

Johdanto

Java-kielessä käytetään poikkeuksia (exceptions) virheiden ja muiden poikkeuksellisten tilanteiden käsittelyyn. Poikkeukset ovat olioita, jotka periytyvät luokasta `java.lang.Throwable`. On myös mahdollista luoda omia poikkeuksia. Poikkeusten käsittely antaa ohjelmoijalle mahdollisuuden käsitellä virheitä hallitusti. Poikkeus tarkoittaa virhetilannetta tai muuten arvaamatonta tilannetta.

Sisällysluettelo

Poikkeukset Javassa	1
Johdanto.....	1
Throwable –luokasta	1
Error ja Exception –luokat.....	2
Poikkeusten käsittely	2
Throw –lause	3
Omat poikkeukset	4
Poikkeusten käyttö.....	4
Mainituista poikkeuksista.....	5
NullPointerException.....	5
ArrayIndexOutOfBoundsException.....	5
LinkageError	5
VirtualMachineError.....	5
Lähteet:	5
Kirjallisuus:	5
Internet:.....	5

Throwable –luokasta

`Throwable` –luokka on kaikkien Java -kielen poikkeusten yläluokka. Javan virtuaalikone ja `throw` -lause voivat heittää (siitä nimi `throwable`) vain tämän luokan – tai jonkin sen alaluokan – instansseja. Lisäksi `catch` –lohkolle voidaan asettaa parametrina vain `Throwable` luokan tai sen alaluokkien tyyppisiä objekteja. `Try-catch-finally` –rakenteesta lisää poikkeusten käsittelyn yhteydessä.

Koska poikkeukset ovat olioita, ne voivat sisältää tietoa ja niille on voitu määritellä metodeja. `Throwable` –luokka sisältää metodeja, jotka joko palauttavat poikkeuksen aiheuttaneen virheen kuvauksen sisältävän merkkijonon tai syyn(cause). Poikkeuksen syyllä tarkoitetaan toista `Throwable` objektia, joka on aiheuttanut kyseisen poikkeuksen heittäminen.

`Throwable` –luokassa ja sen alaluokissa on tavallisesti metodien lisäksi myös kaksi konstruktoria, joista yhdellä ei ole parametreja ja toinen

ottaa parametrinaan `String` -tyyppisen argumentin, jonka avulla voidaan palauttaa poikkeuksen aiheuttaneen virheen kuvaus. Lisäksi, jos alaluokasta on mahdollista palauttaa syy, niin tällä alaluokalla on oltava vielä kaksi konstruktoria. Toinen näistä ottaa parametrinaan `Throwable` (syy) -tyyppisen argumentin, ja toinen sekä `String` (virheen kuvaus) että `Throwable` (syy) -tyyppiset argumentit. [b]

Error ja Exception -luokat

`Error` ja `Exception` -luokat ovat `Throwable` -luokasta suoraan periytyviä aliluokkia. `Error` -luokan virheet ovat ohjelmoijasta riippumattomia peruuttamattomia virheitä. Tyypillisin `Error` -luokan virhe on `OutOfMemoryError`, joka kertoo varatun muistin loppuneen ohjelman ajon aikana. `Error` -luokan poikkeuksia ei tulisi yrittää ottaa kiinni, sillä ne ovat hyvin poikkeuksellisista tilanteista johtuvia eikä niitä tavallisen ajon aikana tulisi syntyäkään. `Exception` -luokka puolestaan on varattu poikkeuksille, joita hyvin toteutetussa ohjelmassa saattaisi olla järkevä yrittää ottaa kiinni.

`Exception` -luokka (pakkauksessa `java.lang.Exception`) jakautuu edelleen `RuntimeException` -luokkaan ja muihin luokkiin. `RuntimeException` -luokka on tehty ohjelmointivirheiden käsittelyyn. Tästä luokasta periytyvät esimerkiksi hyvin tutut `NullPointerException` ja `ArrayIndexOutOfBoundsException`. `RuntimeException` -luokasta tai siitä periytyvistä luokista heitetyt poikkeukset tulee korjata, jonka jälkeen niitä ei enää tarvitse käsitellä. Muun tyyppisistä poikkeuksista esimerkkinä `NoSuchMethodException`, joka ilmenee kun kutsuttavaa metodia ei löydy.

