Abstrakcije v modelih in jezikih

Iztok Savnik

Abstrakcije

- Kaj so abstrakcije?
- Kako uporabljamo abstrakcije pri modeliranju?
- Kako so definirani gradniki podatkovnih modelov?

Oxford (Hornby)

abstract (adj)

1. separated from what is real or concrete; thought of separately from facts, objects or particular examples: *A flower is beautiful, but beauty itself is ~*
2. take out; separate: metal from ore: *~ wallet from sb's pocket.* ed: not paying attention
4. short account e.g. the chief points of a piece of writting

abstraction (n)

1. A concept or idea not associated with any specific instance.
2. The act of withdrawing or removing something.
3. The process of formulating general concepts by abstracting common properties of instances.
4. An abstract painting.
5. Preoccupation with something to the exclusion of all else.
6. A general concept formed by extracting common features from specific examples

Filozofija

- Aristotel v Metafiziki
  - Operacija intelekta (uma):
    - Ločimo se od realnosti in obdržimo samo nekatere lastnosti koncepta medtem ko preostale spustimo
  - O naravi in genezi matematičnih objektov
    - Formiranje koncepta s pomočjo abstrakcije
- Tomaž Akvinski
  - Nadaljeval Aristotelovo delo
  - Dve vrsti abstrakcij
    - Združi dve ali več lastnosti, ki so med sabo različne. Tako lahko konstruiramo sodbo oz. tako lahko sklepamo
    - Razloči med lastnostmi, ki so v realnosti eno. Tako formiramo koncepte.

Uporaba abstrakcij v jeziku

- π(`Del informacijskega sistema za vodenje proizvodnje Lame bomo dali izdelati zunanjim izvajalcem`) = ?
  - π(`informacijski sistem`) = “sistem, ki ...”;
  - π(`vodenje proizvodnje`) = “krmiljenje proizvodne linije...”;
  - π(`zunanje izvajanje`) = “outsourcing”;
  - ...
- Deli stavka imajo pomen kompleksnih konceptov
  - Velikokrat se pričakuje, da poslušalec „razume“ pomen teh konceptov
  - Kompleksni koncepti so lahko razloženi v kontekstu pogovora

Uporaba abstrakcij v jeziku

- Abstrakcije uporabljamo za:
  - Akcije, združenja, institucije, znanja, operacije, postopke, procese, itd, itd.

Kaj je abstrakcija?

Nekaj aspektov abstrakcije:

1 Ni samo postopek identificiranja ključnih lastnosti
2 Abstrakcija je na nek način tudi kreacija.
- Kreiranje abstraktnega modela procesa.

3 Nujno pride do izgube informacij.
- Z uporabo abstrakcij se lahko “izgubimo”?
- Ne vemo natančno kaj delamo?

4 Abstrakcija povleče za sabo pomen.
- Pomen je odvisen od interpretatorja.
- Lahko pride do zabavnih napak pri prevajanju pomenov.

Abstrakcije ter modeli in jeziki

- Abstrakcije so osnovni mehanizem modelov in jezikov
  - Principi za definicijo gradnikov modelov in jezikov
  - Povezave med podatkovnimi modeli
- Premislimo kakšne abstrakcije uporabljamo pri posameznem P[M|J]
  - Klasifikacija, kompozicija, enkapsulacija, ...
  - Pogledali si bomo še abstrakcije v programskih jezikih
  - ...

Hierarhija računalniških jezikov

- Strojna oprema, naprave, krmilniki, procesorji
  - Digitalna stanja in signali
  - Enostaven vgrajen jezik, ukazi
- Zbirnik (assembler)
  - Direktno delo z napravami računalnika, subrutine, zanke, …
- Imperativni in funkcijski programski jeziki
  - Podprogrami, funkcije, zanke, delo s sistemskimi knjižnicami
  - Primer jezika: C, Lisp, Pascal
- Predmetno usmerjeni programski jeziki
  - Predmeti, skupki predmetov, metode, moduli, sporočila, …
  - Primer jezika: Java, C#
- Sistemski integracijski jezik
  - Objekti, relacije, konfiguracije, moduli, tokovi, …
  - Primer jezika: SQL, …

nivo abstrakcij se viša

Nekatere lastnosti

- Čim bližje strojni opremi bolj enostavni jeziki.
  - Enostavne abstrakcije npr. zanke, rutine, …
- Višji nivo abstrakcij jezika bolj usmerjen jezik.
  - Jeziki z visokim nivojem abstrakcije so zelo usmerjeni za specifično področje
- V sredini so splošni programski jeziki.
  - Namembnost zelo široka
  - Med bolj kompleksnimi jeziki

