Elina Juutilainen

Puutteellinen poikkeuksien käyttö Java-ohjelmissa

Tietotekniikan kandidaatintutkielma

20. joulukuuta 2018

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Elina Juutilainen

Yhteystiedot: elina.s.m.juutilainen@student.jyu.fi

Ohjaaja: Sanna Mönkölä

Työn nimi: Puutteellinen poikkeuksien käyttö Java-ohjelmissa

Title in English: Defective exception usage in Java-programs

Työ: Kandidaatintutkielma

Sivumäärä: 18+0

Tiivistelmä: Tutkielmassa tehdään yleisselvitys siitä, millaisia käsityksiä ohjelmoijilla on Javan poikkeuksienhallinnasta sekä mitä havaintoja poikkeuksienhallintamenetelmien käytöstä ja poikkeuksienkäsittelyyn liittyvistä virhetilanteista on saatu. Ohjelmoijilla on harvoin selkeitä poikkeuksienkäsittelyyn liittyviä menettelytapoja. Huonoina pidettyjä käytänteitä esiintyy laajasti Javan ohjelmissa, ja niihin liittyy myös virhetilanteita. Toisaalta poikkeuksienkäsittelyyn liittyviä virheitä on odotettua vähemmän, ja niistä tehtyihin virheraportteihin vastataan nopeasti.

Avainsanat: Java, Javan poikkeukset, ohjelmointi, ohjelmointivirheet

Abstract: The subject of this thesis is to review what developers think about exception handling in Java programming language and ehat observations have been made about defective exception handling and the exception handling faults in Java programs. Programmers rarely have clear procedures for handling exceptions. Used studies also show that bad exception handling practices are common. Exception handling bugs are also often connected to bad practices, however they are not reportedly more common than bugs caused by other issues and they are patched fast.

Keywords: Java, Java exceptions, programming, programming bugs

Kuviot

Taulukot

Sisältö

1 Johdanto

Moderneissa ohjelmistoissa esiintyy aina poikkeuksellisia tilanteita, joissa ohjelman kulku poikkeaa odotetusta polusta. Vakaan ohjelman tulee kuitenkin pystyä palautumaan poikkeuksellisista tilanteista. Poikkeuksenhallinnalla tarjotaan menetelmiä virheistä palautumiseen ja virheviestien välittämiseen. Ohjelmointikielet pyrkivät tarjoamaan kehittäjille rakenteita, joilla pystytään mahdollisimman vaivattomasti käsittelemään poikkeustilanteet ja palautumaan niistä ohjelman normaalitilaan. Tehokasta poikkeuksenhallintaa pidetään tärkeänä osana vakaata ja laadukasta ohjelmistoa.

Poikkeuksenhallintamenetelmät ovat kehittyneet ohjelmointikielten mukana, ja poikkeuksien tehokkaasta ja oikeanmukaisesta käytöstä on kirjoitettu useita suosituksia ja menetelmiä on tutkittu paljon. On kuitenkin huomattu, että ohjelmoijat käyttävät poikkeuksenhallintaan tarkoitettuja rakenteita laajasti suositusten vastaisesti, ja on tärkeää huomioida miten ohjelmoijat käyttävät ohjelmointikielten rakenteita työkalujen ja poikkeuksienhallintamenetelmien kehittämisen kannalta.

Java on yksi laajimmin käytettyjä ohjelmointikieliä, joten siihen keskittymällä voidaan tarkastella laajasti poikkeuksien käyttöä erityyppisissä ohjelmissa. Lisäksi siinä käytetyt menetelmät perustuvat oliopohjaisen ohjelmoinnin periaatteisiin, joten Javan poikkeuksienhallintamenetelmien tarkastelemisella voidaan tarkastella myös muiden oliopohjaisten ohjelmointikielien poikkeuksienkäsittelyä.

Tutkielmassa tehdään yleisselvitys siitä, miten Java-kielen ohjelmoijat käyttävät Javan tarjoamia rakenteita poikkeuksienhallinnassa ja ajonaikaisista virheistä palautumisessa, sekä selvitetään, millaisia poikkeuksenhallintaa koskevia virheitä Java-ohjelmissa esiintyy.

