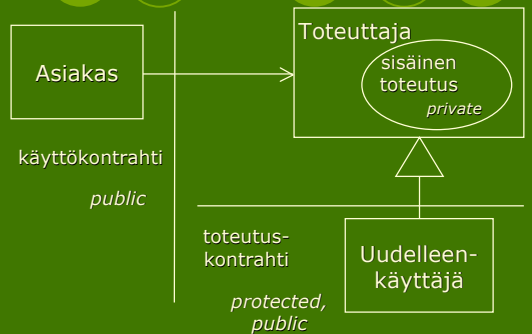


## 5. Periytyminen mekanismina

1. Alityypitys, periytyminen ja polymorfismi
2. Dynaaminen sidonta
3. Määrittelyn erottaminen toteutuksesta
4. Rutiinin korvaus ja ylikuormitus
5. Luokkahierarkia
6. Esimerkkejä
7. Periytyksen käyttö eri tilanteisiin

## Periytyminen



## Alityypitys ja periytyminen

- Liskovin korvausperiaate: alityypit käyttäytyvät siten kuin niiden ylityyppien määrittelyt ilmaisevat
- Oikeaoppinen tyyppiteorian käyttö
  - periytyjä on perimänsä luokan alityyppi
  - periytyjä noudattaa perimänsä luokan operaatioiden määrittelyjä

## Polymorfismi

- Tunnisten kyky viitata moneen erityyppiseen olioon



Henkilö yksilö;  
 Työntekijä duunari;  
 duunari = new Työntekijä("B. Virtanen");  
 yksilö = duunari;

## Sidonta

- Luokka **A** ja sen perijä **B** antavat eri toteutuksen rutiinille **f**,
- Luokan **A** asiakas kutsuu rutiinia käyttäen muuttujaa **x**, johon liittyy polymorfisesti luokan **B** olio
  - ts. **A** `x = new B(); x.f();`
- Mitä tapahtuu?

## Vaihtoehto 1: Staattinen sidonta

- Valintapäätös tehdään käännoaikana
  - kääntäjä ei voi tietää muuttujan **x** dynaamista tyyppiä
  - `x.f()` kutsuu luokan **A** rutiinia
- Oletus mm. C++:ssa (ohjelmoijan muutettavissa)
- Javassa staattisesti sidotaan
  - jäsenmuuttujat
  - static-luokkametodit
  - final-rutiinit
  - private-rutiinit

## Vaihtoehto 2: Dynaaminen sidonta

- Valintapäätös tehdään ajoaikana
  - rutiini *f* valitaan muuttujaan *x* sillä hetkellä liitetyn olion tyyppiin mukaan
  - *x.f()* kutsuu luokan *B* rutiinia
- Oletussidonta Javassa
  - ei ohjelmoijan muutettavissa

## Staattinen ja dynaaminen tyyppi 1(3)

- Staattinen tyyppi
  - ilmaistaan muuttujan esittelyssä
  - pysyy muuttumattomana
- Dynaaminen tyyppi
  - määräytyy sen mukaan minkälaiseen olioon muuttuja kullakin hetkellä viittaa

## Staattinen ja dynaaminen tyyppi 2(3)

```
Henkilö yksilö = new
    Työntekijä("Lennart Nilkén");
Työntekijä duunari = new
    Työntekijä("B. Virtanen");
```

- Dynaaminen tyyppijoukko
  - staattisen tyyppiin ja sen alityyppien muodostama joukko
  - sopivat yhteen staattisen tyyppiin kanssa

## Staattinen ja dynaaminen tyyppi 3(3)

- Mahdottomia asetuksia kun **Henkilö** yksilö ja **Työntekijä** duunari
  - duunari = yksilö;
  - yksilö = duunari;
  - ansio = yksilö.annaPalkka();
- Yleistulkinta (*upcasting*)
  - yksilö = duunari;
- Erikoistulkinta (*downcasting*)
  - duunari = (Työntekijä)yksilö;

## Luokkatyypit

- Rajapintaluokka (*interface*)
- Abstrakti luokka (*abstract class*)
- Konkreetti luokka (*concrete class*)

## Rajapintaluokka

- Luokkamäärittely: *interface*
- Ei sisällä rutiinitoteutuksia
- Kaikki piirteet ovat tyyppiltään
  - public
  - abstract
- Määrittelee roolin toteuttajille

## Rajapintaesimerkki

```
public interface Kirjoittava {
    public void kirjoita(Asiakirja a);
}

public interface Lukeva {
    public void lue(Asiakirja a);
}

public interface Poliisi extends Lukeva, Kirjoittava {
    public void sakota(Henkilö h, double summa);
    public void pidätä(Henkilö h);
}

public class Harjunpää implements Poliisi { /* ... */ }
```

## Abstrakti luokka

- Luokkamäärittely: `abstract class`
- Voi sisältää sekä abstrakteja että konkreetteja rutiineja
- Ei voida konstruoida!
- Voi silti toteuttaa konstruktorin periytymistä varten
- Esimerkki: [RajoitettuPino](#)

## Konkreetti luokka

- Luokkamäärittely: `class`
- Kaikki piirteet on määritelty
- Voidaan konstruoida

## Rutiinien korvaus ja ylikuormitus

Ta	paluutyyppi	nimi	(argumentit)	poikkeukset
↑	↑			↑
Tb	paluutyyppi	nimi	(argumentit)	poikkeukset
↑	⋈		⋈	⋈
Tb	paluutyyppi	nimi	(argumentit)	poikkeukset