1918
TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

IDK0051 Objektorienteeritud programmeerimine Javas

Martin Rebane (martin.rebane@ttu.ee)

Korralduslikud teadaanded

- Järgmisel nädalal teooriatest ja kontrolltöö
- Konsultatsioon K kl 15:45 (ICT-405)
  - Pange nimi doodles kirja!! (link ained.ttu.ee)
  - Muu praks samal ajal? Palun proovige sel nädalal külastada mõnda teise rühma praksi, vajadusel räägin ise õppejõuga kui vaja (andke teada)

Muljeid eelmisest aastast

- „Teoreetiliselt teadsin, et kui ma end väga kindlalt ei tunne, siis lihtsam on konsultatsioonis käia, õppejõult nõu küsida ja kontrolltöö esimese korraga ära teha. Praktikas tahtsin siiski järgi proovida, kas saab ehk hea õnne korral niisama läbi. Nüüd õpin järeltööks.”

Statistika hindamismudeli kohta*

	Tudeng lootis oma teadmiste peale	Tudeng lootis hea õnne peale
Õppejõud hindas kontrolltöös teadmisi	100%	100%
Õppejõud hindas kontrolltöös head õnne	0%	0%
Hea õnn esines tudengi teadmistest sõltumatult	0%	0%

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

* Statistika aluseks on JOOPi tulemused viimasel kolmel aastal

Olulised teemad kontrolltööks

- Kõik seniõpitu ja eriti:

Static factory method

- *Static factory method* on alternatiiv konstruktorile – loote objekti staatilise meetodi sees ja tagastate selle

```
public class MyFactory {
      public static MyFactory getInstance() {
            return new MyFactory();
      }
}
```

NB! Võib tagastada sama klassi objekti kui ta mõne teise klassi objekti

Static factory method

- Viga töös: eeldatakse, et *static factory method* peab olema realiseeritud 1:1 nagu praksis/kodutöös/slaidil
- Lahendus: *static factory method*'i kasutus peab vastama äriloogikale (ülesandele)!

Optional

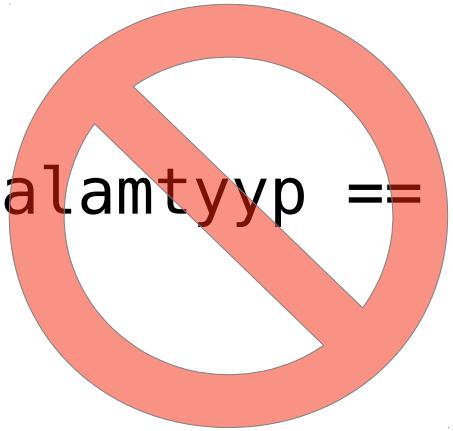
- Võimaldab selgelt väljendada, kui väärtuse puudumine on planeeritud stsenaarium

```
Optional<Product> product;
```

Mis vahet on of(), ofNullable() ja empty() meetoditel?

Polümorfism

- Võimalus käsitleda erinevat alamtüüpi objekte ühtse ülemtüübina
- Väldime if-tingimusi koodi sees, loome alamklasse



```
if (alamtyyp == 1) {
```

Mida saab kontrollida „==” abil ?

- Primitiivsete tüüpide korral sisulist võrdsust:

```
int a = 3;
int b = 3;
if (a == b) {
 // on võrdsed
}
```

Mida saab kontrollida „==” abil ?

- Objektitüüpide korral, kas tegu on sama objektiga:

```
Student a = new Student(„Mary”);
Student b = new Student(„Mary”);
if (a == b) {
 // ei ole samad
}
if (a.equals(b)) {
 // on võrdsed
}
```

equals()