`Error` -luokan (pakkauksessa `java.lang.Error`) heittämät poikkeukset ovat harvinaisempia kuin `Exception` -luokan. Yleisimmin ne ovat joko `LinkageError` - tai `VirtualMachineError` -luokan objekteja. Tällaisten poikkeusten (`Error`) ilmetessä Java-tulkki yleensä tulostaa virheviestin ja poistuu. [1, s. 100-101]

Poikkeusten käsittely

Javassa poikkeuksia käsitellään `try`, `catch`, `finally` -lohkojen avulla. Kyseiseen rakenteen käyttö on hyvin miellyttävää, sillä se ei vaikeuta koodin lukemista. `Try` -lohkon sisään suljetaan koodilohko, joka saattaa heittää virheen ajettaessa. Poikkeukset tulee liittää kyseiseen `try` -lohkoon yhden tai useamman `catch` -lohkon avulla. On syytä mainita, että `try` -lohko ei voi esiintyä yksinään.

`Catch` -lohko seuraa `try` -lohkoa välittömästi, eikä muuta koodia voida kirjoittaa kyseisten lohkojen väliin. Jokainen `catch` -lohko käsittelee vain sille argumenttina annetun tyyppisiä poikkeuksia. Argumentin tyyppi kertoo poikkeuksen tyyppin (esim. `NullPointerException`), joka tulee olla jonkin `Throwable` -luokan alaluokan tyyppinen. nimi -muuttujan avulla voidaan käyttää `Throwable` -luokan ja poikkeuksen oman luokan metodeja.

`Finally` -lohko suoritetaan aina kun `try` -lohko on olemassa, mikä varmistaa sen, että `finally` -lohko tulee suoritettua vaikka odottamaton poikkeustilanne ilmaantuisi (poissulkien sen, että Javan virtuaalikone lopettaa suorituksen tai koodin lopettaminen lakkaa `try` -lohkossa).

Finally –lohkon käyttöä suositellaan myös ”siistimiskoodia” kirjoittaessa. Siistimiskoodilla tarkoitan koodia, joka esimerkiksi poistaa turhia varattuja muistipaikkoja. Tällainen tilanne saattaa ilmaantua, jos try –lohkon sisällä alustetaan olio, mutta oliota ei ehditä käyttää poikkeuksen ilmaantuessa. Tällöin finally –lohkon sisään voidaan sijoittaa komento, joka vapauttaa oliolle varatun muistin muuhun käyttöön.

Try, catch, finally –rakennetta käytettäessä koodi on suunnilleen seuraavanlaista;

```
try {
    koodilohko, joka saattaa heittää virheen
} catch (PoikkeuksenTyyppi1 nimi){
    koodia, joka suoritetaan kyseisessä poikkeustilanteessa
} catch (PoikkeuksenTyyppi2 nimi){
    koodia, joka suoritetaan kyseisessä poikkeustilanteessa
} finally {
    koodiosuus, joka suoritetaan joka tapauksessa
}
```

Edellä kuvattiin kuinka poikkeuksia voidaan käsitellä try, catch, finally –lohkojen avulla. Joskus on kuitenkin soveliaampaa siirtää poikkeusten käsittely metodien kutsuhierarkiassa ylemmälle tasolle ja antaa metodia kutsuneen metodin huolehtia mahdollisista poikkeuksista. Jos siis kutsumassamme metodissa ei huolehdi poikkeusten sieppauksesta, metodin on ilmoitettava poikkeukset, jotka se saattaa heittää. Koodissa se näyttää kutakuinkin tältä;

```
public void jokinMetodi( ) throws PoikkeuksenTyyppi,
    PoikkeuksenTyyppi2{
    koodi, jonka metodin on tarkoitus suorittaa
}
```

Tässä tapauksessa esimerkin metodin on mahdollista heittää kahden tyyppisiä poikkeuksia. Throws –lause koostuu Javan avainsanasta **throws** ja poikkeusten listasta, jossa poikkeuksia erottaa pilkku. throws –lauseella ei tarvitse käsitellä poikkeuksia, joihin vaikuttavat käyttäjän antamat tai esimerkiksi erillisestä tiedostosta luettavat syötet. [c]

Throw –lause

Ennen kuin mikään poikkeus voidaan siepata, jonkin koodilohkon on luotava (poikkeuksethan ovat olioita!) ja heitettävä se. Riippumatta siitä mikä poikkeuksen heittää, heittäminen tapahtuu aina throw –lauseen avulla. Kuten jo edellä mainittiin heitettävien poikkeusten on oltava tyypiltään jonkin Throwable –luokan aliluokan olioita.

