# **Programų versijų valdymo pagrindai įrankiu Git**

<https://www.youtube.com/watch?v=xuB1Id2Wxak>

susikuri git hub repozitorija

tada gitbach

kad pamatyti kas vyra naudojam git status

kad isvalyti cmd puslapy tai naudojam clear

ls – parodo uzkomitintu

suukuriam repozitorija

git init

$ git remote add origin "https://github.com/asag1/react.git"

$ git pull origin master

Atsiranda readme failas

Git add . (visi failai jai norime viena tai git add pvz.tex nurodom konkretu)

$ git commit -m "react pvz first"

$ git push origin master

Jai kuriame branch

Tada

Git branch pavadinimas

Jai noriime i ji persijungti

Git checkout pavadinimas

Taipat sukomitinam

Bet jai norim merge

Git merge pavadinimasBranch (naudojam is master) arba reabase

Jai naudojai add comanda bet nepushinai tai pakeitimus gali tik commit

### Kas yra versijų valdymo sistemos?

Versijų valdymo sistemos (version control system arba VCS) seka bendrų projektų, kuriamų grupės žmonių, pokyčių istoriją. Sistema palaiko projektų vientisumą, padeda sujungti kelių žmonių paraleliai atliktus pakeitimus ir užtikrina, kad bet kokia projekto versija būtų atkurta neprarandant informacijos. Versijų valdymo sistema saugo visų projekto pakeitimų istoriją. Tokią informaciją programuotojai naudoja norėdami sužinoti, kokie pakeitimai buvo padaryti, kas už juos atsakingas, kada jie buvo padaryti, ir kodėl jie buvo reikalingi.

### Paskirstytos versijų valdymo sistemos

Paskirstytą versijos valdymo sistemą (Distributed version control system arba DVCS) sudaro nuotolinis serveris, identifikuojamo kaip centrinė saugykla (central repository) bei lokalios klientų saugyklos. Nuotolinis serveris klientams teikia visą saugyklos turinį, su prieiga prie kiekvieno projekto failo, atšakos ir visų pakeitimų istorijos. Centrinėje saugykloje yra pagrindinė projekto versija, kiekvienas vartotojas įvykdytus pakeitimus sinchronizuoja būtent su ja. DVCS nereikalauja nuolatinio ryšio su centrine saugykla - skirtingų klientų padaryti pakeitimai sinchronizuojami esant poreikiui. Jei nuotolinis serveris miršta, centrinė saugykla gali būti atkurta iš bet kurio kliento saugyklos, nes klientams pateikiamas visas centrinės saugyklos turinys.

### Git

Git yra viena iš paskirstytų versijų valdymo sistemų, sukurta 2005 metais Linus Torvalds siekiant palengvinti „Linux“ branduolio kūrimą. Tai populiariausia versijų valdymo sistema ir pagal Stack Overflow Developers [apklausą](https://insights.stackoverflow.com/survey/2018#work-version-control) 2018 m. ją naudojo 88 proc. programuotojų.

#### Git duomenų saugojimas

Git saugo duomenis kaip momentinius atvaizdus (Snapshot). Momentinis atvaizdas yra projekto failų turinys tam tikru momentu. Kaskart, kai išsaugosite projekto pokyčius, Git sukuria naują momentinį atvaizdą ir išsaugo nuorodą į jį. Git įrankis yra efektyvus, nes nesaugo identiškų failų. Jei naujame momentiniame atvaizde failas išlieka nepakeistas, jis pakartotinai neįrašomas ir naudojamas buvusiame atvaizde esantis failas.

#### Git Struktūra

Git projekto struktūra sudaro 3 sritys - „medžiai“:

* **Darbinė sritis** (Working directory) - kurioje saugome realūs projekto failai, kuriais yra dirbama
* **Režisavimo sritis** (Staging area) – failas, kuriame saugoma informacija apie pakeitimus, kurie bus įtraukti į kitą commitą. Režisavimo sritis taip pat gali būti vadinama indeksu (index)
* **Git sritis** (Git directory) – svarbiausia Git dalis, kurioje yra saugomi momentiniai atvaizdai bei visa meta informacija apie projektą ir įvykdytus pakeitimus. Viena pagrindinių git srities dalių yra galva (head), kuri rodo į paskutinį atliktą commitą. Ši sritis yra nukopijuojama kai Git repozitorija yra klonuojama iš kito kompiuterio

Git repozitorijoje esantys failai gali būti vienos iš šių būsenų:

* **Commited** - reiškia, kad failo pakeitimas yra išsaugotas Git srityje
* **Modified** - reiškia, jog failo turinys yra pakeistas, bet pakeitimas dar nėra išsaugotas Git srityje
* **Staged** - reiškia, jog failas yra pažymėtas ir jo pakeitimai bus išsaugoti kito commito metu

Standartinė darbo Git eiga atrodo taip:

* Darbinėje srityje esantys failai yra modifikuojami
* Pasirenkami modifikuoti failai, kurie turėtų būti įtraukti į sekantį commitą. Informacija apie įvykdytus pakeitimus yra perduodama į režisavimo sritį
* Įvykdomas commitas, kurio metu yra sukuriamas naujas momentinis atvaizdas su pakeitimais, esančiais režisavimo srityje. Commitas su nuoroda į sukurtą atvaizdą yra išsaugomas Git srityje, o galva yra pastumiama, kad rodytų į paskutinį commitą

#### Atšakos (Branches)

Įvykdžius commitą, Git išsaugo commito objektą, rodantį į:

* Naujai sukurtą momentinį atvaizdą su pakeitimais iš režisavimo srities
* Commitus įvykusius anksčiau: vieną tėvinį commitą, jei tai buvo paprastas commitas, kelis tėvinius commitus, jei commitas buvo kelių atšakų apjungimas, arba nulį tėvinių, jei tai yra pirmas commitas

Atšaka yra nuoroda į commitą. Kadangi commitai turi nuorodas į ankstesnius commitus, atšaka tarsi sudaro grandinę su projekto pakeitimų istorija. Sukūrus naują atšaką, ji rodo į tą patį commitą, kaip ir originali atšaka, tačiau nauji vienoje atšakoje sukurti commitai nebus matomi kitoje atšakoje. Darbo eigoje abiejų atšakų turinys išsiskirs ir abi atšakos turės skirtingas įvykusių commitų grandines. Esant poreikiui, skirtingos atšakos gali būti sujungtos ir commitai įvykę skirtingose atšakose vėl sugrąžinami į bendrą grandinę.

Atšakos leidžia programuotojams dirbti ties skirtingomis funkcijomis, izoliuotiems vienas nuo kito. Komandos nariams baigus darbą, jie gali sujungti skirtingus commitus iš įvairių atšakų. **Atšakų vizualizacija**:

Atšaka feature1 su keliais komitais:

"feature1"

|

commit1-commit2

Iš pradinės atšakos sukuriama nauja atšaka pavadinimu feature2 ir padaromi keli nauji komitai:

"feature1"

|

commit1-commit2-commit5

\

commit3-commit4

|

"feature2"

Pakeitimai iš atšakos feature2 yra pridedami prie feature1 atšakos:

"feature1"

|

commit1-commit2-commit5--------commit6

\ /

commit3-commit4

|

"feature2"

Sukūrus naują repozitoriją, atšaka pavadinimu master sukuriama pagal nutylėjimą. Galva taip pat rodo į dabartinę atšaką. Pakeitus atšaką, galvos rodyklė pastumiama į nurodytos atšakos rodyklę. Keičiant atšakas galima matyti commitus, egzistuojančius tik toje atšakoje.

### Git komandos

git init

Darbinėje direktorijoje sukuria tuščią git repozitoriją

git clone /path/to/repository

Klonuoja egzistuojančią repozitoriją iš nurodyto kelio į direktoriją. Ji gali būti lokali arba nuotoliniame serveryje

git add <failo pavadinimas>

git add \* (visiems modifikuotiems failams)

Prideda pasirinktus failus į režisavimo sritį

git commit -m "žinutė"

Padaromas naujas projekto momentinis atvaizdas su pakeitimais, esančiais režisavimo srityje. Sukuriamas commito objektas rodantis į minėtą atvaizdą. Commitas pridedamas prie galvos lokalioje git srityje, tačiau pakeitimai dar nėra nusiunčiami į nuotolini serverį.

git status

Parodomi visi modifikuoti darbinės srities failai, bei failai esantys režisavimo srityje

git push origin <atšakos pavadinimas centrinėje repozitorijoje>

Dabartinės atšakos pakeitimai, išsaugoti lokalioje repozitorijoje, perduodami į centrinę repozitoriją, kur jie tampa matomi kitiems komandos nariams

git pull

Atsiunčiami dabartinės atšakos pakeitimai, išsaugoti nuotolinėje repozitorijoje

git checkout <atšakos pavadinimas>

Pakeičia dabartinę atšaką į nurodytą atšaką

git checkout -b <naujos atšakos pavadinimas>

Sukuria naują atšaką nurodytu pavadinimu

git branch -d <atšakos pavadinimas>

Ištrina nurodytą atšaką

git merge <atšakos pavadinimas>

Prijungia nurodytos atšakos pakeitimus prie dabartinės atšakos

git log

Parodo commitų istoriją, kurioje matoma kiekvieno commito id numeris, žinutė, bei autorius

git reset <failo pavadinimas>

Pašalina failą iš režisavimo srities, tačiau atlikti pakeitimai išlieka

git checkout -- <failo pavadinimas>

Pašalina modifikuoto failo pakeitimus

git reset -–soft <comito id>

Grąžina galvą į nurodytą commitą, tačiau visi pakeitimai padaryti tolesniuose commituose išlieka režisavimo srityje

git reset --mixed <commito id>

arba

git reset <commito id>

Grąžina galvą į nurodytą commitą, tačiau visi pakeitimai padaryti tolesniuose commituose išlieka kaip modifikuoti failai darbinėje srityje

git reset -–hard <commito id>

Grąžina projekto stadiją į nurodytą commitą ir ištrina veliau padarytus pakeitimus

### Užduotys:

* [Bazinės Git užduotys](https://github.com/tadus21/code-academy-java-kursai/blob/master/vsc-basics-with-git/exercises/exercises.md)

### Tolesniam skaitymui

* <https://services.github.com/on-demand/downloads/github-git-cheat-sheet/>
* <http://rogerdudler.github.io/git-guide/>

### Šaltiniai

* <https://guides.github.com/introduction/git-handbook/#version-control>
* <https://git-scm.com/book/en/v2>
* <https://insights.stackoverflow.com/survey/2018>

# **Sąsajos ir paveldėjimai**

### Paveldėjimai

Paveldėjimas labai svarbi Java kalbos dalis. Java kalboje klasės gali būti kilusios iš kitų klasių, taip paveldint jų laukus ir metodus. Klasė, kilusi iš kitos klasės, yra vadinama sub klase arba vaikine klase. Klasė, iš kurios kyla kitos klasės, vadinama super klase arba tėvine klase. Java klasės gali turėti tik vieną super klasę. Jei klasė neturi nurodytos super klasės, ji paveldi klasę Object pagal nutylėjimą.

Vaikinė klasė paveldi narius (laukus, metodus, vidines klases) iš tėvinės klasės. Vaikinės klasės konstruktoriai privalo kviesti superklasės konstruktorių, nes jie nėra paveldimi. Kiekvieno konstruktoriaus pirma eilutė turi kviesti tėvinės klasės konstruktorių. Tačiau, jei kviečiamas standartinis, parametrų neturintis, konstruktorius (super();), to daryti nebutina, tai įvyks automatiškai kompiliavimo metu.

Dėl savybės perduoti narius vaikinei klasei, paveldimumas taikomas siekiant naudoti laukus arba metodus, egzistuojančius kitose klasėse, neperrašant jų pačiam. Vaikinės klasės paveldi visus public ir protected tėvinės klasės narius. Taip pat, jei vaikinė klasė yra tame pačiame kataloge (package) kaip ir tėvinė klasė, ji paveldi visus package-private narius. Paveldėti laukai, metodai, vidinės klasės gali būti pasiekiami tiesiogiai vaikinėje klasėje, taip pat kaip įprasti tos klasės nariai.

Vaikinės klasės gali turėti naują, tik joms būdingą funkcionalumą, tačiau jos taip pat turi tėvinės klasės savybes. Tai reiškia, kad vaikinė klasė gali būti panaudota ten, kur reikalinga jos tėvinė klasė. Pavyzdžiui, klasė TemperatureSensor kuri paveldi klasę Sensor gali būti priskirta TemperatureSensor, Sensor, Object tipo kintamiesiems. Visų šių teiginių sintaksė yra teisinga:

TemperatureSensor temperatureSensor = new TemperatureSensor();

Sensor sensor = temperatureSensor;

Object object = temperatureSensor;

Klasės tipo keitimas į jos tėvinių objektų tipą vadinamas upcast‘inimu. Upcast‘inus objektą į tėvinės klasės tipą, vaikinei klasei būdingi metodai tampa nematomi. Upcast‘inimas vyksta automatiškai.

Tėvinius objektus galima cast‘inti į vaikinės klasės tipus, tai vadinama downcasting. Downcast‘inimas nevyksta automatiškai, tam reikia specifiškai nurodyti cast‘inimo operatorių:

TemperatureSensor temperatureSensor = new TemperatureSensor();

Sensor sensor = temperatureSensor;

TemperatureSensor otherTemperatureSensor = (TemperatureSensor) sensor;

Paprastai Downcast’inimas - tiesiog nurodymas kompiliatoriui, kad tu žinai, koks iš tiesų yra objekto tipas veikimo (runtime) metu. Downcast‘inti objektus reikėtų atsargiai, nes „neatspėjus“ objekto tipo programa pateikia ClassCastException.

Sensor sensor = new HumiditySensor();

TemperatureSensor incorrectSensor = (TemperatureSensor) sensor; // pateikiamas ClassCastException

Išvengti ClassCastException padeda operatorius instanceOf, naudojamas nusakyti objekto tipui:

Sensor sensor = new HumiditySensor()

System.out.println(sensor instanceof HumiditySensor);

System.out.println(sensor instanceof TemperatureSensor);

Tokio kodo rezultatas būtų

true

false

Java leidžia downcast‘inti objektus tik į tokius tipus, kurie paveldi tėvinę klasę. Kodas pavyzdyje nesikompiliuoja, nes klasė Sensor nėra klasės String vaikinė klasė:

String text = “simple text”;

Sensor sensor = (Sensor) text; // Inconvertible types; cannot cast 'java.lang.String' to 'Sensor'

Vaikinės klasės gali perrašyti ne **final**tėvinės klasės metodus. Tai leidžia pritaikyti klasės veikimą pagal savo poreikius. Situacija, kai objekto tipas nulemia metodo veikimą, vadinama polimorfizmu. Metodų perrašymo pavyzdys:

public abstract class Sensor {

public abstract String purpose();

}

public class TemperatureSensor extends Sensor {

@Override

public String purpose() {

return "measures temperature";

}

}

public class HumiditySensor extends Sensor {

@Override

public String purpose() {

return "measures humidity";

}

}

public static void main(String[] args) {

Sensor temperatureSensor = new TemperatureSensor();

Sensor humiditySensor = new HumiditySensor();

System.out.println(temperatureSensor.purpose());

System.out.println(humiditySensor.purpose());

}

Kiekviena Sensor sub klasė išspausdins savo paskirtį:

measures temperature

measures humidity

Metodų perrašymas ypatingas tuo, kad visada kviečiamas perrašytas metodas - net upcastinus klasės tipą į tėvinį, bus kviečiamas ne tėvinis metodas, o perrašyta metodo versija.

public class HumiditySensor extends Sensor {

@Override

public String purpose() {

return "measures humidity";

}

}

public class AdvancedHumiditySensor extends HumiditySensor {

@Override

public String purpose() {

return "precisely measures humidity";

}

}

public static void main(String[] args) {

AdvancedHumiditySensor advancedSensor = new AdvancedHumiditySensor();

HumiditySensor sensor = advancedSensor;

System.out.println(advancedSensor.purpose());

System.out.println(sensor.purpose());

}

Abiem atvejais programa išspausdins tekstą „precisely measures humidity“.

