EETU JUHANI ASIKAINEN

OHJELMOINTIOPAS LAPPEENRANNAN-

LAHDEN TEKNILLISEN YLIOPISTON LUT:N OLIO-OHJELMOINTIKURSSI

LISÄÄ JULKAISIJA?

PUBLISHED BY LISÄÄ JULKAISIJA?

TUFTE-LATEX.GOOGLECODE.COM

First printing, October 2021

Contents

Oppaasta ja sen käytöstä

Esipuhe

Tämä opas on kirjoitettu osaksi Lappeenrannan-Lahden teknillisen yliopiston LUT:n olio-ohjelmoinnin perusteiden kurssin kurssimateriaaleja. Opas käy läpi koko kurssin teoriasisällön lukuunottamatta Android-API:n käyttöä. Opas on tarkoitettu käytettäväksi osana kurssikokonaisuutta, mutta se on suunniteltu niin, että se tarjoaa laadukkaan perehdytyksen oliopohjaiseen ajatteluun, suunnitteluun ja ohjelmointiin myös itsenäisenä kokonaisuutena.

Oppaan tavoitteena on tarjota opiskelijoille helppokäyttöinen, laadukas ja suomenkielinen resurssi kurssin käsittelemien aihealueiden opiskeluun. Opas olettaa käyttäjän hallitsevan proseduraalisen ohjelmoinnin perusteet, jotka käydään läpi esimerkiksi Lappeenrannan-Lahden teknillisen yliopiston LUT:n "Ohjelmoinnin Perusteet" -kurssilla.

Oppaan rakenne

Opas on jaettu kahteentoista sisältölukuun ja kahteentoista bonuslukuun. Sisältöluvut käsittelevät kurssilla käsiteltäviä sisältöjä ja sisältävät kaiken kurssin sisältöön kuuluvan materiaalin. Sisältöluvut on numeroitu välillä 0-11 niin, että luvun numero vastaa viikkoa, jolla luvun sisältöä käsitellään kurssilla ja luku 0 käsittelee itse opasta ja sen käyttöä. Näin opasta on helppo seurata kurssin edetessä.

Jokaista sisältölukua vastaa oppaan toisella puoliskolla yksi bonusluku. Bonusluvut tarjoavat laajempia ja syvempiä katsauksia Javaan ja oliopohjaiseen ohjelmointiin. Ne eivät aina syvennä suoraan vastaavan sisältöluvun materiaaliin liittyviä aiheita, mutta myös niiden järjestys on suunniteltu niin, että käyttäjä voi opiskella yhden luvun viikossa.

BONUSLUVUT ON TARKOITETTU hyödyllisiksi ja hauskoiksi lisälukemistoiksi. Opas on suunniteltu niin, että bonusluvut eivät sisällä kurssin oppimistavoitteisiin luokiteltua sisältöä. Niiden lukeminen on siis täysin vapaaehtoista.

Termistö ja kieli oppaassa

OPPAAN PÄÄASIALLISENA kielenä koodiesimerkkejä lukuunottamatta toimii suomi muun kurssimateriaalin kielivalinnan mukaan. Oppaan käyttämä termistö on tämän vuoksi käännetty Javan avainsanoja lukuunottamatta suomeksi. Olisi kuitenkaan turha kiistää englannin ehdotonta ylivaltaa tietojenkäsittelytieteen pääkielenä. Tämän vuoksi oppaan sanastosta löytyy jokaisen termin selityksen lisäksi termin englanninkielinen vastine.

