# Ohjelmistotyö

Käsite ohjelmistotyö tarkoittaa kaikkea sitä työtä, jota valmiin ohjelman tuottaminen edellyttää. Osittamalla työ hyvin määriteltyihin vaiheisiin päästään systemaattisuuteen, joka auttaa tekemään laadukasta jälkeä.

Ohjelmointi

*Ohjelmointi* on suppeassa mielessä toimintaohjeiden kirjoittamista jollain ohjelmointikielellä. Laajassa mielessä se on ohjelmistotyötä, johon kuuluvat seuraavat työvaiheet: määrittely, suunnittelu, toteutus, testaus, dokumentointi ja ylläpito.

Määrittely

*Ohjelmien määrittelyllä* (program specification) tarkoitetaan ohjelmalle asetettavien vaatimusten ja toivomusten esittämistä suullisessa, kirjallisessa tai graafisessa muodossa. Määrittely sisältää ohjelmien syöttö- ja tulostietojen suunnittelun. Syöttötiedot ovat ihmisen ohjelmalle välittämää tietoa eli dataa, jonka perusteella ohjelma tekee toimenpiteitä. Tulostiedot ovat ohjelman toiminnan tuloksen esitys ihmisen ymmärtämässä muodossa (tekstinä, äänenä tai kuvana).

Suunnittelu

*Ohjelmien suunnittelu* (program design) tarkoittaa ohjelmassa käytössä olevien *toimintamenetelmien* eli *algoritmien* suunnittelua sekä *tietorakenteiden*eli tietojen esitysmuotojen suunnittelua.

Toteutus

*Toteutusvaiheessa* (implementation phase) algoritmit ja tietorakenteet esitetään valitulla ohjelmointikielellä.

Testaus

Kun ohjelma on toteutettu, se on testattava toimintakuntoon jonkin ohjelmointijärjestelmän avulla. Käytännössä *testaus* (testing) on rinnakkainen työvaihe toteutukselle. Ohjelmaa testataan pala palalta samalla, kun sitä rakennetaan valmiiksi.

Dokumentointi

*Dokumentointi* (documentation) on keskeisin osa ohjelmistotyötä. Valmis ja testattu ohjelma on dokumentoitava tulevia käyttäjiä ja ylläpitäjiä varten. Kaikki ratkaisumenetelmät kirjataan. Lisäksi kirjoitetaan ohjelman käyttöohje. Myös testaus on kuvattava. Testauksen kuvauksen perusteella ohjelma on kyettävä milloin tahansa testaamaan uudelleen. Jotta testaus ei olisi liian hankalaa, testausvaiheessa pitää valmistaa riittävästi uudelleen­käytettävissä olevaa testaustyökalustoa ja -materiaalia. Dokumen­tointi­vaiheessa tuotetaan vielä ylläpito-ohje, jossa kuvataan, miten ohjelmaan voidaan lisätä uusia piirteitä ja mitä asioita on otettava huomioon muu­tok­sia tehtäessä.

Käytännön ohjelmointityössä kukin ohjelmoinnin työvaihe (määrittely, suunnittelu jne.) tuottaa joukon dokumentteja. Toteutusvaiheessa syntyvä ohjelma on itse asiassa vain yksi dokumentti muiden joukossa. Se on kaikista laadituista dokumenteista yksityiskohtaisin ja tarkin. Se sisältää täsmälliset ohjeet siitä, miten määrittelyvaiheessa asetetut tavoitteet saavutetaan ohjelmallisesti.

Ohjelmistotyön vaihejakoa voidaan kuvata seuraavasti, jossa vaiheet etenevät ylhäältä alaspäin:

|  |
| --- |
| määrittely |
|  |
| suunnittelu |
|  |
| toteutus |
|  |
| testaus |
|  |
| dokumentointi |

Ohjelmistotyö etenee käytännössä harvoin yksinomaan yhdensuuntaisesti. Käytännössä välivaiheissa ilmenneet ongelmat johtavat johonkin aikaisempaan vaiheeseen palaamiseen. Esimerkiksi toteutusongelma voi johtaa uudelleensuunnittelun tarpeeseen. Lisäksi dokumentointia tehdään jatkuvana prosessina jokaisessa vaiheessa.

