**Programinės įrangos testavimas**

Programinės įrangos testas - kodo dalis, įvykdanti kitą kodo dalį. Testas validuoja, kad įvykdytas kodas grąžino reikiamą rezultatą ir veiksmų seka buvo tokia, kokia tikėtasi.

Testai padeda užtikrinti, kad programos logika yra teisinga. Jei nauji pakeitimai programiniame kode, kuris yra padengtas testais, sugriauna tinkamą programos veikimą, galima tikėtis, kad tai bus pastebėta testų vykdymo metu.

***Unit* testai**

Tai kodas, kuris įvykdo tam tikrą sistemos dalį ir patvirtina tam tikrą rezultatą, elgesį ar būseną. *Unit* testai testuoja mažą kodo dalelę (*unit‘ą*) – metodą arba klasę. Priklausomybės nuo kitų klasių turėtų būti pašalintos, pakeičiant išorines klases testinėmis implementacijomis arba kitaip *mock‘ais*.

Procentas, nurodantis, kiek kodo yra ištestuota naudojant *unit* testus yra vadinamas *test coverage*.

**Integraciniai testai**

Integraciniai testai tikrina ne vienos mažos kodo dalies veikimą, o tam tikrą programos veiksmų seką, vykstančią per kelias skirtingas klases. Integraciniai testai dažniausiai testuoja esminius sistemos funkcionalumus, kaip, pavyzdžiui, internetinės parduotuvės programėlėje, integracinis testas galėtų testuoti programos seką, vykdomą perkant tam tikrą prekę.

***Unit* testai Java programavimo kalboje**

Yra keli populiarūs testavimo karkasai kaip – *JUnit* ir *TestNG*. Ši medžiaga demonstruos testavimą naudojant *JUnit* 5 biblioteką. Paprastai *unit* testai yra kuriami atskiroje direktorijoje, kad testavimo kodas būtų atskirtas nuo pagrindinio kodo. Standartiškai, dirbant su *Maven* arba *Gradle* įrankiais, naudojamos dvi direktorijos klasėms saugoti:

* src/main/java - standartinėms sistemos klasėms
* src/test/java - testų klasėms

Parašyti testai gali būti paleidžiami per naudojamą IDE arba, tinkamai sukonfigūravus *Maven*/*Gradle* įrankius, testai bus automatiškai įvykdomi šių įrankių konstravimo (*Build*) *test* fazėje. Pavyzdžiui vykdant komandą mvn test.

***Unit* testai su JUnit**

*JUnit* testai yra metodai, aprašyti testavimui skirtoje klasėje. Norint nurodyti, kad tam tikras metodas yra testas, reikia jam pridėti anotaciją @Test. Toks metodas įvykdo tam tikras kodo dalis ir patikrina gautus rezultatus. *JUnit* biblioteka pateikia metodus, leidžiančius palyginti gautus rezultatus su tikėtaisiais. Tokie metodai vadinami *asserts* arba *assert statements*.

***Unit* testo pavyzdys**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class ApplicationTests {

@Test

void testPower() {

assertEquals(4, Math.pow(2, 2), "2 power of 2 should be 4");

assertEquals(16, Math.pow(4, 2), "4 power of 2 should be 16");

assertEquals(1, Math.pow(2, 0), "2 power of 0 should be 1");

}

}

***JUnit* 5 funkcionalumai**

Kaip buvo minėta anksčiau, biblioteka pateikia rinkinį *assert* metodų, padedančių įsitikinti, ar gauti rezultatai yra tokie, kaip tikėtasi. *JUnit* biblioteka taip pat palaiko rinkinį anotacijų, leidžiančių praplėsti *unit* testus.

**Pagrindinės anotacijos:**

* @Test - nurodo, kad metodas yra testas
* @BeforeEach, @AfterEach - nurodo, kad metodas bus įvykdomas prieš/po kiekvieno testo
* @BeforeAll, @AfterAll - nurodo, kad metodas bus įvykdomas prieš vykdant/įvykdžius visus klasės testus
* @Disabled - nurodo, kad testas yra išjungtas ir metodas neturėtų būti vykdomas

**Pagrindiniai *assert* metodai:**

* assertEquals(expectedResult, actualResult); - palygina gautą rezultatą su rezultatu, kurio buvo tikimasi. Jei jie nesutampa, testas nėra sėkmingas
* assertNotEquals(unexpectedResult, actualResult); - palygina gautą rezultatą su rezultatu, kurio nebuvo tikimasi. Jei jie sutampa, testas nėra sėkmingas
* assertFalse(condition); - patikrina, ar gauta sąlyga yra neteisinga. Jei ji teisinga, testas nėra sėkmingas
* assertTrue(condition); - patikrina ar gauta sąlyga yra teisinga. Jei ji neteisinga, testas nėra sėkmingas
* assertThrows(Exception.class, executor); – patikrina ar kodas grąžina *exception*, jei ne – testas nėra sėkmingas

**Priklausomybių paslėpimas (*mocking*)**

*Unit* testai vykdo mažas kodo dalis, dažniausiai metodus, vieną arba kelias susijusias klases. Paprastai, jei testuojamoje klasėje yra naudojamos kitos klasės papildomiems veiksmams atlikti, pavyzdžiui testuojama klasė PersonService, kuri naudoja PersonRepository klasę, reikalingą gauti žmonių sąrašui, tos klasės yra „pakeičiamos“ *mock* objektais. *Mock* objektas simuliuoja tam tikrą klasę, tačiau nevykdo jokios logikos, o tiesiog grąžina iš anksto aprašytus rezultatus. Tai leidžia testuoti vieną, konkrečią klasę, išvengiant klaidų testavimo kitose, jos naudojamose klasėse.

**Užduotys**

**Boulingo žaidimo testavimas**

1. Atsisiųskite sukurtą projektą iš [repozitorijos](https://github.com/tadus21/code-academy-java-kursai/blob/master/unit-testing/exercises/unit-testing)
2. Jums reikia baigti įgyvendinti BowlingGame klasę, taip, kad jos pagalba būtų galima imituoti boulingo žaidimą
3. Klasė turėtų būti įgyventina atsižvelgiant į tikras [boulingo taisykles](http://neokaunas.lt/boulingo-taisykles/)
4. Parašykite *unit* testus, įrodančius, kad klasės įgyvendinimas atitinka boulingo taisykles

**Asmens kodo generatoriaus testavimas**

1. Atsisiųskite jau sukurtą projektą iš [repozitorijos](https://github.com/tadus21/code-academy-java-kursai/blob/master/unit-testing/exercises/officialidgenerator)
2. Sukurkite klasę LithuanianOfficialIdProvider, kuri įgyvendintų OfficialIdProvider sąsąją
3. Jūsų sukurta klasė turėtų įgyvendinti metodą, kuris priimtų Person tipo objektą ir grąžintų asmens kodą, atitinkantį Lietuvos valstybės formatą
4. Parašykite *unit* testus, įrodančius, kad visais įmanomais atvejais asmens kodai generuojami teisingai

**Šaltiniai:**

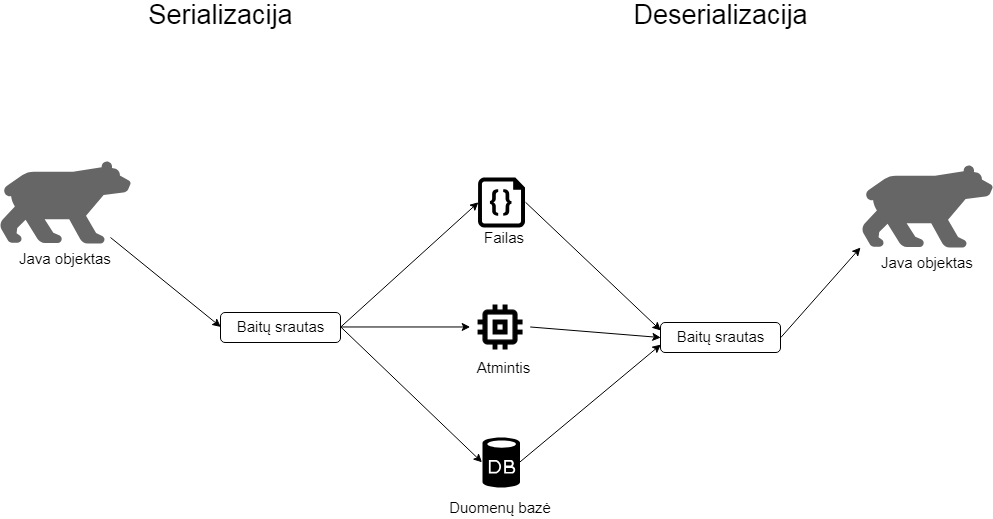
* <http://www.vogella.com/tutorials/JUnit/article.html>
* <https://junit.org/junit5/docs/current/user-guide/>

# Serializacija ir JSON

## Turinys

* [Serializacija ir deserializacija](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#Serializacija-ir-deserializacija)
* [JSON](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#JSON)
  + [Java objekto serializavimas į JSON](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#Java-objekto-serializavimas-%C4%AF-JSON)
  + [JSON deserializavimas į Java objektą](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#JSON-deserializavimas-%C4%AF-Java-objekt%C4%85)
  + [Java objekto konvertavimas į JSON eilutę ir atvirščiai](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#Java-objekto-konvertavimas-%C4%AF-JSON-eilut%C4%99-ir-atvir%C5%A1%C4%8Diai)
  + [Sąrašo konvertavimas į JSON ir atvirščiai](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#S%C4%85ra%C5%A1o-konvertavimas-%C4%AF-JSON-ir-atvir%C5%A1%C4%8Diai)
  + [JSON konvertavimas į Java MAP](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#JSON-konvertavimas-%C4%AF-Java-MAP)
  + [JSON konvertavimas į Java objektą su nežinomais laukais](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#JSON-konvertavimas-%C4%AF-Java-objekt%C4%85-su-ne%C5%BEinomais-laukais)
  + [Individualizuotas serializatorius](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#Individualizuotas-serializatorius)
  + [Individualizuotas deserializatorius](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#Individualizuotas-deserializatorius)
  + [JsonProperty anotacijos naudojimas](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#JsonProperty-anotacijos-naudojimas)
  + [JSON alternatyvos](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#JSON-alternatyvos)
* [Tolesniam skaitymui](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#Tolesniam-skaitymui)
* [Užduotys](https://github.com/tadus21/code-academy-java-kursai/tree/master/serialization-json#U%C5%BEduotys)

## Serializacija ir deserializacija

[](https://github.com/tadus21/code-academy-java-kursai/blob/master/serialization-json/img/ser-deser.png)

Klasės gali būti serializuojamos ir deserializuojamos. Gali būti serializuojamos vienoje sistemoje, o deserializuojamos kitoje. Klasės, kurias norime serializuoti, turi įgyvendinti Serializable interfeisą.

Java klasė ObjectOutputStream gali primityviuosius tipus ir objektus gali paversti į baitų srautą, o baitų srautą skaito ir gali pavesti į Objektus ir primityviuosius klasė ObjectInputStream.

Svarbiausias ObjectOutputStream metodas yra:

public final void writeObject(Object o) throws IOException;

kuris paima serializuojamą objektą ir paverčia jį į baitų srautą.