OPAS PYRKII korostamaan tärkeää termistöä sanaston opettelun helpottamiseksi. Jokainen tärkeä termi on kerätty oppaan sanastoon lyhyen selityksen ja termin englanninkielisen vastineen kera. Sanastosta löytyvät termit on myös korostettu ensimmäisessä esiintymisessään. Sanastotermeillä tämä korostus näkyy kursivoituna kirjoitusasuna ja termin perässä esiintyvänä sulkuihin suljettuna englanninkielisenä vastineena termille. Vastine on niin ikään kursivoitu. Koko korostettu kirjoitusasu termille näyttää siis tältä: *esimerkki (example)*

OPAAN TEORIATEKSTI sisältää sanastosta löytyvien konseptuaalisesti tärkeiden termien lisäksi myös Javan avainsanoja. Nämä on erotettu muusta korostetusta termistöstä alleviivauksilla ja kääntämättömyydellä. Ensimmäisellä esiintymiskerrallaan Javan avainsanat ovat lisäksi kursivoituja. Ensimmäistä kertaa esiintyvä Javan avainsana näyttää tältä: *example* ja uudestaan mainittu tältä: example.

Koodiesimerkit oppaassa

OPAS TARJOAA kattavat koodiesimerkit kaikista siinä käsiteltävistä aihealueista. Koodiesimerkit ovat lyhyitä, toimivia, koodinpätkiä ja ne sijaitsevat heti esitellyn konseptin yhteydessä. Kaikki oppaan koodiesimerkit on koottu yhteen *tänne*.

KOODIESIMERKIT ON KIRJOITETTU Javan virallisen tyylioppaan suosittelemien ohjelinjojen mukaisesti. Kaikki muuttujat, luokat ja metodit on nimetty englanniksi, mutta koodi on kommentoitu

suomeksi. Kommentointi ei noudata hyvää ohjelmointityyliä vaan kommentit ovat koodin opetusmaisen luonteen vuoksi monisanaisempaa ja yleisempää kuin suosituksissa.

Oliopohjainen ajattelu

Abstraktio laadukkaan koodin pohjana

Oliopohjainen ohjelmointi on ohjelmointiparadigma, joka on kehitetty vastaamaan ohjelmistotuotannon peruskysymykseen: kuinka kirjoittaa ymmärrettävää ja ylläpidettävää koodia helposti?

Yksi helpoimmista ja yleisimmistä tavoista selkeyttää koodikantaa on *abstraktio (abstraction)*: yleisten toimintojen ja kutsusarjojen eristäminen ennalta määriteltyihin koodipaloihin. Nämä palat tunnetaan nimellä *funktio (function)*. Tällä tavalla jaettuja kutsusarjoja voidaan uusiokäyttää ja ohjelman muokkaaminen helpottuu, kun kutsusarjan päivittäminen tapahtuu vain yhdessä keskitetyssä paikassa.

Käskysarjat ovat kuitenkin vain puolikas toimivasta ohjelmistosta. Toinen ja aivan yhtä tärkeä puoli on data, jota käskysarjoilla käsitellään. Myös tämän datan abstraktio on mahdollista proseduraalisen ohjelmoinnin puitteissa muokattavilla tietorakenteilla. Tällaisestä hyviä esimerkkejä ovat esimerkiksi Pythonin class-avainsanalla määritellyt luokat, jos niitä käytetään vain datan ryhmittämiseen tai C-kielen *tietue (struct)*.

Nämä abstraktiotyypit ovat kuitenkin erotettuja toisitaan, eivätkä huomioi käskysarjojen ja datan yhteyttä. Lähes poikkeuksetta tiettyjä funktioita kutsutaan koodissa monessa paikkassa parametreinään toistuvasti sama tietorakenne. Eikö siis olisi järkevää yhdistää datan ja sitä käsittelevien funktioiden sijainti koodissa?

Oliopohjaisen ajattelun perusteet

Olio-ohjelmoinnissa ohjelmisto koostuu kokonaan keskenään kommunikoivista yksiköistä. Näille yksiköille vakiintunut kutsuma-

nimi on *olio (object)*. Oliot koostuvat datasta ja sitä manipuloivista käskysarjoista, jotka tunnetaan nimellä *metodi (method)*. Se on oliopohjaisen ohjelmoinnin nimi olioon sidotulle funktiolle, termiä funktio ei juuri käytetä.