Abstrakcije v računalniških jezikih

- Koncepti vgrajeni v programski jezik
  - Funkcija, metoda, APT,…
- Nekateri uveljavljeni koncepti programiranja
  - Klasifikacija, specializacija, agregacija…
- Konceptualno ogrodje za programiranje
  - Imperativni in funkcijski jeziki: procedure in funkcije
  - Objektni jeziki: objekti
- Uporabljene abstrakcije določijo model jezika
  - Osnovno konceptualno ogrodje jezika
  - Jezikovne konstrukcije jezika

Zakaj študij abstrakcij

- Poznavanje izraznih zmožnosti modelov in jezikov
- Pravilna uporaba danih gradnikov podatkovnih modelov in programskih jezikov
- Pravilna uporaba uveljavljenih konceptov

Vrste abstrakcij

- Postopkovne abstrakcije
  - Funkcije, metode, moduli, …
- Podatkovne abstrakcije
  - Enostavni podatki, zapisi, objekti, …

Postopkovne abstrakcije

- Sekvenca
- Iteracija
- Procedure in funkcije
- Objekti in razredi
- Moduli
- Vzorci (parametrizirani razredi)

Sekvenca

- Najbolj osnovna abstrakcija v programiranju je izvajanje operacij v sekvenci.
  - Von-Neumanov model računalnika
  - Abstraktni stroj, ki se premakne vedno za en korak naprej

Iteracija

- Osnovni kontrolni gradnik začetnih programskih jezikov
  - Začetno je iteracija služila kot osnovni gradnik jezika, ki je identificiral celo funkcijske celote programa
- Razne oblike iteracijskih stavkov:
  - for, while, repeat, loop, itd

Funkcija

- Abstrakcija obnašanja.
  - Implementacija ni pomembna iz vidika uporabnika
- Imperativni programski jeziki temeljijo na abstrakciji funkcije.
  - Funkcija je del kode, ki ima: ime, parametre, kodo in rezultat
  - Funkcijo lahko kličemo z različnimi parametri
  - Namesto funkcije se izvrši dani del kode z danimi parametri

Funkcija

- Funkcija ima natančno definiran tip.
  - Tip funkcije (metode); tudi signatura
  - $f(n_1{:}t_1, n_2{:}t_2, \ldots, n_n{:}t_n) \rightarrow t$
  - n_i so imena parametrov in t_i so tipi parametrov; t je tip rezultata
- Klic funkcije nadomesti del kode.
  - Uporabnika ne zanima implementacija funkcije

Primer funkcije

```
// implementacija funkcije

int fakulteta( int n ) {
   if (n==1) return 1;
   else
      return n*fakulteta(n-1);
}

// signatura

fakulteta( n:int ) → int
```

```
// uporaba funkcije

out.print(“Število vseh”+
       “možnosti je “+
       fakulteta(10));
```

Lastnosti funkcij

- Formalen pogled
  - Lambda abstrakcija, lambda račun
  - $\lambda x.f \equiv f(x)$
- Funkcije definirajo strukturo programa.
  - Povezave med funkcijami dajo osnovno strukturo programa ali dela programa
- Skrivanje informacij (inf. hiding)
  - Implementacija funkcije vsebuje lokalne spremenljivke, konstante, lokalne funkcije, …
  - Uporabnika ne zanima implementacija

Lastnosti funkcij

- Parametrični polimorfizem
  - Več funkcij z istim imenom
  - Različni parametri
- Funkcije višjega reda
  - Spremenljivke tipov
  - Vrste (kind)

Objekti

- Osnovna abstrakcija jezika.
  - Jezik sloni na abstraktnem konceptu “objekt”
  - Program je množica medseboj povezanih objektov
- Objekti imajo zunanjo podobo in notranjost.
  - Vmesniki, zakrivanje podatkov (enkapsulacija)
  - Objekti komunicirajo med sabo s sporočili
  - Notranjost (implementacija) objekta, privatne metode
  - Neodvisnost vmesnika od implementacije
  - Vmesnik ima lahko več različnih implementacij
- Objekti so definirani znotraj razreda.