Luvussa 2 esitellään Javan tarjoamia poikkeuksenhallintaan liittyviä rakenteita ja yleisiä määritelmiä. Luvussa 3 käydään läpi poikkeuksien käytölle esitettyjä suosituksia sekä selvitetään, millaisia käsityksiä Java-ohjelmien kehittäjillä on poikkeuksien käsittelystä. Luvussa 4 kerrotaan, mitä havaintoja poikkeuksien käytöstä on

tehty sekä millaisia virhetilanteita esiintyy poikkeuksienhallintaan liittyen ja lopuksi luvussa 5 esitetään yhteenveto johtopäätöksistä.

2 Javan poikkeukset

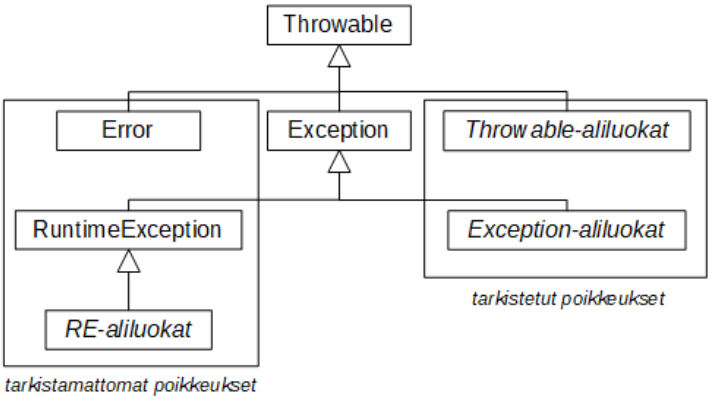
Java tarjoaa poikkeusten käsittelyyn try-catch-rakenteen, jonka avulla pyritään erottamaan poikkeuksia käsittelevä koodi muusta ohjelmakoodista. Try-lohko sisältää yhden tai useamman lausekkeen, joka voi heittää poikkeuksen. Lohkoa seuraa yksi tai useampi catch-lohko, joiden sisällä käsitellään syntynyt poikkeus. Catch-lohkolle annetaan argumenttina poikkeusluokka, johon kuuluvat poikkeukset lohkon halutaan käsittelevän. Yhteen try-lohkoon voidaan liittää useampi catch-lohko eri poikkeusluokille.

Poikkeuksellisessa tilanteessa poikkeuksen voi heittää joko Javan virtuaalikone ajon aikana tai ohjelmoija voi itse aiheuttaa poikkeuksen koodissa throw-avainsanaa käyttäen. Ohjelmoijan aiheuttamat metodia koskevat poikkeukset voidaan dokumentoida javadoc dokumentointityökalun @throws-avainsanalla.

Javan poikkeukset jaetaan oliopohjaisen ohjelmoinnin periaatteiden mukaan eri poikkeusluokkiin, joista nostetaan esiin erityisesti Throwable, Exception, Error ja RuntimeException-luokat, jotka on esitetty kuviossa 1. Kaikki poikkeusluokat periytyvät Throwable-yliluokasta. Error-luokan poikkeukset ovat poikkeuksia, joista tyypillisen ohjelman ei odotetakaan palautuvan, ja usein niiden kohdalla pyritään keskeyttämään ohjelman suoritus mahdollisimman nopeasti. Throwable-luokan Exception-aliluokkaan kuuluvat luokat ovat sen sijaan tarkoitettu ajonaikaisiin virheisiin, joista ohjelman toivotaan palautuvan. Näihin kuuluu esimerkiksi IllegalArgumentException-poikkeus, joka on tarkoitus heittää tilanteissa, joissa metodille annettujen parametrien arvot eivät ole oikeanlaiset.