OHJELMISTO LUO kaikki käyttämänsä oliot ohjelmoijan tekemien muottien pohjalta. Tälläinen muotti, eli *luokka (class)* voi luoda itsestään yleensä rajattoman määrän olioita ajon aikana. Luokan ja olion ero on yksi oliopohjaisen ohjelmoinnin tärkeimmistä käsitteistä ja sen ymmärtäminen alusta lähtien on kriittistä oliopohjaisen ohjelmoinnin opiskelussa.

JOKAINEN OLIO on jonkin luokan *instanssi (instance)*. Saman luokan eri instanssit omistavat identtiset metodit, mutta instansseihin säilötty data on uniikki jokaiselle instansille ja instanssin metodit käsittelevät tätä uniikkia dataa. Luokka määrittää nämä metodit ja kertoo minkä tyyppiset datakentät jokaiselta luokan instanssilta löytyvät. Tästä lähtien opas käyttää termiä "luokka" viitatessaan ohjelmoijan määrittämään olio muottiin eli puhuttaessa suunnitteluperiaatteista ja koodin kirjoittamisesta ja termiä "olio" vain puhuessaan selkeästi koodissa sijaitsevasta luokan ilmentymästä.

TÄLLÄINEN JAOTTELU järjestää koodia proseduraalista koodia luonnollisempiin osiin. Tästä syystä oliopohjaista ohjelmointia esitellessä mainitaan usein sen kyky luoda koodin sisällä järjestelmiä, jotka muistuttavat todellisen maailman rakenteita. Vaikka tämä pitää paikkansa on hyödyllistä ymmärtää alusta lähtien, ettei oliopohjaisen ohjelmoinnin tavoitteena ole luoda yksi yhteen oikean maailman kanssa samoin toimivaa järjestelmää. Sen sijaan oikea oliopohjaisen ohjelmoinnin tavoite on luoda järjestelmä, joka koostuu pienistä tarkasti toisistaan erotelluista osista, jotka kommunikoivat keskenään vain selkeästi määriteltyjä rajapintoja pitkin.

KÄYTÄNNÖSSÄ LAADUKASTA oliopohjaista ohjelmistoa voidaan täten verrata esimerkiksi moderniin modulaarisesti suunniteltuun autoon. Siinä missä auton moottori, pyörät tai verhoilu ovat muokattavissa asiakkaan toiveiden ja budjetin mukaan, on laadukkaan oliopohjaisen ohjelmiston osien, kuten vaikka autorisaatiomoduulin tai tietokantayhteyden laajentaminen ja muokkaaminen mahdollista koskematta muuhun ohjelmistoon.

TÄMÄ KUULOSTAA kunnianhimoiselta ja monimutkaiselta tavoitteelta, joten on tärkeää edetä sopivan pienissä paloissa. Samoin kuin laadukas oliopohjainen ohjelmisto koostuu pienistä paloista, kasataan

ohjelmiston tekemiseen vaadittu tieto pieni pala kerrallaan. Tässä kappaleessa esiteltyjä termejä ja käsitteitä tullaan syventämään oppaan tulevissa kappaleissa yksitellen, joten vaikka tiedon määrä voi tuntua nyt kohtuuttomalta ei kannata huolestua liikaa.

Koodin laadun mittaamisesta

Jotta ohjelmistojen modulaarisuus saadaan toivotulle tasolle on tärkeää miettiä mitä metodeja tai dataa luokka paljastaa muulle ohjelmistolle. Oliopohjaiset kielet sisältävätkin yleensä jonkin keinon rajoittaa datakenttien ja metodien näkyvyyttä luokan ulkopuolelle. Tämä näkyvyyden rajaaminen ja tarkkojen vastuiden määrittäminen tunnetaan nimellä *enkapsulaatio (encapsulation)*.