Visgi, vaikinėje klasėje galima pasiekti net ir perrašytus tėvinius metodus pasitelkiant raktažodį **super.** Pavyzdžiui AdvancedHumiditySensor klasė savo viduje galėtų iškviesti originalų purpose() metodą tokia sintakse: super.purpose().

public class AdvancedHumiditySensor extends HumiditySensor {

@Override

public String purpose() {

return "precisely measures humidity";

}

public String parentPurpose() {

return super.purpose();

}

}

public static void main(String[] args) {

AdvancedHumiditySensor advancedSensor = new AdvancedHumiditySensor();

System.out.println(advancedSensor.purpose());

System.out.println(advancedSensor.parentPurpose());

}

Aukščiau matomo kodo rezultatas būtų:

precisely measures humidity

measures humidity

Norint išvengti paveldimumo, gali būti naudojamas raktažodis final. Klasės su raktažodžiu final negali būti paveldėtos, o final metodai negali būti perrašyti.

### Abstrakčios klasės

Abstrakti klasė – tai klasė, turinti raktažodį abstract. Abstrakčios klasės negali būti inicializuotos, tačiau jos gali būti paveldėtos. Jos turi visus narius, būdingus paprastoms klasėmis, bei abstrakčius metodus. Abstraktūs metodai neturi įgyvendinimo, jie atrodo taip: abstract void printStatus();. Jei klasė turi abstrakčius metodus, ji pati turi būti įvardyta kaip abstrakti. Paveldėjus abstrakčią klasę, vaikinė klasė privalo įgyvendinti visus abstrakčius tėvinės klasės metodus.

Abstrakčios klasės turėtų būti taikomos naudojant tą patį kodą keliose, glaustai susijusiose klasėse. Pavyzdžiui, siuntų tarnybos sistema apdoroja įvairias siuntas. Tai gali būti paprasta siunta, skubi siunta, didelio dydžio siunta. Visos šios siuntos turi bendrus parametrus – gavėjo adresą ir išsiuntimo datą. Taip pat apie siuntas turėtų būti žinoma siuntos gavimo data, siuntimo būdas. Ši informacija būdinga kiekvienai siuntai, tačiau kiekvienam tipui ji bus skirtinga. Žinant šią informaciją, galima sukurti abstrakčią klasę, turinčią visas bendras siuntų savybes. Klasė atrodytų taip:

public abstract class Shipment {

private final String address;

private final LocalDateTime shippingDate;

public Shipment(String address, LocalDateTime shippingDate) {

this.address = address;

this.shippingDate = shippingDate;

}

public abstract LocalDateTime expectedDelivery();

public abstract String shipmentMethod();

public String getAddress() {

return address;

}

public LocalDateTime getShippingDate() {

return shippingDate;

}

}

Metodai expectedDelivery ir shipmentMethod yra abstraktūs, nes jie yra skirtingi kiekvienam siuntos tipui. Standartinės siuntos vežamos autobusiuke, pristatymo trukmė yra dešimt dienų. Klasė, vaizduojanti standartinę siuntą, atrodo taip:

public class StandardShipment extends Shipment {

public StandardShipment(String address, LocalDateTime shippingDate) {

super(address, shippingDate);

}

@Override

public LocalDateTime expectedDeliver() {

return getShippingDate().plusDays(10);

}

@Override

public String shipmentMethod() {

return "by van";

}

}

Abstraktūs metodai įgyvendinami pagal standartinės siuntos suvaržymus. Klasės, vaizduojančios didelius bei skubius siuntinius, taip pat įgyvendinamos pagal joms taikomus reikalavimus

public class LargeShipment extends Shipment {

public LargeShipment(String address, LocalDateTime shippingDate) {

super(address, shippingDate);

}

@Override

public LocalDateTime expectedDelivery() {

return getShippingDate().plusDays(20);

}

@Override

public String shipmentMethod() {

return "by train";

}

}

public class UrgentShipment extends Shipment {

public UrgentShipment(String address, LocalDateTime shippingDate) {

super(address, shippingDate);

}

@Override

public LocalDateTime expectedDelivery() {

return getShippingDate().plusDays(3);

}

@Override

public String shipmentMethod() {

return "by plane";

}

}

Skubi siunta siunčiama lėktuvu ir atkeliauja per tris dienas, o didėlė siunta siunčiama traukiniu. Tai užtrunka ilgiau nei įprastai – dvidešimt dienų.

### Sąsajos

Kompiuterių inžinerijoje dažni atvejai, kai atskiros programuotojų grupės turi sudaryti „kontraktus“, apibrėžiančius, kaip jų programinė įranga bendrauja tarpusavyje. Kiekviena grupė turėtų gebėti kurti savo programinį kodą, nežinodami, kaip parašytas kitų grupių kodas. Sąsajos (interfaces) Java programavimo kalboje ir yra tokie kontraktai.

Įsivaizduokite įmonę, kuriančią dronus. Viena komanda atsakinga už programinę įrangą, skirtą bendrauti su drono technine įranga (hardware). Ši dalis valdo droną, kompiuterinės komandos kontroliuoja drono „geležį“, liepia jam kilti, leistis, suktis į kairę ir t.t. Kita komanda kurią programinę įrangą, kuri priima komandas iš nuotolinio pulto ir perduoda jas anksčiau minėtai sistemai, kad dronas keistų savo skridimo trajektoriją pagal gautas komandas iš pulto. Pirmoji komanda pateikia sąsają OperateDrone skirtą kreiptis į jų programinę įrangą. Sąsaja atrodo taip:

interface OperateDrone {

void up();

void down();

void left(int angle);

void right(int angle);

void accelerate();

void decelerate();

}

Dėl šios sąsajos antroji komanda gali valdyti droną nesirūpindama, kaip tai vyksta žemesniame lygmenyje. Sąsajos įgyvendinimas jai neaktualus. Įsivaizduokite, kad įmonė turi kelis skirtingus dronus – „Drone 2x Lite “ ir „Drone 8x Professional“. Jie turi skirtingą techninę įrangą, kuri programavimo lygmenyje turi būti valdoma skirtingai, dėl to reikalingos skirtingos programinės įrangos jiems valdyti. Pirmoji komanda pateikia du OperateDrone sąsajos įgyvendinimus: Operate2xLiteDrone ir Operate8xProfessionalDrone, skirtus valdyti skirtingų tipų dronus. Antrajai komandai šis suvaržymas nėra aktualus. Jos parašytas komandinis kodas naudoja OperateDrone sąsają ir sąsajos įgyvendinimas jai nėra svarbus.

Šios sistemos pavyzdį galite pamatyti [čia](https://github.com/tadus21/code-academy-java-kursai/blob/master/inheritance/examples/src)

Sąsajos gali turėti abstrakčius metodus bei konstantas (kintamuosius su public, static, final raktažodžiais). Taip pat nuo 8 Javos versijos, sąsajos gali turėti statinius metodus, bei default metodus – metodus su numatytu įgyvendinimu.

Sąsajos ypatingos tuo, kad klasės gali įgyvendinti neribotą kiekį sąsajų ir apribojimas, kad klasė gali paveldėti tik vieną klasę, sąsajoms negalioja. Abstrakčios klasės taip pat gali įgyvendinti sąsajas, tačiau sąsajų metodai gali likti abstraktūs. Tokiu atveju sąsajų metodus turės įgyvendinti klasės, paveldinčios minėtą abstrakčią klasę.

### Abstrakti klasė ar sąsaja? Ką pasirinkti?

Naudokite abstrakčias klases kai:

* norite dalintis kodu tarp kelių, glaustai susijusių klasių
* jums reikia laukų, kurie nebūtų konstantos

Naudokite sąsajas kai:

* tikitės, kad klasės, įgyvendinančios jūsų sąsają, nebus glaustai susijusios
* norite užtikrinti kontraktą, bet jo įgyvendinimo detalės nėra svarbios
* norite pasinaudoti galimybe paveldėti kelias sąsajas

Tvarkingą kodą užtikriną abstrakcijos, nepriklausomi, vienas nuo kito atskirti komponentai/klasės/metodai. Programuojant reikėtų naudoti pačią žemiausią tėvinę klasę iš paveldėjimo grandinės, teikiančią norimą funkcionalumą. Nėra reikalo naudoti vaikinį objekto tipą, jei visi reikalingi metodai būdingi tėviniam duomenų tipui. Pavyzdžiui, nekurkite TemperatureSensor tipo objektų, jei jums tereikia Sensor tipo metodų.

Taip pat, sąsajas reikėtų naudoti tipams nurodyti, o jų įgyvendinimas turėtų būti kiek galima paslėptas. Tvarkingi metodai grąžina sąsajas. Klientui nereikia žinoti, ar jis gauna ArrayList ar LinkedList įgyvendinimą, jam svarbu kad gautas objektas tenkintų List sąsajos kontraktą. Besilaikant tokių praktikų, lengviau išlaikyti tvarkingą ir nuo skirtingų komponentų nepriklausomą kodą.

### Užduotys

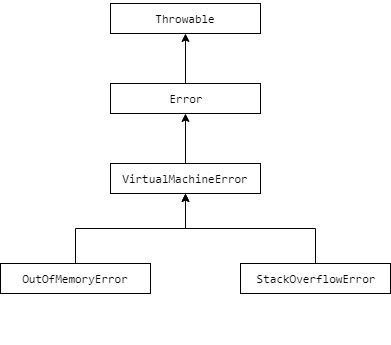
* [Polimorfizmo uzduotis](https://github.com/tadus21/code-academy-java-kursai/blob/master/inheritance/exercises/polymorhism-exercise.md)

### Šaltiniai:

* <https://docs.oracle.com/javase/tutorial/java/IandI/index.html>

# Išimtys (exceptions)

### Teorija

[](https://github.com/tadus21/code-academy-java-kursai/blob/master/exceptions/img/errors-hierarchy.png)

Klaidų klasės OutOfMemoryError ir StackOverflowError yra VirtualMachineError vaikai.

Pavyzdys:

import java.util.ArrayList;

import java.util.List;

public class Main {

public static void main(String[] args) {

try {

notSafeMethod();

} catch (OutOfMemoryError e) {

System.out.println("Out of memory");

} catch (VirtualMachineError error) {

System.out.println("VM error");

}

}

public static void notSafeMethod() {

// kodas, kuris iššauks OutOfMemoryError:

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

while (true) {

list.add(new String("Hello!"));

}

// kodas, kuris iššauks StackOverflowError:

//notSafeMethod();

}

}

Ši kodo dalis

try {

notSafeMethod();

} catch (OutOfMemoryError e) {

System.out.println("Out of memory");

} catch (VirtualMachineError error) {

System.out.println("VM error");

}

reiškia, kad programa kvies galimai nesaugų metodą notSafeMethod ir įvykus klaidai OutOfMemoryError atspausdins Out of memory. Jei įvyks VirtualMachineError arba jos vaikinės klasės tipo klaida, pavyzdžiui StackOverflowError, tada atspausdins VM error. Blokuose catch klaidų ir išimčių klasės turi būti išvardijamos iš eilės - pradedant nuo vaikinių klasių viršuje ir baigiant tėvine klase apačioje.

Aukščiau aprašytas notSafeMethod metodas iššauks OutOfMemoryError klaidą. Metode main šio tipo klaidos tikimasi catch (OutOfMemoryError e), todėl bus atspausdinta Out of memory.

Galime galime sumažinti Java programa išskiriamą atmintį naudjant parametrą -Xmx2M. Užrašas 2M reiškia, kad programa turės 2 MB atminties.

Jei metodą notSafeMethod pakeisime į tokį:

public static void notSafeMethod() {

// kodas, kuris iššauks OutOfMemoryError:

//List<String> list = new ArrayList<>();

//while (true) {

// list.add(new String("Hello!"));

//}

// kodas, kuris iššauks StackOverflowError:

notSafeMethod();

}

metodas rekursiškai begalę kartų kreipsis pats į save ir tai iššauks StackOverflowError klaidą. Įvykus klaidai main metode programa iš eilės patikrins, ar tai ne OutOfMemoryError tipo klaida, bet StackOverflowError neturi jokio ryšio su OutOfMemoryError. Toliau programa eina į kitą catch bloką ir tikrina, ar tai ne VirtualMachineError tipo klaida. Kaip jau žinome StackOverflowError extends VirtualMachineError todėl šis catch blokas aproros klaidą ir atspausdins pranešimą VM error.

Jei try-catch bloką papildysime taip:

try {

notSafeMethod();

} catch (OutOfMemoryError e) {

System.out.println("Out of memory");

} catch (StackOverflowError e) {

System.out.println("Stack overflow");

} catch (VirtualMachineError error) {

System.out.println("VM error");

}

tada programa atspausdins Stack overflow, kai įvyks StackOverflowError klaida.

Jei try-catch bloką supaprastinsime pašalindami dalį catch blokų:

try {

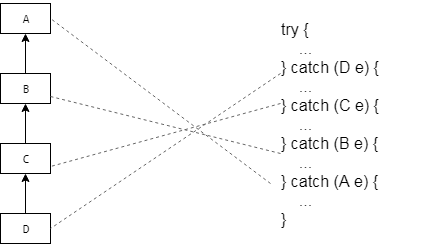
notSafeMethod();

} catch (VirtualMachineError error) {

System.out.println("VM error");

}

Tada įvykus OutOfMemoryError arba StackOverflowError arba kitai VirtualMachineError tipo klaidai, programa spausdins VM error.

[](https://github.com/tadus21/code-academy-java-kursai/blob/master/exceptions/img/abcd-exceptions.png)

### Užduotys

* [Užduotys](https://github.com/tadus21/code-academy-java-kursai/blob/master/exceptions/exercises/readme.md)

### Naudingos nuorodos

* [Java išimčių klasių hierarchija](https://airbrake.io/blog/java-exception-handling/the-java-exception-class-hierarchy)

# **Darbas su failais**

### Kas yra kelias (Path)?

Failų sistemos saugo ir organizuoja failus taip, kad jie būtų lengvai pasiekiami. Dauguma modernių failų sistemų naudoja hierarchinę struktūrą failams saugoti. Pačiame viršuje yra pagrindinis mazgas (root node). Žemiau jo – failai ir direktorijos. Kiekvienoje direktorijoje gali būti failai arba kitos direktorijos.