Algoritmit ja tietorakenteet

Kaikki ohjelmat käsittelevät aina jotain tietoa. Tieto tulee esittää ohjelmissa ohjelmointikielen sisältämien mekanismien avulla. Korkeantason ohjelmointikielet, kuten Java ja C++, tarjoavat välineitä, joilla on suorat vastineet reaalimaailman rakenteisiin.

Ohjelmointikielet tarjoavat välineitä yksittäisten tietojen esittämistä varten, esimerkiksi ihmisen nimeä, kokeen arvosanaa tai päivämäärää varten. Myös monimutkaisempia rakenteita varten on olemassa esitystapoja. Esimerkiksi puhelinluettelon yhtä riviä varten on rakenne, joka mahdollistaa usean tiedon (tässä tapauksessa nimen, osoitteen ja puhelinnumeron) samanaikaisen käsittelyn. On olemassa myös rakenteita, jotka mahdollistavat suurten tietomäärien käsittelyn. Esimerkiksi yhden vuoden jokaisen päivän sademäärät voidaan koota yhteen tietorakenteeseen.

*Tietorakenne*(data structure) on edellä esitettyjen esimerkkien valossa informaatioyksikkö, joka voi sisältää yhden yksittäisen tiedon tai tietojoukon.

Ohjelmat käsittelevät aina käytännössä tietorakenteita. Niitä menettelytapojen ja sääntöjen kokoelmia, joiden avulla rakenteita käsitellään, kutsutaan *algoritmeiksi* (algorithm). Algoritmit käsittelevät aina jotain tietoa (syötteitä, lähtötietoja) siten, että syntyy uutta tietoa (tulosteita, tulostietoja).

Algoritmeille on tyypillistä, että ne eivät ole mistään esitystavasta tai kielestä riippuvaisia. Ne ovat täysin yleisiä ja niitä voi esittää millä välineillä tahansa, esimerkiksi puhekielellä, graafisesti tai ohjelmointikielellä.

On myös huomattava, että algoritmi ei ole yksinomaan ohjelmointiin liittyvä käsite. Esimerkiksi tavallista keittokirjaa voidaan pitää algoritmikokoelmana. Reseptit voidaan tällöin nähdä algoritmeina ja raaka-aineet tietorakenteina.

Ohjelmoijan tehtävänä on suunnitella ohjelmansa käyttäen ”hyviä” algoritmeja. Hyvänä algoritmina voidaan pitää sellaista, joka tuottaa oikean lopputuloksen, on mahdollisimman yksinkertainen, nopea ja käyttää vähän muistia. Algoritmin tulee lisäksi päättyä joskus. On myös huomattava, että algoritmin tulee samoista lähtöarvoista päätyä aina samaan lopputulokseen.

Algoritmien suunnittelu vaatii ajatustyötä. Valitettavasti ei ole olemassa yleistä algoritmia sille, miten ajatustyö saadaan oikein toimivaksi, yksinkertaiseksi, nopeaksi ja vähän muistia tarvitsevaksi.

# 

# Ohjelmien suunnittelu

Ohjelmien kirjoittaminen ei ole sattumanvaraista ohjelmointikielen lauseiden peräkkäin kirjoittamista mielivaltaisessa järjestyksessä vaan systemaattista toimintaa, joka vaatii ajattelua. Vaikka ihminen saa itsensä ymmärretyksi jokapäiväisessä ihmisten välisessä kanssakäymisessä puhumalla asioista epämääräisesti ja jopa sekavasti, tietokone edellyttää, että ohjelmat ovat hyvin systemaattisia ja loogisia. Jotta ohjelma voisi toimia käytännössä, sen logiikan tulee toimia myös paperilla. Jos ohjelmaa ei pystytä suunnittelemaan kynän ja paperin avulla, sen kirjoittaminen ohjelmointikielellä on mahdotonta. Ohjelmoinnissa pätee vanha sanonta ”Hyvin suunniteltu on puoliksi tehty”.