Exception-luokasta periytyy erityisesti RuntimeException-luokan poikkeukset. Näihin kuuluu esimerkiksi IndexOutOfBounds-poikkeusluokka, joka syntyy taulukon ulkopuolelle viittauksesta. RuntimeException-luokan poikkeukset ilmoittavatkin pääasiassa ohjelmointivirheistä, ja niitä käytetään lähinnä ohjelman debuggauksen apuna. Niihin varautumista ei pidetä välttämättömänä.

Javan poikkeukset ovat monista muista ohjelmointikielistä poiketen myös jaettu



Kuvio 1. Javan poikkeusluokkahierarkia.

tarkistettuihin poikkeuksiin (checked exception) ja tarkistamattomiin poikkeuksiin (unchecked exception). Tarkistamattomiin poikkeuksiin kuuluvat kaikki RuntimeException ja Error-luokkien aliluokat, ja ne ovat siis poikkeuksia, joiden annetaan oletusarvoisesti keskeyttää ohjelman suoritus. Tarkistettuihin poikkeuksiin kuuluvat kaikki muut poikkeusluokat, siis Throwable aliluokat, jotka eivät ole RuntimeException tai Error-luokkien aliluokkia. Javan kääntäjä vaatii tarkistettujen poikkeuksien käsittelyn, ja niihin kuuluvat poikkeusluokat ovat tärkeimpiä ajonaikaisista virheistä palautumisen kannalta.

3 Käsityksiä poikkeuksienhallinnasta

Seuraavissa luvuissa käsitellään Javan poikkeuksienkäsittelylle esitettyjä suosituksia. Lisäksi käydään läpi, mitä tietoja on kerätty ohjelmoijien yleisistä käsityksistä poikkeuksienkäsittelyyn liittyen.

3.1 Poikkeuksien käytölle esitettyjä suosituksia

Poikkeuksien käytölle on esitetty useita suosituksia, joilla pyritään tukemaan ohjelmoijia poikkeuksienkäsittelyssä. Tässä luvussa nostetaan esille tutkielman näkökulmasta keskeisiä poikkeuksienhallintaan liittyviä suositeltuja käytänteitä. Suositusten lähteenä käytetään Joshua Blochin kirjaa Effectiva Java, jonka suositukset perustuvat Java-kielen spesifikaatioon sekä yleisiin kokeneiden Java-ohjelmoijien havaintoihin (Bloch 2018).

Älä jätä poikkeuksia huomioimatta
Poikkeuksen sivuuttaminen aiheuttaa ohjelman jatkamisen virheellisessä tilassa. Jos poikkeusta ei voida käsitellä, sen heittäminen eteenpäin aiheuttaa ohjelman kaatumisen nopeammin ja säilyttää tiedot poikkeuksista debuggausta varten. On kuitenkin tilanteita, joissa poikkeuksen jättäminen käsittelemättä on perusteltua, jos voidaan varmistaa, että poikkeuksen jälkeisestä tilasta voidaan jatkaa normaalisti. Tällöinkin tilanne pitäisi kuitenkin kommentoida catch-lohkon sisään ja aiheutuneesta poikkeuksesta on myös suotavaa kirjata tieto lokitiedostoon.

Poikkeusluokkien käytöstä
Bloch kehottaa välttämään yleisten Throwable, Exception, RuntimeError ja Error-luokkien käyttöä suoraan ja suosittelee valitsemaan aina mahdollisimman tilannetta kuvaavan poikkeusluokan. Javan omia poikkeuksia suositellaan kuitenkin käytettäväksi omien poikkeuksien sijaan, sillä niiden käyttö on kannattavaa kirjastojen uudelleenkäytön näkökulmasta.

Heitä oikea poikkeus rajapintaan nähden

Ohjelmarajapintojen välillä pyritään välttämään alemman tason toteutuksen yksityiskohtien vuotaminen korkeammille tasoille, jottei ohjelmistotasojen väliset abstraktitasot rikkoudu. Tällöin ylemmän tason toteutuksen muuttuessa voidaan joutua lähtemään muuttamaan myös alemman tason toteutusta. Bloch suositteleekin käyttämään poikkeuksien ketjuttamista. Ketjuttamisessa korkeamman tason metodi ottaa kiinni alemmalla tasolla aiheutuneet poikkeukset ja sisällyttää ne uuteen poikkeukseen, joka voidaan perustella korkeamman tason toteutuksella. Ketjuttamisesta voi olla myös hyötyä esimerkiksi debuggauksen kannalta, sillä sen avulla saadaan hyödyllistä tietoa poikkeuksen aiheutumiskohdasta.