Razredi

- Razredi so abstraktna predstavitev množice objektov.
  - Lastnosti + metode + implementacija + ekstenzija
- Tip razreda
  - Javni in skriti podatkovni prilastki in metode
  - (formalno) Zapis sestavljen iz imen in tipov prilastkov in signatur metod
- Vmesnik razreda je javni del tipa
  - Javne lastnosti + javne metode
  - (formalno) Zapis sestavljen iz imen in tipov lastnosti in metod (signatur)

Primer razreda

```
class Radio {
   // staticne lastnosti
   private String znamka;
   private int moc;
   private int tip;
   …

   // obnasanje
   prizgi();
   ugasni();
   pojacajGlasnost(int c);
   zmanjsajGlasnost(int c);
   nastaviFerkvenco(real f);
   …
   // vrednosti
   String getZnamka();
   setZnamka(String znamka);
   int getMoc();
   setMoc(int m);
   int getTip();
   setTip(int t);
}
```

- Radio vidimo kot objekt – abstrakcijo dejanskega aparata
- Precej aspektov nas iz danega gledišča ne zanima
- Objektu pošiljamo sporočila kot v realnem svetu
- Od objekta pričakujemo odgovore kot rezultate akcije

Abstrakcije razredov

- Klasifikacija objektov
- Enkapsulacija
- Dedovanje
- Polimorfizem
- Dinamično povezovanje
- Introspekcija
- Zamenljivost

Klasifikacija

- Razred predstavlja skupino objektov, ki imajo podobne lastnosti.
  - Razred definira osnovno strukturo in obnašanje primerkov
  - Primerki so lahko različni oz. pripadajo različnim podrazredom
- Podrazredi bolj natančno klasificirajo objekte.
  - Primerek nekega razreda je lahko bolj natančno predstavljen kot primerek podrazredov

Enkapsulacija

- Objekte in razrede obravnavamo kot celote, ki imajo zunanjost in notranjost
  - Ovojnica okoli objekta zakrije notranjost objekta
  - Metodo sprožimo tako, da pošljemo sporočilo!
  - Notranjost objekta oz. razreda ni nujno znana uporabniku
  - Navzven je viden samo vmesnik, ki je eksplicitno definiran

Dedovanje

- Konceptualno je pod-razred specializacija nad-razreda oz. nad-razred generalizacija pod-razreda.
- Pod-razred podeduje vse lastnosti nad-razreda.
- Pod-razred ima lahko definirane dodatne lastnosti.

Dedovanje

- Prekrivanje (angl. overriding)
  - Pri dedovanju lahko pride do konfliktov imen
  - Nadrazred: <dostopnost> <tip> <ime-metode>(<parametri>);
  - Podrazred: <dostopnost> <tip> <ime-metode>(<parametri>);
  - Primer: kocka in kvadrat imata metodo narisi();
- Večkratno dedovanje
  - Konflikt imen metod ali lastnosti
  - Referenciranje razredov metod
  - Prepoved večkratnega dedovanja

Dedovanje

- Dedovanje med vmesniki.
- Veljajo enaka pravila kot pri razredih.
- Java dovoljuje večkratno dedovanje med vmesniki.
  - V primeru da so parametri definirani znotraj večih nad-vmesnikih morajo imeti isti tip

Polimorfizem

- Grško: več oblik
  - Metode, objekti, … imajo več oblik
  - Odvisno iz katerega zornega kota jih gledaš
- Imamo več različnih oblik polimorfizma
  - Ad hoc, podtipi, parametričen
- V literaturi je več različnih pojmovanj polimorfizma

