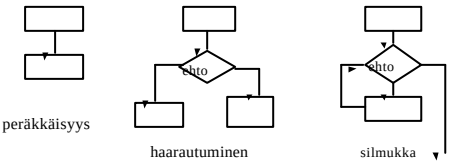
Ohjelmistotuotanto

Ohjelmointi

1

1. Algoritmien kuvaaminen

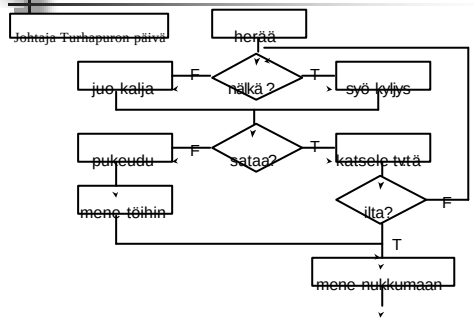
- Algoritmien esitystapoja:
  - Pseudokoodi - luonnollista kieltä ohjelmointikielen kontrollirakentein
  - Vuokaaviot - kontrollin kulku ohjelmassa



peräkkäisyys haarautuminen silmukka

 2

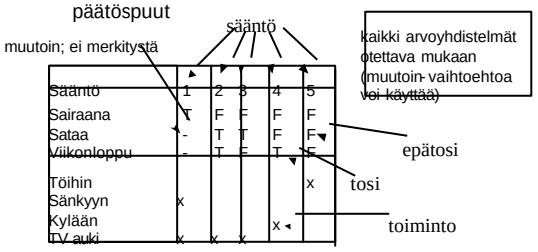
Algoritmien kuvaaminen



 3

Algoritmien kuvaaminen

- Monimutkaiset ehtoyhdistelmät: päätöstaulut / päätöspuut



 4

2. Ohjelmointityyli

- Yleisiä tyyliohjeita:
  - Ohjelmia luetaan useammin kuin kirjoitetaan: optimoi lukijan äläkä kirjoittajan työtä
  - Kuvaavat nimet - vaikuttavat ohjelman luettavuuteen ja ymmärrettävyyteen
  - Käytä yhdenmukaisia nimiä
    - ei sekakieltä (esim. suomi + englanti)
    - yhtenäinen nimirakenne
    - jos lyhennetään, niin aina samoin
      - **ei** average_freq, frequency, frequency_maximum, min_fr, frqTotal
    - osasanat aina samassa järjestyksessä
      - **ei** average_freq, frequency_maximum

 5

Ohjelmointityyli

- Ohjelmassa esiintyvät vakiot on syytä nimetä, mikäli ne voivat jossain ohjelman elinkaaren vaiheessa muuttua
- Tietorakenteille ja metodeille tulisi sopia löytämistä helpottava esittelyjärjestys, esim. aakkosjärjestys
- Parametrit tulisi välittää yhdenmukaisesti (ei tarpeetonta vaihtelua osoittimen ja arvon välillä)
- Sisäkkäiset ehtolauseet voivat aiheuttaa ymmärtämisongelmia, samoin kuin monimutkaiset loogiset lausekkeet - näihin on kiinnitettävä erityistä huomiota
- Pyrittävä mahdollisimman ymmärrettävään koodin - vältettävä kaikkea kikkailua (esimerkiksi mitättömän tehostamisen takia)

 6

Ohjelmointityyli

- Koodi on kommentoitava
  - Moduulin / luokan otsakekommenttiin:
    - käyttötarkoitus
    - liittymän kuvaus
      - parametrit & globaalit rakenteet
    - miten moduulia käytetään liittymän kautta
    - mitä ulkopuolisia moduuleja itse käyttää
    - käytön rajoitukset (esiehdot)
    - tärkeimmät operaatiot ja niiden toimintaperiaate
    - kehityshistoria
      - laatija, hyväksyntä, muutokset
    - nykytila
      - protoasteella, testauksessa, tuotantoasteella (valmis)

Ohjelmointityyli

- Pseudokielen lauseet kommenteiksi
  - kytkee koodin vastaavaan algoritmi/moduulisuunnitelmaan
  - ei kommentoida itsestäänselvyyksiä (kuten: "muuttujan a arvoa kasvatetaan yhdellä") vaan syitä ja perusteita
- Tietorakenteiden (ja niiden kenttien) merkitys
- Oikeellisuustarkistusten vaatimat kommentit
- Tarvittavien kommenttien määrä riippuu
  - ohjelmointikielestä
  - muuttujanimien valinnasta ym. lisäinformaatiosta
  - ohjelmistokehittäjien määrästä
- Kommentti kertoo, mitä koodin on tarkoitus tehdä – ei välttämättä, mitä todella tehdään