Dokumentoi kaikki metodin poikkeukset

Sekä tarkistettujen että tarkistamattomien poikkeuksien dokumentointi on tärkeää niiden tehokkaan käsittelyn kannalta. Ilman mahdollisten poikkeuksien dokumentointia, poikkeuksien käsittely on vaikeaa tai jopa mahdotonta luokkia ja rajapintoja käytettäessä. Dokumentointi on myös tärkeää tarkistamattomien poikkeuksien kohdalla, sillä ne syntyvät yleisimmin ohjelmointivirheistä. Tarkistamattomien poikkeuksien dokumentointi on siten osa metodin kuvausta ja antaa edellytykset metodin onnistuneelle käytölle.

3.2 Java-ohjelmoijien käsityksiä poikkeuksienhallinnasta

Yksi tärkeä tutkimuskohde on myös Java-ohjelmien kehittäjien käsitykset poikkeustenhallinnasta. Poikkeuksienkäsittelyyn liittyvää koodia pidetään vähän ymmärrettynä osana ohjelmistojen kehittämistä. Ohjelmoijien käsitykset auttavat ymmärtämään poikkeuksien käyttöä sekä voivat olla tukena uusien menetelmien ja toimintatapojen kehittämisessä. Viitatuissa tutkimuksissa kehittäjien käsityksiä on kerätty ohjelmoijille annetuilla kyselyillä, joilla on kerätty tietoa muun muassa siitä, miten ja miksi ohjelmoijat itse kertovat käyttävänsä poikkeuksia, miten miellyttävää poikkeuksien käyttö on ja millaisia poikkeuksienhallintaan liittyviä virhetilanteita ohjelmoijat ovat korjanneet.

Shah, Gorg ja Harrold (2010) nostivat esille erityisesti, miten kehittäjien käsitykset poikkeuksien käsittelystä eroavat aloittelevien ja kokeneempien ohjelmoijien välillä. Tutkimuksessa aloittelevilla ohjelmoijilla oli keskimäärin kahden vuoden kokemus ja kokeneemmilla ohjelmoijilla yli viiden vuoden kokemus ohjelmistokehittämisestä. Yleisesti huomattiin kokeneempien ohjelmoijien pitävän poikkeuksienkäsittelyä tärkeämpänä kuin aloittelevien ohjelmoijien, jotka kertovat usein sivuuttavansa poikkeuksien käsittelyn ja pitävät sitä myös yleisesti epämiellyttävänä. Aloittelevat ohjelmoijat kertovat myös käyttävänsä poikkeuksienhallintaan liittyviä menetelmiä lähinnä debuggauksen tukena, eikä virheistä palautumisessa.

Kokeneemmat ohjelmoijat pitävät poikkeuksista johtuvaa polkua ja ohjelman pääfunktionaalisuutta yhtä tärkeinä, ja he myös pitävät poikkeuksienhallintaan liittyvää koodia huonolaatuisempana kuin aloittelevat ohjelmoijat (Shah ym. 2010). Shah ym. (2010) tutkimuksessa myös huomattiin, että kokeneet ohjelmoijat pitävät poikkeuksien käsittelyä tärkeänä sekä virheistä palautumisen kannalta että myös välittämään käyttäjälle hyödyllistä tietoa tapahtuneesta virheestä. Kehittäjien mukaan on tärkeää, että käyttäjälle muodostuu kuva mahdollisimman vakaasta ohjelmasta.