ObjectInputStream atveju metodas, kuris skaito baitų srautą ir paverčia jį atgal į Java objektą yra:

public final Object readObject() throws IOException, ClassNotFoundException;

Pavyzdys. Turime klasę Asmuo su keliais klasės kintamaisiais:

import java.io.Serializable;

public class Asmuo implements Serializable {

private String vardas;

private int amzius;

// getteriai ir setteriai

}

Tada sukuriame Asmuo objektą ir jį serializuojame į failą, vėliau iš failo deserializuojame atgal į Java objektą ir atspausdiname objekto reikšmes:

Asmuo asmuo = new Asmuo();

asmuo.setVardas("Jurgis");

asmuo.setAmzius(50);

FileOutputStream fileOutputStream = new FileOutputStream("asmens\_failas.txt");

ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);

objectOutputStream.writeObject(asmuo);

objectOutputStream.flush();

objectOutputStream.close();

FileInputStream fileInputStream = new FileInputStream("asmens\_failas.txt");

ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

Asmuo asmuo2 = (Asmuo) objectInputStream.readObject();

objectInputStream.close();

System.out.println(asmuo2.getVardas());

System.out.println(asmuo2.getAmzius());

Jurgis

50

Java objektas serializuojamas į baitų srautą:

objectOutputStream.writeObject(asmuo);

Iš baitų srauto deserializuojama į Java objektą:

Asmuo asmuo2 = (Asmuo) objectInputStream.readObject();

Operacija cast šioje vieotje yra būtina.

Naudojant transient operatorių galime nurodyti kurios objekto dalies nenorime serializuoti. Pavyzdžiui nenorime serializuoti slaptažodžių dėl saugumo.

public class Asmuo implements Serializable {

private transient String vardas;

private int amzius;

// getteriai ir setteriai

}

Tada pagal anstesnį serializavimo, deserialiazavimo ir objekto reikšmių atspausdinimo pavyzdį gautume rezultatą:

null

50

Jei klasė įgyvendina Serializable interfeisą, tai visų tos klasės vaikų (paveldinčių klasių) objektai taip pat galės būti serializuojami. Tačiau kompozicijos atveju kiekviena klasė turi įgyvendinti Serializable interfeisą. Pavyzdys:

public class Asmuo implements Serializable {

transient private String vardas;

private int amzius;

private Adresas adresas;

// getteriai ir setteriai

}

Tada klasė Adresas taip pat turi įgyvendinti Serializable interfeisą.

Serializacijoje yra naudojamas serialVersionUID. Pavyzdys:

public class Asmuo implements Serializable {

private static final long serialVersionUID = 1L;

transient private String vardas;

private int amzius;

// getteriai ir setteriai

}

serialVersionUID yra skaičius, kuris užtikrina, kad objektas, kurį deserilizuojame, atitiks klasės sandarą, kuri buvo serializuojant objektą. serialVersionUID yra didelis skaičius, paskaičiuojamas pagal klasės pavadinimą, klasės atributus ir klasėje naudojamus prėjimo modifikatorius. Jei šis skiačius nėra parašytas klasėje, tada JVM tai padaro už mus. Jei serializuosime objektą, o vėliau pakeisime klasę, pvz. parašysime naują metodą, tada deserializuojant gausime InvalidClassException klaidą, nes serialVersionUID, kuriuos generuoja pati JVM, nesutaps. To išvengti padėtų, jei mes patys priskitume serialVersionUIDreikšmę.

Jei bandysime serializuoti objektą, kurio klasė neįgyvendina Serializable interfeiso, tada gausime NotSerializableExceptionklaidą.

Serializavimo/deserializamino procesą galime šiek tiek modifikuoti. Pavyzdžiui turime klasę Adresas:

public class Adresas {

private String miestas;

private String gatve;

// getteriai ir setteriai

}

ir turime klasę Asmuo, kurioje adresas yra pažymėtas kaip transient, t.y. neserializuojamas:

public class Asmuo implements Serializable {

private String vardas;

private int amzius;

private transient Adresas adresas;

// getteriai ir setteriai

}

bet norime serializuoti adresą. Tada klasėje Asmuo turime sukurti writeObject ir readObject metodus:

public class Asmuo implements Serializable {

private String vardas;

private int amzius;

private transient Adresas adresas;

private void writeObject(ObjectOutputStream oos) throws IOException {

oos.defaultWriteObject();

oos.writeObject(adresas.getMiestas());

oos.writeObject(adresas.getGatve());

}

private void readObject(ObjectInputStream ois) throws ClassNotFoundException, IOException {

ois.defaultReadObject();

String miestas = (String) ois.readObject();

String gatve = (String) ois.readObject();

Adresas adresas = new Adresas();

adresas.setMiestas(miestas);

adresas.setGatve(gatve);

this.setAdresas(adresas);

}

// getteriai ir setteriai

}

Tokiu atveju adresas bus serializuotas ir galėsime deserializuojant gauti miestą bei gatvę.

Asmuo asmuo = new Asmuo();

asmuo.setVardas("Jurgis");

asmuo.setAmzius(50);

Adresas adresas = new Adresas();

adresas.setMiestas("Vilnius");

adresas.setGatve("Sauletekio");

asmuo.setAdresas(adresas);

FileOutputStream fileOutputStream = new FileOutputStream("asmens\_failas.txt");

ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);

objectOutputStream.writeObject(asmuo);

objectOutputStream.flush();

objectOutputStream.close();

FileInputStream fileInputStream = new FileInputStream("asmens\_failas.txt");

ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

Asmuo asmuo2 = (Asmuo) objectInputStream.readObject();

objectInputStream.close();

System.out.println(asmuo2.getVardas());

System.out.println(asmuo2.getAmzius());

System.out.println(asmuo2.getAdresas().getMiestas());

System.out.println(asmuo2.getAdresas().getGatve());

Rezultatas:

Jurgis

50

Vilnius

Sauletekio

## JSON

Naudosime Jackson:

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.4</version>

</dependency>

### Java objekto serializavimas į JSON

Java objektą serializuosime į JSON failą.

Turime enum Salis:

public enum Salis {

LIETUVA,

LATVIJA,

ESTIJA

}

Turime klasę Adresas:

public class Adresas {

private Salis salis;

private String miestas;

private String gatve;

private String namoNr;

public Adresas() {

// default konstruktorius bus reikalingas norint is JSON pagaminti Java objektą

}

public Adresas(Salis salis, String miestas, String gatve, String namoNr) {

this.salis = salis;

this.miestas = miestas;

this.gatve = gatve;

this.namoNr = namoNr;

}

// getteriai ir setteriai

}

Atskiroje kalsėje main metode sukuriame adreso objektą:

Adresas adresas = new Adresas(Salis.LIETUVA, "Vilnius", "Sauletekio", "15");

Taip pat sukuriame Jackson Object mapper'į:

ObjectMapper mapper = new ObjectMapper();

Norėdami serializuoti adreso objektą į json failą sukuriame failo objektą ir jį kartu su adreso objektu paduodame Object mapper'io metodui writeValue(...):

File file = new File("target/adresas.json");

mapper.writeValue(file, adresas);

Tada kataloge target bus sukurtas adresas.json failas:

{

"salis": "LIETUVA",

"miestas": "Vilnius",

"gatve": "Sauletekio",

"namoNr": "15"

}

### JSON deserializavimas į Java objektą

JSON failą deserializuosime į Java objektą.

Turime failą, kuriame yra JSON formatu saugomi duomenys apie adresą. Šį failą ir adreso klasę paduosime Object mapper'io metodui readValue(...):

ObjectMapper mapper = new ObjectMapper();

File file = new File("target/adresas.json");

Adresas adresasIsJsonFailo = mapper.readValue(file, Adresas.class);

System.out.println(adresasIsJsonFailo.getSalis());

System.out.println(adresasIsJsonFailo.getMiestas());

System.out.println(adresasIsJsonFailo.getGatve());

System.out.println(adresasIsJsonFailo.getNamoNr());

Rezultatas bus:

LIETUVA

Vilnius

Sauletekio

15

### Java objekto konvertavimas į JSON eilutę ir atvirščiai

Turime adreso objektą:

Adresas adresas = new Adresas(Salis.LIETUVA, "Vilnius", "Sauletekio", "15");

Norėdami konvertuoti jį į String eilutę kviečiame Object mapper'io metodą writeValueAsString(...) paduodami jam adreso objektą:

ObjectMapper mapper = new ObjectMapper();

String jsonInString = mapper.writeValueAsString(adresas);

System.out.println(jsonInString);

Rezultatas bus:

{"salis":"LIETUVA","miestas":"Vilnius","gatve":"Sauletekio","namoNr":"15"}

Turime JSON eilutę:

String jsonEilute = "{\"salis\":\"LIETUVA\",\"miestas\":\"Vilnius\",\"gatve\":\"Sauletekio\",\"namoNr\":\"15\"}";

Norėdami konveruoti JSON eilutę į Java objektą Objeck mapper'io metodui readValue(...) paduodame JSON eilutą ir objekto, į kurį konvertuosime, klasę:

Adresas adresasIsJsonEilutes = mapper.readValue(jsonEilute, Adresas.class);

System.out.println(adresasIsJsonEilutes.getSalis());

System.out.println(adresasIsJsonEilutes.getMiestas());

System.out.println(adresasIsJsonEilutes.getGatve());

System.out.println(adresasIsJsonEilutes.getNamoNr());

Rezultatas bus:

LIETUVA

Vilnius

Sauletekio

15

### Sąrašo konvertavimas į JSON ir atvirščiai

Turime sąrašą su adresais ir norėdamį jį konvertuoti į JSON dirbame taip pat kaip ir su bet kokiu kituJava objektu:

List<Adresas> adresai = new ArrayList<>();

Adresas adresas1 = new Adresas(Salis.LIETUVA, "Vilnius", "Sauletekio", "15");

Adresas adresas2 = new Adresas(Salis.LIETUVA, "Vilnius", "Antakalnio", "17");

adresai.add(adresas1);

adresai.add(adresas2);

ObjectMapper mapper = new ObjectMapper();

String jsonAdresai = mapper.writeValueAsString(adresai);

System.out.println(jsonAdresai);

Rezultatas bus:

[

{

"salis": "LIETUVA",

"miestas": "Vilnius",

"gatve": "Sauletekio",

"namoNr": "15"

},

{

"salis": "LIETUVA",

"miestas": "Vilnius",

"gatve": "Antakalnio",

"namoNr": "17"

}

]

Turime JSON eilutę su sąrašu adresų. Norėdami tokią eilutę konvertuoti į Java objektų sąrašą naudojame Object mapper'io metodą readvalue(...) paduodant jam JSON eilutę ir List klasę:

String jsonAdresai = "[{\"salis\":\"LIETUVA\",\"miestas\":\"Vilnius\",\"gatve\":\"Sauletekio\",\"namoNr\":\"15\"},{\"salis\":\"LIETUVA\",\"miestas\":\"Vilnius\",\"gatve\":\"Antakalnio\",\"namoNr\":\"17\"}]";

ObjectMapper mapper = new ObjectMapper();

List<Adresas> adresuSarasas = mapper.readValue(jsonAdresai, new TypeReference<List<Adresas>>(){});

### JSON konvertavimas į Java MAP

Turime JSON eilutę ir norime ją konvertuoti į Java Map'ą:

String jsonEilute = "{\"salis\":\"LIETUVA\",\"miestas\":\"Vilnius\",\"gatve\":\"Sauletekio\",\"namoNr\":\"15\"}";

ObjectMapper mapper = new ObjectMapper();

Map<String, Object> map = mapper.readValue(jsonEilute, new TypeReference<Map<String,Object>>(){});

### JSON konvertavimas į Java objektą su nežinomais laukais

Sakykime JSON eilutėje yra laukas pastoKodas su reikšme 00000. Tokio lauko klasėje Adresas nėra. Konvertuojant tokį JSON į adreso objketą:

String jsonEilute = "{\"salis\":\"LIETUVA\",\"miestas\":\"Vilnius\",\"gatve\":\"Sauletekio\",\"namoNr\":\"15\",\"pastoKodas\": \"00000\"}";

ObjectMapper mapper = new ObjectMapper();

Adresas adresas = mapper.readValue(jsonEilute, Adresas.class);

Gausime klaidą:

Exception in thread "main" com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field "pastoKodas" (class lt.codeacademy.Adresas), not marked as ignorable (4 known properties: "salis", "gatve", "miestas", "namoNr"])

Tokiu atveju Object mapper'iui galime nurodyti ignoruoti nežinomus laukus nustatant konfigūracijos parametrą FAIL\_ON\_UNKNOWN\_PROPERTIES į false:

String jsonEilute = "{\"salis\":\"LIETUVA\",\"miestas\":\"Vilnius\",\"gatve\":\"Sauletekio\",\"namoNr\":\"15\",\"pastoKodas\": \"00000\"}";

ObjectMapper mapper = new ObjectMapper();

mapper.configure(DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES, false);

Adresas adresas = mapper.readValue(jsonEilute, Adresas.class);

### Individualizuotas serializatorius

Galime sukurti savo individualizuotą serializatorių (custom serializer).

