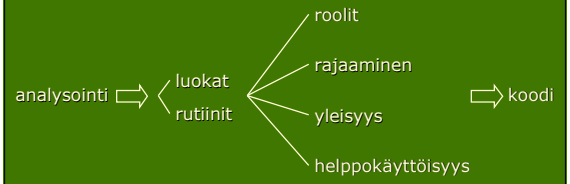
2. Rutiinin muodostaminen

1. Rutiinin määrittely
2. Sopimuspohjainen ohjelmointi
3. Määrittelyjen kirjoittaminen
4. Erikoistilanteiden hallinta

Rutiinin muodostaminen



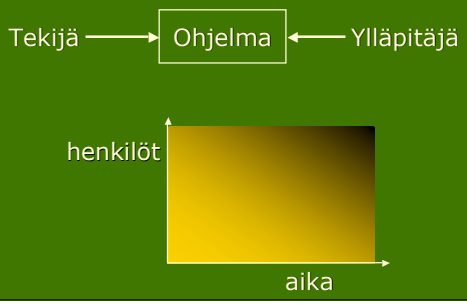
Hyvän ohjelman tunnusmerkit

- Oikeellisuus
- Ymmärrettävyys
- Vankkuus

Määrittely (*specification*)

- Ohjelmaan kirjoitettu teksti
- Kuvaa jonkin ohjelmistokomponentin
  - toimintaa ja merkitystä
  - abstraktisti ja kompaktisti

Määrittelyn osapuolet



Rutiinin määrittely

- Signatuuri
  - nimi
  - parametrit
  - paluuarvo
  - poikkeukset
- Toiminnan kuvaus
  - sanallinen kuvaus
  - alkuehto
  - loppuehto

Esimerkki: Neliöjuurirutiinin määrittely

```
/**
 *  Palauttaa x:n neliöjuuren.
 *  @.pre  x >= 0
 *  @.post Math.abs(RESULT * RESULT - x)
 *         < 1.0e-10.0 & RESULT >= 0.0
 */
public static double neliöjuuri(double x)
```

Rutiinimäärittelyn merkitys 1(2)

- Kommunikointi rutiinin toteuttajan ja käyttäjän välillä
  - toteutus on vapaasti muunneltavissa
  - kutsuja ei tarvitse salattua tietoa
- Suunnittelu ja dokumentointi
  - suunnittelu mahdollista pelkkiä rutiinimäärittelyjä käyttäen
  - sama määrittelyformalismi koko projektissa
- Odotukset
  - mihin toteuttaja sitoutuu?
  - johtaa yleiskäyttöisyyteen

Rutiinimäärittelyn merkitys 2(2)

- Toimivuus
  - vastaako toteutus määrittelyä?
- Virheiden paikallistaminen
  - määrittely antaa ”rajat” korjausta varten
  - iäkkäämpi koodi usein uutta luotettavampaa
- Luokkakokonaisuus
  - piirteiden valinta ja roolit
- Periytymismekanismin oikea käyttö

Sopimuspohjainen ohjelmointi

- Sopimuspohjainen ohjelmointi/suunnittelu = engl. *programing/design by contract*
- Määrittely muodostaa sopimuksen rutiinin (luokan tms.) asiakkaan ja toteuttajan välille

Sopimuspohjaisuus

	Velvoitteet	Hyödyt
Asiakas	Alkuehdon on oltava voimassa ennen kutsua	Kutsun jälkeen loppuehto on voimassa
Toteuttaja	Toteutuksen on saatettava loppuehto voimaan	Alkuehdon karsimia tilanteita ei tarvitse käsitellä

Alkuehto

- Rutiini voi olla
  - osittainen: alkuehdossa rajoituksia
  - totaalinen: alkuehdossa ei rajoituksia
- Rajoitustyyppejä:
  - argumenttien rajoittaminen
  - operaation käytön rajoittaminen
  - toteutukseen liittyvät ehdot
- Kutsujan on pystyttävä tarkistamaan asetetut rajoitukset!