Ohjelmoijien kokemustaustasta riippumatta he ilmoittavat saavansa poikkeuksien käsittelyyn vain vähän organisaation tukea, eikä organisaatioissa, joissa ohjelmoijat toimivat, ole usein kehitetty yleisiä toimintatapoja poikkeuksien testausta ja dokumentointia varten (Ebert, Castor ja Serebrenik 2015). Shah ym. (2010) kertovat lisäksi, että myös kokeneemmat ohjelmoijat sanovat sivuuttavansa poikkeuksienkäsittelyn ohjelmakoodin testauksen aikana.

Ebert ym. (2015) tutkivat myös, mikä käsitys ohjelmoijilla on puutteellisesta poikkeuksienhallinnasta johtuvista virheistä. Vaikka poikkeuksien käsittelyä pidetään yleisesti huonona, tutkimukseen osallistuneista ohjelmoijista noin puolet on sitä mieltä, että poikkeuksienkäsittelyyn liittyvä koodi on laadultaan hyvää tai erittäin hyvää. Vain alle viidesosa pitää koodin laatua huonona (Ebert ym. 2015). Ohjelmoijat myös pitivät poikkeuksiin liittyviä virheitä helpommin korjattavina kuin muita ohjelmissa esiintyviä virheitä, mutta ohjelmoijien arvioiden mukaan niitä esiintyy enemmän kuin muista syistä johtuvia virheitä.

4 Havaittu poikkeuksien käyttö

Poikkeuksien käyttöä käytännössä on tutkittu louhimalla eri versionhallintatietokantojen Java-projektien lähdekoodia ja metatietoja. Seuraavissa luvuissa esitellään yleisiä huomioita poikkeuksien käytöstä sekä mitä poikkeuksiin liittyviä virhetilanteita on esiintynyt.

4.1 Havaitut erot suositeltuihin käytänteihin

Poikkeuksien käytöstä kiinnostavin tutkimuskohde on poikkeuksien käsittely eli catch-lohkojen sisältö. Eri menettelytapoja on luokiteltu muun muassa sen suhteen, paljonko koodia on käytetty poikkeusten käsittelyyn, millaisia poikkeusluokkia on käytetty sekä mitä metodeja poikkeusten käsittelyssä on käytetty. Havaintoja on myös verrattu erilaisiin poikkeuksien käytölle esitettyihin suosituksiin.

Taulukossa 1 koottu yleisimmät havainnot. Poikkeuksien sivuuttaminen on kaikissa tutkimuksissa erittäin yleinen havaittu tapa (Cabral ja Marques 2007; Asaduzzaman ym. 2016; Nakshatri, Hegde ja Thandra 2016; Kery, Le Goues ja Myers 2016). Yksinkertainen ja yleinen tapa sivuuttaa aiheutuneen poikkeuksen käsittely on yksinkertaisesti jättää sitä vastaava catch-lohko tyhjäksi. Lohkoja, joissa ei ole yhtään riviä koodia, esiintyy yleisesti tutkituissa ohjelmissa.

Huomioitavaa on myös tapaukset, joissa ei tehdä muuta kuin kutsuta Throwable-luokan printStackTrace-metodia, ja joita nähdään käytettävän myös useasti (Cabral ym. 2007, Asaduzzaman ym. 2016, Nakshatri ym. 2016, Kery ym. 2016).

Taulukko 1. havaitut käytänteet

Poikkeuksien sivuuttaminen
Liian yleiset luokat
Ketjuttaminen tyyppien välillä
Ei riittävää dokumentointia

Metodin avulla voidaan päätellä poikkeuksen lähtökohta, mutta ilman muita toimia poikkeuksia ei kuitenkaan käsitellä mitenkään. Kery ym. (2016) huomioivat myös, että printStackTrace luodaan oletussisältönä esimerkiksi suositun Eclipse-kehitysympäristön automaattisesti generoituihin try-catch-rakenteisiin, mikä viittaa siihen, että se on jätetty luotuun poikkeuksenkäsittelijään. Pelkkä printStackTrace-kutsun käyttö viittaa siis myös poikkeuksen sivuuttamiseen.