Luokan paljastamat datakentät ja metodit kertovat ulkopuoliselle ohjelmistolle, mitä luokasta luodut oliot voivat tehdä ja mitä niiltä voi odottaa. Tämä muodostaa sopimuksen luokan ja muiden luokan paljastamia metodeja tai datakenttiä käyttävien luokkien välillä. Tätä sopimusta kutsutaan nimellä *rajapinta (interface)*. Rajapinnan abstraktia käsitettä ei pidä sekoittaa Javan samankaltaiseen "interface"-avainsanaan, joka esitellään myöhemmin.

Yksi olio-ohjelmoinnin nyrkkisäännöistä on pitää huolta, että jokainen luokka suorittaa yhden ja vain yhden tehtävän. Tämä tarkoittaa luokan rajapinnan pitämistä minimaalisena, eli korkeaa enkapsulaatiota. Tämän lisäksi ohjelmiston peruslaatua voidaan arvioida käsitteillä *koheesio (cohesion)* ja *pariutuminen (coupling)*.

Koheesio voidaan arvioida luokkakohtaisesti luokan metodien ja datakenttien yhteistoiminnan perusteella. Mitä tiiviimmin metodit ja datakentät ovat yhteistyössä ja mitä vähemmän luokassa on metodeja ja datakenttiä, jotka eivät ole vuorovaikutuksissa muun luokan kanssa, sitä korkeampi koheesio luokalla on. Korkea koheesio kulkee siis pitkälti käsi kädessä tiukan enkapsulaation kanssa. Mitä vähemmän luokka paljastaa itsestään muulle ohjelmistolle, sitä tiukemmin sen metodit ja datakentät yleensä komminkoivat keskenään.

Pariutuminen tarkoittaa ohjelmiston luokkien riippuvuutta toisista ohjelmiston luokista. Siitä puhutaan usein asteikolla *löysä (loose)-tiukka (tight)*. Löysästi pariutineessa ohjelmistossa jokainen luokka on riippuvainen vain muutamasta muusta luokasta ja riippuvaisuudet muistuttavat usein enemmän ketjua tai puuta kuin verkkoa.

Koheesio ja pariutuminen kulkevat usein käsi kädessä: Korkea koheesio johtaa löysään pariutumiseen ja matala koheesio tiukkaan pariutumiseen. Enkapsulaatio mukailee yleensä koheesiota niin että korkeasta koheesiosta seuraa korkea enkapsulaatio ja toisin päin. Kaikki kolme ovat tärkeitä mittareita puhuttaessa oliopohjaisen ohjelmiston laadusta. Vaikka oppaan tärkein tehtävä onkin opettaa lukija koodaamaan Javalla, eikä niinkään koodaamaan huipputason koodia, on näiden käsitteiden olemassaolo ja merkitys hyvä tiedostaa. Tämä auttaa ymmärtääkseen oliopohjaisten kielten suunnittelua ja etuja suhteessa muihin ohjelmointikieliin ja saattaa myös parantaa tuotetun koodin laatua huomaamattomasti.

Javan perusteet

Javan syntaksin alkeet

Java ohjelmointikielenä

JAVA on staattisesti tyypitetty korkean tason oliopohjainen ohjelmointikieli, joka pyrkii standardisoimaan ohjelmien kirjoittamisen alustasta riippumatta. Java-ohjelmat käännetään bittikoodiksi, joka ajetaan Java-virtuaalikoneella siten, että tismalleen sama koodi voidaan ajaa jokaisella alustalla, jolle virtuaalikone voidaan asentaa.

TÄSTÄ SYYSTÄ Java toimii tismalleen samoin kaikilla käyttöjärjestelmillä. Tämä yhdessä Javan helposti opittavan syntaksin ja oliopohjaiseen suunnitteluun pakottavan luonteen ohella on tehnyt Javasta yhden käytetyimmistä ohjelmointikielistä teollisessa ohjelmoinnissa.

JAVAN SYNTAKSI on saanut paljon vaikutteita C-kielestä. Muun muassa kaarisulkeiden käyttö koodin osien erotteluun ja suurin osa ohjausrakenteiden nimistä on suoraan C-kielestä kopioitua.

