linkedList);
}

private Float listGet(List<Float> list) {
    Float sum = 0f;
    for (int i : indexes) {
        sum += list.get(i);
    }
    return sum;
}
```

```
@Benchmark
public void linkedListGet(Blackhole bh) {
    listGet(linkedList, bh);
}

private void listGet(List<Float> list,
Blackhole bh) {
    for (int i : indexes) {
        bh.consume(list.get(i));
    }
}
```

JMH matavimas – geresni rezultatai

ktu

	ArrayList, µs	LinkedList, µs
4000	5.277	1232.701
8000	5.284	2484.535
16000	5.314	5061.168
32000	5.341	9990.668



O jei testas keičia būseną?

ktu

- Galima būsenos atstatymą įtraukti į patį testą
- Jei testas ilgesnis nei **1 ms**, galima naudoti *Level.Invocation*

```
ArrayList<Float> newElements = …


@Setup(Level.Invocation)
public void clearList() {
    arrayList = new ArrayList<>();
}


@Benchmark
public void arrayListAdd() {
    for (Float e : newElements)
        arrayList.add(e);
}
```

Dar apie JMH

ktu

- Rekomenduojama JMH naudoti **Maven** projekte
- JMH pagal kodo anotacijas sugeneruoja pagalbinį kodą, todėl pakeitus anotacijas, reiktų **perkompiliuoti** projektą (*Rebuild*)
- JMH – tik **pagalbinė** priemonė ir jo *benchmark*'ų patikimumu reikia rūpintis patiems

Kur rasti

ktu

- JMH
- https://openjdk.java.net/projects/code-tools/jmh/
- JHM dokumentacija/pavyzdžiai
- https://github.com/openjdk/jmh/tree/master/jmh-samples/src/main/java/org/openjdk/jmh/samples
- Šio pristatymo aprašymas ir kodas
- https://github.com/dastruktu/benchmark
- ir Duomenų struktūrų **Moodle** kurse

Klausimai?