Polimorfizem

- Parametrični polimorfizem
  - Metodi z istim imenom in tipom vendar z različnim naborom parametrov
  - Signatura metod se razlikuje samo v parametrih
  - Metodi sta lahko definirani znotraj istega razreda ali v hierarhiji dedovanja
  - V času prevajanja vemo za katero metodo gre
- Primer:
  - Razred kocka ima dve metodi
    - izpiši();
    - izpiši(int rob); // debelina roba

Polimorfizem

- Polimorfizem zaradi podtipov (angl. subtype polymorphism)
  - Toneta lahko vidimo kot osebo, krojača, …
  - Objekt je član vseh nad-razredov $\Rightarrow$ima različne tipe
  - Razred, nad-razredi po pod-razredi imajo lahko več metod z istim imenom in enakimi ali različnimi parametri
- Primer:
  - Primerek kocke lahko obravnavamo kot kocko ali kot kvadrat
  - Kvadrat in kocka imata več metod nariši()

Polimorfizem

- “Ad hoc” polimorfizem
  - Različni razredi imajo lahko metodo z isto signaturo
  - Razredi ni potrebno, da so med seboj povezani
  - Prekrivanje - prej predstavljeno
  - V času prevajanja ne vemo vedno za katero metodo gre

Dinamično povezovanje

- Problem:

```
foreach (obj in geometricObjectCollection) {
    obj je lahko kocka ali kvadrat;
    prevajalnik zve kateri včasu izvajanja;
}
```

- Rešitev:
  - Dinamično povezovanje (angl. dynamic binding)
  - Med izvajanje se metoda dinamično poveže s kodo

Introspekcija

- Razredi so obravnavani kot objekti.
- Tip razreda je vrednost razreda.
  - Abstraktna vrednost, ki definira lastnosti razreda.
  - N-terica, ki opisuje lastnosti razreda
  - Java ima java.lang.reflect
- Večina objektnih jezikov obravnava razreda kot objekte.
  - Smalltalk, Java, C#

Zamenljivost

- Zamenljivost je princip računalniških jezikov.
- S je podtip T $\Longrightarrow$ objekte tipa T lahko zamenjamo z objekti tipa S brez spremembe programa.
- Zamenljivost ni povsod enako implementirana
  - Objektni jeziki
  - Proceduralni ali funkcijski jeziki

Parametrizirani razredi

- Jeziki s parametriziranimi razredi
  - C++, Java, Ocaml
- Primer v Javi:
  - `List<String>` je parametrični tip
  - Iz parametričnega tipa lahko izpeljemo raznolike tipe seznamov:
    - `List<Int>, List<Oseba>, List<String>, List<Object>`
  - `Int,Oseba,String,Object` so v vlogi (formalnih) parametrov tipa
- Parametrični tipi izhajajo iz funkcijskih jezikov – Lisp, ML, Schema

Java Generics

```
public interface IntegerList {
      void add(Integer x)
      Iterator<Integer> iterator();
}
```

- Parametrični tipi v Javi
  - Nastajali so v zadnjih desetletjih
  - C++ templates, ML
- Metode
  - Formalni parametri vrednosti
  - Ob klicu se formalni zamenjajo z dejanskimi
- Podobno z generičnimi tipi
  - Formalni parametri tipa
  - Ob uporabi se spremenijo v dejanske
- Razlika s C++ templates
  - Java vsebuje samo eno instanco kode

```
public interface List<E> {
      void add(E x);
      Iterator<E> iterator();
}
```

Moduli

- Modul združuje kodo definirano okoli nekega koncepta.
  - Delo s specifičnimi podatkovnimi strukturami
  - Implementacije specifičnih algoritmov
- Modul združuje množico razredov sorodnega ali istega tipa
- Modul ima lahko:
  - Lasten naslovni prostor
  - Vmesnik – kaj je dostopno od zunaj
  - Implementacija - koda
- Abstrakcijo modula uporabljajo:
  - Ada, Modula, Perl, ML, Ocaml, (nekateri) Pascal, …