Kommentointi

SUURIN OSA ohjelmointikielistä tukee ainakin yhtä kommentointisyntaksia. Javassa on kaksi pääasiallista tapaa merkata koodissa oleva teksti kommentiksi. Yhden rivin kommentti voidaan aloittaa "//"-merkkiparilla, kun taas monen rivin kommentti suljetaan "/*"- ja "*/"-merkkien väliin. Seuraava esimerkki esittää kommentointisyntaksin vielä tarkemmin. Älä välitä vielä itse koodista, sen tehtävä on vain selventää miten kommenttien sijoittelu vaikuttaa koodin toimintaan.

Primitiiviset tietotyypit ja String

YKSIÄ TÄRKEIMMISTÄ koodin osista ovat ehdottomasti *muuttujat (variable)*. Ne ovat koodissa määriteltyjä tietokenttiä, jotka kykenevät säilömään joko jonkin luokan instanssin, tai kokonaislukumuodossa esitetyn simppelin datayksikön. Koska Java on *staattisesti tyypitetty kieli (statically typed language)*, täytyy siinä jokaisen muuttujan tietotyyppi määritellä muuttujan määrittelyn yhteydessä. Muuttujan tyyppi voi olla joko jokin käyttäjän määrittelemä luokka, jokin Javan standardikirjaston luokka, tai jokin *primitiivinen tietotyyppi (primitive data type)*.

JAVA SISÄLTÄÄ yhteensä kahdeksan primitiivistiä tietotyyppiä. Nämä määritellään pienellä alkukirjaimella kirjoitetulla tyypin nimellä. Seuraava taulukko ja koodiesimerkki sisältävät kaikki primitiivisten tietotyyppien nimet, arvojoukot, oletusarvot ja määrittelytavan koodissa.

Avainsana	Oletusarvo	Arvojoukko
bool	false	true ja false
byte	0	-2^7:stä $2^7 - 1$:een
short	0	-2^{15}:stä $2^{15} - 1$:een
int	0	-2^{31}:stä $2^{31} - 1$:een
long	0L	-2^{63}:stä $2^{63} - 1$:een
float	0.0f	noin $1.4e-45$:stä noin $3.4e+38$:aan
double	0.0d	noin $4.9e-324d$:stä noin $1.8e+308d$:hen
char	0	0:sta $2^{16} - 1$:een

```java
// Javassa on kaksi tapaa kommentoida kirjoitettua koodia. Ensimmäinen on aloittaa kommentti
// kahdella kauttaviivalla. Tämä merkitsee kääntäjälle rivin loppuosan olevan kommenttia, jollon
// kääntäjä jättää sen huomiotta Huomioi, että tämä toimii vaikka rivillä olisi ollut aikaisemmin
// koodia, eikä riko toiselle riville jatkuvia rakenteita. Tämän oppaan kommentit käyttävät
// pääasiassa tätä tyyliä. Välin jättäminen kaksoiskauttaviivan ja sitä seuraavan kommentin
// väliin ei ole pakollista, mutta sitä pidetään monessa tyylioppaassa oikeaoppisena tyylinä.

//Myös tämä on siis toimiva kommentti

package week2; // Pakettimääritelmä toimii vaikka sitä seuraa kommentti

public class CommentingExample { // Luokan määritelmän jälkeen laitettu kommentti toimii myös

    /* Toinen tapa kommentoida on ympäröidä kommentti kauttaviivoilla ja tähdillä tähän tapaan */

    /* Tälläinen kommentti voi ulottua monelle riville.
     * Monesti jatketut kommenttirivit aloitetaan tähdellä.
     * Tämä erottaa kommentin selkeämmin muusta koodista.
     */

    /* Mutta esimerkiksi myös tälläinen kommentti toimii,
    joskin se ei näytä kovin mukavalta ja on hankalampi lukea.
     */

    public static void main(){

        // /*-merkillä aloitetut kommentit voivat sijaita myös koodirivin alussa tai keskellä.
        // Tällainen asettelu toimii kunhan kommentit eivät katkaise avainsanoja.
        // Tämä ei ole suositeltavaa viimeistellyssä koodissa, mutta saatta olla hyödyllistä
        // kehitysvaiheessa.

        /* Tämä toimii */ System.out.println(/* Tämäkin toimii */"Toimii");
    }
}
```