Syntyneen poikkeuksen kirjaaminen lokitiedostoon on yleisesti toivottava osa poikkeuksen dokumentointia. Poikkeuksenkäsittelyssä on kuitenkin huomattu myös paljon poikkeuksen kirjaamista lokitiedostoihin ilman muuta käsittelyä (Cabral ym. 2007, Nakshatri ym. 2016, Kery ym. 2016). Pelkkä kirjaaminen ja printStackTrace-kutsun käyttö onkin liitetty kehittäjien tapaan käyttää poikkeuksenhallintamenetelmiä laajasti debuggauksen apuna.

Ketjuttamisessa havaitaan myös paljon tilanteita, joissa tarkistettu poikkeus lähetetään eteenpäin RuntimeException-luokan tarkistamattomana poikkeuksena (Nakshatri ym. 2016). Se voi olla toivottavaa, mutta havaitaan tilanteita, joissa poikkeuksen tyyppi vaihtuu aina peräkkäin, kun se heitetään uudelleen eteenpäin, siis Tarkistettu – Runtime - Tarkistettu – Runtime -rakenteina (Ebert ym. 2015). Ebert ym. (2015) huomioivat, että vaikka tyyppien välinen ketjuttaminen ei ole välttämättä huono tapa, jatkuva tarkistettujen ja tarkistamattomien poikkeustyyppien välinen vaihto ei vaikuta tarkoituksenmukaiselta ja lisää sekavuutta.

Ohjelmoijat ottavat paljon kiinni Exception-luokan poikkeuksia tai jopa Throwable-luokan poikkeuksia ilman, että try-lohkoon olisi lisätty käsittelijöitä myös muille poikkeusluokille (Nakshatri ym. 2016, Kery ym. 2016). Yleisen poikkeusluokan käyttäminen vaikeuttaa poikkeuksen käsittelyä tai siitä palautumista.

Sena ym. (2016) huomasivat tutkituissa Javan kirjastoissa laajasti dokumentoimattomia poikkeuksia. Poikkeuksien dokumentointi on erityisen tärkeää Javan kirjastoissa, sillä kirjaston asiakasohjelman on vaikeaa tai täysin mahdotonta valmistautua poikkeukseen, jos siitä ei ole tarpeeksi hyödyllistä tietoa.

Huonoja käytänteitä havaittiin yleisesti kaikissa tutkimuksissa ja Asaduzzaman

ym. (2016) havaitsivat ohjelmoijien käyttävän poikkeuksienhallintamenetelmiä suositusten vastaisesti ohjelmoijan kokemustasosta riippumatta. Yleisesti voidaan huomata, ettei poikkeuksienkäsittelyyn näytetä käyttävän paljon huomiota, ja huonot käytänteet liittyvät paljon debuggaukseen eivätkä virheistä palautumiseen.

4.2 Poikkeuksenhallintaan liittyvät virhetilanteet

Virhetilanteita tarkastelemalla voidaan saada tietoa poikkeusten käsittelyn tärkeydestä vertailemalla virheiden määrää ja vakavuutta havaittuihin käytänteisiin. Poikkeusten tarkoitus on myös auttaa virheistä palautumisessa, joten on tärkeää tarkastella mahdollisesta huonosta poikkeustenhallinnasta nousevia virhetilanteita. Poikkeuksenhallintaan liittyvillä virhetilanteilla tarkoitetaan virheitä, jotka liittyvät puutteelliseen poikkeuksienkäsittelyyn. Esimerkiksi nollalla jakamisesta syntyvä poikkeus on ohjelmointivirhe, eikä kuulu poikkeuksenhallintaan liittyviin virhetilanteisiin, mutta poikkeuksen sivuuttamisesta johtuva virhe tai väärin heitetty poikkeus voidaan katsoa aiheutuneeksi huonosta poikkeuksienkäsittelystä.