Ocaml moduli

- Vmesnik modula definira stvari dostopne od zunaj
  - Implementacija modula je skrita; lahko skrijemo definicijo tipov, definicijo podatkovni struktur, itd.
  - Implementacijo lahko spreminjamo brez spremembe vmesnika
  - Prevajamo in razvijamo lahko ločeno od sistema
- Lasten naslovni prostor
- Modul uporablja druge module
  - Dobimo lahko tudi ciklične strukture
- Imamo tudi funktorje (parametrizirane module)

Java paketi

- Grupiranje razredov v direktorije primer
  - java.lang, java.io, …
- Java paketi nimajo posebnih lastnosti
  - Ni samostojnega naslovnega prostora (Perl,Ocaml)
  - Ni vmesnika modula (Ada,Ocaml)
- Veliko število gradnikov naredi jezik kompleksen
  - Ada, C++, …

Funktorji

- Parametrizirani moduli
- Modul je lahko parameter modula
  - V modulu lahko delamo z generičnim modulom katerega implementacija ni znana
  - Ključ podatkovne strukture zakrit z modulom
  - Urejenost podatkov zakrita z modulom
- Jeziki, ki vsebujejo funktorje:
  - Ocaml, SML

Podatkovne abstrakcije

- Števila, znaki, osnovni tipi
- Podatkovne strukture
- Razredi, klasifikacija
- Asociacija
- Kompozicija / dekompozicija
- Specializacija / generalizacija

Števila, znaki

- Tudi običajni simboli so abstrakcije
  - Število “10” ima lahko veliko možnih interpretacij
    - Binarni zapis 0101
    - Šestnajstiški zapis A
    - Znakovni zapis
  - Znaki so zelo ustaljene abstrakcije
    - Vseeno imamo zelo veliko vrst zapisov

Podatkovne strukture

- (Kartezijski) produkt definira podatkovno strukturo n-teric
  - Tip $T=T_1 x ... x T_n$
  - Primerek $(v_1,...,v_n)$
  - Semantični podatkovni modeli in funkcijski jeziki
- Zapisi definirajo agregacijo komponent zapisa
  - Komponenta je lahko poljubnega tipa
  - Tip $T=\{a_1:T_1,...,a_n:T_n\}$
  - Primerek $\{a_1=v_1,...,a_n=v_n\}$
  - Zapise vsebujejo praktično vsi podatkovni modeli
    - Relacijski, ER, semantični modeli, UML, ...
    - ... in večino programskih jezikov

