POIKKEUKSET JA TIEDOSTOT

Tavoite

Oppia poikkeusten käsittelyä sekä tiedosto-operaatioita Java-sovelluksissa.

Käsitteitä

poikkeus, try-catch-finally, try-with-resources, throw, tietovirta

Alkutoimet: Kopioi tehtäväsarjan koodit (**OlsoTehtSarja4**) kurssin tietovarastosta omaan repoosi ja luo Eclipse-projekti ensimmäisen tehtäväsarjan ohjeita soveltaen. Projektissa on tehtävässä mainitut luokat, mutta koodi puuttuu. Tests-hakemistossa olevat koodit eivät käänny, ennen kuin olet kirjoittanut tehtävien koodit.

1. Merkkijonomuuttujassa olevan numerotekstistä voi poimia kokonaisluvun käyttäen `Integer`-käreluokan metodia `Integer.parseInt(String merkkijono)`. Näin esimerkiksi merkkijono ”42” saadaan muunnettua `int`-tyyppiseksi kokonaisluvuksi 42. Jos muunnettavassa merkkijonossa on muita kuin numeromerkkejä, ei muunnos onnistu vaan syntyy poikkeus. Selvitä API-dokumentaation avulla mikä poikkeus on kyseessä (tee www-haku Java 8 API Integer).

   Kirjoita sitten ohjelma, joka lukee String-tyyppiseen muuttujaan henkilön iän ja reagoi ei-numeerisiin syötteisiin alla olevan esimerkin mukaisesti, kunnes syötteeksi saadaan kelvollinen ikä. Versionhallinnassa on tyhjä tehtäväpohja (`Poikkeus.java`). Esimerkissä syötteet on kursivoitu. Käytä `try/catch`-rakennetta.

   Tapaus 1:

   ```
   Anna ikäsi.
   23
   Vuoden päästä olet jo 24-vuotias.
   ```

   Tapaus 2:

   ```
   Anna ikäsi.
   eläkeläinen
   Antamasi ikä eläkeläinen ei ole kelvollinen.
   Anna ikäsi.
   99
   Vuoden päästä olet jo 100-vuotias.
   ```

   Tehtävään ei liity JUnit-testiä.

   Hyvin käyttäytyvän ohjelman tulee aina varautua siihen, että pyydettäessä käyttäjältä numeerista tietoa saatetaankin saada tekstiä. Käytännössä on siis luettava aina tekstinä, ja siitä sitten parsitaan numeroita try/catch-rakenteessa.

2. Tehtävä pohjustaa seuraavia tiedostonkäsittelytehtäviä.

   Toteuta luokka **Valtio**, jossa instanssimuuttujina ovat valtion nimi, pääkaupunki ja asukasluku. Toteuta niille setterit ja getterit. Tee kolmen parametrin konstruktori, jolla voi Valtio-oliota luotaessa asettaa arvot instanssimuuttujille (järjestys: nimi, pääkaupunki, asukasluku). Kirjoita myös metodi toString(), joka palauttaa merkkijonoesityksen valtion tiedoista.

   Aja testit (ValtioTest.java).

   Laadi luokkaan **ValtioMain** pääohjelma, joka kysyy käyttäjältä toistossa valtioiden tietoja, luo niiden pohjalta **Valtio**-olioita ja tallentaa oliot `HashMap`-tietorakenteeseen (katso OMA-työtilasta

Dokumentit | Tuntimateriaalit | 01 - Kertaus). Avaimena on valtion nimi. Tietojen syöttäminen lopetetaan antamalla syötteeksi tyhjä rivi. Muista käsitellä myös tilanne, jossa saman valtion tiedot syötetään useita kertoja. Toiston jälkeen ohjelma tulostaa valtioiden tiedot hajautustaulusta.

3. Laajenna edellistä pääohjelmaa siten, että valtioiden tulostuksen jälkeen se kirjoittaa tiedot olio-virtana tiedostoon `valtiot.dat`.

   Ohjelmoi tietojen kirjoittamista varten luokkaan **TiedostonKäsittely** staattinen metodi `kirjoitaTiedosto()`. Versionhallinnassa on valmis tehtäväpohja.

   Tässä pääsee helpoimmalla kirjoittamalla levylle kerralla koko `HashMap`-olion. Tiedosto tallentuu oletuksena projektin hakemistoon samalle tasolle kuin hakemistot src ja tests.

   Toteuta kaikki tarvittava poikkeustenkäsittely (try/catch-lohkot).

   Huom: Tee oliovirran käsittelyä varten Valtio-luokasta `java.io.Serializable`-rajapinnan toteuttaja. Tämän rajapinnan toteuttaminen ei edellytä itse kirjoitettavia metodeja.

   Aja testit (TiedostonKasittelyTest.java). Tässä vaiheessa riittää, että kirjoitustesti menee läpi.

4. Laajenna tehtävän 2 pääohjelmaa siten, että se lukee ensin tiedoston `valtiot.dat` sisältönä olevat valtiot hajautustauluun ennen valtiotietojen kysymisen aloittamista.

   Ohjelmoi lukemista varten luokkaan **TiedostonKäsittely** staattinen metodi `lueTiedosto()`, joka palauttaa lukemansa HashMap-olion. Koska HashMap-olio on kirjoitettu tiedostoon yhdellä kutsulla, niin lue se myös yhdellä kutsulla. Jos tiedostoa ei ole olemassa, metodi palauttaa arvon null.

   Toteuta kaikki tarvittava poikkeustenkäsittely (try/catch-lohkot).

   Aja testit (TiedostonKasittelyTest.java). Nyt molempien testien tulee mennä läpi.

5. Tutustu tietovirtojen tavutason- ja tietotyyppitason IO-operaatioihin (kirja s. 278-282, linkki PDF-kirjaan löytyy OMA-työtilasta). [Näille on käyttöä Lego-projektissa.]

   Annettuna on pääohjelman keskeneräinen runko (`PerusTyypitettyIO.java`), jossa kirjoitetaan ennalta määrättyyn tiedostoon kokonaislukutaulukossa olevat luvut. Sen jälkeen ohjelma lukee ja tulostaa tiedoston sisällön erikseen tavuina ja kokonaislukuina.

   Kirjoita pääohjelma valmiiksi ja toteuta staattiset metodit, jotka luokassa ovat tyhjiä.

   Vihjeitä:
   Ohjelma voi selvittää tiedoston tavumäärän `FileInputStream`-luokan metodilla `available()`. Huomaa, että tavujen määrä on eri kuin kokonaislukujen määrä, sillä Javan int on aina 32 bittiä eli 4 tavua pitkä.

   Varaudu tiedosto-operaatioiden poikkeuksiin (mm. `FileNotFoundException`) ja tulosta sopiva virheilmoitus.

   Aja testit (PerusTyypitettyIO.java)