Paprastas direktorijų medžio pavyzdys:

+--------------------+

| C:\ (Windows root) |

| / (Linux root) |

+-----------+--------+

|

+---v--+

+-----+ home +------+

| +------+ |

| |

+-----v-----+ +------v------+

| documents | |Program Files|

+-----+-----+ +------+------+

| |

+-----v-----+ +----v----+

|dokumentas1| | Java |

+-----------+ +---------+

Failas yra identifikuojamas pagal kelią nuo tėvinio medžio. Pavyzdžiui, failas dokumentas.txt iš ankstesnio pavyzdžio Windows sistemoje būtų aprašytas taip: C:\home\documents\dokumentas1, o Linux sistemoje jis būtų aprašytas taip: /home/documents/dokumentas1

Ženklas, skirtas atskirti direktorijų pavadinimus, vadinamas delimiter, kiekvienoje sistemoje jis yra kitoks. Windows sistemoje tai \, o Linux sistemoje /.

Kelias gali būti pilnas (absolute) arba reliatyvus (relative). Pilnas kelias prasideda nuo pagrindinio mazgo iki failo. Pavyzdžiui C:\home\documents\dokumentas1 yra pilnas kelias. Pilnas kelias turi visą reikalingą informaciją, reikalingą pasiekti failui. Norint pasiekti failą, reliatyvus kelias turi būti sujungtas su kitu keliu. Be papildomos informacijos programa negali pasiekti documents\dokumentas1 failo failų sistemoje.

### Path klasė

Path klasė yra failų sistemos kelio reprezentacija Java programavimo kalboje. Path objektas yra naudojamas rasti bei manipuliuoti failais. Failas nurodytas Path objekte gali neegzistuoti. Norint sukurti naują Path objektą naudojamas statinis Paths klasės metodas get. Pavyzdžiui Paths.get("C:\home\documents\dokumentas");.

Path objekto metodas resolve naudojamas sulipdyti du kelius. Pavyzdžiui kodas:

Path path = Paths.get("/home");

Path anotherPath = Paths.get("documents");

Path newPath = path.resolve(anotherPath);

System.out.println(newPath);

išspausdins /home/documents.

### Files klasė

Files klasė naudojama dirbant su failais. Ši klasė turi rinkinį statinių metodų, leidžiančių manipuliuoti failais.

#### Files klasės metodai

boolean exists(Path path)

Patikrina, ar nurodytas failas arba direktorija egzistuoja

boolean isRegularFile(Path path)

Patikrina, ar nurodytas kelias rodo į paprastą failą

boolean isDirectory(Path path)

Patikrina, ar nurodytas kelias rodo į direktoriją

Path createDirectory(Path dir)

Sukuria direktoriją nurodytame kelyje

Path createFile(Path path)

Sukuria failą nurodytame kelyje

long size(Path path)

Grąžina failo dydį

void delete(Path path)

Pašalina failą/direktoriją nurodytame kelyje

Path copy(Path source, Path target)

Nukopijuoja failą į nurodytą direktoriją

Path move(Path source, Path target)

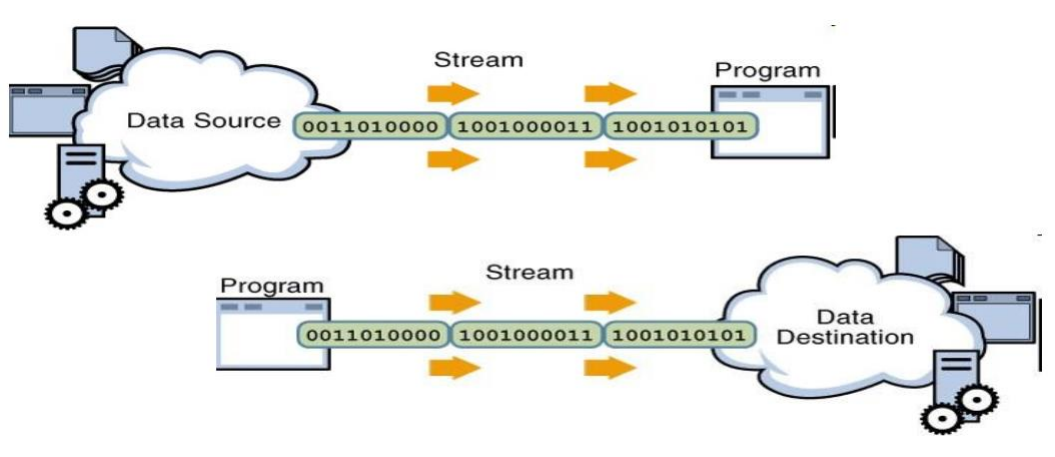
Perkelia failą į nurodytą direktoriją

List<String> readAllLines(Path path)

Grąžina sąrašą su failų eilutėmis

### Įvesties/Išvesties srautai

Java srautas - duomenų (baitų, simbolių) seka. Srauto klasės leidžia operuoti skaitymo / rašymo operacijomis.

[](https://github.com/tadus21/code-academy-java-kursai/blob/master/files/stream.png)

Java kalboje srautą pavaizduoja InputStream, OutputStream sąsajos. Srauto sąsajos turi daug skirtingų įgyvendinimų skirtingiems srautų tipams. Pagrindinės srautų klasės yra java.io pakete.

Galimų srautų pavyzdžiai:

* Byte stream
* Character stream
* Buffered stream

Srautai yra efektyvūs, nes jie leidžia skaityti/apdoroti duomenis mažomis dalimis, neįkeliant viso turinio į atmintį.

Dirbant su dideliais failais paprastai naudojami Files klasės metodai

BufferedReader newBufferedReader(Path path)

BufferedWriter newBufferedWriter(Path path)

grąžinantys Buffered tipo srautus įvesties ir išvesties.

Failo eilutės nuskaitymas iš srauto atrodo taip:

try (BufferedReader reader = Files.newBufferedReader(file)) {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

}

Failo eilutės įrašymas atrodo taip:

try (BufferedWriter bufferedWriter = Files.newBufferedWriter(file)) {

bufferedWriter.write("First line");

bufferedWriter.newLine();

bufferedWriter.write("Second line");

bufferedWriter.newLine();

bufferedWriter.write("Third line");

}

Darbo su failais programėles pavyzdį galite pamatyti [čia](https://github.com/tadus21/code-academy-java-kursai/blob/master/files/examples/files-example/src)

### Užduotys

* [Užduotis darbui su failais](https://github.com/tadus21/code-academy-java-kursai/blob/master/files/exercises/files-exercise.md)

### Nuorodos

* <https://docs.oracle.com/javase/tutorial/essential/io/path.html>

***Generic* klasės**

***Generic* klasės sintaksė**

class KlasesVardas<T1, T2, ..., Tn> { /\* ... \*/ }

T1, T2, ..., Tn - tai yra tipų kintamieji (klasių vardai). Tipų kintamieji gali būti bet kokie neprimityvieji tipai: bet kokios klasės tipas, bet kokio interfeiso tipas, bet kokio masyvo tipas ar bet koks kitas tipas.

**Tipo parametro vardo konvencija**

Pagal Java konvenciją sutarta, kad *generic* tipo parametro vardas turi būti viena didžioji. Dažniausiai naudojami vardai:

* E - Element
* K - Key
* N - Number
* T - Type
* V - Value

**Ne *generic* klasės pavyzdys**

Pavyzdys paprastos (ne *generic*) Box klasės, kuri turi bet kokio tipo klasės kintamąjį. Klasė Box turi du metodus - set, kuris nustato viduje esantį objektą ir get, kuris grąžina viduje esantį objektą.

public class Box {

private Object object;

public void set(Object object) { this.object = object; }

public Object get() { return object; }

}

Tokiai klasei galime paduoti bet kokio neprimityvaus tipo objektus. Pavyzdžiui į klasę Box įdėsime Integer tipo objektą. Vėliau iš Box klasės pasiėmę objektą tikėsimės, kad tai yra skaičius. Viskas gerai. Bet jei kas nors į Box klasą įdės pavyzdžiui Stringtipo objektą, o mes pasiėmę tą objektą to nežinosime ir tikėsimės, kad tai yra skaičius, bandysime objektą *cast*'inti į Integer ir gausime RuntimeException. Situaciją vaizduojantis pavyzdys:

public class BoxTest {

public static void main(String[] args) {

Integer number;

Box box = new Box();

box.set(10);

number = (Integer)box.get(); // Viskas gerai, nes Integer

box.set("Java");

number = (Integer)box.get(); // RuntimeException

}

}

***Generic* klasės pavyzdys**

Pakeitus klasę Box į *generic*, ji atrodo taip:

public class Box<T> {

private T t;

public void set(T t) { this.t = t; }

public T get() { return t; }

}

**Objekto sukūrimas**

Box<Integer> integerBox;

Taip yra apibrėžiamas integerBox kintamasis. Kampiniuose skliaustuose <> yra nurodoma, kokio tipo objektą saugos klasė Box.

Naujo objekto sukūrimas:

Box<Integer> integerBox = new Box<Integer>();

Nuo Java 7-os versijos dešinėje galima ir netgi rekomenduojama (dėl kodo supaprastinimo) nerašyti tipo. Galime palikti tuščius kampinius skliaustus <>.

Box<Integer> integerBox = new Box<>();

Anksčiau sukurtos *generic* klasės Box naudojimo pavyzdys

public class GenericBoxTest {

public static void main(String[] args) {

Integer number;

Box<Integer> integerBox = new Box<>();

integerBox.set(10);

number = integerBox.get(); // privalumas: nereikia naudoti cast operacijos

integerBox.set("Java"); // kompiliavimo klaida

number = integerBox.get();

}

}

Naudodami *generic* tipus kode padarytą klaidą pastebės kompiliatorius. Taip pat didelis privalumas, kad norint panaudoti Box klasės viduje esantį objektą mums nereikia atlikti *cast* operacijos.

***Key* ir *value* poros implementacijos naudojant *generic* pavyzdys**

public interface Pair<K, V> {

public K getKey();

public V getValue();

}

public class OrderedPair<K, V> implements Pair<K, V> {

private K key;

private V value;

public OrderedPair(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() { return key; }

public V getValue() { return value; }

}

Klasės OrderedPair objekto sukūrimas:

Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Eight", 8);

Pair<String, String> p2 = new OrderedPair<String, String>("hello", "world");

Eilutė new OrderedPair<String, Integer> nurodo, kad OrderedPair klasės K tipo kintamasis tampa String, V tampa Integer, todėl galime kviesti OrderedPair konstruktoriu jam perdodant atitinkamas reikšmes new OrderedPair<String, Integer>("Eight", 8). Kaip buvo minėta anksčiau, dešinėje pusėje nėra būtina rašyti *generic* tipų. Taip kodą supaprastiname iki

Pair<String, Integer> p1 = new OrderedPair<>("Eight", 8);

Taip pat kaip tipą galima nurodyti tokį, kuris taip turi *generic* parametrą. Pavyzdys:

Pair<String, Box<Integer>> p = new OrderedPair<>("Primes", new Box<Integer>(...));

***Raw* (neapdoroti) tipai**

*Raw* tipu vadinamas *generic* klasės arba interfeiso vardas be tipo parametro. Pavyzdžiui turime tą pačią klasę Box:

class Box<T> {

private T object;

public void set(T object) { this.object = object; }

public T get() { return object; }

}

Jei kuriant klasės Box objektą nurodome koks tipas atitinka T generic parametrą, tada turime parametrizuotą Box<T> tipą:

Box<Integer> intBox = new Box<>();

Bet jei kuriant tos pačios klasės objektą nenurodome *generic* tipo, tada turime Box<T> *raw* tipą:

Box rawBox = new Box();

Kaip buvo minėta anksčiau *generics* atsirado nuo Java 5-os versijos, todėl *raw* tipai yra daugiau skirti kodui parašytam naudojant ankstesnes Java versijas palaikyti.

Tačiau rašant naują kodą, reikia vengti naudoti *raw* tipus. Pavyzdys dempnstruojantis galimą problmą:

public class RawBoxTest {

public static void main(String[] args) {

Box rawBox = new Box(); // raw tipas

rawBox.set("java"); // kompiliatorius: warning: unchecked invocation to set(T)

Box<Integer> intBox = rawBox; // keičiame į parametrizuotą tipą.

// kompiliatorius: warning: unchecked conversion

intBox.get().intValue(); // gauname RuntimeException

}

}

Parašius tokį kodą kompiliatorius perspėja, kad kompiliavimo metu negali nuspręsti tipo ir gali kilti problemų. Jei tokius perspėjimus ignoruojame, vykdymo metu gauname RuntimeException klaidą. Norint ignoruoti ir paslėpti tokius kompiliatoriaus perspėjimus galime naudoti anotaciją @SuppressWarnings("unchecked"). Pavyzdys:

public class RawBoxTest {

@SuppressWarnings("unchecked")

public static void main(String[] args) {

Box rawBox = new Box();

rawBox.set("java");

Box<Integer> intBox = rawBox;

intBox.get().intValue(); // gauname RuntimeException

}

}

Tačiau šiuo atveju tik paslepiamas perspėjimas, o vykdymo klaidos neišvengiama. Todėl patariama nenaudoti raw tipų.

***Generic* metodai**

Java *generic* taip pat gali būti nadojami metodų tipams nurodyti. Sintaksė tokia pati kaip ir naudojant *generic* klasių tipams nurodyti, tačiau skirias tipo matomumo ribos - tipas matomas tik to metodo viduje.

***Generic* metodo pavyzdys**

Turime klasė Pair su K ir V tipų patametrais.

public class Pair<K, V> {

private K key;

private V value;

public Pair(K key, V value) {

this.key = key;

this.value = value;

}

public void setKey(K key) { this.key = key; }

public void setValue(V value) { this.value = value; }

public K getKey() { return key; }

public V getValue() { return value; }

}

Ir turime Util klasę, kurios metodas compare(...) yra generic.

public class Util {

public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {

return p1.getKey().equals(p2.getKey()) &&

p1.getValue().equals(p2.getValue());

}

}

Sukuriame du klasės Pair objektus.

Pair<Integer, String> p1 = new Pair<>(1, "apple");

Pair<Integer, String> p2 = new Pair<>(2, "pear");

*Generic* metodo iškvietimo pavyzdys:

boolean same = Util.<Integer, String>compare(p1, p2);

Šiu atveju tipų aprašyti dešinėje pusėje nėra būtina. Java kompiliatorius pats nuspręs kokie tai yra tipai. Todėl galime supaprastinti kodą:

boolean same = Util.compare(p1, p2);

***Bounded* (apriboto) tipo parametrai**

Gali būti situacijų, kai norime apriboti tipą, kuris naudojamas kaip *generic* parametras.

Pavyzdys: norime turėti klasę su *generic* parametru, kuris yra skaičius. Number yra tokių klasių kaip Integer, Double, Long ir keletos kitų skaičių tipų superklasė.

+--------+

| Number |

+---+----+

^

+-----------------------------+----------+

| | | |

+-----+---+ +----+---+ +---+--+ +--+--+

| Integer | | Double | | Long | | ... |

+---------+ +--------+ +------+ +-----+

Todėl klasėje Calculator galima nurodyti T extends Number taip pažymint, kad klasei tinka tik tokie tipai, kurie yra Number vaikai. Tai yra apriboto tipo parametrai.

public class Calculator<T extends Number> {

private T n;

private T m;

public Calculator(T n, T m) {

this.n = n;

this.m = m;

}

public Number sum() {

return n.doubleValue() + m.doubleValue();

}

}

Kuriant Calculator klasės objektą konstruktoriui galime perduoti tik skaičius. Jei bandysime perduoti kitokio tipo objektus, pavyzdžiui eilutės String tipo, gausime kompiliavimo klaidą.