Sakykime norime, kad raktai JSONé būtų angliski. Turime sukurti klasę AdresoSerializeris , kuri paveldi Jackson klasę StdSerializer. Sukurtoje klasėje sukuriame du konstruktorius - vienas be parametrų, kitas Class tipo parametru. kadangi klasėje StdSerializer yra anstraktus metodas serialize(...) todėl turime jį įgyvendinti paveldinčioje klasėje AdresoSerializeris.

public class AdresoSerializeris extends StdSerializer<Adresas> {

public AdresoSerializeris() {

this(null);

}

public AdresoSerializeris(Class<Adresas> t) {

super(t);

}

@Override

public void serialize(Adresas adresas, JsonGenerator jsonGenerator, SerializerProvider serializerProvider) throws IOException {

jsonGenerator.writeStartObject();

jsonGenerator.writeStringField("country", String.valueOf(adresas.getSalis()));

jsonGenerator.writeStringField("city", adresas.getMiestas());

jsonGenerator.writeStringField("street", adresas.getGatve());

jsonGenerator.writeStringField("flatNo", adresas.getNamoNr());

jsonGenerator.writeEndObject();

}

}

Tada tokio serializatoriaus panadojimas:

ObjectMapper mapper = new ObjectMapper();

SimpleModule module = new SimpleModule("AdresoSerializeris");

module.addSerializer(Adresas.class, new AdresoSerializeris());

mapper.registerModule(module);

Adresas adresas = new Adresas(Salis.LIETUVA, "Vilnius", "Sauletekio", "15");

String adresasJson = mapper.writeValueAsString(adresas);

System.out.println(adresasJson);

Rezultatas:

{

"country": "LIETUVA",

"city": "Vilnius",

"street": "Sauletekio",

"flatNo": "15"

}

### Individualizuotas deserializatorius

Viską darome panašiaip kaip serializatoriuje, tik kuriant deserializatrių klasė turi paveldėti StdDeserializer ir įgyvendinti jos abstraktų metodą deserialize(...).

public class AdresoDeserializeris extends StdDeserializer<Adresas> {

protected AdresoDeserializeris(Class<?> vc) {

super(vc);

}

protected AdresoDeserializeris(JavaType valueType) {

super(valueType);

}

protected AdresoDeserializeris(StdDeserializer<?> src) {

super(src);

}

@Override

public Adresas deserialize(JsonParser jsonParser, DeserializationContext deserializationContext) throws IOException, JsonProcessingException {

Adresas adresas = new Adresas();

ObjectCodec codec = jsonParser.getCodec();

JsonNode node = codec.readTree(jsonParser);

JsonNode salisNode = node.get("country");

String salisString = salisNode.asText();

Salis salis = Salis.valueOf(salisString);

adresas.setSalis(salis);

JsonNode miestasNode = node.get("city");

String miestas = miestasNode.asText();

adresas.setMiestas(miestas);

JsonNode gatveNode = node.get("street");

String gatve = gatveNode.asText();

adresas.setGatve(gatve);

JsonNode numerisNode = node.get("flatNo");

String numeris = numerisNode.asText();

adresas.setNamoNr(numeris);

return adresas;

}

}

Tokio deserializatoriaus panaudojimas:

String adresasJson = "{\"country\":\"LIETUVA\",\"city\":\"Vilnius\",\"street\":\"Sauletekio\",\"flatNo\":\"15\"}";

ObjectMapper mapper = new ObjectMapper();

SimpleModule module = new SimpleModule("AdresoDeserializeris");

module.addDeserializer(Adresas.class, new AdresoDeserializeris(Adresas.class));

mapper.registerModule(module);

Adresas adresas = mapper.readValue(adresasJson, Adresas.class);

### JsonProperty anotacijos naudojimas

Norėdami pakeisti lauko, kuris bus konvertuotas į JSON nebūtinai turime rašyti savo serializatorių ir deserializatorių. Tam galime naudoti anotaciją @JsonProperty. Pavyzdys:

Norime, kad klasės Adresas laukas namoNr JSON'e turėtų kitą pavadinimą - gyvenamosiosVietosNumeris.

public class Adresas {

private Salis salis;

private String miestas;

private String gatve;

@JsonProperty("gyvenamosiosVietosNumeris")

private String namoNr;

// konstruktoriai, getteriai ir setteriai

}

Adresas adresas = new Adresas(Salis.LIETUVA, "Vilnius", "Sauletekio", "15");

String adresasJson = mapper.writeValueAsString(adresas);

System.out.println(adresasJson);

Rezultatas:

{

"salis": "LIETUVA",

"miestas": "Vilnius",

"gatve": "Sauletekio",

"gyvenamosiosVietosNumeris": "15"

}

### JSON alternatyvos

Trumas JSON struktūros palyginimas su alternatyvomis.

{

"salis": {

"pavadinimas": "Lietuva",

"kodas": "LT"

},

"miestas": "Vilnius",

"gatve": "Sauletekio",

"namoNr": "15"

}

#### XML

<adresas>

<salis>

<pavadinimas>Lietuva</pavadinimas>

<kodas>LT</kodas>

</salis>

<miestas>Vilnius</miestas>

<gatve>Sauletekio</gatve>

<namoNr>15</namoNr>

</adresas>

#### YAML

adresas:

salis:

pavadinimas: Lietuva

kodas: LT

miestas: Vilnius

gatve: Sauletekio

namoNr: 15

## Tolesniam skaitymui

* Serializacija: <https://dzone.com/articles/serialization-amp-de-serialization-in-java>
* Joshua Bloch: Effective Java (85-90 skyriai)
* Jackson anotacijos: <https://www.baeldung.com/jackson-annotations>
* Jackson: <https://www.baeldung.com/jackson>

## Užduotys

## Užduotys: Serializacija ir JSON

## Nr. 1

### Užduotis

1. Sukurkite klases Sender ir Receiver, kurios paveldi klasę Asmuo.
2. Sukurkite klasę Payment kuri turi klasės atributus sender ir receiver.
3. Sukurkite ir užpildykite Payment objektą duomenimis.
4. Serializuokite sukurtą objektą į failą.
5. Deserializuokite iš failo į Payment objektą ir patikrinkite ar nepraradote duomenų.

## Nr. 2

### Užduotis

1. Naudoti ankstesnės užduoties sąlygas, tik serializuoti į JSON failą.
2. Deserializuoti iš JSON failo atgal į Payment Java objektą.
3. Konvertuokite Payment objektą į JSON formato String eilutę.

## Nr. 3

### Užduotis

1. Sukurkite klasę Imone, kuri tokius duomenis kaip įmonės pavadinimas, įmonės kodas, darbuotojų skaičius, vidutinis atlyginimas ir panašiai.
2. Sukurkite sąrašą, į kurį galėtume įdėti klasės Imone tipo objektus.
3. Sukurkite 100 įmonės objektų. Duomenis reikalingus įmonei generuokite. Galite naudoti [java-faker](https://github.com/DiUS/java-faker).
4. Sąrašą serializuokite į JSON failą pavadinimu imoniu\_sarasas.json

## Nr. 4

### Užduotis

1. Sukurkite Maven projektą internetas
2. Maven projekte internetas sukurkite modulius narsykle, serveris ir helper
3. internetas turi turėti jackson priklausomybę (dependency)
4. Modulyje helper sukurkite klasę Uzklausa, kuri turės vieną lauką imonesKodas. Taip pat sukurkite klasę Atsakymas, kuri turės laukus - imonesKodas, imonesPavadinimas ir vidutinisAtlyginimas
5. Modulyje helper sukurtos klasės turi būti prieinamos moduliuose narsykle ir serveris
6. Sukurkite folderį pvz. and Desktop su pavadinimu internetas
7. Modulyje narsykle sukurkite klasę Narsykle su main metodu.
8. Modulyje serveris sukurkite klasę Serveris su main metodu.
9. Klasės Narsykle veiksmu seka:
   * Papraso vartotojo įvesti imones kodą
   * Vartotojo įvestą kodą panaudoja pagaminti Uzklausa objektui
   * Objektą Uzklausa serializuoja į JSON failą ir išsaugo į internetas folderį su pavadinimu uzklausa.json
   * Laukia failo tame pačiame folderyje su pavadinimu atsakymas.json Patikrinimas ar failas su tokiu vardu egzistuoja:
   * File file = new File("C:/Users/USERNAME/Desktop/internetas/atsakymas.json");

boolean failasEgzistuoja = file.exists();

Jei patikrinus failo dar nėra, palaukti sekundę ir tikrinti vėl Thread.sleep(1000);

* + Sulaukus failo, jis yra deserializuojamas iš JSON į objektą Atsakymas
  + Failas pavadinimu atsakymas.json ištrinamas iš folderio internetas. file.delete()
  + Atsakymo informacija atspausdinama į ekraną ir toliau leidžiama vartotojui įvesti kitą įmonės kodą

1. Kasės Serveris veiksmų seka:
   * Serveris turi sąrašą įmonių objektų. Startuojant serveriui panaudoti ankstesnėje užduotyje sukurtą imoniu\_sarasas.json failą jį deserializuojant iš JSON į imonių sąrašą.
   * Serveris taip pat kaip ir naršyklė - naudoja folderį internetas ir laukia ten failo pavadinimu uzklausa.json
   * Sulaukęs failo, serveris failą uzklausa.json deserializuoja į objektą Uzklausa ir ištrina failą uzklausa.json
   * Gautą užklausą spausdina į log'us. Naudoti org.apache.logging.log4j.
   * Pasinaudodamas informacija iš Uzklausa objekto, atlieka paiešką turimame įmonių sąrašą
   * Suradęs įmonę sukuria objektą Atsakymas ir jį serializuoja į failą atsakymas.json, kurį išsaugo į internetas folderį
   * Toliau laukia kito failo užklausa.json. Tikrina folderį kas sekundę.
2. Pirmiausia paleiskite serverį, o po to naršyklę ir pabandykite surasti informaciją apie įmonę naudojantis savo sukurta naršykle.