Ohjelmien suunnitteluun on olemassa erilaisia formaaleja (muodollisia, tarkkojen sääntöjen mukaisia) välineitä: lohkokaavioita, vuokaavioita ja pseudokoodeja. Tässä kirjassa käytetään paikoin vapaamuotoista sanallista esitystapaa eli pseudokoodia.

Jäsentävä ajattelu

Ohjelmien suunnittelu ja toteuttaminen edellyttävät reaalimaailman ilmiöiden hahmottamis- ja jäsentämiskykyä. Jäsentämisellä tarkoitetaan tässä yhteydessä kokonaisuuden näkemistä osiensa summana. Ihminen joutuu jäsentämään asioita rat­koessaan kaikenlaisia ongelmia. Jäsentäminen on välttämätöntä ihmisen ajattelutoiminnan kan­nalta, koska ihmisen muistijärjestelmä on sellainen, että se ei helposti kykene käsittelemään useita asioita kerrallaan.

TOP-DOWN –suunnittelu

Jäsentämistä voidaan tarkastella systemaattisena tapahtumana, josta esi­merk­kinä on ohjelmien suunnittelussa käytetty TOP−DOWN−-mene­tel­mä. Tässä menetelmässä noudatetaan seuraavia sääntöjä:

- ensin mietitään pääkohdat

- seuraavaksi tarkennetaan pääkohtia

- uusia alakohtia tarkennetaan asteittain.

Menetelmässä lähdetään ylhäältä ja päädytään alas eli menetelmässä lähdetään suurista suuntalinjoista ja päädytään yksityiskohtiin. Esimerkiksi talon rakentaminen voidaan jäsentää seuraaviin pääkohtiin:

osta tontti

hanki rakennuslupa

hanki työvoima

kaiva perustuksille kuoppa

osta rakennustarpeet

rakennuta talo

teetä lopputarkastus

Pääkohtia tarkennetaan edelleen seuraavasti:

hanki rakennuslupa:

mene rakennuslupavirastoon

täytä rakennuslupa-anomus

maksa leimavero postiin

jätä kaavake rakennuslupavirkailijalle

Alakohtia voidaan tarkentaa, elleivät ne ole vielä riittävän yksityiskohtaisia. Esimerkiksi rakennuslupavirastoon meneminen voidaan toteuttaa kävellen, bussilla, polkupyörällä tai omalla autolla. Ennen kuin tällainen vaihtoehtojen joukko ja niihin liittyvät toimintamallit voidaan esittää, tarvitaan kuitenkin käyttöön lisää uusia rakenteita.

TOP−DOWN-suunnittelussa voidaan keskittyä suuriin linjoihin ja yksityiskohtia suunniteltaessa yhteen osa-alueeseen kerral­laan. Lopputuloksena on selkeä, ymmärrettävä ja ylläpidettävä ratkaisu. Ylläpidettävyys tarkoittaa sitä, että ratkaisua voidaan helposti muuttaa, jos jokin lisäseikka tätä vaatii.

BOTTOM−UP-suunnittelu

TOP−DOWN-suunnittelulle vastakkainen menetelmä on yksityiskohdista lähtevä BOTTOM**−**UP**−**-menetelmä. Siinä tuotetaan ensin kokoelma työkaluohjelmia, ennen kuin lähdetään varsinaisesti rakentamaan mitään laajoja kokonaisuuksia. Ohjelmistojen suunnittelumenetelmät lähtevät yleisistä suuntalinjoista (TOP−DOWN), mutta käytännön ohjelmistoprojekteissa on usein perusteltua lähteä liikkeelle kokonaan tai TOP**−**DOWN-menetelmän rinnalla BOTTOM**−**UP-periaatteella.

# Ohjelmien perusrakenteet

Kantavana ajatuksena ohjelmistotyössä on pidettävä vanhaa tuttua sanontaa ”pyörää ei kannata keksiä uudestaan”. Ohjelmistoteollisuus on tuottanut ohjelmia jo vuosikymmeniä ja kehittynyt sekä hyviä ohjelmointitapoja että lukuisia valmiita kirjasto-ohjelmia, joita ohjelmoija voi käyttää suoraan omissa ohjelmissaan. Ohjelmointikielten kehittyessä on käynyt niin, että hyvät toimintamallit ovat tulleet osiksi ohjelmointikieliä. Seuraavaksi tutustutaan perusrakenteisiin, jotka sisältyvät käytännössä kaikkiin ohjelmointikieliin ja joita kaikissa ohjelmissa käytetään.