Calculator calc = new Calculator(5, 8.9f);

calc.sum();

Apriboto tipo parametrus taip pat galima naudoti ir su *generic* metodais. Pavyzdys:

public class Calculator {

public <T extends Number> Number sum(T n, T m) {

return n.doubleValue() + m.doubleValue();

}

}

Calculator calc = new Calculator();

calc.sum(5, 8.9f);

calc.sum("5", "8.9f"); // kompiliavimo klaida

***Generic* ir paveldėjimas**

Situacija: turime metodą someMethod, kuriam galime perduoti Number tipo objektą.

public void someMethod(Number n) { /\* ... \*/ }

Žinome, kad tokiam metodui galime perduoti ne tik Number tipo objektus, bet ir Number subklasių (vaikinių klasių) tipų objektus:

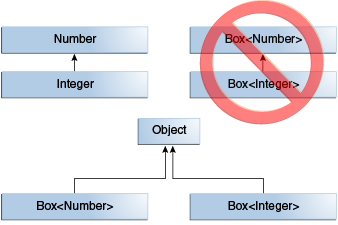
someMethod(new Integer(10));

someMethod(new Double(10.1));

Jei turime metodą someGenericMethod, kurio parametras yra *generic* tipo Box<Number>.

public void someGenericMethod(Box<Number> n) { /\* ... \*/ }

Tada tokiu atveju galime kviesti metodą perduodant Box<Number> tipo parametrą, tačiau negalima kviesti perduodant Box<Integer> tipo parametrą. Nors klasė Number yra klasės Integer superklasė, bet Box<Number> nėra Box<Integer>superklasė.

[](https://github.com/tadus21/code-academy-java-kursai/blob/master/generics/img/generics-subtypeRelationship.gif)

Pavyzdys:

public class Box<T> {

private T object;

public void set(T object) { this.object = object; }

public T get() { return object; }

}

public class BoxNumberTest {

public static void boxTest(Box<Number> n) { /\* ... \*/ }

public static void main(String[] args) {

Box<Number> boxNumber = new Box();

Box<Integer> boxInteger = new Box();

boxTest(boxNumber);

boxTest(boxInteger); // kompiliavimo klaida

}

}

***Generic* klasės ir potipiai**

public class Collection<E> { /\* ... \*/}

public class List<E> extends Collection<E> { /\* ... \*/}

public class ArrayList<E> extends List<E> { /\* ... \*/}

List<Integer> integerList = new ArrayList<>();

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

integerList = stringList; // kompiliavimo klaida: Incompatible types

Negalima List<Integer> tipo kintamajam integerList priskirti kintamojo stringList, kuris yra List<String> tipo, nes abu tipai priklauso skirtingoms klasių hierarchijoms.

+--------------------+ +---------------------+

| Collection<String> | | Collection<Integer> |

+---------^----------+ +---------^-----------+

| |

+---------+----------+ +---------+-----------+

| List<String> | | List<Integer> |

+---------^----------+ +---------^-----------+

| |

+---------+----------+ +---------+-----------+

| ArrayList<String> | | ArrayList<Integer> |

+--------------------+ +---------------------+

**Pakaitos simboliai *(wildcards)***

**Viršutinio apribojimo pakaitos simboliai**

*Generic* tipą galime apriboti iš viršaus naudojant tokią sintaksę <? extends VirsutinisTipas>. Pavyzdys: norime turėti metodą, kuris priima List tipo parametrą su elementais, kurie yra bet kokie skaičiai Numbers. Užrašas List<Numbers> yra per daug griežtas, nes tokiu atveju negalėsime metodui perduoti List<Integer> sąrašo. Šiuo atveju naudojame iraišką List<? extends Number>.

+--------+

| Object |

+---^----+

|

+-------------------------------------------------+

| | |

| +---+----+ |

| | Number | |

| +---^----+ |

| | |

| +--------------+----------------+ |

| | | |

| +----+----+ +----+---+ |

| | Integer | | Double | |

| +---------+ +--------+ |

| |

| |

| |

+-------------------------------------------------+

Number with subclasses

public static void printList(List<? extends Number> list) {

for (Number n: list) {

System.out.println(n.doubleValue());

}

}

List<Integer> integerList = Arrays.asList(1, 5, 10);

List<Double> doubleList = Arrays.asList(1.5, 5.9, 10.2);

printList(integerList);

printList(doubleList);

**Apatinio apribojimo pakaitos simboliai**

*Generic* tipą galime apriboti iš apačios naudojant tokią sintaksę <? super ApatinisTipas>. Panaudojimo pavyzdys:

+----------------------------------------+

| |

| +--------+ |

| | Object | |

| +---^----+ |

| | |

| | |

| | |

| +---+----+ |

| | Number | |

| +---^----+ |

| | |

| +--------------+----------------+

| | | |

| +----+----+ | +----+---+

| | Integer | | | Double |

| +---------+ | +--------+

| |

+----------------------------------------+

Integer with superclasses

public static void addTenNumbers(List<? super Integer> list) {

for (int i = 1; i <= 10; i++) {

list.add(i);

}

}

List<Integer> integerList = Arrays.asList(1, 5, 10);

List<Double> doubleList = Arrays.asList(1.5, 5.9, 10.2);

addTenNumbers(integerList);

addTenNumbers(doubleList); // kompiliavimo klaida.

// Double nėra Integer klasės superklasė

**Neribojantys pakaitos simboliai**

Jei metodo logika nepriklauso nuo to, pavyzdžiui kokio tipo elementai bus metodui paduodame sąraše, tada galime naudoti List<Object> list, nes Object yra visų Java klasių superklasė;

public static void printList(List<Object> list) {

for (Object elem : list)

System.out.println(elem + " ");

}

Bet tada galėsime metodui perduoti tik tokius sąrašus, kurių elementai yra Object tipo.

List<Object> objectList = Arrays.asList(new Object(), new Object());

List<Integer> integerList = Arrays.asList(1, 5, 10);

List<Double> doubleList = Arrays.asList(1.5, 5.9, 10.2);

printList(objectList); // OK

printList(integerList); // kompiliavimo klaida

printList(doubleList); // kompiliavimo klaida

Sprendimas yra naudoti neribojantį pakaitos simblį <?>

public static void printList(List<?> list) {

for (Object elem : list)

System.out.println(elem + " ");

}

printList(objectList); // OK

printList(integerList); // OK

printList(doubleList); // OK

## Tolesniam skaitymui

* M Naftalin, P Wadler: *Java Generics and Collections*
* Joshua Bloch: *Effective Java* (26-33 skyriai)
* Venkat Subramaniam: *The Good, Bad, and Ugly of Java Generics* <https://www.youtube.com/watch?v=34oiEq9nD0M>
* <https://docs.oracle.com/javase/tutorial/java/generics/index.html>

## Užduotys

**Užduotys: *Generics***

**Nr. 1**

**Užduotis**

Turime du masyvus - viename skaičiaus tipo elementai { 1, 2, 3 }, kitame eilutės tipo elementai {"Hello", "World"}.

Užduotis: parašyti **vieną** metodą printArray, kuris gali atspausdinti visus abiejų turimų masyvų elementus. Metodas turi priimti ir masyvą su skaičiais, ir masyvą su eilutėmis.

Tikėtinas rezultatas:

1

2

3

Hello

World

**Nr. 2**

**Užduotis**

Sukurti tokią klasių/interfeisių hierarchiją:

+-------+

| Medis |

+---^---+

|

+-----------------+-------------------+

| |

+-----+----+ +-----+-------+

| Lapuotis | | Spygliuotis |

+-----^----+ +-----^-------+

| |

+-------+-------+ +-------------+-+------------+

| | | | |

+----+----+ +----+---+ +--+---+ +---+---+ +-----+---+

| Azuolas | | Berzas | | Egle | | Pusis | | Kadagys |

+---------+ +--------+ +------+ +-------+ +---------+

Kiekvienam objektui turime turėti galimybę iškviesti metodą turi(), kuris aptspausdina informaciją apie objektą. Pavyzdžiui, jei sukurtume ąžuolo objektą Azuolas azuolas = new Azuolas(); ir jam iškviestume metodą azuolas.turi(); tada turi būti atspausdinta Azuolas turi lapus. Nekurti kiekvienam to paties turi() metodo kiekvienoja klasėje, o panaudoti paveldėjimo savybes.

Panaudojant *generics* atskiroje klasėje sukuri metodus:

* ivairusMiskas(...) - priima bet kokių medžių sąrašą
* spygliuociuMiskas(...) - priima bet kokių spgliuočių sąašą
* berzuMiskas(...) - priima tik beržų sąrašą

Visų metodų logika panaši: atspausdina koks tai miškas ir kiekvienam medžiui iškviečia metodą turi()

Sukurti sąrašą, į kurį įdėti ivairių mežių ir iškviesti metodą ivairusMiskas(...).

Sukurti sąrašą, į kurį įdėti ivairių mežių ir iškviesti metodą spygliuociuMiskas(...).

Sukurti sąrašą, į kurį įdėti ivairių mežių ir iškviesti metodą berzuMiskas(...).

Tikėtinas rezultatas:

Ivairus miskas:

Berzas turi lapus

Azuolas turi lapus

Kadagys turi spyglius

Egle turi spyglius

Pusis turi spyglius

Spygliuociu miskas:

Kadagys turi spyglius

Egle turi spyglius

Pusis turi spyglius

Berzu miskas:

Berzas turi lapus

Berzas turi lapus

Berzas turi lapus

**Nr. 3**

**Užduotis**

Sukurkite klasę Pora, su klasės atributais Raktas ir Reiksme. Abu klasės atributai gali būti bet koks objektas.

Sukurkite klasę Mapas su vienu klasės atributu sarasasPoru, kuris bus sarasas, skirtas saugoti klasės Pora objektams. Klasė Mapas turi turėti metodą ideti, kuriam per parametrus perdavus raktą ir reikšmę, iš jų sukonstruoja porą ir ją įdeda į map'ą. Taip pat reikalingas metodas gauti, kuriam per parametrą perdavus raktą, gausime reikšmę pagal tą raktą iš sąrašo.

Sukurkite klasę DnsServer, kuri turi atributus ip1 ir ip2.

Sukurkite enum DnsProvider, kuris saugos keletą reikšmių, pvz. GOOGLE.

Atskiroje klasėje main metode sukurti map'ą, kuris saugo DnsProvider -> DnsServer poras. Sukurti kitą map'ą, kuris saugo String -> String poras. Map'us užpildyti duomenimis ir pabandyti gauti reikšmes pagal raktus.

Panaudojimo pavyzdys:

Mapas<DnsProvider, DnsServer> dnsMapas = new Mapas<>();

dnsMapas.ideti(GOOGLE, new DnsServer("8.8.8.8", "8.8.4.4"));

dnsMapas.ideti(CLOUDFLARE, new DnsServer("1.1.1.1", "1.0.0.1"));

// ...

DnsServer googleDns = dnsMapas.gauti(GOOGLE);

System.out.println(googleDns);

Mapas<String, String> zodynoMapas = new Mapas<>();

zodynoMapas.ideti("Labas", "Hello");

zodynoMapas.ideti("Pasaulis", "World");

// ...

String labas = zodynoMapas.gauti("Labas");

System.out.println(labas);

programa atspausdina:

DnsServer{ip1='8.8.8.8', ip2='8.8.4.4'}

Hello

**Nr. 4**

**Užduotis**

Sukurkite klasę ListOfNumbers kuri saugo sąrašą Double tipo skaičių. Turi metodą addNumber, kuris perduotą skaičių įdeda į sąrašą ir getAverage, kuris grąžina sąrašo skaičių vidurkį.

Kitoje klasėje sukurkite metodą findWithMaxAverage, kuris gali priimti iš anksto nežinomą skaičių ListOfNumbers tipo parametrų. Metodas turi surasti ir grąžinti objektą, kurio skaičių vidurkis yra didžiausias.

Pavyzdys:

ListOfNumbers listObject1 = new ListOfNumbers();

listObject1.addNumber(1.5d);

listObject1.addNumber(10d);

listObject1.addNumber(2.5d);

ListOfNumbers listObject2 = new ListOfNumbers();

listObject2.addNumber(0d);

listObject2.addNumber(1d);

ListOfNumbers listObject3 = new ListOfNumbers();

listObject3.addNumber(100d);

ListOfNumbers result1 = findWithMaxAverage(listObject1, listObject2, listObject3);

ListOfNumbers result2 = findWithMaxAverage(listObject1, listObject2);

System.out.println(result1);

System.out.println(result2);

atspausdina rezultatą

ListOfNumbers{numbers=[100.0]}

ListOfNumbers{numbers=[1.5, 10.0, 2.5]}

**Nr. 5**

**Užduotis**

Sukurkite klasę SuperPair su trimis *generic* tipais taip, kad trečiasis tipas galėtų būti tik koks nors Number skaičius. Pirmieji du tipai gali būti bet kokie. Klasė SuperPair turi turėti kontruktorių su trimis parametrais. Sukurkite kelis tokių porų objektus. Pabandykite trečią parametrą paduoti ne skaičių.

# ***Kolekcijų karkasas (Collections framework)***

„Java“ kolekcijų karkasas yra klasių ir sąsajų (interface) rinkinys, įgyvendinamas per panaudojamas kolekcijų duomenų struktūras. Kolekcija yra objektas, turintis nuorodą į objektų grupę ir pateikiantis rinkinį metodų jiems valdyti. Kolekcijos yra panašios į masyvus, tačiau, skirtingai nei masyvai, jos nėra fiksuoto dydžio. Jų talpa automatiškai keičiasi, pridėjus ar pašalinus objektus.

Kolekcijų karkasas susideda iš:

* **Sąsajų (interfaces)** - abstrakčiųjų duomenų tipų, atspindinčių skirtingas kolekcijas ir pateikiančių vieningą metodų rinkinį, kuris leidžia manipuliuoti kolekcijose talpinamais objektais, nepriklausomai naudojamos kolekcijos tipo.
* **Įgyvendinimų (Implementations)** - konkrečių abstrakčių kolekcijų sąsajų įgyvendinimų.
* **Pagalbinių funkcijų** - statinių metodų, atliekančių naudingas funkcijas su kolekcijomis, pvz. kolekcijos objektų rūšiavimas.

java.util.Collection yra viena pagrindinių kolekcijų sąsajų. Beveik visi kolekcijos objektai paveldi šią sąsają. Esminiai šios sąsajos palikuonys:

* Sąrašas (List)
* Komplektas (Set)
* Eilė (Queue)

Kita reikšminga sąsaja yra java.util.Map. Šį sąsaja nėra tikra kolekcija, nes ji neįgyvendina java.util.Collection sąsajos. Tačiau ji pateikia metodus, kurie grąžina sąsajos turinį kolekcijos objektuose. Tai leidžia java.util.Map sąsaja manipuliuoti kaip kolekcija.