# Java Date Time API

## Turinys

* [LocalDate](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#LocalDate)
* [LocalTime](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#LocalTime)
* [LocalDateTime](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#LocalDateTime)
* [Period](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#Period)
* [Duration](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#Duration)
* [LocalDate ir LocalTime formatavimas](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#LocalDate-ir-LocalTime-formatavimas)
* [LocalDateTime ir JSON](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#LocalDateTime-ir-JSON)
* [Nuorodos](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#Nuorodos)
* [Užduotys](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-date-time-api#U%C5%BEduotys)

## LocalDate

LocalDate reiškia datą ISO formatu yyyy-MM-dd be laiko. Data gali būti sukuriama naudojant sistemo laikrodį:

LocalDate localDate = LocalDate.now();

System.out.println(localDate); // 2019-02-15

Jei žinome metus, mėnesį ir dieną, taip pat galime sukurti datą:

LocalDate localDate = LocalDate.of(2019, 02, 15);

System.out.println(localDate); // 2019-02-15

arba

LocalDate localDate = LocalDate.parse("2019-02-15");

System.out.println(localDate); // 2019-02-15

Prie turimos Java datd galime pridėti arba iš jos atimti dienas, savaites, mėnesius ir metus.

LocalDate localDate = LocalDate.parse("2019-02-15");

LocalDate tomorrow = localDate.plusDays(1);

System.out.println(tomorrow); // 2019-02-16

LocalDate yesterday = localDate.minusMonths(2);

System.out.println(yesterday); // 2018-12-15

Taip pat galima naudoti ChronoUnit enum'ą:

LocalDate previousMonthSameDay = LocalDate.parse("2019-02-15").minus(1, ChronoUnit.DECADES);

System.out.println(previousMonthSameDay); // 2009-02-15

Galima gauti savaitės ir mėnesio dieną:

DayOfWeek dayOfWeek = LocalDate.parse("2018-12-15").getDayOfWeek();

System.out.println(dayOfWeek); // SATURDAY

int fifteen = LocalDate.parse("2018-12-15").getDayOfMonth();

System.out.println(fifteen); // 15

Galime gauti požymį, ar tai yra keliamieji metai:

boolean leapYear = LocalDate.parse("2018-12-15").isLeapYear();

System.out.println(leapYear); // false

Galima patikrinti, ar viena data yra vėlesnė už kitą:

boolean notBefore = LocalDate.parse("2018-12-15").isBefore(LocalDate.parse("2018-12-14"));

System.out.println(notBefore); // false

boolean isAfter = LocalDate.parse("2018-12-15").isAfter(LocalDate.parse("2018-12-14"));

System.out.println(isAfter); // true

Galime gauti dienos pradžio laiką:

LocalDateTime beginningOfDay = LocalDate.parse("2019-02-15").atStartOfDay();

System.out.println(beginningOfDay); // 2019-02-15T00:00

arba mėnesio pradžios datą

LocalDate firstDayOfMonth = LocalDate.parse("2019-02-15").with(TemporalAdjusters.firstDayOfMonth());

System.out.println(firstDayOfMonth); //2019-02-01

## LocalTime

LocalTime yra laika sbe datos.

LocalTime now = LocalTime.now();

System.out.println(now); // 23:23:17.594

Galima nurodyti tikslų laiką:

LocalTime sixThirty = LocalTime.parse("06:30");

System.out.println(sixThirty); // 06:30

arba

LocalTime sixThirty = LocalTime.of(6, 30);

System.out.println(sixThirty); // 06:30

Galime nurodyti prie turimo laiko pridėti laiko vienetą naudojant ChronoUnit:

LocalTime sevenThirty = LocalTime.parse("06:30").plus(1, ChronoUnit.HOURS);

System.out.println(sevenThirty); // 07:30

Galima gauti laiko dalį - valandą, minutę, sekundė ir t.t.:

LocalTime time = LocalTime.parse("06:30");

int hour = time.getHour();

int minute = time.getMinute();

System.out.println(hour); // 6

System.out.println(minute); // 30

Galima patikrinti, ar vienas laikas yra ankstesnis už kitą:

boolean isBefore = LocalTime.parse("06:30").isBefore(LocalTime.parse("07:30"));

System.out.println(isBefore); // true

boolean isAfter = LocalTime.parse("06:30").isAfter(LocalTime.parse("07:30"));

System.out.println(isAfter); // false

Galima gauti minimalų ir makslimalų laiką:

System.out.println(LocalTime.MIN); // 00:00

System.out.println(LocalTime.MAX); // 23:59:59.999999999

## LocalDateTime

LocalDateTime yra data su laiku.

LocalDateTime now = LocalDateTime.now();

System.out.println(now); // 2019-02-15T23:38:27.859

LocalDateTime now1 = LocalDateTime.of(2019, Month.FEBRUARY, 15, 23, 39);

System.out.println(now1); // 2019-02-15T23:39

LocalDateTime now2 = LocalDateTime.parse("2019-02-15T23:39:00");

System.out.println(now2); // 2019-02-15T23:39

Galima naudoti plus ir minus metodus:

LocalDateTime now = LocalDateTime.parse("2019-02-15T23:39:00");

now = now.plusDays(1);

now = now.minusHours(5);

System.out.println(now); // 2019-02-16T18:39

## Period

Galime paskaičiuoti periodą tarp dviejų datų:

LocalDate now = LocalDate.parse("2019-02-15");

LocalDate nextWeek = now.plus(Period.ofDays(7));

long seven = Period.between(now, nextWeek).getDays();

System.out.println(seven); // 7

seven = ChronoUnit.DAYS.between(now, nextWeek);

System.out.println(seven); // 7

## Duration

Galime paskaičiuoti trukmę tarp dviejų laikų:

LocalTime initialTime = LocalTime.of(6, 30, 0);

LocalTime finalTime = initialTime.plus(Duration.ofSeconds(30));

long thirty = Duration.between(initialTime, finalTime).getSeconds();

System.out.println(thirty); // 30

thirty = ChronoUnit.SECONDS.between(initialTime, finalTime);

System.out.println(thirty); // 30

Naudojant Duration galime apskaičiuoti per kiek laiko programa atliko veiksmus:

LocalTime startTime = LocalTime.now();

for (int i = 0; i < 1000000; i++) {

System.out.println(i);

}

LocalTime endTime = LocalTime.now();

long trukme = Duration.between(startTime, endTime).getSeconds();

System.out.println("Pradžia: " + startTime); // Pradžia: 13:24:42.666

System.out.println("Pabaiga: " + endTime); // Pabaiga: 13:24:51.272

System.out.println("Trukmė: " + trukme + " sec"); // Trukmė: 8 sec

## LocalDate ir LocalTime formatavimas

LocalDateTime localDateTime = LocalDateTime.now();

System.out.println(localDateTime); // 2019-02-15T23:30:35.407

String localDateString = localDateTime.format(DateTimeFormatter.ISO\_DATE);

System.out.println(localDateString); // 2019-02-15

String formattedLocalDateTime = localDateTime.format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));

System.out.println(formattedLocalDateTime); // 2019/02/15

## LocalDateTime ir JSON

Norint konvertuoti LocalDateTime objektą į JSON naudojant Jackson turime sukurti datos ir laiko serializerį ir deserializerį.

LocalDateTimeSerializer klasė:

public class LocalDateTimeSerializer extends StdSerializer<LocalDateTime> {

public LocalDateTimeSerializer() {

super(LocalDateTime.class);

}

protected LocalDateTimeSerializer(Class<LocalDateTime> t) {

super(t);

}

@Override

public void serialize(LocalDateTime localDateTime, JsonGenerator jsonGenerator, SerializerProvider serializerProvider) throws IOException {

jsonGenerator.writeString(localDateTime.format(DateTimeFormatter.ISO\_LOCAL\_DATE\_TIME));

}

}

LocalDateTimeDeserializer klasė:

public class LocalDateTimeDeserializer extends StdDeserializer<LocalDateTime> {

public LocalDateTimeDeserializer() {

super(LocalDateTime.class);

}

protected LocalDateTimeDeserializer(Class<?> vc) {

super(vc);

}

@Override

public LocalDateTime deserialize(JsonParser jsonParser, DeserializationContext deserializationContext) throws IOException {

return LocalDateTime.parse(jsonParser.readValueAs(String.class));

}

}

Objekto su laiku klasė ObjektasSuLaiku:

public class ObjektasSuLaiku {

@JsonDeserialize(using = LocalDateTimeDeserializer.class)

@JsonSerialize(using = LocalDateTimeSerializer.class)

private LocalDateTime laikas;

public ObjektasSuLaiku() {

this.laikas = LocalDateTime.now();

}

public LocalDateTime getLaikas() {

return laikas;

}

public void setLaikas(LocalDateTime laikas) {

this.laikas = laikas;

}

}

Tokio objekto serializavimo į JSON ir deserializavimo pavyzdys:

ObjektasSuLaiku objektasSuLaiku = new ObjektasSuLaiku();

ObjectMapper mapper = new ObjectMapper();

String jsonString = mapper.writeValueAsString(objektasSuLaiku);

System.out.println(jsonString); // {"laikas":"2019-02-16T19:26:28.751"}

ObjektasSuLaiku objektasSuLaikuIsJson = mapper.readValue(jsonString, ObjektasSuLaiku.class);

System.out.println(objektasSuLaikuIsJson.getLaikas()); // 2019-02-16T19:26:28.751

## Nuorodos

* <https://www.baeldung.com/java-8-date-time-intro>
* <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>
* <https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

## Užduotys

## Užduotys: Java Date Time API

## Nr. 1

### Užduotis

* Sukurti programą, kuri paskaičiuotų Jūsų tikslų amžių metais, mėnesiais ir dienomis.
* Raskite visų Jūsų gmtadienių savaitės dienas. Sugrupuokite ir atspausdinkite kurią savaitės dieną kiek kartų šventėte gimtadienį.

## Nr. 2

### Užduotis

1960-06-10 yra laikoma magiška data, nes jos dienos skaičių padauginus iš mėnesio skaičiaus gauname metų skaičių. Originaliai 6/10/60. Parašyti programą, kuriai galime nurodyti datų rėžius. Duotuose rėžiuose surasti visas magiškas datas.

# Regex

## Turinys

* [Regex](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Regex)
* [Regex sudarymo taisyklės](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Regex-sudarymo-taisykl%C4%97s)
* [Bendrieji simboliai](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Bendrieji-simboliai)
* [Metasimboliai](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Metasimboliai)
* [Kiekybiniai simboliai](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Kiekybiniai-simboliai)
* [Grupavimas ir nuorodos](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Grupavimas-ir-nuorodos)
* [String metodų naudojimas su regex](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#String-metod%C5%B3-naudojimas-su-regex)
* [Nuorodos](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#Nuorodos)
* [Užduotys](https://github.com/tadus21/code-academy-java-kursai/tree/master/regex#U%C5%BEduotys)

## Regex

Regex yra santruma reškianti reguliarią išrašką. Regex apibrėžia paieškos šabloną eilutėje arba tekste. Paieškos šablonas gali būti bet kas nuo simbolio ar fiksuotos eilutės iki sudėtingos išraiškos su specialiais simboliais.