Poikkeuksenhallintaan liittyviä virheitä pidetään vaikeasti raportoitavina (Ebert ym. 2015). Lisäksi kehittäjät yliarvioivat virheiden määrää, mutta aliarvioivat virheiden korjaamisen vaikeutta (Ebert ym. 2015). Ebert ym. (2015) toisaalta huomioivat, että virheiden yliarviointiin voi vaikuttaa vähäinen poikkeuskoodin testaus, minkä takia virheitä raportoidaan vähemmän kuin niitä todellisuudessa esiintyy. Virheraporttien mukaan poikkeuksiin liittyvät virheet ovat kuitenkin pääosin yhtä vaikeita korjata kuin muista syistä johtuvat virheet ja ne myös korjataan keskimäärin yhtä nopeasti kuin muut virheet (Ebert ym. 2015).

Sena ym. (2016) huomasivat, että suurin osa tutkituista ohjelmista ei dokumentoineet eksplisiittisesti heitettyjä RuntimeException-luokan poikkeuksia. Koska luokan poikkeukset johtuvat usein ohjelmointivirheistä, niiden käsittely ei ole välttämätöntä, mutta niiden dokumentointi on erittäin tärkeää. Coelho ym. pitivätkin puutteellista dokumentointia riskitekijänä ohjelman vakauden kannalta. Myös Ebert ym. (2015) tutkimuksen kyselyissä kehittäjät mainitsivat korjanneensa

virheitä, jotka johtuivat ajantasalla olemattomasta dokumentoinnista. Puutteelliseen dokumentointiin liittyy huonojen virheilmoituksen liittäminen poikkeuksiin sekä tilanteet, joissa poikkeuksia ei kirjata ylös lokitiedostoon (Barbosa, Garcia ja Barbosa 2014, Ebert ym. 2015).

Monissa tutkimuksissa huomattiin poikkeuksien virheellistä ketjutusta, joissa poikkeusta ei kapseloida oikein (Barbosa ym. 2014, Ebert ym. 2015). Tällöin alkuperäisen poikkeuksen alkuperää on vaikeampi selvittää. Samoin havaittiin paljon väärien poikkeusluokkien käyttämistä, kuten tilanteeseen sopimattoman tai liian yleisen poikkeuksen heittämistä eteenpäin. Barbosa ym. (2014) kategorisoivat ketjutusta ja poikkeusluokkia koskevat virheet yleisesti tilanteiksi, jossa poikkeuksen käsittelyssä hukataan hyödyllistä tietoa tapahtuneesta poikkeuksesta, mikä voi aiheuttaa epätoivottua käyttäytymistä ja vaikeuttaa ongelman diagnosointia ja virheestä palautumista.

Barbosa ym. (2014) luokittelivat toisen yleinen kategorian alle virhetilanteet, joissa ohjelma jatkaa poikkeuksen jälkeen suoritusta virheellisessä tilassa. Näihin kuuluvat tilanteet, joissa ohjelma päätyy poikkeukselliseen tilaan, mutta ohjelmoija ei heitä tarvittavaa poikkeusta. Lisäksi tyhjät catch-lohkot ja liian yleisten poikkeusluokkien ottaminen kiinni aiheuttavat ohjelman jatkamista virheellisessä tilassa. Samoja virheitä joita havaitsivat myös Ebert ym. (2015) tutkimissaan ohjelmissa. Molemmissa tutkimuksissa huomattiin myös tilanteita, joissa poikkeuksia käytettiin ei-poikkeuksellisissa tilanteissa ohjaamaan ohjelman kulkua.

Kaikissa tapauksissa huonoista poikkeuksia koskevista menettelytavoista ei kuitenkaan nostettu virheraporttia. Tyhjiä catch-lohkoja ja geneerisien tyyppien käyttöä catch-lohkoissa pidetään yleisesti huonoina käytänteinä ja niitä myös käytetään paljon. Niistä johtuvia virhetilanteita esiintyy paljon odotettua vähemmän (Ebert ym. 2015, Barbosa ym. 2014). Lisäksi virheraporttien mukaan tyhjiä catch-lohkoja käytettään myös raportoitujen virheiden korjaamiseen (Ebert ym. 2015). Ebert ym. (2015) pitävät kuitenkin tyhjiä catch-lohkoja kuitenkin riskinä ohjelman vakaudelle, koska ne eivät ole yleisiä poikkeuksia koskevia menettelytapojen mukaisia. Tällöin ei voida taata, että poikkeuksen sivuuttaminen ei riko ohjelmaa jatkossa.