#### Pagrindiniai java.util.Collection sąsajos metodai

add(E e)

Skirtas pridėti E tipo elementui į kolekciją

addAll(Collection<? extends E> c)

Prideda visus nurodytos kolekcijos elementus į šią kolekciją

clear()

Pašalina visus elementus, esančius šioje kolekcijoje

contains(Object o)

Grąžina true, jei ši kolekcija talpina nurodyta elementą

isEmpty()

Grąžina true jei ši kolekcija neturi jokių elementų

size()

Grąžina elementų, esančių kolekcijoje, skaičių

remove(Object o)

Jei toks elementas egzistuoja, pašalina jį iš kolekcijos

### Pagrindiniai kolekcijų tipai

### Sąrašas (List)

Sąrašas - vienas dažniausiai naudojamų kolekcijų tipų. Sąrašo kolekcijos objektai įgyvendina java.util.List sąsają. Sąrašas vaizduoja surikiuotą elementų seką. Kadangi sąraše svarbi elementų tvarka, naudotojas gali pasiekti saugomus elementus pagal indeksą arba ieškoti specifinio elemento sąraše. Sąrašo ideologija nedraudžia jam turėti pasikartojančių elementų bei null reikšmių, tačiau nėra garantuota, kad kažkas neįgyvendins sąrašo, draudžiančio dublikatus. Sąrašai, kaip ir masyvai, prasideda nuo 0 indekso.

#### java.util.List sąsają papildo java.util.Collection šiais metodais:

get(int index)

Grąžina elementą esantį nurodytame indekse

set(int index, E element)

Pakeičia nurodytame indekse esantį elementą duotuoju

add(int index, E element)

Įdeda elementą į nurodytą indeksą ir pastumia visus po jo išsidėsčiusius elementus per vieną poziciją

remove(int index)

Pašalina nurodytame indekse esantį elementą

#### java.util.List įgyvendinimai

Dažniausiai naudojamos klasės, įgyvendinančios sąrašą yra java.utils.ArrayList ir java.utils.LinkedList.

**ArrayList** implementacija naudoja masyvą elementų saugojimui. ArrayList objekto inicializacijos metu sukuriamas fiksuoto dydžio masyvas. Iškvietus add metodą su nauju elementu, elementas išsaugomas sukurtame masyve. Pasibaigus laisvai vietai masyve, ArrayList objektas automatiškai sukurs naują, pusantro karto didesnį masyvą, nukopijuos į jį seno masyvo elementus ir pridės naują elementą.

Kita populiari sąrašo implementacija yra **LinkedList**. Tai [Linked List](https://en.wikipedia.org/wiki/Linked_list) duomenų struktūros įgyvendinimas „Java“ kalboje. Kiekvienas LinkedList elementas turi sekančio elemento adresą, taip kartu jie sudaro elementų seką. Pridėjus naują elementą į LinkedList kolekciją, paskutinis kolekcijos elementas gauna nuorodą į naująjį elementą. Vizualiai tai atrodo taip:

[elementas1]-->[elementas2]-->null

[elementas1]-->[elementas2]-->[elementas3]-->null

ArrayList kolekcija daug efektyvesnė norint pasiekti kolekcijos elementus. Masyvo elementai kompiuterio atmintyje yra saugomi vienas šalia kito, todėl ši kolekcija yra daug efektyvesnė skaitant visus kolekcijos elementus. Taip pat ArrayList kolekcija yra daug greitesnė norint gauti specifinio indekso elementą. LinkedList kolekcijoje norint gauti n indekso elementą, einama per visus elementus kol pasiekiamas norimas elementas. ArrayList kolekcijoje tai vyksta daug greičiau – tiesiog paimamas n indekso elementas iš masyvo. Dauguma atvejų yra naudojama ArrayList sąrašo implementacija. LinkedList kolekcija naudojama labai retais atvejais - kai svarbus našumas įterpiant, pašalinant elementus iš sąrašo. LinkedList atveju norint įterpti elementą į sąrašo n poziciją, elementas pozicijoje n-1 pradeda rodyti į naują elementą, o naujasis elementas pradeda rodyti į elementą, buvusį n pozicijoje. Vizualiai įterpimas atrodo taip:

[naujas elementas]-->null

[elementas N-1]-->[elementas N]-->null

[elementas N-1]-->[naujas elementas]-->null

[elementas N]-->null

[elementas N-1]-->[naujas elementas]-->[elementas N]-->null

Norint pašalinti elementą indeksu n, elementas n-1\* nustoja rodyti į jį ir pradeda rodyti į elementą n+1

[elementas N-1]-->[elementas N]-->[elementas N+1]-->null

[elementas N]-->null

[elementas N-1]-->[elementas N+1]-->null

ArrayList kolekcijoje tai vyksta daug sudėtingiau. Tiek įterpimo, tiek pašalinimo atveju yra sukuriamas naujas masyvas ir nukopijuojami visi elementai bei nauja reikšmė arba elementai be pašalintos reikšmės.

**Dauguma atvejų reikėtų naudoti ArrayList kolekciją. LinkedList įgyvendinimas naudojamas tik išskirtiniais atvejais, kai yra svarbus objektų įterpimo į vidurį arba pašalinimo greitis.**

### Žemėlapis (Map)

Žemėlapio tipo kolekcijos įgyvendina java.util.Map sąsają. Žemėlapio kolekcijos saugo reikšmes pagal paduotą raktą. Raktas yra naudojamas pridedant ir pasiimant reikšmes. Žemėlapis negali turėti pasikartojančių raktų, kiekvienas raktas gali rodyti tik į vieną reikšmę.

Nors java.util.Map sąsaja neįgyvendina java.util.Collection sąsajos, žemėlapis yra laikomas kolekcijų karkaso dalimi, nes visi java.util.Map įgyvendinimai pateikia 3 metodus, grąžinančius žemėlapio turinį kaip kolekcijas:

* Metodas grąžinantis visas žemėlapio reikšmes kaip komplektą (set)
* Metodas grąžinantis žemėlapio reikšmių kolekciją
* Metodas grąžinantis žemėlapio raktų ir reikšmių porų kolekciją

Žemėlapio sąsaja yra naudojama norint efektyviai gauti specifinį elementą. Žemėlapio sąsaja pateikia metodus leidžiančius ieškoti reikiamo elemento pagal duotą raktą. Todėl, jei dirbama ne su visais kolekcijos elementais iš karto, o norima pasiekti specifinį elementą, jį naudoti yra efektyviau. Tikrinimui, ar toks raktas jau egzistuoja, žemėlapio kolekcija naudoja hashCode() ir equals() metodus, todėl objektai, naudojami kaip raktas, turėtų perrašyti šiuos metodus.

#### Žemėlapio sąsajai būdingi metodai

containsKey(Object key)

Grąžina true jei toks raktas egsizstuoja kolekcijoje

containsValue(Object value)

Grąžina true jei nors vienas raktas rodo į nurodytą elementą

put(K key, V value)

Susieja nurodytą raktą su duota reikšme šiame žemėlapyje

get(Object key)

Grąžina nurodyto rakto elementą

remove(Object key)

Pašalina reikšmę, susietą su nurodytu raktu

keySet()

Grąžina žemėlapio raktų kolekciją

values()

Grąžina žemėlapio reikšmių kolekciją

entrySet()

Grąžina žemėlapio raktų ir reikšmių porų kolekciją

Yra du pagrindiniai žemėlapio įgyvendinimai: java.util.HashMap ir java.util.TreeMap.

#### java.util.HashMap

HashMap įgyvendinimas yra paremtas [hash lentele](https://en.wikipedia.org/wiki/Hash_table). Dauguma šio įgyvendinimo operacijų yra labai greitos, tačiau šis įgyvendinimas neužtikrina nekintančios saugojamų elementų tvarkos

#### java.util.TreeMap

Tree map įgyvendinimas yra paremtas [Raudono-Juodo medžio](https://en.wikipedia.org/wiki/Red%E2%80%93black_tree) duomenų struktūra. Šio įgyvendinimo metodai nėra tokie greiti kaip HashMap įgyvendinimo, tačiau saugojami elementai yra surikiuoti ir jų tvarka nekinta. Taip pat šis įgyvendinimas negali turėti rakto su null reikšme.

### Komplektas (Set)

Komplektas yra kolekcija, įgyvendinanti java.util.Set sąsają, neturinčią vienodų elementų. Tiksliau, komplektas neturi elementų porų e1 ir e2 kur e1 = e2 ir daugiausiai vieną null reikšmę. Kaip matoma iš pavadinimo, komplektas yra [matematinio komplekto](https://en.wikipedia.org/wiki/Set_(mathematics)) įgyvendinimas „Java“ programavimo kalboje.

Komplektai paprastai naudojami vietoje sąrašų, norint užtikrinti, kad kolekcijoje nebus pasikartojančių elementų. Užtikrinti, kad komplekte dar nėra tokio elemento, naudojamas equals() metodas, todėl norint užtikrinti, kad komplektas veikia taip kaip norima, komplekte saugomų objektų equals() ir hashCode() metodai turėtų būti teisingai perrašyti (overriden).

Vienas iš komplekto naudojimo atvejų – norint pašalinti pasikartojančias reikšmes iš kolekcijos:

public static List<String> removeDuplicateNames(List<String> namesWithDuplicates) {

final Set<String> uniqueNames = new HashSet<>(namesWithDuplicates);

return new ArrayList<>(uniqueNames);

}

Komplekto įgyvendinimai, į kuriuos reikėtų atkreipti dėmesį yra HashSet ir TreeSet.

* **Įgyvendinimas java.util.HashSet** naudoja hash lentelę (tiksliau HashMap kolekciją) elementų unikalumui užtikrinti. Kaip ir HashMap kolekcijos, šio įgyvendinimo metodai yra greiti, tačiau ši kolekcija neužtikrina griežtos elementų tvarkos.
* **Įgyvendinimas java.util.TreeSet** naudoja TreeMap kolekciją elementų unikalumui užtikrinti. Todėl jis nėra toks greitas kaip HashSet įgyvendinimas, tačiau užtikrina surikiuotus kolekcijos elementus.

### Pagalbinės funkcijos

Kolekcijų karkasas pateikia rinkinį statinių funkcijų, kurios manipuliuoja arba grąžina kolekcijas. Šį rinkinį sudaro:

* Funkcijos, kurios grąžina naują kolekciją paremtą nurodyta kolekcija
* Funkcijos, keičiančios kolekcijos sekos tvarką (pvz. rūšiuoja elementus didėjimo tvarka)
* Funkcijos, grąžinančios specifinius kolekcijos elementus (pvz. grąžina didžiausią sąrašo skaičių)

Šios pagalbinės funkcijos yra klasėse java.util.Collections bei java.util.Arrays.

#### java.util.Collections

Klasė java.util.Collections pateikia tokias funkcijas:

emptyList()

emptyMap()

emptySet()

Grąžinančias tuščią nurodytą kolekciją

fill(List<? super T> list, T obj)

Pakeičia visus sąrašo elementus, nurodytu elementu

max(Collection<? extends T> coll)

Grąžina didžiausią kolekcijos elementą (visi kolekcijos elementai turi įgyvendinti Comparable sąsają)

copy(List<? super T> dest, List<? extends T> src)

Nukopijuoja sąrašo src elementus į dest sąrašą

Collections klasė turi dar daug kitų naudingų funkcijų, kurias vertėtų panagrinėti.

#### java.util.Arrays

Klasė java.util.Arrays pateikia funkcijas, skirtas dirbti su masyvais, tokias kaip masyvo elementų rūšiavimas arba reikiamo elemento paieška. Tačiau ši klasė turi naudingą ir dažnai naudojamą funkciją dirbant su sąrašais:

asList(T... a)

Kuri leidžia masyvus konvertuoti į sąrašus arba greitai sukurti sąrašą su nurodytais elementais.

String[] raidziuMasyvas = {"a", "b", "c"};

List<String> raidziuSarasas = Arrays.asList(raidziuMasyvas);

raidziuSarasas = Arrays.asList("A", "B", "C");

### Patobulintas for ciklas (foreach loop)

Patobulintas for ciklas leidžia pereiti per kiekvieną masyvo/kolekcijos elementą be poreikio nurodyti for ciklo salygą.

Paprasto ciklo pavyzdys:

char[] vowels = {'a', 'e', 'i', 'o', 'u'};

for (int i = 0; i < vowels.length; ++ i) {

System.out.println(vowels[i]);

}

Patobulinto ciklo pavyzdys:

char[] vowels = {'a', 'e', 'i', 'o', 'u'};

for (char item: vowels) {

System.out.println(item);

}

Abejų ciklų rezultatas bus toks pats:

a

e

i

o

u

Patobulinto ciklo sintaksė yra lengviau suprantama ir greičiau parašoma, todėl, kai nėra svarbus iteracijos indeksas, vertėtų naudoti patobulintą for ciklą.

#### foreach ciklo sintaksė

for(KolekcijosDuomenųTipas elementas : kolekcija) {

...

}

Sintaksėje aukščiau:

* kolekcija yra kolekcijos arba masyvo objektas, per kurio elementus bus einamas ciklas
* elementas yra vienas kolekcijos elementas
* KolekcijosDuomenųTipas yra elementų, saugomų kolekcijoje, tipas

### Užduotys

### 1. Užduotis

* Sukurkite klasę Payment su laukais id, name, paymentDate, recipientId, payerId
* Sukurkite statinį metodą, kuris priimtų int tipo parametrą. Parametras nurodo, kiek Payment reikšmių reikia sugeneruoti. Dinamiškai generuokite Payment objektus ir grąžinkite sąrašą su nurodytu reikšmių kiekiu
* Padarykite, kad generuojami id laukai kartais pasikartotų
* Naudokite metodo grąžinamus duomenis sukurti ArrayList bei LinkedList sąrašus (duomenys abiejose kolekcijose turi būti tokie patys). Sukurtos kolekcijos turėtų būti įvairaus dydžio - nuo kelių iki kelių tūkstančių elementų
* Paskaičiuokite List sąsajos metodų (add, delete, get) vykdymo laikus skirtingiems įgyvendinimams. Atkreipkite dėmesį, kaip laikai skiriasi nuo sąrašo dydžio
* Paverskite sąrašą į rinkinį, įsitikinkite, kad besikartojančios reikšmės dingsta

### 2. Užduotis

* Sukurkite klasę PeopleDatabase
* Pridėkite jai Map tipo lauką ir duokite jam reikšmingą pavadinimą
* Klasės inicializavimo metu užpildykite lauką duomenimis apie žmones, kur raktas yra String tipo asmens kodas, o reikšmė yra String tipo žmogaus vardas. Tegul vardai būna reikšmingi, o asmens kodai atitinka reikalavimus

Bonus: sukurkite funkciją, generuojančią atsitiktinius ir teisingus asmens kodus

* Sukurkite metodą, kuris grąžintų visų žemėlapyje esančių žmonių vardus ir kitą metodą, kuris grąžintų visus asmens kodus
* Sukurkite metodą, kuris priima vieną parametrą – asmens kodą ir grąžina asmens kodo savininko vardą
* Sukurkite metodą, kuris grąžintų visus unikalius vardus
* Patobulinkite metodą, grąžinantį visus žmonių vardus taip, kad jis grąžintų sąrašą sudarytą iš asmens kodo + vardo elementų
* Sukurkite metodą, kuris grąžintų žemėlapį su visais asmens kodais, sudėtais pagal raktus vyras ir moteris
* Sukurkite metodą, kuris priima vieną parametrą - žmogaus vardą ir grąžina sąrašą asmens kodų su tokiu vardu
* Sukurkite programėlę, naudojančią PeoplDatabase klasę. Programėlė paleidimo metu priimtų argumentą – norimo metodo numerį ir atspausdintų iškviesto metodo rezultatą. Jei metodui reikia papildomų parametrų, pridėkite juos kaip papildomus argumentus
* Argumentas help turėtų išspausdinti visus galimus metodus, bei jų iškvietimo numerius