## Regex sudarymo taisyklės

## Bendrieji simboliai

* . atitinka bet kokį simbolį:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll(".", "#");
* System.out.println(text);

// ###########################################

* ^regex suranda regex, kuris turi eilutės pradžios simbolį:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("^fox", "#");
* System.out.println(text);
* // The quick brown fox jumps over the lazy dog
* text = text.replaceAll("^The", "#");
* System.out.println(text);

// # quick brown fox jumps over the lazy dog

* regex$ suranda regex, kuris turi eilutės pabaigos simbolį:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("fox$", "#");
* System.out.println(text);
* // The quick brown fox jumps over the lazy dog
* text = text.replaceAll("dog$", "#");
* System.out.println(text);

// The quick brown fox jumps over the lazy #

* [abc] suranda simbolius a, b arba c:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("[abc]", "#");
* System.out.println(text);

// The qui#k #rown fox jumps over the l#zy dog

* [abc][kz] suranda simbolius a, b arba c, po kurių būtinai turi sekti raidė k arba z:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("[abc][kz]", "#");
* System.out.println(text);

// The qui# brown fox jumps over the l#y dog

* [^abc] regex priešingas [abc], t.y. tinka bet koks simbolis išskyrus a, b arba c:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("[^abc]", "#");
* System.out.println(text);

// #######c##b#########################a######

* [a-d1-7] tinka visos raidės nuo a iki z ir skaičiai nuo 1 iki 7:
* String text = "The quick brown fox jumps over the lazy dog1230";
* text = text.replaceAll("[a-d1-7]", "#");
* System.out.println(text);

// The qui#k #rown fox jumps over the l#zy #og###0

* x|y|z tinka x, y arba z raidės:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("x|y|z", "#");
* System.out.println(text);

// The quick brown fo# jumps over the la## dog

* zy tinka raidės z ir y sekančios viena po kitos:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("zy", "#");
* System.out.println(text);

// The quick brown fox jumps over the la# dog

* ^ tinka eilutės pradžios simbolis:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("^", "#");
* System.out.println(text);

// #The quick brown fox jumps over the lazy dog

* $ tinka eilutės pabaigos simbolis:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("$", "#");
* System.out.println(text);

// The quick brown fox jumps over the lazy dog#

## Metasimboliai

* \d tinka bet koks skaitmuo, atitinka regex [0-9]:
* String text = "The quick brown fox jumps over the lazy dog123";
* text = text.replaceAll("\\d", "#");
* System.out.println(text);

// The quick brown fox jumps over the lazy dog###

* \D tinka bet koks skibolis, kuris yra ne skaitmuo, atitinka regex [^0-9]:
* String text = "The quick brown fox jumps over the lazy dog123";
* text = text.replaceAll("\\D", "#");
* System.out.println(text);

// ###########################################123

* \s tinka tarpo simboliai:
* String text = "The quick brown fox jumps over the lazy dog ";
* text = text.replaceAll("\\s", "#");
* System.out.println(text);

// The#quick#brown#fox#jumps#over#the#lazy#dog###

* \S tinka bet kokie simboliai išsyrus tarpą:
* String text = "The quick brown fox jumps over the lazy dog ";
* text = text.replaceAll("\\S", "#");
* System.out.println(text);

// ### ##### ##### ### ##### #### ### #### ###

* \w tinka simboliai sudarantys žodžius, atitinka regex [a-zA-Z\_0-9]:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("\\w", "#");
* System.out.println(text);

// ### ##### ##### ### ##### #### ### #### ###

* \W tinka visi simboliai, kurie nesudaro žodžių, atitinka regex [^\w]:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("\\W", "#");
* System.out.println(text);

// The#quick#brown#fox#jumps#over#the#lazy#dog

* \b randa žodžio ribas:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("\\b", "#");
* System.out.println(text);

// #The# #quick# #brown# #fox# #jumps# #over# #the# #lazy# #dog#

## Kiekybiniai simboliai

* {X} kartojasi X kartų. Pvz. \d{3} tinka trys iš eilės sekantys skaitmenys:
* String text = "The quick brown fox jumps over the lazy dog 1234";
* text = text.replaceAll("\\d{3}", "#");
* System.out.println(text);

// The quick brown fox jumps over the lazy dog #4

arba \w{5} tinka visi 5 simbolių ilgio žodžiai:

String text = "The quick brown fox jumps over the lazy dog";

text = text.replaceAll("\\w{5}", "#");

System.out.println(text);

// The # # fox # over the lazy dog

* {X,Y} kartojasi nuo X iki Y kartų. Pvz. \w{4,5} tinka visi žodžiai, kurių ilgis nuo 4 iki 5 simbolių.
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("\\w{4,5}", "#");
* System.out.println(text);

// The # # fox # # the # dog

* \* kartojasi 0 arba daugiau kartų, atitinka regex {0,}:
* String text = "The quick brown fox jumps over the lazy dog bb";
* text = text.replaceAll("[b\*]", "#");
* System.out.println(text);

// The quick #rown fox jumps over the lazy dog ##

* + kartojasi 1 arba daugiau kartų, atitinka regex {1,}:
* String text = "The quick brown fox jumps over the lazy dog bb";
* text = text.replaceAll("[b+]", "#");
* System.out.println(text);

// The quick #rown fox jumps over the lazy dog ##

* ? kartojasi 0 arba 1 kartą, atitinka regex {0,1}:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("\\s.?ro..", "#");
* System.out.println(text);

// The quick# fox jumps over the lazy dog

## Grupavimas ir nuorodos

Galima šabloną sugrupuoti, pvz. šabloną \w\s+[\.,] pakeisti į (\w)(\s+)([\.,]), kurį sudaro trys grupės. Tada tas grupes galima naudoti pvz. replaceAll metode. Turime tekstą, kuriame po žodžių yra tarpas ir kablelis. Norime pašalinti tarpus prieš kablelius:

String pattern = "(\\w)(\\s+)([\\.,])";

String text = "The quick brown fox , jumps over , the lazy dog";

text = text.replaceAll(pattern, "$1$3");

System.out.println(text);

// The quick brown fox, jumps over, the lazy dog

## String metodų naudojimas su regex

* matches("regex") galime patikrinti ar toks regex yra tekste:
* String text = "The quick brown fox jumps over the lazy dog";

System.out.println(text.matches(".\*brown.\*")); // true

* split("regex") galime suskaidyti eilutę į keletą eilučių ties regex atitinkančiomis teksto vietomis:
* String text = "The quick brown fox jumps over the lazy dog";
* String[] textArray = text.split("\\s");
* for (String s : textArray) {
* System.out.println(s);
* }
* //The
* //quick
* //brown
* //fox
* //jumps
* //over
* //the
* //lazy

//dog

* replaceFirst("regex", "replacement") galime pakeisti pirmą teksto vietą, kuri tenkina regex:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceFirst("\\s", "#");

System.out.println(text); // The#quick brown fox jumps over the lazy dog

* replaceAll("regex", "replacement") galime pakeisti visas teksto vietas, kurios tenkina regex:
* String text = "The quick brown fox jumps over the lazy dog";
* text = text.replaceAll("\\s", "#");

System.out.println(text); //The#quick#brown#fox#jumps#over#the#lazy#dog

Telefono numerio tikrinimas:

String phone1 = "+37061234567";

String phone2 = "861234567";

String phone3 = "+61234567";

String pattern = "^(\\+370|8)\\d{8}$";

System.out.println(phone1.matches(pattern)); // true

System.out.println(phone2.matches(pattern)); // true

System.out.println(phone3.matches(pattern)); // false

## Nuorodos

* <https://www.oreilly.com/library/view/java-cookbook-3rd/9781449338794/ch04.html>
* Įrankis Regex patikrinti: <https://regex101.com/>

## Užduotys

## Užduotys: Regex

## Nr. 1

### Užduotis

Tekstas:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

* Visus žodžius, kurie prasideda raide L arba l pakeiskite tekstu \*\*\*\*\*\*.

Išskleisti

* Pakeiskit eilutę taip, kad ją atspausdinus kiekvienoje eilutėje būtų tik vienas žodis.

Išskleisti

* Pakeisti eilutę taip, kad kalblelis ir taškas būtų paskutiniai kiekvienos eilutės simboliai.

Išskleisti

\* užduoties sąlygos yra viena su kita nesusijusios.

## Nr. 2

### Užduotis

Sukurkite validatorių, kuris patikrina email adresą, pvz. vardas@codeacademy.lt turi būti validus, o vardas@codea@cademy.lt - nevalidus.

## Nr. 3

### Užduotis

Tekstas:

tN6WgkPg

eX9qZRwr

nQfMQjJz

eAEwVSfh

vLCnbwXK

HUTCjjfs

bRATtkPw

VxGSzR8y

qYhWYquL

Kk3GajkG

VZ3s2U9v

QGJXQxHc

AA6WZVpA

Cws4xyXz

QeY8p4aX

Visi šie slaptažodžiai turi būti vienoje eilutėje ir atskirti tik kableliu, o išorėje - laužtiniai skliaustai.

Išskleisti

## Nr. 4

### Užduotis

Sąrašas slaptažodžių:

7YeSYKrPMLd4rSAW

ah4uDJAWxBkX

720477471992

FJU8Phr7KAcd

en8wdFtunqDkmpAL

kkcnf95pyt99sjyz

2tvy5t6qj5z6htt7

ldxorybgheyx

0021673575111166

48esda7dd9wn9rg9

3QmgwjfKPMEL

kQe366B6X5KcXM9G

NsBRaeX2vDRmjfU2

JCWNY54NJLVA

5ggqS2uQgZ59Tqct

xibrjbktsbew

Naudojant srautus ir regex sugrupuoti šiuos šlaptažodžius į grupes:

* STIPRUS - jei slaptažodžio ilgis yra bent 16 smbolių ir jame yra bent dvi mažosios raidės, bent dvi didžiosios raidės ir bent du skaitmenys.
* VIDUTINIS - jei jame yra bent dvi mažosios raidės, bent dvi didžiosios raidės ir bent du skaitmenys.
* SILPNAS - visais kitais atvejais.

Pavyzdys:

{VIDUTINIS=[FJU8Phr7KAcd], SILPNAS=[ah4uDJAWxBkX, 720477471992, en8wdFtunqDkmpAL, kkcnf95pyt99sjyz, 2tvy5t6qj5z6htt7, ldxorybgheyx, 0021673575111166, 48esda7dd9wn9rg9, 3QmgwjfKPMEL, JCWNY54NJLVA, xibrjbktsbew], STIPRUS=[7YeSYKrPMLd4rSAW, kQe366B6X5KcXM9G, NsBRaeX2vDRmjfU2, 5ggqS2uQgZ59Tqct]}

# Lygiagretus programavimas (Concurrency)

Kompiuterių naudotojams savaime suprantama, kad jų sistemos gali daryti daug dalykų vienu metu. Jie tikisi, kad galės naršyti internete, kol kitos programėlės siunčia failus, groja muziką, spausdina dokumentus. Net ir paprastos programėlės paprastai daro skirtingus dalykus tuo pačiu metu – pavyzdžiui teksto redaktorius reaguoja į klaviatūros bei pelės judesius tuo pat metu redaguodamas tekstą ir atnaujindamas vaizdą. Programinė įranga, galinti daryti daug dalykų vienų metu vadinama concurrent software arba lygiagrečia.

Java programavimo kalba sukurta taip, kad galėtų palaikyti lygiagretų programavimą java.util.concurrent pakete.

Lygiagrečiame programavime yra du svarbūs vykdymo vienetai – procesai (processes) ir gijos (threads)

Sistema paprastai turi kelis aktyvius procesus bei gijas, tačiau procesorius, turintis tik vieną vykdymo branduolį, vienu metu gali vykdyti tik vieną giją. Norint pasiekti lygiagretumą, vieno branduolio procesoriuje procesoriaus vykdymo laikas paskirstomas per visas gijas bei procesus. Tokia operacinės sistemos funkcija vadinama slicing.

Procesoriai su keliais branduoliais pagerina lygiagrečių procesų vykdymą, nes kiekvienas branduolys gali vykdyti procesus vienu metu, paraleliai.

### Procesai

Procesas yra vykdymo aplinka, turinti privačius resursus. Kiekvienas procesas turi sau išskirtą vietą atmintyje. Java programėlė yra operacinės sistemos procesas.