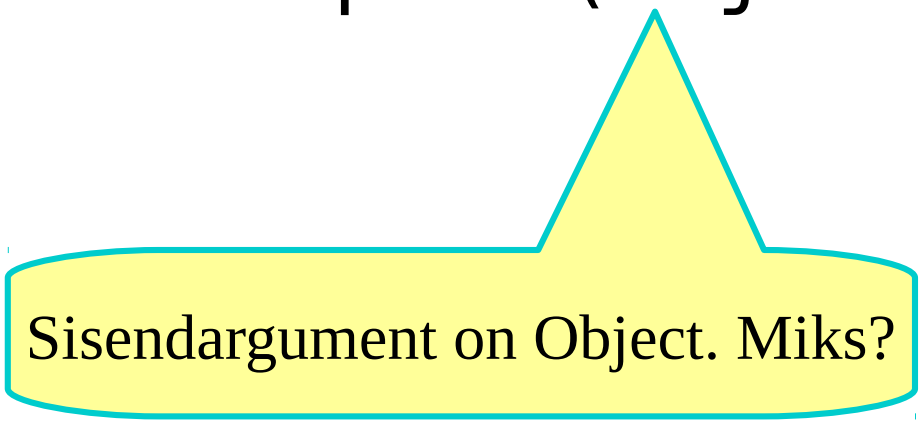
- Objektitüübi jaoks tuleb equals() üle kirjutada, vaikeimplementatsioon Object klassis kontrollib, kas tegu on sama objektiga

- Stringi (jt Java tüüpide) jaoks on Java arendajad selle töö teinud. Enda tüüpide jaoks peate ise equals()-i üle kirjutama

equals()

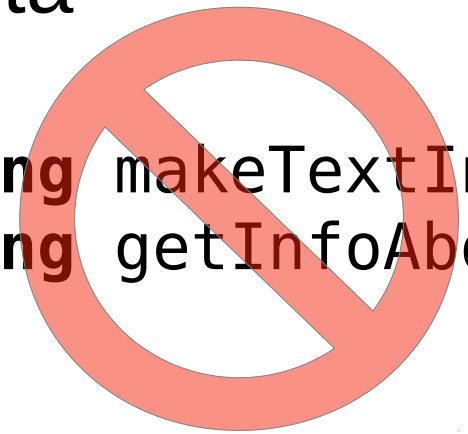
- NB! equals() meetodi signatuur on:

```
public boolean equals(Object o)
```



toString()

- Mugav võimalus anda tekstilist infot objekti kohta



```
String makeTextInfo()
String getInfoAboutObject()
```

Puhas kood!

- Clean code!

- NB! Paketi nimed väikese tähega
  - car
  - car.estate

Teemad

- Vaadake üle aine senised materjalid
- Segaseks jäänud kohtades lugege juurde Oracle Java Tutorialist

KT ja teooriatest

- Teooriatest 17. oktoobri loengu lõpus
  - Järelvastamist ei ole. Hiljem on võimalik teha ainult mõjuval põhjusel ja aegsasti enne 17. oktoobrit kokku leppides
  - Halvasti läinud tööd saab jaanuaris parandada teooriaeksamil
  - Kes mõlemad teooriatestid hästi sooritab, ei pea teooriaeksamit tegema (tavaline eksamitöö on ikka)
- Kontrolltöö teie praktikumi ajal
  - Programmeerimisülesanne, sarnaselt praktikumi- ja kodutöödega

Teooriatest

- Kordamisküsimused ained.ttu.ee lähipäevil

Eelmine nädal: objektide võrdlemine

- == kontrollib primitiivsete tüüpide korral sisulist võrdsust
- Objektitüüpide korral kontrollib, kas tegu on sama objektiga kuhjas (*heap*)
- Objektide sisuliseks võrdlemiseks tuleb kasutada equals() meetodit

equals()

- Objektitüübi jaoks tuleb equals() üle kirjutada, vaikeimplementatsioon Object klassis kontrollib, kas tegu on sama objektiga
- equals()-iga koos tuleb realiseerida hashCode()

hashCode()