Ohjelmoijan käytössä on kolme perusrakennetta, joilla kaikki perinteiset ohjelmointiongelmat voidaan ratkaista. Nämä kolme perusrakennetta ovat

1. peräkkäisrakenne

2. valintarakenne

3. toistorakenne.

Rakenteita kutsutaan usein *ohjausrakenteiksi* tai *kontrollirakenteiksi* (control structure), koska niiden avulla ohjataan ohjelman kulkua.

Peräkkäisrakenne

*Peräkkäisrakenne*on yksinkertaisin ohjelmassa käytettävistä rakenteista. Siinä toteutetaan sarja toimenpiteitä peräkkäin.

Valintarakenne

*Valintarakennetta* käytetään silloin, kun ohjelmassa halutaan suorittaa jotain vain silloin, kun jokin ehto on voimassa. Valintatilanteesta on esimerkkinä seuraava virke:

Jos sinulla on rahaa, käy elokuvissa.

Sama voidaan esittää muodollisemmin seuraavasti:

JOS (sinulla on rahaa)

{

käy elokuvissa

}

Tällainen rakenne voidaan liittää peräkkäisrakenteen sisälle, esimerkiksi päivittäisrutiinien joukkoon, kuten

herää

syö aamiainen

mene kouluun

opiskele

palaa koulusta

tee läksyt

JOS (sinulla on rahaa)

{

käy elokuvissa

}

käy iltapesulla

mene nukkumaan

JOS-rakenne voi sisältää useita toimintoja:

JOS (sinulla on rahaa)

{

käy elokuvissa

käy nakkikioskilla

}

JOS-rakenne voi sisältää vaihtoehtoisen MUUTEN-haaran:

JOS (sinulla on rahaa)

{

käy elokuvissa

käy nakkikioskilla

}

MUUTEN

{

katso tv:tä

}

Tässä MUUTEN-haara suoritetaan tilanteessa, jossa rahaa ei ole ts. kahdesta vaihtoehtoisesta toimintamallista valitaan toinen ja vain toinen.

JOS−MUUTEN-rakenteen sisässä voi olla toinen JOS-rakenne:

JOS (sinulla on rahaa)

{

käy elokuvissa

käy nakkikioskilla

}

MUUTEN

{

JOS (tv:stä tulee jotain katsomisen arvoista)

{

katso tv:tä

}

MUUTEN

{

käy lenkillä

}

}

Edellisen rakenteen perusteella lenkille päästään, jos lompakko on tyhjä eikä televisiosta tule mitään mielenkiintoista.

Toistorakenne

Jos tehtäväkokonaisuuden suorittaminen edellyttää samanlaisina toistuvien yksittäis­tapahtumien toistamista, voidaan käyttää *toistorakennetta*. Esimerkiksi kuoppaa kaivettaessa toistuu lapiointi. Kuoppaa pitää kaivaa niin kauan, että se on tarpeeksi syvä. Luonnollisella kielellä vaatimus (ja toimintaohje) voitaisiin esittää seuraavasti:

Kaiva kuoppaa, kunnes se on tarpeeksi syvä.

Sama asia voidaan esittää muodollisemmin seuraavasti:

NIIN KAUAN KUIN (kuoppa on liian pieni)

{

kaiva lapiollinen

}

Toistossa on periaatteena, että ennen jokaista toistokertaa tut­ki­taan jatkoehto uudelleen ts. ”pyöritään silmukassa”, kunnes voidaan lopettaa.