### Gijos

Gijos, kaip ir procesai, sukuria vykdymo aplinką, tačiau jos neturi privačių resursų. Gijos egzistuoja procesuose ir naudoja proceso resursus. Kiekvienas procesas turi bent vieną arba daugiau gijų.

Java programėlė, kuri po vieną skaito sistemos failus ir apdoroja jų turinį, yra procesas iš vienos gijos. Java internetinis puslapis yra Multi threaded procesas, sukuriantis naują giją, kiekvienai naudotojo užklausai.

### Gijų sukūrimas

Gijos sukuriamos naudojant klasę Thread. Į klasės konstruktorių reikia paduoti sąsajos Runnable įgyvendinimą. Runnableklasė turi vieną metodą public void run(), kuris yra vykdomas naujoje gijoje, iškvietus Thread klasės metodą start.

### Thread metodai

Thread klasės metodas sleep sustabdo klasės veikimą milisekundėmis nurodytam laiko tarpui. sleep metodas grąžina InterruptedException, kuris turi būti suvaldytas, jei gijos veikimas yra sustabdytas iš kitos gijos.

interrupt metodas gali būti iškviestas iš kitos gijos, norint nurodyti, kad dabartinis gijos veiksmas turi būti nutrauktas.

Thread klasė turi metodą join kuris liepia dabartinei gijai palaukti, kol nurodyta gija baigs darbą. Įvykdžius join funkciją visi pakeitimai, įvykdyti nurodytoje gijoje, tampa matomi tai gijai, kuri iškvietė minėtą metodą.

Iki tol veiksmai, vykdomi skirtingose gijose, nėra matomi viena kitai.

### Sinchronizavimas

Jei kelios gijos vienu metu dirba su tos pačios klasės laukais, galima tikėtis nenumatytų klaidų. Norint to išvengti naudojama sinchronizacija. Metodai gali būti paženklinti synchronized raktažodžiu. Jis nurodo, kad metodas negali būti iškviestas tol, kol jis yra vykdomas kitos gijos. Taip pat, veiksmai, įvykdyti šiuo raktažodžiu pažymėtuose metoduose yra matomi visose gijose.

### Instrict Locks

Sinchronizacija yra įgyvendinta naudojant užraktą vadinamą instrict lock arba monitor lock. Kiekvienas objektas turi užraktą, gija, norėdama pasiekti objekto synchronized metodus, pirma turi gauti jo užraktą ir grąžinti jį, kai darbas su laukais yra baigtas. Kol gija turi užraktą, jokia kita gija negali dirbti su objekto laukais. Kai gija kviečia objekto synchronized metodą, ji automatiškai gauna jo užraktą, kuri grąžina, kai metodas yra įvykdomas.

### Užduotis

Sukurkite klasę ProgressBar kuris turėtų int lauką progress. Jūsų programėles main metodas turėtų sukurti ProgressBar tipo objektą ir vykdyti ciklą, kuris didintų progress lauko reikšmę vienetu kas sekundę, tol, kol reikšmė pasiekia 100. Sukurkite kitą giją, kuri kas 3 sekundes į konsolę išvestų ProgressBar objekto, progress lauko reikšmę, tol, kol programa veikia.

### Šaltiniai

* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

# Java 9-11

## Turinys

* [Java 9](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Java-9)
  + [Modulių sistema](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Moduli%C5%B3-sistema)
  + [VisualVM perkeltas į GraalVM](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#VisualVM-perkeltas-%C4%AF-GraalVM)
  + [JShell įrankis](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#JShell-%C4%AFrankis)
  + [Factory metodai kolekcijoms](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Factory-metodai-kolekcijoms)
  + [ObjectInputFilter](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#ObjectInputFilter)
  + [Naujas HTTP klientas](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Naujas-HTTP-klientas)
  + [Patobulintas datų iteravimas](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Patobulintas-dat%C5%B3-iteravimas)
* [Java 10](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Java-10)
  + [var](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#var)
  + [Optional.orElseThrow() metodas](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Optional.orElseThrow()-metodas)
* [Java 11](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Java-11)
  + [Nemodifikuojamų kolekcijų sukūrimas](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Nemodifikuojam%C5%B3-kolekcij%C5%B3-suk%C5%ABrimas)
  + [Vieno failo programa](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Vieno-failo-programa)
  + [Nauji String metodai](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Nauji-String-metodai)
  + [Files.writeString() ir Files.readString() metodai](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Files.writeString()-ir-Files.readString()-metodai)
  + [TimeUnit.convert() metodas](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#TimeUnit.convert()-metodas)
  + [Kitos naujienos](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Kitos-naujienos)
* [Nuorodos](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#Nuorodos)
* [Užduotys](https://github.com/tadus21/code-academy-java-kursai/tree/master/java-9-11#U%C5%BEduotys)

## Java 9

### Modulių sistema

Modulis - naujas programavimo komponentas. Programa gali būti suskirstyta į modulius. Leidžiama valdyti matomumą tarp skirtingų programos modulių. Java JDK taip pat buvo suskirstyta į modulius. Plačiau: <http://openjdk.java.net/jeps/261>

### VisualVM perkeltas į GraalVM

Įrankis VisualVM perkeltas į GraalVM ir nebėra tarp kitų JAVA\_HOME/bin esančių įrankių. VisualVM galima toliau naudoti parsisiuntus iš <https://visualvm.github.io/>

### JShell įrankis

Kataloge JAVA\_HOME/bin galime rasti naują įrankį jshell.exe. Su šiuo įrankiu galime naudoti Java komandas interaktyviame lange, nekuriant jokios klasės.

Jei Java įdiegta į C:\Program Files\Java\jdk-11.0.2, tai jshell įrankį galite iškviesti komandinėje eilutėje įvedus: "C:\Program Files\Java\jdk-11.0.2\bin\jshell.exe"

C:\ROOT

λ "C:\Program Files\Java\jdk-11.0.2\bin\jshell.exe"

| Welcome to JShell -- Version 11.0.2

| For an introduction type: /help intro

jshell> String eilute = "Labas";

eilute ==> "Labas"

jshell> System.out.println(eilute);

Labas

jshell>

Daugiau apie jshell: <https://docs.oracle.com/javase/9/jshell/toc.htm>

### Factory metodai kolekcijoms

Palengvinimas norint užpildyti naujus List, Set ir Map objektus duomenimis.

Java 8:

List<String> list8 = new ArrayList<>();

list8.add("a");

list8.add("b");

list8.add("c");

Java 9:

List<String> list9 = List.of("a", "b", "c");

Tačiau toks sukurtas sąrašas yra nemodifikuojamas, t.y. jei norėsime įdėti dar vieną elementą list9.add("d");, ištrinti ar pakeisti esamą, to padaryti negalėsime ir gausime klaidą UnsupportedOperationException. Toks būdas sukurti kolekciją naudojant factory metodą of() naudingas tik tada, kai visi kolekcijos elementai yra žinomi ir nenorėsime modifikuoti kolekcijos vėliau.

Set ir Map taip pat galime sukurti naudojant factory metodą of():

Set<String> set9 = Set.of("a", "b", "c");

Map<String, Integer> map9 = Map.of("one", 1, "two", 2, "three", 3);

Sukurti Set ir Map objektai taip pat yra nemodifikuojami.

Plačiau: <https://docs.oracle.com/javase/9/core/creating-immutable-lists-sets-and-maps.htm>

### ObjectInputFilter

Sukurta ObjectInputFilter klasė, kuri leidžia patikrinti objektą dar prieš jį deserializuojant. Panaudojimo pavyzdys:

public class Main {

public static void main(String[] args) throws Exception {

objectOutputInputTest(new GeraKlase());

objectOutputInputTest(new BlogKlase());

}

private static void objectOutputInputTest(Object objectToSerialize) throws Exception {

// objekto serializavimas

Path path = Files.createTempFile("failas", "");

ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream(path.toFile()));

try (objectOutputStream) {

objectOutputStream.writeObject(objectToSerialize);

}

// objekto deserializavimas

ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream(path.toFile()));

objectInputStream.setObjectInputFilter(createObjectFilter()); // nustatomas filtras

try (objectInputStream) {

Object o = objectInputStream.readObject();

System.out.println("Deserializuotas objektas: " + o);

}

}

private static ObjectInputFilter createObjectFilter() {

return filterInfo -> {

Class<?> theClass = filterInfo.serialClass();

if (GeraKlase.class.isAssignableFrom(theClass)) {

System.out.println("Leistina klase: " + theClass.getSimpleName());

return ObjectInputFilter.Status.ALLOWED;

} else {

System.err.println("Neleistina klase: " + theClass.getSimpleName());

return ObjectInputFilter.Status.REJECTED;

}

};

}

}

class GeraKlase implements Serializable {

}

class BlogKlase implements Serializable {

}

Rezultatas:

Leistina klase: GeraKlase

Deserializuotas objektas: lt.ca.GeraKlase@2d928643

BlogKlase class is not allowed for serialization: BlogKlase

Exception in thread "main" java.io.InvalidClassException: filter status: REJECTED

at java.base/java.io.ObjectInputStream.filterCheck(ObjectInputStream.java:1287)

...

### Naujas HTTP klientas

Pakeitė seną HttpURLConnection. Paprastas GET užklausos pavyzdys:

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI("https://postman-echo.com/get"))

.GET()

.build();

HttpResponse<String> response = HttpClient.newHttpClient()

.send(request, HttpResponse.BodyHandlers.ofString());

System.out.println(response.body());

Plačiau: <https://openjdk.java.net/groups/net/httpclient/intro.html>

### Patobulintas datų iteravimas

public class Main {

public static void main(String[] args) {

iteruotiTarpDatu7(new Date(2019, 1, 1), new Date(2019, 1, 15));

iteruotiTarpDatu8(LocalDate.of(2019, 1, 1), LocalDate.of(2019, 1, 15));

iteruotiTarpDatu9(LocalDate.of(2019, 1, 1), LocalDate.of(2019, 1, 15));

}

// JAVA 7

public static void iteruotiTarpDatu7(Date start, Date end) {

Date current = start;

while (current.before(end)) {

System.out.println(current);

Calendar calendar = Calendar.getInstance();

calendar.setTime(current);

calendar.add(Calendar.DATE, 1);

current = calendar.getTime();

}

}

// JAVA 8

public static void iteruotiTarpDatu8(LocalDate start, LocalDate end) {

for (LocalDate date = start; date.isBefore(end); date = date.plusDays(1)) {

System.out.println(date);

}

}

// JAVA 9

public static void iteruotiTarpDatu9(LocalDate start, LocalDate end) {

start.datesUntil(end).forEach(System.out::println);

}

}

## Java 10

### var

Galime kairėje pusėje prieš kintamojo pavadinimą nerašyti tipo. Tipas bus nusprendžiamas pagal dešinėje pusėje esančią išraišką. Pavyzdys:

var i = 10;

System.out.println(i); // 10

var list = new ArrayList<>();

list.add(2);

list.add("one");

System.out.println(list); // [2, one]

Bet negalime daryt taip:

var i;

var i, j = 0;

var i = null;

nes Java kompiliatorius nežinos, koks yra kintamojo 'i' tipas.