Listing 1: Kommentointi Javassa

```java
package week2;

// Luokan luonnin notaatio käydään läpi myöhemmin tässä luvussa
public class BasicDataTypes {

    // Primitiivisten tietotyyppien luominen

    // boolean määritetäänd "true" tai "false" avainsanalla
    boolean booleanValue = true;
    // byte, short ja int voidaan kaikki määrittää normaaleilla lukuarvoilla
    byte maxByteValue = 127;
    short maxShortValue = 32767;
    int maxIntValue = 2147483647;
    // Huomaa long-tietotyypin määrittämiseen vaadittu L-pääte
    long maxLongValue = 9223372036854775807L;
    // float ja double ilmoitetaan f- ja d-päätteillä
    float floatValue = 1.1f;
    double doubleValue = 3.14159265358979323846264338327950288419716939937510582097494592d;
    // char-arvot määritellään ympäröimällä yksittäinen merkki tai unicode-koodi yksittäisillä
    // lainausmerkeillä (')
    char charValue = 'a';
    char unicodedCharValue = '\u0123';
}
```

Listing 2: Primitiiviset tietotyypit Javassa

NÄIDEN TIETOTYYPPIEN lisäksi Javasta löytyy monta valmiiksi määriteltyä luokkaa datan säilömistä varten. Näistä tärkein kielen opettelua aloitettaessa on ehdottomasti merkkijonoja säilövä *String*-luokka. Tämä luokka mahdollistaa dynaamisten merkkijonojen helpon luomisen ja manipulaation. String-luokka on voimakas työkalu ja sen toimintaan kannattaa kiinnittää huomiota Javan opiskelun alussa.

STRING-TYYPPISEN muuttujan arvo kerrotaan ohjelmalle antamalla String-tyypin muuttujalle toivottu merkkijono suljettuna normaalien lainausmerkkien (") sisään. Yksittäiset hipsut (′) eivät toimi, koska niitä käytetään char-tyypin muuttujien määrittämiseen.

```
package week2;

public class BasicString {

    public static void main(){
        String exampleString = "Merkkijonoesimerkki";
        System.out.println(exampleString);
    }
}
```

Listing 3: String-tyypin muuttujan määrittäminen

Luokkien ja olioiden perusteet

Oliopohjaisen ohjelman ajaminen

LUOKAT OVAT Javan ydin ja kaiken koodin rakennuspala. Kaikki koodi Javassa on suljettava luokan sisään, niin myös ohjelman ajon aloittava koodi. Tästä syystä jokaisessa Java-ohjelmistossa on yksi erikseen määritelty pääluokka, joka sisältää main-metodin. Tämä metodi määritellään avainsanoilla public static void ja sen *signatuuri (signature)* on main(String args[]). Void-avainsana selitetään myöhemmin kappaleessa ja static-avainsana kappaleessa *x*. Toistaiseksi riittää sisällyttää nämä avainsanat annetussa järjestyksessä main-metodin määritelmään.

Näkyvyysmääreet

Muuttujan luominen Javassa tukee tietotyypin määrittelyn lisäksi muuttujan näkyvyyden määrittämistä. Tämä määritelmä on nimeltään *näkyvyysmääre (access modifier)*. Kuten oppaassa aiemmin todettiin, yksi olio-ohjelmoinnin ydinideoista on *enkapsulaatio*, eli koodin osien vastuiden tarkka rajaaminen. Näkyvyysmääreet ovat tärkein työkalu tämän suhteen.