#### Veikimo pavyzdys

Komanda:

java uzduotis -h

Išspausdina rezultatą:

1) Metodas gauti unikaliems vardas

2) Metodas gauti vardui pagal asmens kodą (papildomas argumentas - asmens kodas)

Komanda:

java uzduotis 1

Išspausdina:

Jonas

Tomas

Tadas

Komanda:

Java uzduotis 2 39102110000

Išspausdina:

Jonas

### Šaltiniai

* <https://en.wikipedia.org/wiki/Java_collections_framework>
* <https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>

# **Comparator**

## Pavyzdys

Turime klasę Employee:

import java.time.LocalDate;

public class Employee {

private int id;

private String name;

private double salary;

private LocalDate joiningDate;

public Employee(int id, String name, double salary, LocalDate joiningDate) {

this.id = id;

this.name = name;

this.salary = salary;

this.joiningDate = joiningDate;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public LocalDate getJoiningDate() {

return joiningDate;

}

public void setJoiningDate(LocalDate joiningDate) {

this.joiningDate = joiningDate;

}

}

Norime turėti galimybę surūšiuoti Employee tipo elementus saugantį sąrašą. Kad tą galėtume daryti, mūsų klasė turi implemntuoti interfeisą Comparable, per generics nurodant, kad tipas bus Employee:

public class Employee implements Comparable<Employee> {

/\* ... \*/

}

Tada turime implementuoti interfeiso Comparable metodą compareTo. Jei norime rūšiavimui naudoti tik klasės atributą id, tada compareTo metodas atrodo taip:

@Override

public int compareTo(Employee anotherEmployee) {

return this.getId() - anotherEmployee.getId();

}

Jei norime rūšiuoti pagal du klasės Employee atributus id ir name, tada compareTo metodas atrodo taip:

@Override

public int compareTo(Employee anotherEmployee) {

int i = this.getId() - anotherEmployee.getId();

if (i != 0) {

return i;

}

return this.getName().compareTo(anotherEmployee.getName());

}

Po modifikacijų, mūsų Emplayee klasė atrodo taip:

import java.time.LocalDate;

class Employee implements Comparable<Employee> {

private int id;

private String name;

private double salary;

private LocalDate joiningDate;

public Employee(int id, String name, double salary, LocalDate joiningDate) {

this.id = id;

this.name = name;

this.salary = salary;

this.joiningDate = joiningDate;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public LocalDate getJoiningDate() {

return joiningDate;

}

public void setJoiningDate(LocalDate joiningDate) {

this.joiningDate = joiningDate;

}

@Override

public int compareTo(Employee anotherEmployee) {

int i = this.getId() - anotherEmployee.getId();

if (i != 0) {

return i;

}

return this.getName().compareTo(anotherEmployee.getName());

}

@Override

public String toString() {

return "Employee{" +

"id=" + id +

", name='" + name + '\'' +

", salary=" + salary +

", joiningDate=" + joiningDate +

'}';

}

}

Kitoje klasėje šalia metodo main turime kitą metodą, kuris spausdina informaciją apie sąrašo elementus:

private static void printEmployees(List<Employee> employees, String message) {

System.out.println(message);

for (Employee employee : employees) {

System.out.println(employee);

}

}

Susikurkime sąrašą su Employee elementais:

List<Employee> employees = new ArrayList<>();

employees.add(new Employee(1010, "Rajeev", 100.00, LocalDate.of(2010, 7, 10)));

employees.add(new Employee(1004, "Chris", 900.50, LocalDate.of(2017, 3, 19)));

employees.add(new Employee(1015, "David", 134000.00, LocalDate.of(2017, 9, 28)));

Tada atpausdinus sąrašą:

printEmployees(employees, "Unsorted list:");

gausime nesurūšuotą sąrašą:

Unsorted list:

Employee{id=1010, name='Rajeev', salary=100.0, joiningDate=2010-07-10}

Employee{id=1004, name='Chris', salary=900.5, joiningDate=2017-03-19}

Employee{id=1015, name='David', salary=134000.0, joiningDate=2017-09-28}

Surušiuojame sąrašo elementus:

Collections.sort(employees);

printEmployees(employees, "Sorted list:");

gausime

Sorted list:

Employee{id=1004, name='Chris', salary=900.5, joiningDate=2017-03-19}

Employee{id=1010, name='Rajeev', salary=100.0, joiningDate=2010-07-10}

Employee{id=1015, name='David', salary=134000.0, joiningDate=2017-09-28}

Jei norime turėti pasirinkimą pagal ką rūšiuoti tada susikurikime keletą klasių, kurios implementuoja Comparator interfeisą ir įgyvendina comapare metodą.

import java.util.Comparator;

public class IdComparator implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

return o1.getId() - o2.getId();

}

}

import java.util.Comparator;

public class SalaryComparator implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

return Double.compare(o1.getSalary(), o2.getSalary());

}

}

Tada galime rūšiuoti pagal id:

Collections.sort(employees, new IdComparator());

printEmployees(employees, "Sorted list by ID:");

Sorted list by ID:

Employee{id=1004, name='Chris', salary=900.5, joiningDate=2017-03-19}

Employee{id=1010, name='Rajeev', salary=100.0, joiningDate=2010-07-10}

Employee{id=1015, name='David', salary=134000.0, joiningDate=2017-09-28}

arba pagal salary:

Collections.sort(employees, new SalaryComparator());

printEmployees(employees, "Sorted list by salary:");

Sorted list by salary:

Employee{id=1010, name='Rajeev', salary=100.0, joiningDate=2010-07-10}

Employee{id=1004, name='Chris', salary=900.5, joiningDate=2017-03-19}

Employee{id=1015, name='David', salary=134000.0, joiningDate=2017-09-28}

Norint surūšiuoti atvirkšine tvarka, turime naudoti Collections.reverseOrder:

Collections.sort(employees, Collections.reverseOrder(new SalaryComparator()));

printEmployees(employees, "Sorted list by salary reversed:");

Sorted list by salary reversed:

Employee{id=1015, name='David', salary=134000.0, joiningDate=2017-09-28}

Employee{id=1004, name='Chris', salary=900.5, joiningDate=2017-03-19}

Employee{id=1010, name='Rajeev', salary=100.0, joiningDate=2010-07-10}

## Tolesniam skaitymui

* <https://www.callicoder.com/java-comparable-comparator/>
* <https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html>

## Užduotys

## Užduotys: Comparator

## Nr. 1

### Užduotis

Sukurti klasę JavaStudentGroup, kuri turi sąrašą studentų. Klasė Student turi vardą, pavardę ir studento numerį. Suskurti klasės su sąrašu objektą ir į jį įdėti 21 studentą. Studentus išrūšiuoti pagal vardą, pagal pavardę ir pagal studento nr. Studentus suskirstyti į 4 grupes. Kodas turi būti parašytas taip, kad lengvai galėtumėte suskirstyti studentus į kitokį skaičių grupių.

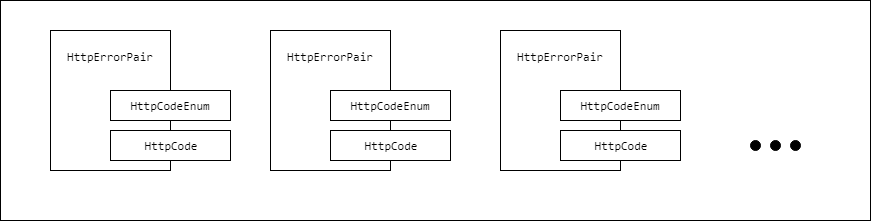
## Nr. 2

### Užduotis

Sukurti klasę X, kuri turi sąrašą skaičių. Sukurti klasę Y, kuri saugo prieš tai sukurtos klasės objektų sąrašą. Surūšiuoti skaičius sąraše. Surūšiuoti Y elementus saugantį sąrašą pagal tai, kokia yra jo sąraše esančių skaičių suma. Programa turi mokėti rūšiuoti tiek didėjimo, tiek ir mažėjimo tvarka.

## Nr. 3

### Užduotis

[](https://github.com/tadus21/code-academy-java-kursai/blob/master/comparator/exercises/img/task1.png)

1. Sukurkite enum'ą HttpCodeEnum su reikšmėmis CODE\_404, CODE\_401, CODE\_500 ir pan.
2. Sukurkite enum'ą ErrorLevels su reikšmėmis HIGH, LOW, MEDIUM ir pan. Kiekviena enum'o reikšmė turi turėti kažkokį skaičių, pvz HIGH yra 2, LOW yra 0.
3. Sukurkite klasę HttpCode, kuri turi klasės atributą ErrorLevels tipo.
4. Sukurkite keletą HttpCode klasę paveldinčių klasių, tokių kaip NotFoundHttpCode, BadRequestHttpCode, InternalServerErrorHttpCode ir pan.
5. Sukurkite klasę HttpErrorPair, kuri turi generics tipo du klasės kintamuosius (key ir value). Pirmasis gali būti bet kokio tipo, antrasis turi galėti būti tik HttpCode tipo arbą šios klasės vaikinių klasių tipo.
6. Sukurkite tuščią sąrašą ir į jį įdėkite keletą HttpErrorPair objektų.
7. pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_500, new InternalServerErrorHttpCode(ErrorLevels.HIGH)));
8. pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_401, new BadRequestHttpCode(ErrorLevels.MEDIUM)));
9. pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_404, new NotFoundHttpCode(ErrorLevels.LOW)));

pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_403, new ForbiddenHttpCode(ErrorLevels.MEDIUM)));

1. Atspausdinkite sąrašą, surūšiuokite pagal ErrorLevels enum'o reikšmes ir atspausdinkite surūšiuotą sąrašą. Gausite panašiai tokį rezultatą:
2. [HttpErrorPair{key=CODE\_500, code=HttpCode{level=HIGH}}, HttpErrorPair{key=CODE\_401, code=HttpCode{level=MEDIUM}}, HttpErrorPair{key=CODE\_404, code=HttpCode{level=LOW}}, HttpErrorPair{key=CODE\_403, code=HttpCode{level=MEDIUM}}]
3. [HttpErrorPair{key=CODE\_404, code=HttpCode{level=LOW}}, HttpErrorPair{key=CODE\_401, code=HttpCode{level=MEDIUM}}, HttpErrorPair{key=CODE\_403, code=HttpCode{level=MEDIUM}}, HttpErrorPair{key=CODE\_500, code=HttpCode{level=HIGH}}]
4. Nepamirškite metodo toString teisingai panaudoti Comparable.

### prendimas

Išskleisti

package lt.codeacademy.learn;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class Application {

public static void main(String[] args) {

List<HttpErrorPair> pairList = new ArrayList<>();

pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_500, new InternalServerErrorHttpCode(ErrorLevels.HIGH)));

pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_401, new BadRequestHttpCode(ErrorLevels.MEDIUM)));

pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_404, new NotFoundHttpCode(ErrorLevels.LOW)));

pairList.add(new HttpErrorPair(HttpCodeEnum.CODE\_403, new ForbiddenHttpCode(ErrorLevels.MEDIUM)));

System.out.println(pairList);

Collections.sort(pairList);

System.out.println(pairList);

}

}

enum HttpCodeEnum {

CODE\_404,

CODE\_401,

CODE\_500,

CODE\_403

}

enum ErrorLevels {

HIGH(2),

MEDIUM(1),

LOW(0);

private final int level;

ErrorLevels(int level) {

this.level = level;

}

public int getValue() {

return level;

}

}

class HttpCode implements Comparable<HttpCode> {

private ErrorLevels level;

public HttpCode(ErrorLevels level) {

this.level = level;

}

public ErrorLevels getLevel() {

return level;

}

public void setLevel(ErrorLevels level) {

this.level = level;

}

@Override

public int compareTo(HttpCode p) {

return Integer.compare(this.level.getValue(), p.getLevel().getValue());

}

@Override

public String toString() {

return "HttpCode{" +

"level=" + level +

'}';

}

}

class NotFoundHttpCode extends HttpCode {

public NotFoundHttpCode(ErrorLevels level) {

super(level);

}

}

class BadRequestHttpCode extends HttpCode {

public BadRequestHttpCode(ErrorLevels level) {

super(level);

}

}

class InternalServerErrorHttpCode extends HttpCode {

public InternalServerErrorHttpCode(ErrorLevels level) {

super(level);

}

}

class ForbiddenHttpCode extends HttpCode {

public ForbiddenHttpCode(ErrorLevels level) {

super(level);

}

}

class HttpErrorPair<K, V extends HttpCode> implements Comparable<HttpErrorPair> {

private K key;

private V code;

public HttpErrorPair(K key, V code) {

this.key = key;

this.code = code;

}

public K getKey() {

return key;

}

public void setKey(K key) {

this.key = key;

}

public V getCode() {

return code;

}

public void setCode(V code) {

this.code = code;

}

@Override

public int compareTo(HttpErrorPair o) {

return this.getCode().compareTo(o.getCode());

}

@Override

public String toString() {

return "HttpErrorPair{" +

"key=" + key +

", code=" + code +

'}'

# Lambda išraiškos bei Java 8 srautai

### Higher-order funkcijos

Matematikoje ir kompiuterių moksle, funkcija vadinama higher-order funkcija jei ji tenkina bent vieną iš šių sąlygų:

* Priima vieną ar daugiau funkcijų kaip parametrą
* Grąžina funkciją kaip rezultatą

Įsivaizduokite funkciją sort, kuri kaip parametrus priima du argumentus – elementų sąrašą ir funkciją, palyginančią, kuris iš sąrašo elementų didesnis, o funkcijos grąžinamas rezultatas – naujas elementų sąrašas, išrūšiuotas naudojant pateiktą funkciją.

sort(elements : List<T>, comparator : (t1 : T, t2 : T) -> int) -> List<T>

Tokia funkcija išrūšiuoja nurodytą sąrašą, keliaudama per visus jo elementus ir lygindama juos vienas su kitu, naudodama nurodytą palyginimo funkciją. Rūšiavimo funkcija yra higher-order funkcija.

Programavime dažnai reikalinga nurodyti tam tikrą veiksmų seką, tačiau tuos veiksmus įgyvendinti reikia vėliau. Tokiose situacijose, pavyzdžiui, rūšiuojant sąrašo elementus, yra naudojamos higher-order funkcijos.

Kitose kalbose, pavyzdžiui Scala arba Javascript, metodai yra tokie pat lygiaverčiai nariai kaip ir klasės arba duomenų tipai. Tai reiškia, kad metodai gali būti priskirti kintamiesiems arba grąžinami kaip metodų rezultatai. Java programavimo kalba tokia savybe nepasižymi. Norint paduoti funkciją kaip metodo parametrą, Java kalboje naudojamos sąsajos, turinčios vieną reikalingą metodą. Tokios sąsajos vadinamos funkcinėmis sąsajomis (functional interface).

Kiekvieną kartą įgyvendinti funkcinę sąsają užima daug laiko, todėl 8 Java programavimo kalbos versija pridėjo lambdaišraiškas, palengvinančias darbą su funkcinėmis sąsajomis.