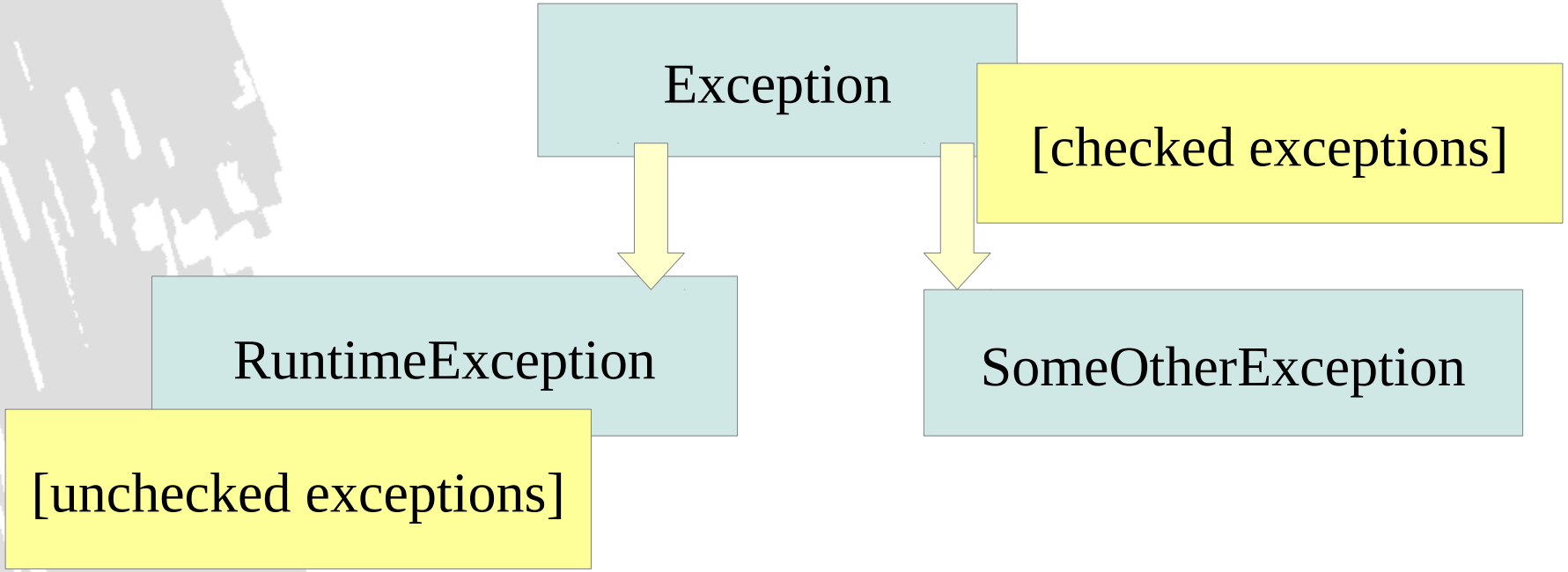
- Meetod, mis tagastab objekti väljadele vastava räsiväärtuse (*hash*i), mis ei tohi muutuda kui objekti olek ei muutu

- Räsiväärtused jaotuvad ühtlaselt üle võimalike väärtuste hulga. Väga suure tõenäosusega on kahe erineva olekuga objekti räsid erinevad, kuid võivad siiski kattuda.

Täna

- Erindite käsitlemine
- Sõltuvuse sisestamine
- Konstandid

Erindid on laiendatavad



Erindi loomine

- Esmalt mõtle, **kas erindit on vaja**!
- **Võib-olla** saab asendada if-else blokiga
- ```
  obj.doSomeStuff();
  ```
  versus
  ```
  if (obj.someStuffPossible()) {
      obj.doSomeStuff();
  }
  ```
  ```

Erindi loomine

- **throw** new StudentException();
- meetod peab informeerima kontrollitud erindi võimalikkusest:
  throws StudentException
- Kui veaolukord tekib – loo kõige sobivamat tüüpi erind
- Üldist tüüpi Exception objekti ei tohiks kunagi ise luua

Olulised meetodid

- `printStackTrace();`
  - kuvab informatsiooni vea esinemise kohta – nn veapinu
- `getMessage();`
  - kuvab veateate
- getCause()
  - tagastab pakitud erindi (kui on)

Enda erindite loomine

- Esmalt püüa kasutada Java erindeid
- Loomise eesmärgiks võib olla
  - pakkuda spetsiifilisemaid meetodeid vea põhjuste kohta
  - kasutamiseks programmisiseses veatöötluses (throw – catch / catch – rethow)

Erindite aheldamine e chaining

- Liiga üldine viga täpsemaks
- Liiga täpne viga üldisemaks
- Mitme vea liitmine

Kus püüda erindit?

- Kui püüda probleemi tekkekohale lähemal, siis on rohkem võimalusi probleemi lahendamiseks
- Kui püüda võimalikult kõrgel tasemel, siis ei pea allpool liiga palju koodi erinditele kulutama (tsentraliseeritud veahaldus)

Säilita veaeelne olukord (nn failure atomicity)

- Kui erind tekib, püüa see luua nii, et objekti olek ei saaks muudetud
  - nii on võimalik objekti peale vea põhjuse kõrvaldamist edasi kasutada

Try with resources

- Kui soovite avada ressurssi, siis ei pea muretsema selle saatuse pärast

- ```
  try (FileInputStream i = new
  FileInputStream("file.inf")) {

  }
  ```
  ```

Finally..