Avainsana	Ominaisuuden näkevät luokat
private	Vain tämä luokka
Ei avainsanaa (default)	Kaikki luokat tässä paketissa
protected	Kaikki luokat tässä paketissa ja kaikki tämän luokan perivät luokat
public	Kaikki luokat kaikkialla

Sanasto

abstraktio (abstraction) Ohjelmoinnin perustekniikka, jossa ongelman tarkka ratkaisu piilotetaan kutsuttavan koodirakenteen, kuten funktion, tietorakenteen tai luokan taakse. 9

enkapsulaatio (encapsulation) Datan piilottaminen olion sisään niin ettei muu ohjelmisto näe kyseistä dataa. Mitataan asteikolla matala-korkea, niin että korkea enkapsulaatio tarkoittaa pientä määrää julkisia metodeja tai datakenttiä ja matala taas suurta määrää julkisia metodeja ja datakenttiä see. 11, 17

funktio (function) Ohjelmoijan määrittelemä käskysarja, eli koodin osa, joka on rajattu, ottaa tietyn määrän parametreja ja mahdollisesti palauttaa paluuarvon. Tunnetaan olio-ohjelmoinnissa nimellä metodi. 9, 10, *see* parametri & metodi

instanssi (instance) Olio, joka on luotu jonkin luokan pohjalta on kyseisen luokan instanssi. 10, *see* olio & luokka

koheesio (cohesion) Ohjelmiston laadun mittaamiseen käytetty käsite. Mittaa luokkien sisäistä yhtenäisyyttä akselilla matala-korkea. Matala koheesio tarkoittaa että luokassa on paljon metodeja jotka eivät keskustele toisten luokan metodien kanssa ja matala että luokan kaikki metodit käyttävät useita muita luokan metodeja. Matala koheesio on toivottavaa, koska luokan tehtävä on tehdä yksi ja vain yksi asia. 11

luokka (class) Ohjelmoijan kirjoittama muotti, jonka pohjalta ohjelmisto luo olioita. Voi sisältää metodeja ja datakenttiä mutta yleensä näiden käyttämiseksi vaaditaan olion luontia. 10, *see* olio, metodi & instanssi

metodi (method) Luokkaan sidottu käskysarja, joka suorittaa ottamiensa parametrien ja luokan omien datakenttien perusteella

jonkin tietyn toiminnon. 10, *see* parametri, funktio & luokka

muuttuja (variable) koodissa määritelty tietokenttä, joka sisältää jonkin ohjelman käyttämän arvon. 14

näkyvyysmääre (access modifier) Muuttujan näkyvyyden määrittävä avainsana. 17, *see* muuttuja

olio (object) Luokan instanssi. Yksittäinen koodissa luotu toimija, joka sisältää datakenttiä ja metodeita. Luokka, jonka pohjalta olio luodaan määrittää olion käytettävissä olevat metodit ja sihen tallennetut datatyypit, mutta vain olio pääsee käsiksi omiin metodeihin ja datakenttiinsä. 10, *see* luokka, metodi & instanssi

parametri (parameter) Arvo, jonka funktio tai metodi ottaa muulta koodilta vastaan. *see* funktio & metodi

pariutuminen (coupling) Ohjelman laadun mittaamiseen käytetty käsite. Mittaa luokkien keskenäisten riippuvuuksien määrää akselilla löysä-tiukka. Löysässä pariutumisessa ohjelmiston luokkien väliset riippuvuudet ovat harvassa, jolloin ohjelmiston muokkaaminen on helppoa. Tiukassa pariutumisessa puolestaan jokaisella ohjelmiston luokalla on riippuvuus moneen muuhun ohjelmiston luokkaan, jolloin ohjelmison muokkaus hankaloituu ja täten voidaan katsoa ohjelmiston laadun laskevan. 11

primitiivinen tietotyyppi (primitive data type) Tietotyyppi, jonka ohjelmointikieli kykenee säilömään suoraan muistipaikkaan raakana numeerisena datana. Ainoat tietotyypit Javassa, jotka eivät ole jonkin luokan instansseja. 14, *see* luokka & instanssi