Rakenteiden edut

Rakenteiden etu on siinä, että niitä voidaan yhdistää: rakenteita voidaan kirjoittaa peräkkäin ja kunkin rakenteen sisässä voi olla mikä tahansa rakenne. Rakenteita voidaan kirjoittaa myös useita sisäkkäin. Peräkkäisyyttä ja sisäkkäisyyttä yhdistelemällä syntyy rakenteinen (tai lohkorakenteinen) ohjelma. Ohjelmointikieli, jolla voidaan ohjelmoida edellä esitettyjä rakenteita hyväksi käyttäen on ns. *lohkorakenteinen* kieli. Java, C, ja C++ ovat kaikki lohkorakenteisia kieliä. Java ja C++ ovat lisäksi olio-kieliä. Olio-pohjaisuus tuo ohjelmointiongelmien ratkaisuun uudenlaisen näkökulman, mutta se ei tarkoita etteikö kolmea perusrakennetta tarvittaisi myös olio-ohjelmissa. Siihen, kuinka näitä rakenteita toteutetaan Javalla, palataan myöhemmin. Sitä ennen on tutustuttava vuorovaikutteisten ohjelmien perusmekanismeihin.

Vuorovaikutteiset ohjelmat

Käytännössä tietokoneohjelmat ovat apuvälineitä, jotka tekevät tehtäviä ihmisen antamien ohjeiden mukaan. Ohjelmissa on keskeistä ihmisen ja ohjelman vuorovaikutus, jossa ihminen eli ohjelman käyttäjä antaa ohjeita tai pyytää ohjelmaa tekemään jotain ja ohjelma toteuttaa annetut tehtävät antaen käyttäjälle sopivan palautteen.

Vuorovaikutuksen toteuttamisessa on keskeistä kommunikaation toteuttaminen. Kommunikaatioon kuuluu tiedon lähettäminen ja sen vastaanottaminen. Perusrakenteeltaan kukin ohjelma toimii seuraavan kaavan mukaan:

**tietokone puhuu** eli lähettää tietoa ja **ihminen kuuntelee** eli vastaanottaa tietoa

**ihminen puhuu** eli lähettää tietoa ja **tietokone kuuntelee** eli vastaanottaa tietoa

tietokone puhuu ja ihminen kuuntelee

ihminen puhuu ja tietokone kuuntelee

tietokone puhuu ja ihminen kuuntelee

jne.

Sen jälkeen kun ihminen on puhunut eli (yleensä) **syöttänyt näppäimistöltä** tietokoneelle tietoa, tietokone käsittelee eli prosessoi tiedon, ennen kuin se antaa vastauksen ihmiselle eli (yleensä) **tulostaa** tiedon **kuvaruudulle**. Tällaisen keskustelevan ohjelman perusajatus voidaan tiivistää seuraavaan kuvaan:

INPUT TEMPUT OUTPUT

Kuvassa tietojen syöttövaihe on nimetty INPUT-vaiheeksi, tulostus OUTPUT-vaiheeksi ja tiedon prosessointi TEMPUT-vaiheeksi. Tietokoneohjelma voidaankin nähdä tehtaan kaltaisena tuotantolaitoksena, joka tietyistä raaka-aineista tuottaa haluttuja tuotteita. Näitä tuotantoprosessin vaiheita yhteen ohjelmaan voin liittyä useita. Tosin sanoen OUTPUT-vaihetta voi seurata uusi INPUT-vaihe jne. Keskustelevassa ohjelmassa siis toistuvat systemaattisesti seuraavat kolme vaihetta:

1. tietojen syöttö (käyttäjä syöttää, ohjelma ottaa vastaan)

2. laskenta eli prosessointi

3.tietojen tulostus.

Jotta osattaisiin kirjoittaa ohjelma, joka toimii edellisen kaavan mukaan on tutustuttava välineisiin, joilla vuorovaikutus tehdään mahdolliseksi. Aloitetaan yksinkertaisella ohjelmalla, jossa kommunikaatio on yksisuuntaista: ohjelma puhuu, ihminen kuuntelee. Toisin sanoen tutustutaan, kuinka ohjelma saadaan tulostamaan tietoa kuvaruudulle.