Daugiau apie var naudojimą: <https://www.baeldung.com/java-10-local-variable-type-inference>

### Optional.orElseThrow() metodas

Jei Optional yra be reikšmės galima naudoti metodą orElseGet, kuris grąžins nurodytą numatytąją reikšmę arba nuo Java 9-os versijos metodą orElseThrow, kuris iššauks klaidą. Pavyzdys:

Optional<String> test = Optional.empty();

test.orElseThrow();

Exception in thread "main" java.util.NoSuchElementException: No value present

at java.base/java.util.Optional.orElseThrow(Optional.java:382)

at lt.ca.Main.main(Main.java:12)

### Nemodifikuojamų kolekcijų sukūrimas

Map, List ir Set klasės turi naują statinį metodą copyOf(...), kuris nukopijuoja paduotos kolekcijos elementus ir sukuria nemodifikuojamą kolekciją.

Pavyzdys:

List<String> sarasas = new ArrayList<>();

sarasas.add("hello");

sarasas.add("world");

List<String> nemodifikuojamasSarasas = List.copyOf(sarasas);

nemodifikuojamasSarasas.add("!!!");

Gausime UnsupportedOperationException klaidą.

## Java 11

### Vieno failo programa

Turime paprastą klasę Greeting, kuri reikalauja jai paduoti vardą per argumentus. Ši programa yra saugome faile pavadinimu Greeting.java:

public class Greeting {

public static void main(String[] args) {

if (args == null || args.length < 1) {

System.err.println("Name required");

System.exit(1);

}

System.out.println(String.format("Hello %s!!", args[0]));

}

}

Tada tokią programą galime paleisti taip:

java Greeting.java John

t.y. nereikia naudoti kompiliavimo komandos javac ir vėliau naudoti java komandą sukompiuliuotos programos paleidimui. Nuo Java 11-os versijos komanda java supranta, kad norime paleisti vieno failo Java programą ir ją sukompiliavusi paleidžia.

### Nauji String metodai

String klasė turi naujus metodus:

* isBlank() - patikrina ar eilutė yra tuščia arba yra sudaryt iš tuščių simbolių:
* String eilute = "";
* System.out.println(eilute.isBlank()); // true
* eilute = " ";

System.out.println(eilute.isBlank()); // true

* lines() - grąžina sąrašą eilučių kaip srautą:
* String daugEiluciu = "Hello\nworld";
* daugEiluciu.lines()
* .forEach(System.out::println); // Hello

// world

* repeat() - grąžina tiek kartų padidintą eilutę, kokį skaičiu padavėme per parametrus:
* final String desimtIksu = "X".repeat(10);

System.out.println(desimtIksu); // XXXXXXXXXX

* strip(), stripLeading() ir stripTrailing() - grąžina eilutę be pradžioje ir pabaigoje buvusių tarpų:
* System.out.println("|" + eiluteSuTarpais + "|"); // | Labas Vakaras |
* System.out.println("|" + eiluteSuTarpais.strip() + "|"); // |Labas Vakaras|
* System.out.println("|" + eiluteSuTarpais.stripLeading() + "|"); // |Labas Vakaras |

System.out.println("|" + eiluteSuTarpais.stripTrailing() + "|");// | Labas Vakaras|

### Files.writeString() ir Files.readString() metodai

Klasės Files metodas writeString leidžia turimą String eilutę lengvai įrašyti į failą:

Files.writeString(Path.of("failas.txt"), "Labas!");

Klasės Files metodas readString leidžia lengvai nuskaityti failą ir tekstą priskirti String tipo kintamajam:

String eiluteIsFailo = Files.readString(Path.of("failas.txt"));

System.out.println(eiluteIsFailo); // Labas!

### TimeUnit.convert() metodas

Naudojant klasės TimeUnit metodą convert() galime konvertuoti iš vieno laiko vieneto į kitą. Pavyzdys:

TimeUnit unit = TimeUnit.DAYS;

long fiftyHoursInDays = unit.convert(Duration.ofHours(50));

System.out.println(fiftyHoursInDays); // 2

unit = TimeUnit.SECONDS;

long oneYearInSeconds = unit.convert(Duration.ofDays(365));

System.out.println(oneYearInSeconds); // 31536000

### Kitos naujienos

Daugiau naujų funkcijų su pavyzdžiais galima rasti čia: <https://dzone.com/articles/90-new-features-and-apis-in-jdk-11>

## Nuorodos

* Release notes:
  + JDK 9 <https://www.oracle.com/technetwork/java/javase/9all-relnotes-3704433.html>
  + JDK 10 <https://www.oracle.com/technetwork/java/javase/10all-relnotes-4108743.html>
  + JDK 11 <https://www.oracle.com/technetwork/java/javase/11all-relnotes-5013287.html>

## Užduotys

## Užduotys: Java 9-11

## Nr. 1

### Užduotis

Surasti jshell įrankį ir atlikti keletą nesudėtingų Java operacijų:

* sukurti sąrašą
* į sąrašą įdėti keletą elementų
* elementus surūšiuoti ir atspausdinti

## Nr. 2

### Užduotis

Naudojant List.of() sukurkite sąrašą su duomenimis. Sukurtas sąrašas yra nemodifikuojamas. Padarykite taip, kad į sąrašą galėtumėte įdėti naujų elementų ir atspausdinkite papildytą sąrašą.

## Nr. 3

### Užduotis

Naudodami HttpClient ir GET užklausą iškvieskite šį adresą http://api.population.io:80/1.0/population/2019/Lithuania/. Atspausdinkite gautą rezultatą. Galite iš JSON formato gautus duomenis paversti į Java objektus ir atspausdinti informaciją struktūrizuotai. Daugiau šio serviso metodų: <http://api.population.io/>

## Nr. 4

### Užduotis

Naudojantis datų iteravimo pavyzdžiu ir String klasės metodu repeat(...) atspausdinkite visas datas nuo 2019-01-01 iki 2019-02-15 tokiu formatu:

2019-01-01

2019-01-02

2019-01-03

2019-01-04

2019-01-05

2019-01-06

2019-01-07

2019-01-08

2019-01-09

2019-01-10

2019-01-11

2019-01-12

2019-01-13

2019-01-14

2019-01-15

2019-01-16

2019-01-17

2019-01-18

2019-01-19

2019-01-20

2019-01-21

2019-01-22

2019-01-23

2019-01-24

2019-01-25

2019-01-26

2019-01-27

2019-01-28

2019-01-29

2019-01-30

2019-01-31

2019-02-01

2019-02-02

2019-02-03

2019-02-04

2019-02-05

2019-02-06

2019-02-07

2019-02-08

2019-02-09

2019-02-10

2019-02-11

2019-02-12

2019-02-13

2019-02-14

## Nr. 5

### Užduotis

Sukurkite Map'ą nurodydami kintamojo tipą var, į jį įdėkite keletą porų, Map'ą atspausdinkite. Atspausdinkite Map kintamojo tipą.

## Nr. 6

### Užduotis

Sukurkite paprastą Java programą viename faile ir jį paleiskite su Java 11-os versijos komanda java FailoPavadinimas.java

## Nr. 7

### Užduotis

Naudojant klasės TimeUnit metodą convert parašykite programą, kuriai įvedus skaičių (dienų, valandų ir pan.) atspausdins tą patį laiką tik kitais matavimo vienetais. Pavyzdžiui įvedus 15 dienų programa turi atspausdinti:

15 dienu

360 valandu

21600 minuciu

1296000 sekundziu

1296000000 milisekundziu

1296000000000 mikrosekundziu

1296000000000000 nanosekundziu

## Nr. 8

### Užduotis

Parašykite vieną kodo eilutę naudojant Java 11-os versijos naujus String metodus, kad gautumėte:

|\_

\_|

|\_

\_|

|\_

\_|

|\_

\_|

|\_

\_|

Gautą rezultatą įrašykite į failą naudojant Files metodą writeString.

# Lygiagretus programavimas (Concurrency)

Kompiuterių naudotojams savaime suprantama, kad jų sistemos gali daryti daug dalykų vienu metu. Jie tikisi, kad galės naršyti internete, kol kitos programėlės siunčia failus, groja muziką, spausdina dokumentus. Net ir paprastos programėlės paprastai daro skirtingus dalykus tuo pačiu metu – pavyzdžiui teksto redaktorius reaguoja į klaviatūros bei pelės judesius tuo pat metu redaguodamas tekstą ir atnaujindamas vaizdą. Programinė įranga, galinti daryti daug dalykų vienų metu vadinama concurrent software arba lygiagrečia.

Java programavimo kalba sukurta taip, kad galėtų palaikyti lygiagretų programavimą java.util.concurrent pakete.

Lygiagrečiame programavime yra du svarbūs vykdymo vienetai – procesai (processes) ir gijos (threads)

Sistema paprastai turi kelis aktyvius procesus bei gijas, tačiau procesorius, turintis tik vieną vykdymo branduolį, vienu metu gali vykdyti tik vieną giją. Norint pasiekti lygiagretumą, vieno branduolio procesoriuje procesoriaus vykdymo laikas paskirstomas per visas gijas bei procesus. Tokia operacinės sistemos funkcija vadinama slicing.

Procesoriai su keliais branduoliais pagerina lygiagrečių procesų vykdymą, nes kiekvienas branduolys gali vykdyti procesus vienu metu, paraleliai.

### Procesai

Procesas yra vykdymo aplinka, turinti privačius resursus. Kiekvienas procesas turi sau išskirtą vietą atmintyje. Java programėlė yra operacinės sistemos procesas.

### Gijos

Gijos, kaip ir procesai, sukuria vykdymo aplinką, tačiau jos neturi privačių resursų. Gijos egzistuoja procesuose ir naudoja proceso resursus. Kiekvienas procesas turi bent vieną arba daugiau gijų.

Java programėlė, kuri po vieną skaito sistemos failus ir apdoroja jų turinį, yra procesas iš vienos gijos. Java internetinis puslapis yra Multi threaded procesas, sukuriantis naują giją, kiekvienai naudotojo užklausai.

### Gijų sukūrimas

Gijos sukuriamos naudojant klasę Thread. Į klasės konstruktorių reikia paduoti sąsajos Runnable įgyvendinimą. Runnableklasė turi vieną metodą public void run(), kuris yra vykdomas naujoje gijoje, iškvietus Thread klasės metodą start.

### Thread metodai

Thread klasės metodas sleep sustabdo klasės veikimą milisekundėmis nurodytam laiko tarpui. sleep metodas grąžina InterruptedException, kuris turi būti suvaldytas, jei gijos veikimas yra sustabdytas iš kitos gijos.

interrupt metodas gali būti iškviestas iš kitos gijos, norint nurodyti, kad dabartinis gijos veiksmas turi būti nutrauktas.

Thread klasė turi metodą join kuris liepia dabartinei gijai palaukti, kol nurodyta gija baigs darbą. Įvykdžius join funkciją visi pakeitimai, įvykdyti nurodytoje gijoje, tampa matomi tai gijai, kuri iškvietė minėtą metodą.

Iki tol veiksmai, vykdomi skirtingose gijose, nėra matomi viena kitai.

### Sinchronizavimas

Jei kelios gijos vienu metu dirba su tos pačios klasės laukais, galima tikėtis nenumatytų klaidų. Norint to išvengti naudojama sinchronizacija. Metodai gali būti paženklinti synchronized raktažodžiu. Jis nurodo, kad metodas negali būti iškviestas tol, kol jis yra vykdomas kitos gijos. Taip pat, veiksmai, įvykdyti šiuo raktažodžiu pažymėtuose metoduose yra matomi visose gijose.