3. Koodin parantelu – Refactoring

- "Refaktorointi": Refactoring
  - ohjelmakoodin rakenteen muuttamista muuttamatta koodin toiminnallisuutta
  - parantaa rapautuneen koodin arkkitehtuuria
  - helpottaa koodin ymmärtämistä
  - helpottaa koodin ylläpitoa
  - helpottaa koodin testaamista ja virheiden jäljitystä
  - helpottaa projektiryhmän sisäistä yhteistyötä
  - keskeinen vaihe Extreme Programming -prosessissa

Koodin parantelu – Refactoring

- M. Fowler: *Refactoring – Improving the Design of Existing Code*. Addison-Wesley, 1999
  - 72 refaktorointi-sääntöä
  - toteutus Javalla, mutta perusideat kieliriippumattomia
  - yhtenäinen kuvaustapa: sääntöluettelo
  - toisiaan hyödyntäviä refaktorointeja: laajemmat muutokset voidaan tehdä systemaattisesti vaihe vaiheelta

Koodin parantelu – Extract Method

Muunnetaan koodinpätkä kutsuttavaksi metodiksi

```
void printOwing (double amount) {
    printBanner( );

    // print details
    System.out.println ("name: Jukka");
    System.out.println
    ("amount: " + amount);
}
```

⇨

```
void printOwing (double amount) {
    printBanner( );
    printDetails(amount);
}

void printDetails (double amount) {
    System.out.println ("name: Jukka");
    System.out.println
    ("amount: " + amount);
}
```


Koodin parantelu – Extract Method

Säännöt

1. Luo uusi metodi ja anna sille kuvaava nimi.
2. Kopioi koodinpätkä ***k*** lähtömetodista ***m*** uuden metodin rungoksi.
3. Tee ***k*** :ssa viitatuista ***m*** :n paikallisista muuttujista ja parametreista uuden metodin muodollisia parametreja.
4. Tee ***k*** :ssa käytetyistä väliaikaisista muuttujista uuden metodin väliaikaisia paikallisia muuttujia.
5. Mikäli ***k*** :ssa muutetaan ***m*** :n paikallisten (tai väliaikaisten) muuttujien arvoa, sovella niihin jotakin sopivaa refaktorointia.
6. Muuta ***m*** :ssä oleva ***k*** uuden metodin kutsuksi siten, että viittaukset ***m*** :n paikallisiin muuttujiin ja parametreihin muutetaan kutsun parametreiksi.
7. Käännä ja testaa.

Koodin parantelu – Inline Method

Muunnetaan metodi koodinpätkäksi

```
int getRating ( ) {
    return (moreThan5( )) ? 2 : 1 ;
}

boolean moreThan5 ( ) {
    return numberOfLate > 5 ;
}
```

⇨

```
int getRating ( ) {
    return (numberOfLate > 5) ? 2 : 1 ;
}
```

 13

Koodin parantelu – Inline Method

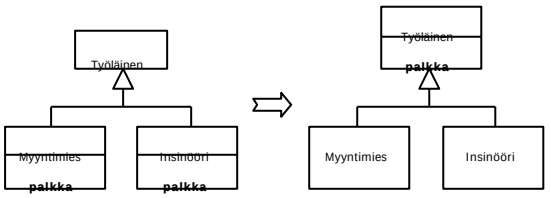
Säännöt

1. Tarkista, ettei metodi ole monimuotoinen (polymorphic).
   - Metodia ei saa poistaa, mikäli se määritellään uudelleen (syrjäytetään, override) aliluokissa.
2. Etsi kaikki ko. metodin kutsut.
3. Korvaa jokainen kutsu metodin rungolla.
4. Käännä ja testaa.
5. Poista metodi.