Todettakoon aluksi, että Javalla voidaan tehdä täysin merkkipohjaisia (ei graafisia) ohjelmia. Nämä ovat useimmiten ohjelmointikurssien lähtökohta ja tärkein pohjatieto siirryttäessä kohti tapahtumapohjaista ohjelmointia, jossa ohjelmaan rakennetaan graafinen käyttöliittymä. Esitetään merkkipohjaista käyttöliittymää käyttävä yksinkertaisin mahdollinen Java-ohjelma. Windowsissa seuraava ohjelma ajetaan DOS-ikkunassa ja Unixissa pääteikkunassa, joita kutsutaan usein yhteisellä nimellä **konsoliksi** viitaten merkkipohjaiseen tilaan.

public class HeiMaailma

{

public static void main(String args[]) {

System.out.println("Hei maailma!");

}

}

Ohjelma tulostaa kuvaruudulle tekstin ”Hei maailma!”

Esimerkkiohjelmasta voidaan todeta seuraavat Java-ohjelmoinnin piirteet: Jokainen Java-ohjelma kirjoitetaan luokan sisään. **class**-avainsanalla annetaan luokalle nimi (edellä HeiMaailma), joka on luokan yksilöllinen tunniste. Luokan on oltava julkinen (**public**) ja sen on sijaittava tiedostossa *HeiMaailma.java*. Tämän nimeämissäännön avulla on pyritty siihen, että jokainen luokka sijaitsee omassa tiedostossaan, jolloin ohjelmistojen rakentaminen koostuu luonnollisemmin osista eli ns. komponenteista. Jos luokan nimi ja tiedoston nimi poikkeavat toisistaan, Java-kääntäjä antaa kuvaavan virheilmoituksen ("Public class HeiMaailma must be defined in a file called HeiMaailma").

Luokka rajataan aaltosulkeilla, ja sen sisälle tulee kaikki luokassa määriteltävät asiat. Yleisesti aaltosulkeiden rajaamaa aluetta kutsutaan ohjelmoinnissa **lohkoksi** (block). Luokan sisällä olevaa ohjelmaa kutsutaan pääohjelmaksi ja se nimetään aina seuraavasti:

public static void main(String[] args)

Hakasulkeet (edellä String-sanan jälkeen) voidaan kirjoittaa myös args-sanan perään:

public static void main(String args[])

Itse asiassa args-sanan paikalla voi lukea mitä tahansa. Nämä asiat selviävät myöhemmin.

Pääohjelma sisään sijoitetaan suoritettavat **lauseet** (statement). Javassa jokainen lause päättyy aina puolipisteeseen. Lauseet ovat tavallaan ohjelmointikielen virkkeitä. Koska **tyhjillä merkeillä** tai rivinvaihdoilla ei ole Java-ohjelmassa merkitystä, yhdellä rivillä voi sijata useampi lause. Samaan tapaan lauseen pituudella ei ole merkitystä, joten lause voi jatkua toisella rivillä.

Javassa konsolille tulostus toteutetaan **System.out**-olion **println**-metodilla. Metodi-sana tarkoittaa Javassa rutiinia tai aliohjelmaa, joka suorittaa annetun tehtävän. Yksinkertaisuuden vuoksi voidaan tässä vaiheessa ajatella että käytetään **käskyä**, jonka nimi on *System.out.println.* Tulostuslauseelle annetaan merkkijono sulkeiden sisällä. Merkkijono kirjoitetaan aina lainausmerkkien sisään. Edellisessä ohjelmassa on yksi ainoa lause:

System.out.println("Hei maailma!");

Lause tulostaa tekstin ”Hei maailma”.

Kun käytössä on Java-ohjelmien kehitys- ja suoritusympäristö, JDK (Java Development Kit), ohjelma voidaan kääntää javac-kääntäjällä (javac = java compiler) seuraavasti:

javac HeiMaailma.java

Tämän jälkeen tavukoodiksi käännetty ohjelma suoritetaan seuraavasti:

java HeiMaailma

Ohjelma tulostaa kuvaruudulle tekstin:

Hei Maailma

Tietojen syöttämiseen palataan hieman myöhemmin. Sitä ennen on tutustuttava mekanismeihin, joilla saadaan määriteltyä ohjelman välineitä, joihin syötettävää tietoa tallennetaan. Toisin sanoen on tutustuttava sellaisiin asioihin kuin tunnukset, muuttujat ja tyypit. Näitä asioita käsitellään seuraavaksi.

(teksti: Peltomäki – Silander: Java 2, Docendo)