### Instrict Locks

Sinchronizacija yra įgyvendinta naudojant užraktą vadinamą instrict lock arba monitor lock. Kiekvienas objektas turi užraktą, gija, norėdama pasiekti objekto synchronized metodus, pirma turi gauti jo užraktą ir grąžinti jį, kai darbas su laukais yra baigtas. Kol gija turi užraktą, jokia kita gija negali dirbti su objekto laukais. Kai gija kviečia objekto synchronized metodą, ji automatiškai gauna jo užraktą, kuri grąžina, kai metodas yra įvykdomas.

### Užduotis

Sukurkite klasę ProgressBar kuris turėtų int lauką progress. Jūsų programėles main metodas turėtų sukurti ProgressBar tipo objektą ir vykdyti ciklą, kuris didintų progress lauko reikšmę vienetu kas sekundę, tol, kol reikšmė pasiekia 100. Sukurkite kitą giją, kuri kas 3 sekundes į konsolę išvestų ProgressBar objekto, progress lauko reikšmę, tol, kol programa veikia.

### Šaltiniai

* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

# Java virtuali mašina

Java virtuali mašina (trumpiau JVM) - kompiuterinė aplinka, leidžianti vykdyti sukompiliuotas Java programas, vadinamas bytecode. Java virtuali mašina yra apibrėžtą specifikacija kuri aprašo kaip ji turėtų veikti. Teisingai įgyventinus specifikaciją galima sukurti savo JVM kuri galėtų vykdyti sukompiliuotą Java baitų kodą. Yra daug skirtingų JVM įgyvendinimų – iš skirtingų tiekėjų, įvairioms platformoms.

Geras JVM supratimas padeda geriau išmanyti pačios Java veikimą.

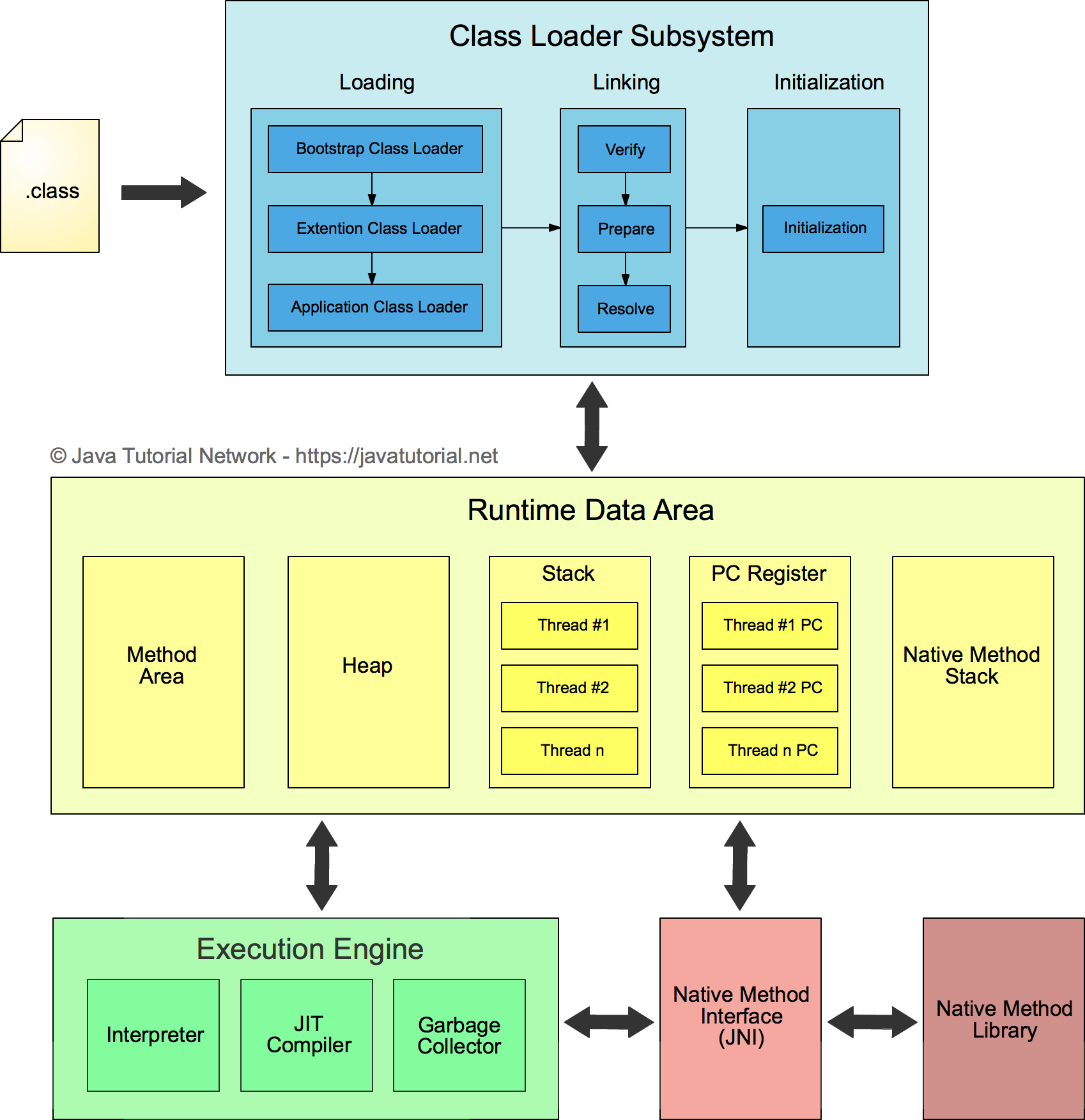
Vienas iš svarbių Java privalumų – gebėjimas veikti įvairiose platformose. Ji kurta remiantis idėja ‘‘Parašyti kartą, leisti bet kur“. Skirtingai nei kitos kalbos, Java šaltinio kodas pirmiausia sukompiliuojamas į bytecode - .class failą, kurį ir interpretuoja JVM.

### Architektūra

Java virtualią mašiną sudaro trys pagrindinės dalys:

* **Class loader subsystem** - klasių užkrovimo posistemė
* **Runtime data area** - vykdymo sritis
* **Execution engine** - vykdymo variklis

Diagrama iš [javatutorial.net](https://javatutorial.net/jvm-explained) vaizduojanti JVM architektūrą

[](https://camo.githubusercontent.com/cc988e853d0a664463fdec1cbee4594f5859bf0e/68747470733a2f2f6a6176617475746f7269616c2e6e65742f77702d636f6e74656e742f75706c6f6164732f323031372f31302f6a766d2d6172636869746563747572652e706e67)

#### Klasių užkrovimo posistemė (Class loader subsystem)

Kaip jau minėta, sukompiliuotos klasės saugomos kaip .class failai. Naudojantis class failu, classLoader užkrauna reikalingą klasę į atmintį. Klasės įvedamos į Java aplinką, kai jas iškviečia jau vykstanti klasė. Jau veikiant pirmajai klasei, kitų klasių užkrovimą vykdo classLoader. Pirmoji klasė dažniausiai paleidžiama naudojant \* static main()\* metodą.

Yra trys ClassLoader rūšys:

* Bootstrap Class Loader – užrauna vidines JDK klases, tokias kaip klases esančias java.lang.\* pakete
* Extensions Class Loader – užkrauna papildomas JDK klases paprastai iš JRE lib/ext direktorijos
* System Class Loader – užkrauna klases iš classpath

##### Susiejimas

Jungiant klases ar sąsajas vyksta reikalingų klasių ar sąsajų, jų tiesioginių superklasių, supersąsajų, elementų tipų patvirtinimas ir paruošimas. JVM reikalauja, kad būtų laikomasi šių savybių:

* Klasė ar sąsaja prieš siejant turi būti visiškai užkrauta
* Klasė ar sąsaja turi būti pilnai patvirtinta ir paruošta prieš inicializavimą
* Klaidos, aptiktos siejant, grąžinamos toje programos vietoje, kurioje tiesiogiai ar netiesiogiai gali būti reikalingas siejimas tai klasei ar sąsajai

##### Inicializavimas

Klasės ar sąsajos inicializavimas yra klasės objekto sukūrimas atmintyje. Inicializavimas sudarytas iš klasės ar sąsajos inicializavimo metodo, kitaip klasės konstruktoriaus, iškvietimo. Dėl JVM multithreadingo klasės ar sąsajos inicializavimas turi būti kruopščiai sinchronizuotas. Priešingu atveju skirtingos gijos gali mėginti inicializuoti klasę tuo pat metu.

### Vykdymo duomenų sritis (Runtime data area)

Vykdymo sritį sudaro penki pagrindiniai komponentai:

* Metodo sritis (Method area) - šioje srityje saugomi visi klasių lygio duomenys ir statiniai kintamieji. JVM yra tik viena, bendra metodo sritis
* Grupės sritis (Heap area) - šioje srityje saugomi visi objektai, masyvai ir jų atitinkami kintamieji. JVM turi tik vieną grupės sritį. Kadangi method irheap sritys yra bendros tarp skirtingų gijų, saugomi duomenys nėra thread safe
* Steko sritis (Stack area) Kiekvienai gijai yra sukuriamas atskiras vykdomasis stekas. Kiekvienam kviečiamam metodui sukuriamas atskiras įrašas steko atmintyje, vadinamas Stack Frame. Steko atmintyje kuriami visi lokalūs metodo kintamieji. Ši dalis yra thread safe nes ja nėra dalijamasi tarp skirtingų gijų. Stack Frame skirstoma į tris dalis:
  + Lokalių kintamųjų masyvas – lokalus metodo kintamieji yra saugomi šiame masyve
  + Operandų stekas – naudojamas tarpinėms (skaičiavimo) operacijoms. Operandų stekas veikia kaip vykdymo vieta kurioje atliekamos operacijos
  + Frame data – naudojama pasiekti konstantas bei metode įvykusius exception
* Registrai (PC Registers) Kiekvienas gija turi atskirus registrus, saugančius tuo metu vykstančios operacijos adresą. Įvykdžius operaciją, į registrą pakraunamas naujos operacijos adresas.
* Native Method stacks Laiko informaciją apie gijos naudojamus native metodus

### Vykdymo variklis (Execution engine)

Vykdymo duomenų sričiai priskirtas bytecode bus vykdomas atlikimo variklio, kuris iš eilės skaito ir vykdo gautą kodą.

#### Interpretatorius

Interpretatorius interpretuoja baitų kodą ir pateikią jį vykdyti kompiuteriui kaip mašinį native kodą . Interpretatorius bytecodeinterpretuoja greitai, tačiau jo vykdymas yra gan lėtas. Interpretatoriaus trūkumas – kviečiant tą patį metodą kelis kartus, interpretavimas kaskart bus kartojamas.

#### JIT kompiliatorius

JIT kompiliatorius neutralizuoja interpretatoriaus minusus. Konvertuojant bytecode atlikimo variklis naudoja interpretatorių, tačiau aptikus pasikartojantį kodą, pasitelkiamas JIT kompiliatorius. Tokiu atveju sukompiliuojamas visas bytecode ir pakeičiamas į native code. Native code naudojamas pasikartojantiems metodo iškvietimams, taip pagreitinant vykdymą.

#### Garbage collector

The Garbage collector (GC) surenka ir pašalina nenaudojamus objektus – atmintyje sukurtus objektus į kuriuos nėra jokių nuorodų. Garbage Collection gali būti įjungiama iškviečiant System.gc(), tačiau tai negarantuoja paleidimo.

#### Java Native Interface (JNI)

Sąsaja skirta bendrauti su Native method libraries, norint vykdyti native metodus

#### Native Method Libraries

Kolekcija native bibliotekų reikalingų vykdymo varikliui

### Šaltiniai

* <https://javatutorial.net/jvm-explained>
* <https://docs.oracle.com/javase/specs/jvms/se8/html/index.html>