 14

Koodin parantelu – Pull Up Field

Aliluokkien yhteinen kenttä siirretään yliluokkaan



 15

Koodin parantelu – Pull Up Field

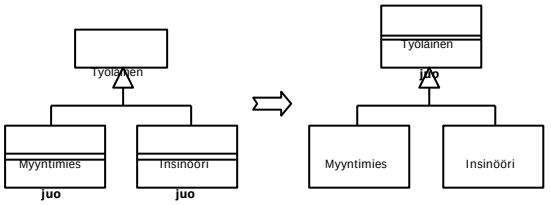
Säännöt

1. Tarkista, että yhteisellä kentällä on sama tyyppi ja sitä käytetään samalla tavalla kaikissa aliluokissa.
2. Mikäli kenttä esiintyy aliluokissa eri nimillä, korvaa ne kaikki yhteisellä nimellä.
3. Käännä ja testaa.
4. Määrittele yliluokkaan uusi valitunniminen ja –tyyppinen kenttä.
5. Poista kenttä kaikista aliluokista.
6. Käännä ja testaa.
7. Suojaa halutessasi kenttä ja luo sille tarvittavat saantimetodit soveltamalla jotakin sopivaa refaktorointia.

 16

Koodin parantelu – Pull Up Method

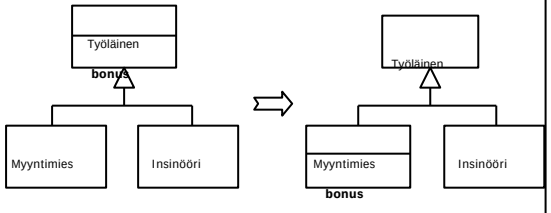
Aliluokkien identtinen metodi siirretään yliluokkaan



 17

Koodin parantelu – Push Down Field

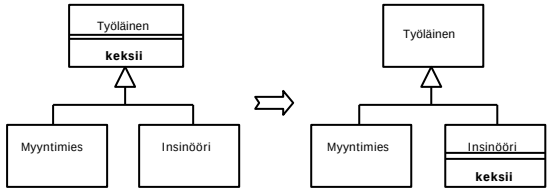
Kenttä siirretään yliluokasta sitä käyttäviin aliluokkiin



 18

Koodin parantelu – Push Down Method

Metodi siirretään yliluokasta sitä käyttäviin aliluokkiin

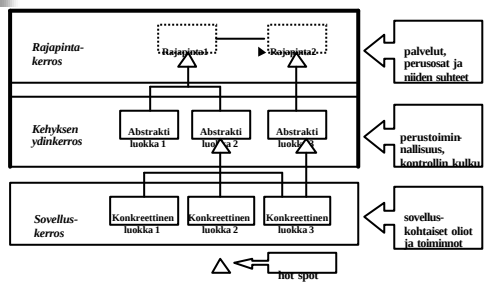


4. Sovelluskehykset

Sovelluskehys (application framework): kiinteästi toisiinsa liittyvien ohjelmistokomponenttien kokoelma, joka toteuttaa tietyn tuoteperheen arkkitehtuurin ja josta voidaan erikoistaa mikä tahansa perheen jäsen

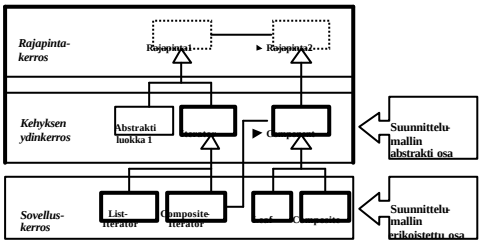
- tuoteperheiden toteutustekniikka
- uudelleenkäytettävä "pääohjelman luuranko" + joukko erikoistamisrajapintoja (*hot spot*)
- yleensä olioperustainen: aliluokitus ja perintä, rajapinnat, koostaminen, monimuotoisuus, dynaaminen sidonta
- iteratiivinen kehitys: sovelluksista kehykseen (yleistäminen), kehyksestä sovellukseen (erikoistaminen), ja takaisin
- tehokasta koodin uudelleenkäyttöä, tyypillisesti 50-80% sovelluksen koosta

Sovelluskehykset – Rakenne



Sovelluskehykset ja suunnittelumallit

- suunnittelumallit dokumentoivat sovelluskehyksen arkkitehtuuria
- suunnittelumallit määrittelevät kehyksen sovellusrajapinnan



Sovelluskehykset – Fred

- Fred (Framework Editor)
  - laitoksen tutkimushanke
  - (1) sovelluskehysten systeemaattinen rakentaminen (suunnittelu)mallien avulla
  - (2) sovellusten tuottaminen kehyksestä ohjatusti ja valvotusti
  - ohjaus: dynaamisesti mukautuva lista (ohjelmointi-) tehtäviä, joilla kehyksestä saadaan sovellus
  - valvonta: (suunnittelu)malleihin liittyvien rajoitteiden dynaaminen tarkistaminen

Fred – Käyttöliittymä