Loppuehto

- Kuvaus siitä miten rutiini käyttäytyy laillisten kutsun yhteydessä
- Loppuehdon väittämät:
  - rutiinin toiminnan vaikutukset asiakkaalle
  - olion sisäiseen esitysmuotoon liittyvät ehdot
- Tunnisteisiin, joita ei ole mainittu loppuehdossa, ei ole tehty muutoksia

Esimerkki: Neliöjuurirutiinin määrittely

```
/**
 *  Palauttaa x:n neliöjuuren.
 *  @.pre  x >= 0
 *  @.post Math.abs(RESULT * RESULT - x)
 *         < 1.0e-10.0 & RESULT >= 0.0
 */
public static double neliöjuuri(double x)
```

Neliöjuurifunktion toteutus

```
//-- Alkuehto
assert x >= 0.0 : "Alkuehtorikkomus";
//-- Totetus
double tulos;
// Lasketaan neliöjuuren arvo muuttujaan tulos...
//-- Loppuehto
assert tulos >= 0.0 : "Loppuehtorikkomus";
assert Math.abs(tulos * tulos - x) < 1.0e-10.0 :
       "Loppuehtorikkomus";
//-- Paluuarvo
return tulos;
}
```

Esimerkki: Tehtävä 2-7 (a)

Rutiini palauttaa tiedon, onko parametrina annettu kokonaisluku laillinen postinumero (ts. väliltä 00000–99999).

Ratkaisuesimerkkejä

```
/**
 * @.pre  true
 * @.post RESULT == (0 <= n <= 99999)
 */
boolean onPostinumero(int n)

/**
 * @.pre  n != null
 * @.post RESULT == (n.length() == 5)
 *       & FORALL(c : n; Character.isDigit(c))
 */
boolean onPostinumero(String n)
```

Esimerkki: Tehtävä 2-7 (b)

Rutiini tarkistaa onko annetussa kokonaislukutaulukossa yhdelläkään alkiolla duplikaattia (ts. rutiini palauttaa `true` jos ja vain jos kukin luku esiintyy taulukossa vain kerran).

Ratkaisuesimerkki

```
/**
 * @.pre  t != null
 * @.post RESULT ==
 *          FORALL(i : 0 <= i < t.length - 1;
 *              !EXISTS(j : i < j < t.length;
 *                  t[i] == t[j]))
 */
boolean onDuplikaatiton(int[] t)
```

Esimerkki: Tehtävä 2-7 (c)

Rutiini palauttaa annetun double-tyyppisiä alkioita sisältävän taulukon pienimmän alkion indeksin. Mikäli pienimmällä alkiolla on duplikaatteja, rutiini palauttaa niistä indeksiltään pienimmän alkion.

Ratkaisuesimerkki

```
/**
 * @.pre  t != null && t.length >= 1
 * @.post
 *  FORALL(i : 0 <= i < RESULT; t[RESULT] < t[i]) &
 *  FORALL(i : RESULT <= i < t.length; t[RESULT] <= t[i])
 */
int miniminIndeksi(double[] t)
```

Onko tästä kaikesta nyt sitten mitään iloa?

- Tutkitaanpa seuraavien Java-rutiinien määrittelyjä
  - String.charAt(int)
  - Arrays.fill(int[], int, int, int)
  - Arrays.binarySearch(int[], int, int, int)

String.charAt()

```
/**
 * @.pre  0 <= index < length()
 * @.post RESULT ==
 *           (positiossa index
 *            oleva merkki)
 */
public char charAt(int index)
```

Arrays.fill()

```
/**
 * @.pre  a != null && (0 <= fromIndex
 *        <= toIndex <= a.length)
 * @.post FORALL(i : fromIndex <= i
 *              < toIndex; a[i] = val)
 */
public static void fill(int[] a,
                        int fromIndex,
                        int toIndex,
                        int val)
```

Arrays.binarySearch()

```
/**
 * @.pre  a!= null && (a on lajiteltu ei-laskevaan
 *                     järjestykseen)
 * @.post
 *   EXISTS(i : 0 <= i < a.length; a[i] == key) ?
 *       a[RESULT] == key :
 *       (FORALL(i : 0 <= i < -(RESULT + 1);
 *               a[i] < key) &
 *        FORALL(i : -(RESULT + 1) <= i < a.length;
 *               key < a[i]))
 */
public static int binarySearch(int[] a,
                               int fromIndex,
                               int toIndex,
                               int key)
```