### Lambda išraiškos

Lambda išraiškos tai funkcijos, kurios gali būti sukurtos nepriskiriant jų jokiai klasei. Lambda išraiškos įgyvendina funkcinių sąsajų metodus ir gali būti paduodamos kaip objektai, ten, kur reikia funkcinių sąsajų.

#### Lambda išraiškų sintaksė

Lambda išraiškos sintaksę sudaro **argumentų sąrašas**, **rodyklė** ir **metodo korpusas**:

() -> 10

nepriima argumentų ir grąžina int tipo argumentą. Funkcijos atitikmuo būtų int getNum(). Jei metodo korpusas yra iš vienos eilutės, figūriniai skliaustai bei return raktažodis nėra būtinas.

(int x) -> x \* x

Priima int tipo argumentą ir grąžina int tipo argumentą. Funkcijos atitikmuo būtų int square(int x).

x -> x \* x

Tokia pati išraiška kaip ir ankstesniame pavyzdyje. Lambda išraiškose skliaustai bei argumento tipas nėra būtini: kai jis nenurodytas, kompiliatorius tipą nuspėja automatiškai,.

(x, y) -> x + y

Priima du int tipo argumentus ir grąžina jų sumą kaip int reikšmę. Funkcijos atitikmuo būtų int sum(int x, int y). Kai lambdaišraiška priima daugiau nei vieną parametrą, skliaustai yra būtini.

(x, y) -> {

return x + y;

}

Tokia pati išraiška kaip ir ankstesnė, tačiau jei metodo korpusas yra ilgesnis nei vienos eilutės, figūriniai skliaustai yra būtini. Esant figūriniams skliaustams, jei išraiška grąžina reikšmę, return raktažodis turi būti nurodytas specifiškai.

Lambda išraiškos gali būti priskirtos kintamiesiems, paduodamos kaip funkcijos argumentai arba grąžinamos kaip funkcijų rezultatai. Pavyzdžiui funkcinė sąsaja

interface IntComparator{

int compare(int x, int y);

}

galėtų būti įgyvendinta naudojant lambda išraišką. Pavyzdžiui:

IntComparator comparator = (x, y) -> x – y;

arba

public IntComparator reversedComparator() {

return (x, y) -> y – x;

}

#### Paketas java.util.function

Java teikia didelį standartinių funkcinių sąsajų rinkinį java.util.function pakete. Prieš aprašant naują funkcinę sąsają, reikėtų pasitikrinti ar standartinių sąsajų pakete nėra tinkamos sąsajos.

Standartinės funkcinės sąsajos yra suskirstytos į 4 grupes:

* Consumers - priima argumentą ir nieko negrąžina (Consumer, BiConsumer, IntConsumer, ...)
* Suppliers - nepriima jokių argumentų, bet grąžina reikšmę (Supplier, IntSupplier, ...)
* Predicates - priima argumentą ir grąžina boolean tipo reikšmę (\*Predicate, BiPredicate, LongSupplier, ...)
* Functions - priima argumentą ir grąžina reikšmę (Function, BiFunction, UnaryOperator)

#### Metodų nuorodos

Jei metodo parašas atitinka funkcinėje sąsajoje aprašyto metodo parašą, nuoroda į tą metodą gali būti paduota vietoj lambda išraiškos.

public class Math {

public static double multiply(double a, double b) {

return a \* b;

}

}

Aukščiau aprašytos klasės metodas \* multiply \*, priimantis du double tipo parametrus ir grąžinantis jų sandaugą kaip double rezultatą, gali būti naudojamas vietoj lambda išraiškos, ten kur reikalinga funkcinė sąsaja, su metodu, priimančiu dvi double reikšmes ir grąžinančiu double tipo rezultatą.

BiFunction<Double, Double, Double> f;

f = (val1, val2) -> val1 \* val2;

f = Math::multiply;

### Srautai

Srautas - klasė, palaikanti funkcinio tipo operacijas, paremtas funkcinėmis sąsajomis, leidžiančias transformuoti kolekcijas.

Pavyzdžiui:

int totalSalary = people.stream()

.filter(person -> person.isEmployed())

.mapToInt(person -> person.getSalary())

.sum();

Šis kodas naudoja žmonių kolekciją kaip šaltinį srautui. Srautas filtruojamas ir pašalinami nedirbantys asmenys, tada srautas transformuojamas iš asmenų srauto į asmenų atlyginimo srautą ir srauto elementai yra sudedami.

Srauto metodai suskirstyti į dvi grupes - intermediate ir terminal. Kartu šie tipai formuoja srauto grandines (pipelines). Grandinę sudaro šaltinis (pavyzdžiui kolekcija arba masyvas), nulis arba daugiau intermediate operacijų, tokių kaip filter arba map, keičiančių srauto turinį ir terminal operacija, pavyzdžiui reduce arba collect.

Intermediate metodai aprašo taisykles srauto elementų manipuliavimui ir kaip rezultatą gražina naują srautą. Jie yra tingūs (lazy) – pavyzdžiui iškvietus operaciją filter, filtravimas iškarto nevyksta, tiesiog apibrėžiama taisyklė, kuri, prasidėjus keliavimui per šaltinio elementus, pašalins sąlygos netenkinančius elementus.

Terminal metodai pradeda šaltinio elementų ėjimą per srauto grandinę vykdant intermediate operacijas, manipuliuojančias srauto turiniu. Terminal operatoriai grąžina rezultatą. Srautas įvykdžius terminal operatorių, laikomas panaudotu. Norint iš naujo pereiti per šaltinio elementus, reikia sukurti naują srautą.

Srauto ir kolekcijų skirtumai:

* Srautas nesaugo duomenų. Srautas tik apdoroja duomenis iš šaltinių, tokių kaip kolekcijos, masyvai ar I/O kanalai, per įvairias skaičiavimo, transformavimo operacijas
* Srautas laikosi funkcinio programavimo ideologijos. Kiekviena srauto operacija grąžina rezultatą, tačiau nemodifikuoja originalaus šaltinio
* Srautas yra tingus (lazy). Intermediate operacijos nebus vykdomos tol, kol nebus iškviestas terminal metodas
* Srautas neturi apibrėžto ilgio, ir gali būti begalinis
* Srautas yra suvartojamas (consumable) - srauto elementai yra aplankomi tik vieną kartą ir norint antrą kartą pereiti per šaltinio elementus, turi būti sugeneruotas naujas srautas.

Srautai gali būti sukurti keliais būdais, pavyzdžiui:

* Iš Collection objekto per metodą stream()
* Iš Arrays klasės metodo stream(Object[])
* Iš static factory metodų kaip Stream.of(Object[])
* Iš BuffererdReader objekto metodo lines()

#### Pagrindinės intermediate operacijos

* **Filter** – priima Predicate sąsają ir filtruoja jos netenkinančius elementus
* **Map** – priima Function sąsają ir transformuoja srauto elementus
* **FlatMap** – priima Function sąsają, kuri iš kiekvieno elemento sukuria naują srautą ir sujungia juos į vieną bendrą srautą

#### Pagrindinės terminal operacijos

* **Collect** - surenka srauto elementus į nurodytą kolekcijos tipą
* **Reduce** priima BiFunction sąsają, kurios dėka sumažina srautą iki vieno elemento
* **Min** - grąžina mažiausią srauto elementą
* **Max** - grąžina didžiausią srauto elementą

### Užduotys

# Srautų užduotis

1. Atsisiųskite [projektą](https://github.com/tadus21/code-academy-java-kursai/blob/master/streams-and-lambdas/exercises/src) su užduočiai reikalingais failais
2. Projekte sukurkite naują klasę DefaultShopService, įgyvendinančią ShopServicesąsąją
3. ShopService metodų įgyvendinimui naudokite kolekciją, gaunamą iš ShopRepository.getShop() metodo
4. Kad gautumėte reikiamus rezultatus, manipuliuokite kolekcijos elementais pasitelkdami srautus

### Šaltiniai

* Effective Java 3rd edition
* <https://docs.oracle.com/javase/8/docs/>

***Maven***

*Maven* yra projekto valdymo įrankis padedantis valdyti:

* Artefaktus/bibliotekas (*Builds*)
* Dokumentaciją (*Documentation*)
* Ataskaitas (*Reporting*)
* Priklausomybes (*Dependencies*)
* Programinės įrangos konfigūracijas (*SCMs*)
* Versijų leidimus (*Releases*)
* Paskirstymą (*Distribution*)

**Artefakto gyvavimo ciklo fazės**

| **Fazė** | **Paaiškinimas** |
| --- | --- |
| validate | Patikrina ar projektas korektiškas ir visa reikiama informacija yra pasiekiama |
| compile | Kompiliuoja projekto išeities kodą |
| test | Naudojant nurodytą testavimo karkasą testuoja sukompiliuotą išeities kodą |
| package | Supakuoja sukompiliuotą kodą į paskirstymo formatą, pvz JAR |
| integration-test | Įdiegia paketus į aplinką, kurioje vykdomi integraciniai testai |
| verify | Patikrina ar paketas atitinka kokybės kriterijus |
| install | Įdiegia paketą į lokalią saugyklą tam, kad kiti lokalūs projektai galėtų jį naudoti |
| deploy | Įdiegia paketą į nutolusią saugyklą tam, kad kiti projektai ar programuotojai galėtų jį naudoti |
| clean | Pašalina visus failus, kurie buvo sugeneruoti ankstesnio build'o metu |
| site | Generuoja projekto dokumentaciją |

**POM**

POM yra projekto objektinis modelis (*Project Object Model*). POM yra pagrindinė *Maven* projekto dalis, joje yra visa projekto konfigūracija. POM yra rašomas XML ženklinimo kalba ir įprastai failo pavadinimas yra pom.xml.

**POM struktūra**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<!-- Pagrindai -->

<groupId>...</groupId>

<artifactId>...</artifactId>

<version>...</version>

<packaging>...</packaging>

<dependencies>...</dependencies>

<parent>...</parent>

<dependencyManagement>...</dependencyManagement>

<modules>...</modules>

<properties>...</properties>

<!-- Artefakto sudarymo nustatymai -->

<build>...</build>

<reporting>...</reporting>

<!-- Papildoma informacija apie projektą -->

<name>...</name>

<description>...</description>

<url>...</url>

<inceptionYear>...</inceptionYear>

<licenses>...</licenses>

<organization>...</organization>

<developers>...</developers>

<contributors>...</contributors>

<!-- Aplinkos nustatymai -->

<issueManagement>...</issueManagement>

<ciManagement>...</ciManagement>

<mailingLists>...</mailingLists>

<scm>...</scm>

<prerequisites>...</prerequisites>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<distributionManagement>...</distributionManagement>

<profiles>...</profiles>

</project>

**POM elementai**

* groupId: Elementas, kuris yra bendras visai organizacijai arba projektui, pvz.: lt.codeacademy.learn
* artifactId: Projekto vardas, pvz.: my-project. Kombinacija groupId ir artifactId turėtų būti unikali visame pasaulyje.
* version: Projekto versija
* packaging: Paketo formatas. Galimos reikšmės: pom, jar, maven-plugin, ejb, war, ear, rar. Numaytoji reikšmė - jar.
* dependencies: kitų artefaktų priklasomybės. Pavyzdys:
* <project xmlns="http://maven.apache.org/POM/4.0.0"
* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
* xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
* https://maven.apache.org/xsd/maven-4.0.0.xsd">
* ...
* <dependencies>
* <dependency>
* <groupId>junit</groupId>
* <artifactId>junit</artifactId>
* <version>4.12</version>
* <type>jar</type>
* <scope>test</scope>
* <optional>true</optional>
* </dependency>
* ...
* </dependencies>
* ...

</project>

* + groupId, artifactId ir version: artefakto pagrindiniai duomenys.
  + type: Galimos reikšmės: pom, jar, test-jar, maven-plugin, ejb, ejb-client, war, ear, rar, java-source ir javadoc. Numatytoji reikšmė - jar.
  + scope: Galimos reikšmės: compile, provided, runtime, test ir system. Numatytoji reikšmė - compile.
  + systemPath: Naudojamas tik tada, kai scope reikšmė yra system.
  + optional: true arba false.
  + exclusions: Galima nurodyti, kokios priklausomubės dalis nereikia. Pavyzdys:
  + <project xmlns="http://maven.apache.org/POM/4.0.0"
  + xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  + xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  + https://maven.apache.org/xsd/maven-4.0.0.xsd">
  + ...
  + <dependencies>
  + <dependency>
  + <groupId>org.apache.maven</groupId>
  + <artifactId>maven-embedder</artifactId>
  + <version>2.0</version>
  + <exclusions>
  + <exclusion>
  + <groupId>org.apache.maven</groupId>
  + <artifactId>maven-core</artifactId>
  + </exclusion>
  + </exclusions>
  + </dependency>
  + ...
  + </dependencies>
  + ...

</project>

* parent: Paveldėjimas. Pavyzdys: Turime "tėvinį" POM:
* <project xmlns="http://maven.apache.org/POM/4.0.0"
* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
* xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
* https://maven.apache.org/xsd/maven-4.0.0.xsd">
* <modelVersion>4.0.0</modelVersion>
* <groupId>org.codehaus.mojo</groupId>
* <artifactId>my-parent</artifactId>
* <version>2.0</version>
* <packaging>pom</packaging>

</project>

"Tėviniame" POM'e elementas packaging turi būti pom. Tada "vaikinis" POM atrodys taip:

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.codehaus.mojo</groupId>

<artifactId>my-parent</artifactId>

<version>2.0</version>

<relativePath>../my-parent</relativePath>

</parent>

<artifactId>my-project</artifactId>

</project>

* dependencyManagement: Supaprastina priklausomybių aprašymą, kai yra vienas "tėvinis" POM ir keli "vaikiniai" POM, kurie naudoja tas pačias priklausomybes. Pavyzdys:
* <!-- ParentProj pom -->
* <project>
* <dependencyManagement>
* <dependencies>
* <dependency> <!-- not much benefit defining alpha here, as we only use in 1 child, so optional -->
* <groupId>alpha</groupId>
* <artifactId>alpha</artifactId>
* <version>1.0</version>
* <exclusions>
* <exclusion>
* <groupId>zebra</groupId>
* <artifactId>zebra</artifactId>
* </exclusion>
* </exclusions>
* </dependency>
* <dependency>
* <groupId>charlie</groupId> <!-- not much benefit defining charlie here, so optional -->
* <artifactId>charlie</artifactId>
* <version>1.0</version>
* <type>war</type>
* <scope>runtime</scope>
* </dependency>
* <dependency> <!-- defining betaShared here makes a lot of sense -->
* <groupId>betaShared</groupId>
* <artifactId>betaShared</artifactId>
* <version>1.0</version>
* <type>bar</type>
* <scope>runtime</scope>
* </dependency>
* </dependencies>
* </dependencyManagement>
* </project>
* <!-- Child Proj1 pom -->
* <project>
* <dependencies>
* <dependency>
* <groupId>alpha</groupId>
* <artifactId>alpha</artifactId> <!-- jar type IS DEFAULT, so no need to specify in child projects -->
* </dependency>
* <dependency>
* <groupId>betaShared</groupId>
* <artifactId>betaShared</artifactId>
* <type>bar</type> <!-- This is not a jar dependency, so we must specify type. -->
* </dependency>
* </dependencies>
* </project>
* <!-- Child Proj2 -->
* <project>
* <dependencies>
* <dependency>
* <groupId>charlie</groupId>
* <artifactId>charlie</artifactId>
* <type>war</type> <!-- This is not a jar dependency, so we must specify type. -->
* </dependency>
* <dependency>
* <groupId>betaShared</groupId>
* <artifactId>betaShared</artifactId>
* <type>bar</type> <!-- This is not a jar dependency, so we must specify type. -->
* </dependency>
* </dependencies>

</project>

* modules: Naudojamas kelių projektų agregavimui. Pavyzdys:
* <project xmlns="http://maven.apache.org/POM/4.0.0"
* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
* xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
* https://maven.apache.org/xsd/maven-4.0.0.xsd">
* <modelVersion>4.0.0</modelVersion>
* <groupId>org.codehaus.mojo</groupId>
* <artifactId>my-parent</artifactId>
* <version>2.0</version>
* <packaging>pom</packaging>
* <modules>
* <module>my-project</module>
* <module>another-project</module>
* <module>third-project/pom-example.xml</module>
* </modules>

</project>

* properties: Galime turėti savybes, kurias galime naudoti visame POM'e. Pavyzdys:
* <project xmlns="http://maven.apache.org/POM/4.0.0"
* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
* xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
* https://maven.apache.org/xsd/maven-4.0.0.xsd">
* ...
* <properties>
* <maven.embedder.version>3.1.0</maven.embedder.version>
* <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
* <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
* </properties>
* ...
* <dependencies>
* <dependency>
* <groupId>org.apache.maven</groupId>
* <artifactId>maven-embedder</artifactId>
* <version>${maven.embedder.version}</version>
* </dependency>
* ...
* </dependencies>
* ...