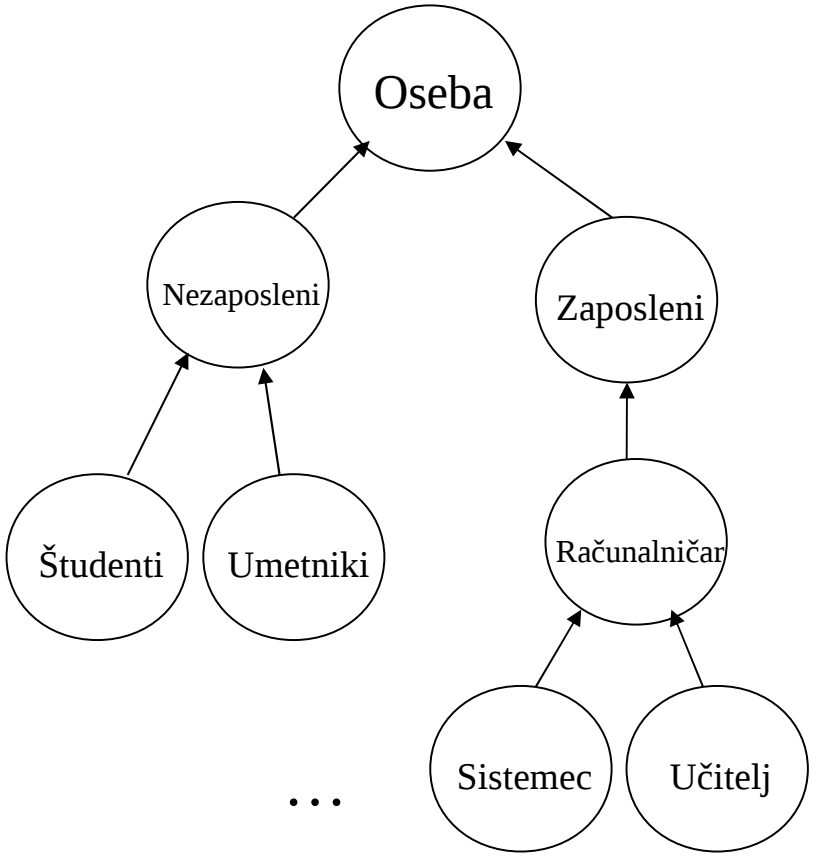
Podatkovne strukture

- Unija definira tip, ki predstavlja več tipov
  - Tip $T=T_1+...+T_n$
  - Vsak primerek T je primerek T_i za nek i
- Unijo vsebujejo semantični podatkovni modeli
  - SDM, IFO in nekateri programski jeziki

Razredi in klasifikacija

- Razred predstavi množico predmetov ali konceptov iz neke realnosti
  - Razred vsebuje opis statičnih lastnosti
  - Razred vsebuje opis obnašanja
- Zasnova programa
  - Razvrščanje konceptov v razrede
  - Razredi so povezani preko mehanizma dedovanja
- Klasifikacijska struktura razredov da osnovno strukturo (okostje) programa

Razredi in klasifikacija



- Relacija → je dedovanje
- Lahko je kaj drugega npr. specializacija/generalizacija (modeli)
- Običajno dedovanje – drevo
- Večkratno dedovanje – DAG
- Kompleksne hierarhije – ontologije
- Koncepti, področja, kniga, članki, zdravila, recepti, …
- Pogosta uporaba ontologij v novejših informacijskih sistemih

Asociacije

- Binarne asociacije
  - Razmerje v ER, XML Link, Trojice v RDF, ...
  - Semantični modeli: asociacije
  - Lastnosti razreda v programskih jezikih
- Asociacije med več objekti
  - Razmerja v ER, UML asociacije, XML Link, ...
  - Asociacijo lahko implementiramo kot razred, ki referencira razrede v razmerju

Specializacija/generalizacija

- Koncept (razred) lahko
  - Posplošimo (G)
    - miza → pohištvo, mačka → žival, svinčnik → pisalo
  - Specializiramo (S)
    - knjiga → kuharska knjiga, škatla → računalnik
- Ena od primarnih abstrakcij
  - Aristotel - tvorjenje konceptov: genus – differentiae
- Gradniki za definicijo strukture programov in sistemov
  - G je “obratno” S ?
  - Zamenjava G in S ne da vedno ekvivalentne predstavitve
  - Včasih je sam proces nastajanja koncepta razreda pomemben