Throw –lause vaatii argumenttinaan ainoastaan yhden throwable –tyyppisen objektin. Yksinkertaisuudessaan;

```
throw new jokinThrowableOlio;
```

Kun poikkeus on heitetty, ohjelman suoritus keskeytyy ja siirrytään lähimpään `try, catch`-lohkoon. Heitettäessä poikkeusta on myös mahdollista liittää poikkeukseen virheviesti, johon päästään myöhemmin käsiksi `Exception`-luokan metodilla `getMessage()`.

```
throw new jokinThrowableOlio("Virhe! Mogari!!");
```

[c] [1, s. 105]

Omat poikkeukset

Kuten aikaisemmin on jo mainittu, ohjelmoijan on mahdollista luoda omia poikkeusluokkia, jonka tulee olla `Exception`-luokan aliluokka. Omien poikkeuksien käsittely on samanlaista kuin valmiidenkin. Poikkeusluokan tulee sisältää valmiiden poikkeusluokkien tavoin vastaavia konstruktoreja. Tavanomaisesti konstruktorin, joka ei vaadi parametreja ja `String`-tyyppisen olion parametrinaan vaativan konstruktorin. Omien poikkeusluokkien luomisen tarpeellisuutta kannattaa tosin harkita.

Poikkeusten käyttö

Poikkeusten käyttö on hyvin mielekästä, mutta kannattaa kuitenkin pohtia, onko käyttö perusteltua ja voiko poikkeuksellisen tilanteen välttää tai käsitellä jollakin muulla tavalla. Poikkeusten luonti ja heittäminen syö resursseja, mikä saattaa hidastaa ohjelman suorittamista varsinkin kun samanaikaisesti ohjelmassa on tapahtunut virhetilanne. Esimerkiksi jo edellä mainittu ystävämme `NullPointerException` on vältettävissä `if`-lauseen oikeanlaisilla ehdoilla. Vastaavasti jos tilanne on odotettavissa, kuten se että taulukosta ei joka kerta löydy haettavaa arvoa, poikkeuksen heittäminen ei ole tyylikästä.

Poikkeuksia tulee käyttää oikeasti poikkeuksellisissa tilanteissa. Esimerkiksi jos vain metodin kutsuja tietää miten poikkeuksellisessa tilanteessa tulee menetellä tai metodin kutsumisen seurauksia ei ole määritelty jossakin tilanteessa. Esimerkiksi `jono`-olion metodi `pop()` palauttaa normaalioloissa jonon seuraavan olion. Kuitenkin jos kyseinen `jono` on tyhjä, `pop()`-metodin palauttama arvo ei ole määritelty. Tilanne voidaan välttää ainoastaan sillä, että tyhjästä jonosta ei alunperinkään yritetä ottaa mitään. Tässä tapauksessa poikkeuksen käyttö on perusteltua.

Lisäksi poikkeusten käyttöä suunniteltaessa kannattaa huolehtia siitä, että poikkeus aiheutetaan mahdollisimman varhain. Tällä tavoin vältetään virheen hukkaamiselta eikä virhe ehdi aiheuttamaan toista poikkeusta. Virhelähde on myös huomattavasti helpommin löydettävissä, mitä aikaisemmin poikkeus ilmenee. [3, s. 330-332]

Mainituista poikkeuksista

NullPointerException

`NullPointerException` on `RuntimeException` -luokan aliluokka. Kyseinen poikkeus heitetään, kun ohjelma yrittää käyttää arvoa `null` tilanteessa, jossa vaaditaan objektia. Esimerkiksi jos metodilta pyydetään jotakin oliota, joka täyttää tietyt ehdot, mutta haluamaamme oliota ei löydy ja palauttaa arvon `null`. Tämän jälkeen yritetään kutsua haluamamme oliolle (jota meillä ei ole, vaan se on `null`) vaikkapa nimen antavaa metodia – aiheutuu `NullPointerException`.

ArrayIndexOutOfBoundsException

`ArrayIndexOutOfBoundsException` on `IndexOutOfBoundsException` -luokan aliluokka, joka on `RuntimeException` -luokan aliluokka. Se heitetään, kun ohjelma yrittää ottaa taulukosta arvoa indeksiltä joka on taulukon ulkopuolella.

LinkageError

`LinkageError` on `Error` -luokan aliluokka. Poikkeus aiheutuu, kun luokalla on jokin riippuvuus johonkin toiseen luokkaan, joka ei ole enää yhteensopiva ensimmäisen luokan kanssa ensimmäisen luokan kääntämisen jälkeen.

VirtualMachineError

`VirtualMachineError` on `Error` -luokan aliluokka. Poikkeus heitetään tilanteessa, jossa Javan virtuaalikone on rikki tai sen ohjelman ajoa varten tarvittavat resurssit ovat loppuneet.

[a] [b]

Lähteet:

Kirjallisuus:

- [1] Niemeyer, Patrick; Knudsen, Jonathan. Learning Java. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.
- [2] Peltomäki, Juha; Malmirae Pekka. Java – Java-ohjelmoinnin peruskirja. Teknolit, Porvoo, 2000
- [3] Vesterholm, Mika; Kyppö, Jorma. Java-ohjelmointi. Talentum, Helsinki 2006

Internet:

- [a] <http://www.freshsources.com/Apr01.html>
- [b] <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Throwable.html>
- [c] <http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html>