- Finally blokk võimaldab lõputegevusi, mis tehakse igal juhul, sõltumata sellest, kas try täideti edukalt või mitte
- Üldiselt ei kasutata eriti

Erindi ignoreerimine

- Erindit saab ignoreerida, kui catch blokk tühjaks jätta

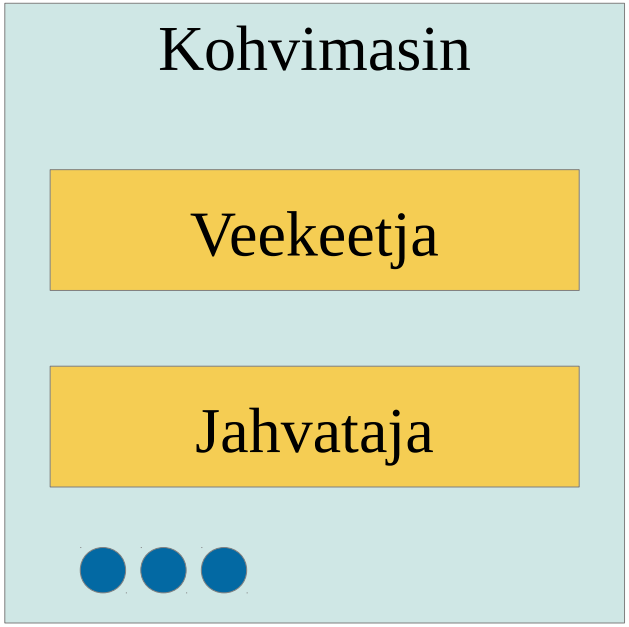
- See on **äärmiselt halb stiil** ja maksab teile varem või hiljem kätte!

Erindid - kokkuvõte

- Kaalu hoolikalt, kas erindit on vaja
- Kui erind luuakse, siis **ära ignoreeri**!
- Kontrollitud erind püüa alati kinni!

Dependency injection
Sõltuvuse sisetamine

Kohvimasin



Probleem

- Meil on kahte tüüpi kohvijahvatajaid – tavaline „põhjamaa keskmine” ja itaaliapärane espressojahvataja
- Mõlema API on samasugune
- Kuidas panna kohvimasin ära tundma, milline jahvataja tema sees on?

Probleem

- Masin on sisuliselt sama, tehases pannakse sisse erinevat sorti jahvataja

Lahendus

- Kui API (e avalikud meetodid) on sama, siis kasutame objekte polümorfselt
- Kohvimasin ei peagi teadma, milline jahvataja tema sees on ➔ ei pea ka ise otsustama, millist objekti luua

Dependency injection (sõltuvuse sisestamine)

- Sõltuvuse sisestamine klassi
- Anname klassile mingi objekti, mida ta oma töös kasutab
- Vastuvõttev klass ei pea ise mõtlema, millist tüüpi objekti luua või kasutada

Näide

- Klass DataParser töötleb andmeid
- Andmed tulevad ja kirjutatakse tagasi DataService-tüüpi teenusega
- Olemas on mitu erinevat DataService teenust – millist kasutada?

Sõltuvuse sisestamine

- Mille poolest erineb sõltuvuse sisestamine tavalisest argumendist?

Näide

- **Sõltuvuse sisestamine (dependency injection):** klass kasutab mingit teenust, mis on spetsifitseeritud liidesega. Klassi kasutaja tarnib (sisestab) sellele liidesele vastava teenuse (sõltuvuse).

- Näiteks teie klass krüpteerib andmeid, kuid kasutaja saab määrata algoritmi, millega andmeid krüpteeritakse.