Specializacija/generalizacija

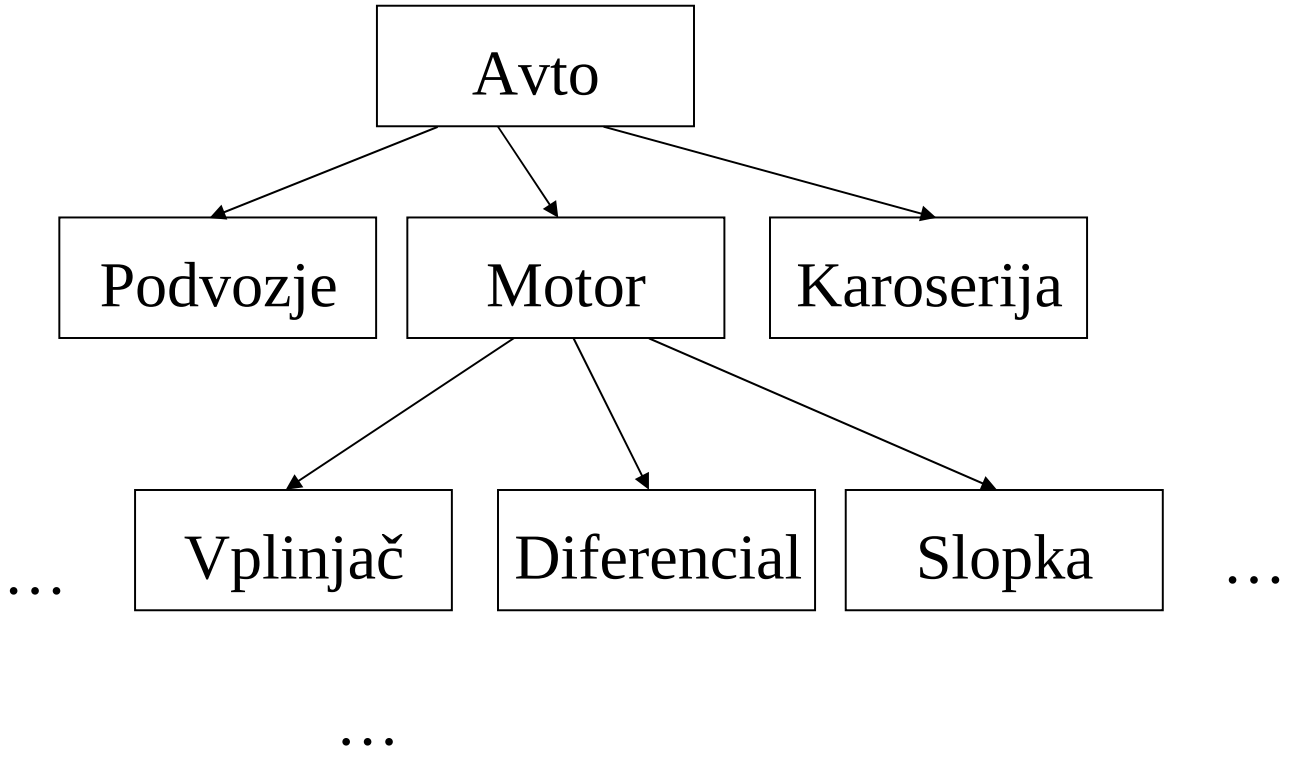
```
class Oseba {…}
class Nezaposleni
      extends Oseba { … }
class Zaposleni
      extends Oseba { … }
class Racunalnicar
      extends Zaposleni { … }
class Sistemec
      extends Racunalnicar { … }
class Ucitelj
      extends Racunalnicar { … }
```

- V Javi ni eksplicitnega gradnika za predstavitev abstrakcij S / G
- “extends” bolj asocira na “specializacijo”
- Unified Modelling Language (UML) se pogosto uporablja za načrtovanje programov
- UML vsebuje:
  - specializacijo
  - generalizacijo
  - kompozicijo
  - dekompozicjo

Kompozicija/dekompozicija

- Sestava objektov
  - Objekt je fizično ali logično sestavljen iz komponent
  - Model naj čim bolj verno odraža dejansko stanje !
- Ena od primarnih abstrakcij
- Implementacija
  - Komponente objektov so objekti
  - Reference med razredi (prilastki so uporabniški razredi)
  - Uporaba večkratnega dedovanja za kompozicijo ni v redu !
    zakaj? Drugačen namen gradnikov.

Kompozicija/dekompozicija



Kompozicija/dekompozicija

```
class Avto {
   // komponente
   Podvozje podvozje= null;
   Motor motor = null;
   Karoserija karoserija = null;
   …
}

class Podvozje {
    Dimenzije dim;
   …
}

class Karoserija {
   int barva;
   int tip;
   int oblika;
   …
}
```

```
class Motor {
   // lastnosti
   int moc;
   int teza;

   // komponente
   Sklopka skl;
   Diferencial dif;
   Vplinjac vpl;
   …
}

class Sklopka {
…
}

class  Diferencial {
…
}
…
```

v Javi ni eksplicitnega gradnika za kompozicijo

Struktura modela

- Kaj definira strukturo modela?
- Moduli in paketi
  - Povezave med moduli in paketi (uporaba)
- Razredi (abstraktni podatkovni tipi)
  - Povezave med razredi (uporaba)
- Klasifikacijska hierarhija razredov
  - Specializacija/generalizacija, Dedovanje
- Kompozicijska hierarhija razredov
  - Kompozicija/dekompozicija, Reference med objekti
- Metode (funkcije)
  - Povezave med metodami (klici)