</project>

***Maven* alternatyvos**

* [Gradle](https://gradle.org/)
* [Ant](https://ant.apache.org/)
* Kitos alternatyvos <https://alternativeto.net/software/maven/>

**Naudingos nuorodos**

* Saugyklos:
  + <https://mvnrepository.com/>
  + <https://search.maven.org/>
* IntelliJ Maven konfigūracija <https://www.jetbrains.com/help/idea/maven.html>
* Naudingos bibliotekos: <https://github.com/akullpp/awesome-java>

**Tolesniam skaitymui**

* Maven <https://maven.apache.org/>
* POM <https://maven.apache.org/pom.html>

**Užduotys**

## Užduotys: Maven

## Nr. 1

### Užduotis

1. Atsisiųskite [javafaker](https://search.maven.org/search?q=a:javafaker) biblioteką ir pridėkite ją į Intellij projektą nenaudojant Maven.
2. Panaudokite javafaker <https://github.com/DiUS/java-faker> pvz. sugeneruokite 100 gatvių pavadinimų, 100 celsijaus laipsnio temperatūrų ir pan.
3. Sukurkite Maven projektą, įdėkite javafaker priklausomybę ir perkelkite klases iš ankstesio projekto, t.y. nerašykite kodo kviečiančio javafaker dar kartą.

## Nr. 2

### Užduotis

1. Sukurkite naują Maven projektą.
2. Į POM kaip priklausomybę įtraukite [lingua](https://github.com/pemistahl/lingua)
3. Sukurkite Java klasę
4. Java klasėje main metode sukurkite LanguageDetector
5. Leiskite vartotojui įvesti 10 žodžių ar sakinių
6. Vartotojui įvedus žodį ar sakinį iškart rodyti kokia kalba jis įvedė tekstą. Panaudoti detector.detectLanguageOf(...)

## Nr. 3

### Užduotis

Naudojant commons-codec biblioteką su Maven padarykite konsolinį login.

Pvz.: vartotojui rodome meniu: [1] registruotis; [2] prisijungti Tada pasirinkus [1] bus prašoma įvesti vartotojo vardą, vėliau - sugalvotą slaptažodį, ir dar po to - tą slaptažodį įvesti dar kartą. Po šių veiksmų vartotojo vardas ir užšifruotas slaptažodis išsaugomi į Map. Kai vartotojas pasirinks [2], tada programa turi paprašyti įvesti vartotojo vardą ir slaptažodį. Slaptažodį tikrinti su jau įvestu ir išsaugotu Map'e. Atspausdinti informaciją - pavyko prisijungti ar ne

Naudoti naujausią commons-codec versiją:

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

</dependency>

Užšifravimas:

String sha256hex = DigestUtils.sha256Hex("tekstas");

**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.

Priklausomybių paslėpimo naudojant mockito karkasą pavyzdys:

* Reikalingas mockito dependency:
* <dependency>
* <groupId>org.mockito</groupId>
* <artifactId>mockito-all</artifactId>
* <version>2.0.2-beta</version>

</dependency>

* Turime klasę Service:
* public class Service {
* private List<String> list = new ArrayList<>();
* public void loadList(List<String> list) {
* this.list = list;
* }
* public List<String> getList() {
* return list;
* }

}

* Turime klasę KlaseX, kuri naudoja Service:
* public class KlaseX {
* private Service service;
* public KlaseX(Service service) {
* this.service = service;
* }
* public int sarasoDydis() {
* return service.getList().size();
* }

}

* Turime testą, kuris tesuoja kalsės KlaseX metodą sarasoDydis:
* @Test
* public void sarasoDydisTest() {
* Service service = new Service();
* KlaseX klaseX = new KlaseX(service);
* assertEquals(3, klaseX.sarasoDydis());

}

tačiau toks testas neveiks, nes sąrašo dydis yra 0. Mes norime, kad klasė KlaseX kviesdama klasės Service metodą getList gautų netuščią sąrašą elementų, bet nenorime kviesti kitų Service klasės metodų, kurie leistų užpildyti sąrašą.

* Naudojant mockito toks veikianti testas atrodys taip:
* @Test
* public void sarasoDydisTest() {
* Service service = mock(Service.class);
* KlaseX klaseX = new KlaseX(service);
* List<String> mockedList = new ArrayList<>();
* mockedList.add("1");
* mockedList.add("2");
* mockedList.add("3");
* when(service.getList()).thenReturn(mockedList);
* assertEquals(3, klaseX.sarasoDydis());

}

mock(Service.class); nurodome, kad mūsų Service bus užmockintas.when(service.getList()).thenReturn(mockedList); nurodome kokį sąrašą grąžinti, kai bus kviečiamas serviso getListmetodas.

### Užduotys

* [Boulingo žaidimas](https://github.com/tadus21/code-academy-java-kursai/blob/master/unit-testing/exercises/bowling-game.md)
* [Asmens kodo generatorius](https://github.com/tadus21/code-academy-java-kursai/blob/master/unit-testing/exercises/official-id-generator.md)

### Šaltiniai:

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

**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:**

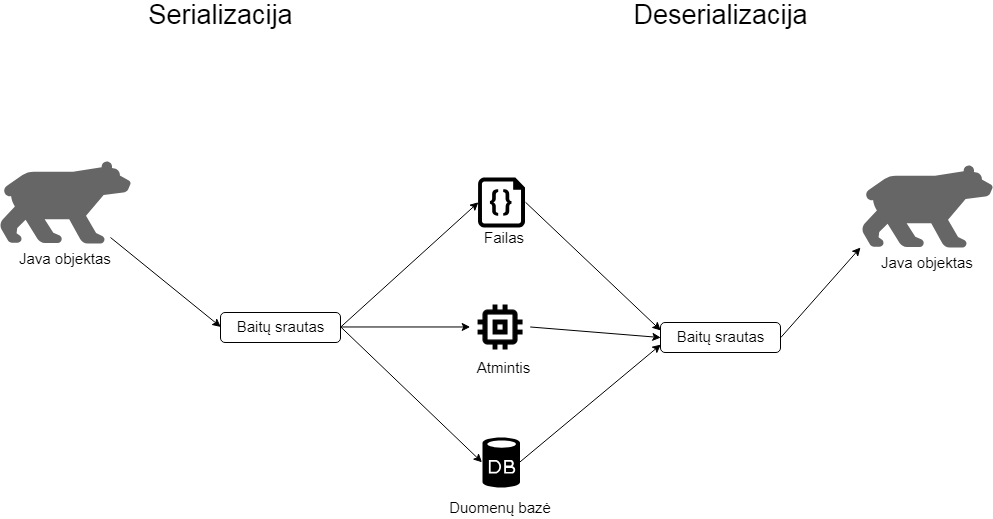
* <http://www.vogella.com/tutorials/JUnit/article.html>
* <https://junit.org/junit5/docs/current/user-guide/>

# 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.

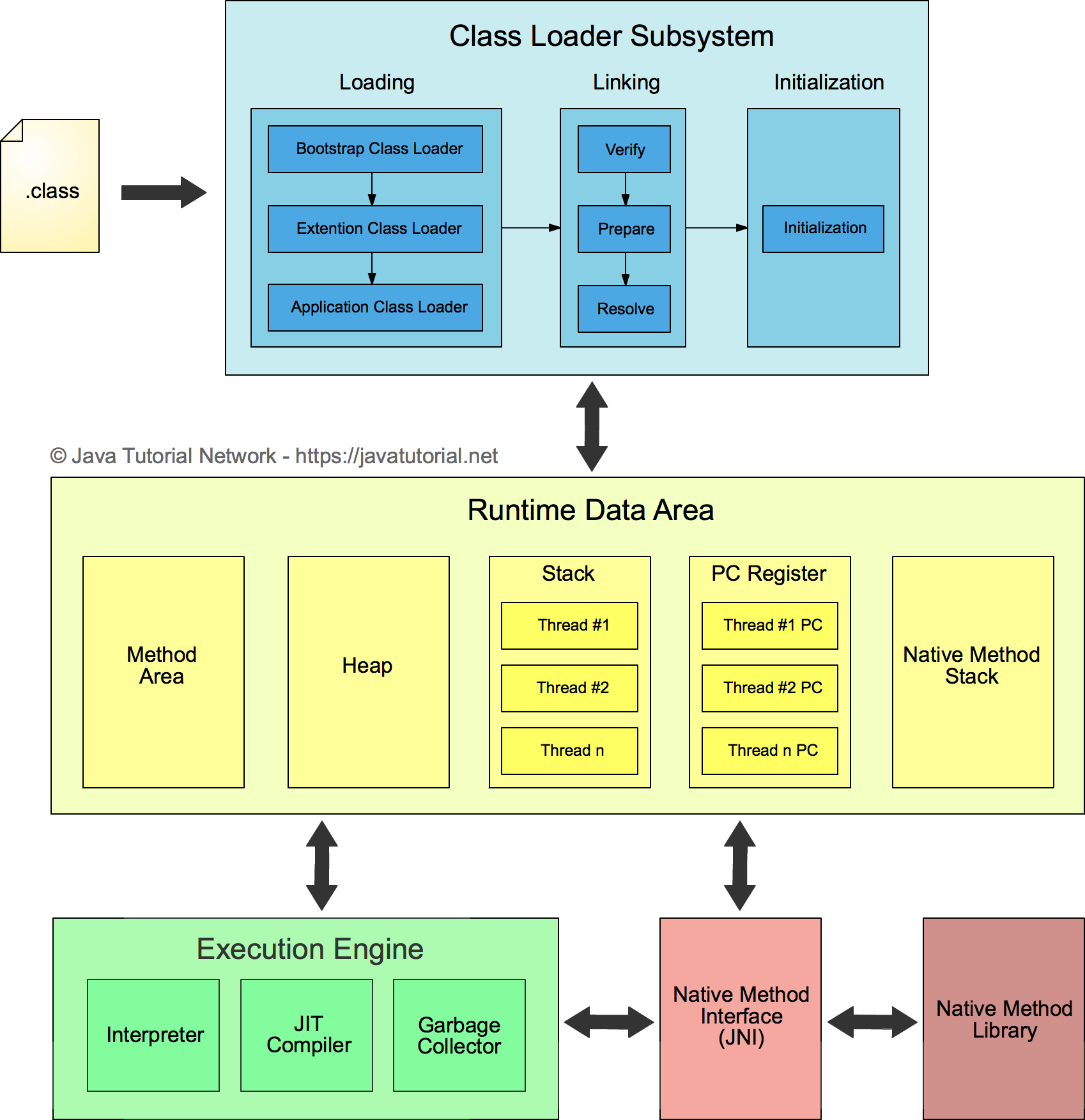
Vienas iš svarbių Java privalumų – gebėjimas veikti įvairiose platformose. Ji kurta remiantis idėja ‘‘Parašyti kartą, leisti bet kur“. Skirtingai nei kitos kalbos, Java šaltinio kodas pirmiausia sukompiliuojamas į bytecode - .class failą, kurį ir interpretuoja JVM.

### 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>

## Nr. 1

### Užduotis

1. Panaudojant HttpClient iškvieskite <https://jsonvat.com/> ir gaukite šalių PVM duomenis.
2. Remiantis JSON struktūra sukurkite atitinkamas klases su reikiamais laukais. Naudojant Jackson ObjectMapperdeserializuokite JSON į Java objektus. Klasės turi būti logiškai išskirstytos į paketus.
3. JSON esančią datą deserializuokite į LocalDate.
4. Naudojant srautus tarp duomenų palikite tik galiojančius periodus, t.y. kai šalis turi kelis periodus, turi likti tik naujausias. Surūšiuokite pagal šalies pavadinimą ir atspausdinkite tokiu formatu:

| Country | VAT |

-------------------------

| Austria | 20 |

| Belgium | 21 |

| Bulgaria | 20 |

| Croatia | 25 |

| Cyprus | 19 |

...

1. Sukurkite klasę CustomVatInfo su laukais name, reduced1, reduced2, standard ir kitais, jei reikia. Transformuokite Rateklasės objektus į CustomVatInfo. Turite gauti CustomVatInfo sąrašą
2. CustomVatInfo sąrašą sugrupuokite pagal standard ir atspausdinkite tokiu formatu:

17:

Luxembourg

18:

Malta

19:

Cyprus

Romania

Germany

...

## Nr. 2

### Užduotis

Sukurkite dvigubai surištą sąrašą DoublyLinkedList. Kiekvienas sąrašo elemntas turi turėti nuorodą į prieš jį esantį elementą ir nuorodą į po jo esantį elementą. Pražioje sukurkite klasę Node, kuri turi reikšmę ir turi nuorodas į elementus-kaimynus. Klasėje Node panaudokite Generics, kad į sukurtą sąrašą galėtumėte įdėti bet kokio tipo objektus. Po to sukurkite klasę DoublyLinkedList, kuri turi du klasės kintamuosius - pirmąjį sąrašo elementą ir paskutinįjį. DoublyLinkedList turi turėti tokius metodus"

* push - įdeda elementą į sąrašo pabaigą;
* pop - išima elementą iš sąrašo pabaigos ir grąžina elementą;
* shift - išima elementą iš sąrašo pradžios ir grąžina elementą;
* unshift - įdeda elementą į sąrašo pradžią.

Pavyzdys:

DoublyLinkedList<String> myList = new DoublyLinkedList<>();

myList.push("Code");

myList.push("Academy");

myList.unshift("Java");

System.out.println(myList.pop());

System.out.println(myList.shift());

System.out.println(myList.pop());

Rezultatas turi būti:

Academy

Java

Code