Ebert ym. (2015) esittivät poikkeuksienkäsittelystä johtuvien virheiden syyksi sen, ettei organisaatioissa, joissa ohjelmoijat toimivat, poikkeuksien käsittelyä eikä niiden testausta pidetä tärkeänä, eikä poikkeuksienhallintaan ole kehitetty selkeitä menettelytapoja. Barbosa ym. (2014) tulivat samanlaiseen johtopäätökseen, ja ehdottivat syyksi selkeiden ohjeiden puuttumisen poikkeuksienkäsittelyn toteuttamisessa.

5 Yhteenveto

Poikkeuksien käyttöön liittyviä huonoina pidettyjä käytänteitä esiintyy Java-ohjelmissa paljon, ja virheraporteista huomataan, että niihin liittyy myös korjattavia virhetilanteita. Myös kehittäjät kertovat korjanneensa huonoihin käytänteisiin liittyviä virhetilanteita. Virhetilanteet eivät ole kuitenkaan sen vakavampia kuin muut virheet, ja raportoituihin virheisiin vastataan nopeasti.

Huonoina pidettyjä käytänteitä havaitaan kaikilla ohjelmoijilla kokemustasosta riippumatta. Yleisesti vaikuttaa siltä, että poikkeuksienhallintaan liittyviä menetelmiä käytetään debuggauksen tukena, ja varsinkin vähemmän kokemusta omaavat ohjelmoijat kertovat olevansa kiinnostuneempia debuggauksesta kuin virheistä palautumisessa. Lisäksi kehittäjillä on harvoin annettu selkeitä poikkeuksienhallintaan liittyviä toimintatapoja. Yhtenä ongelmakohtana nostettiinkin esille organisaatioiden huomion puute poikkeuksienhallinnassa.

Lähteet

Asaduzzaman, M., M. Ahasanuzzaman, C.K. Roy ja K. A. Schneider. 2016. "How Developers Use Exception Handling in Java?" Teoksessa *Proceedings of the 13th International Conference on Mining Software Repositories*, 516–519. Austin, Texas.

Barbosa, E. A., A. Garcia ja S. D. J. Barbosa. 2014. "Categorizing Faults in Exception Handling: A Study of Open Source Projects". Teoksessa *2014 Brazilian Symposium on Software Engineering*, 11–20. Maceio, Brazil.

Bloch, J. 2018. *Effective Java (3rd Edition).* Addison-Wesley.

Cabral, B., ja P. Marques. 2007. "Exception Handling: A Field Study in Java and .NET". Teoksessa *ECOOP 2007 – Object-Oriented Programming*, 151–175. Berlin, Heidelberg: Springer Berlin Heidelberg.

Ebert, F., F. Castor ja A. Serebrenik. 2015. "An exploratory study on exception handling bugs in Java programs". *Journal of Systems and Software* 106:82–101.

Kery, M. B., C. Le Goues ja B. A. Myers. 2016. "Examining Programmer Practices for Locally Handling Exceptions". Teoksessa *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 484–487. Austin, Texas.

Nakshatri, S., M. Hegde ja S. Thandra. 2016. "Analysis of Exception Handling Patterns in Java Projects: An Empirical Study". Teoksessa *Proceedings of the 13th International Conference on Mining Software Repositories*, 500–503. Austin, Texas.

Sena, D., R. Coelho, U. Kulesza ja R. Bonifácio. 2016. "Understanding the Exception Handling Strategies of Java Libraries: An Empirical Study". Teoksessa *Proceedings of the 13th International Conference on Mining Software Repositories*, 212–222. Austin, Texas.

Shah, H., C. Gorg ja M. J. Harrold. 2010. "Understanding Exception Handling: Viewpoints of Novices and Experts". *IEEE Transactions on Software Engineering* 36 (2): 150–161.