rajapinta (interface) Termi ohjelmistossa toteutuvalle sopimukselle, jonka jokin metodi, luokka tmv. toteuttaa. Myös kahden ohjelmiston osan välinen taso. Myös Javan Interface-avainsana. 11, *see* interface

signatuuri (signature) Metodin määritelmä, joka sisältää sekä metodin nimen, että sen ottamien argumenttien tyypit.. 17

staattisesti tyypitetty kieli (statically typed language) Ohjelmointikieli, joka tietää jokaisen koodissa esiintyvän muuttujan tietotyypin koko ajan. 14

tietue (struct) Vanhahko muokattava tietotyyppi, yleinen esimerkiksi C-kielessä. Käyttäjä voi määritellä tietueen sisältämään mitä tahansa vakiokokoisia datakenttiä. Luokkien edeltäjä. 9

Javan avainsanat

bool Primitiivinen tietotyyppi, joka sisältää totuusarvon (true tai false). 14, *see* primitiivinen tietotyyppi

byte Primitiivinen tietotyyppi, joka sisältää tavun kokoisen merkillisen kokonaisluvun. 14, *see* primitiivinen tietotyyppi

char Primitiivinen tietotyyppi, joka sisältää kahden tavun kokoisen unicode-koodatun merkin esitettynä merkittömänä kokonaislukuna. 14, *see* primitiivinen tietotyyppi

double Primitiivinen tietotyyppi, joka sisältää 64 bitin kokoisen liukumaesitetyn desimaaliluvun. 14, *see* primitiivinen tietotyyppi

float Primitiivinen tietotyyppi, joka sisältää 32 bitin kokoisen liukumaesitetyn desimaaliluvun. 14, *see* primitiivinen tietotyyppi

int Primitiivinen tietotyyppi, joka sisältää 32 bitin kokoisen merkillisen kokonaisluvun. 14, *see* primitiivinen tietotyyppi

long Primitiivinen tietotyyppi, joka sisältää 64 bitin kokoisen merkillisen kokonaisluvun. 14, *see* primitiivinen tietotyyppi

main Varattu metodinimi metodille, jonka Java ajaa ensimmäisenä ajaessaan ohjelmistoa. Ohjelmistossa voi olla useampi-main niminen metodi, mutta vain määritellyn juuriluokan main-metodi ajetaan. Metodin täytyy olla muotoa public static void ja ottaa yhden taulukon String-luokan instansseja.. 17

private Näkyvyysmääre, joka määrittää ominaisuuden olevan käytettävissä vain ominaisuuden omistaman luokan sisällä.. 17

protected Näkyvyysmääre, joka määrittää ominaisuuden olevan käytettävissä ominaisuuden omistavan luokan sisältävässä packagessa ja kaikissa ominaisuuden omistavan luokan perivissä luokissa.. 17

public Näkyvyysmääre, joka määrittää ominaisuuden olevan käytettävissä kaikkialla ohjelmistossa.. 17

short Primitiivinen tietotyyppi, joka sisältää kahden tavun kokoisen merkillisen kokonaisluvun. 14, *see* primitiivinen tietotyyppi

static Staattinen metodi tai muuttuja näkyy kaikille sen omistavan luokan instansseille jaetusti. Tunnetaan luokkamuuttujana tai luokkametodina.. 17

String Javan standardikirjaston merkkijonoimplementaatioluokka. Suositellaan käytettäväksi merkkijonojen säilömiseen koodissa. Pystyy säilömään dynaamisen merkkijonon, jonka alustus- tai maksimikokoa ei tarvitse määrittää erikseen. Lisäksi sisältää lukuisia merkkijonon käsittelyä ja muokkausta helpottavia metodeja. 16, 17

void Muuttujan paluuarvotyyppi muuttujalle, joka ei palauta mitään.. 17