Sõltuvuse sisestamine

```
public class DigiDocContainer {

private CryptoAlgorithm crypto;

    public DigiDocContainer(
            CryptoAlgorithm crypto) {
            this.crypto = crypto;
    }

}
```

Sõltuvuse sise

```
public class Di

private CryptoAlgorithm crypto;

    public DigiDocContainer(
            CryptoAlgorithm crypto) {
        this.crypto = crypto;
    }

}
```

DigiDocContainer kasutab krüpteerimiseks mingit **liidesele** CryptoAlgorithm vastavat algoritmi objekti

Konkreetne realisatsioon (algoritm) sisestatakse nt konstruktoris või setteriga

Soovitud tulemus

- Klass, kuhu sõltuvus sisestatakse, vabaneb loogikast, mis korraldab sõltuvuse tüübi valimist ja loomist
- Klass tegeleb vaid enda vastutusalaga

Inversion of control

- Põhimõte, et üldisem kood kontrollib spetsiifilisemat
- Sõltuvuse sisetamine on levinuim vorm
- Tihti kasutatakse tänapäeval sünonüümina (nt Spring raamistikus)

Unit-testimine

- Mida tuleb meetodi juures testida?

Mida testida?

- Kõiki erineva käitumisega vahemikke
- Piirväärtusi (ülemik ühelt käitumiselt teisele)
- Veaolukordi

Test-driven development

- Test enne, kood pärast
- Mis on selle mõte?

Test on kui spetsifikatsioon.
Kogu käitumine saab testitud.
Uue käitumise lisamisel jääb
vana tööle või läheb test katki.

final

final võtmesõna

```
final int i = 1;
```

- final võtmesõnaga tähistatud elementi saab vaid ühe korra väärtustada

`final` klassi ja meetodi korral

- final class – ei saa luua alamklasse
- final meetod – ei saa üle kirjutada

`final` võtmesõna välja korral

- Primitiivse muutuja korral on `final` fikseeritud kogu programmi töö ajaks
- Objektitüübi korral on viit fikseeritud programmi töö ajaks (objekti olek võib muutuda)

`final` võtmesõna välja korral

- Final muutuja initsialiseerimine peab toimuma deklareerimisel **või konstruktoris**

Final konstandi defineerimisel

- Konstant – primitiivset tüüpi `final static` väli

```
public static final double PI =

3.14159265358979323846;
```

NB! Nimi kokkuleppeliselt suure tähe ja alakriipsudega

Võimalikke nimekujusid

- AMOUNT
- MIN_LENGTH
- MAX_INBOX_SIZE
- APPLICATION_NAME

Konstandi kasutuskohad

- Muutumatud suurused klassis:
  - Fikseeritud tüübid
  - Arvulised väärtused
  - Piirid (igasugu maksimum- ja miinimumväärtused)
  - Töö ajal muutumatud nimed
  - Kõikvõimalikud muud mittemuutuvad väljad

Konstandi kasutamine

- Reeglina **staatiliselt**:

  Math.PI

  Student.MIN_EAP

  Klassinimi.KONSTANDI_NIMI

Praktiline nipp

- Eclipses kirjutage toores väärtus koodi
- Refaktoreerige:
  - Refactor – Extract Constant

Konstandi piirangud

- Kui väärtusi on palju, siis on jama!

```
private static final String CAR_REGULAR =
                                    "Auto";
private static final String CAR_SPORT =
                                "Sportauto";
private static final String CAR_ESTATE =
                  "Universaalkerega auto";
```

Lahendus - enum

- Enumeraator on spetsiaalset tüüpi klass
- Selle kõige lihtsam vorm ja kasutus on erinevate tüüpide spetsifitseerimine

Nüüd...

Kevad Tartus?

Aasta välismaal?

- Tartust naastes planeerige aasta vahetusüliõpilasena välismaal :-)
- Üliõpilasvahetus on väga oluline hariduskvaliteedi tagamisel ja valdkond, kus TTÜ on praegu üsna nõrk
- Valige tugev ülikool

Korralduslikud teadaanded

- **Lahendage kindlasti selle nädala praktikumi töö!**
- Tähtis on asjast aru saada, mitte definitsioone õppida

Korralduslikud teadaanded

- Järgmisel nädalal **teooriatest** loengu lõpus ja **kontrolltöö**
- Konsultatsioon K 15:45 ICT-405 – pane ennast kirja ained.ttu.ee

Foorum

- Aitäh kõigile, kes on juba ained.ttu.ee foorumit kasutanud!
- Kasutage julgesti ja võimalusel vastake ka oma kaastudengite küsimustele :)