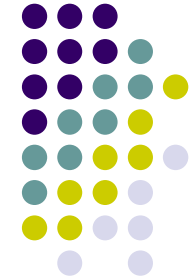
# Niz koraka u oblikovanju komponente



- 1) Određivanje koncepata/klasa i temeljnih relacija među njima
- 2) Profinjenje klasa određivanjem skupa operacija nad njima
  - Klasificiranje operacija. Posebno konstruktora, destruktora, kopiranja
- 3) Profinjenje klasa određivanjem zavisnosti
  - mogućnosti parametrizacije, nasljeđivanja, korištenje zavisnosti
- 4) Definiranje sučelja
  - Odvajanje funkcija u javne i privatne operacije
  - Određivanje točnog tipa operacija na klasama

# 1) Određivanje klasa

- Razgovor s budućim korisnicima sustava ili (nezadovoljnim) korisnicima postojećeg sustava
  - imenice >> klase i objekti
  - glagoli >> operacije na objektima
- Odmak od postojećeg sustava
- Generalizacija i klasifikacija (primjer kružnice i elipse)
- Analiza primjera korištenja (Iniciranje poziva = podigni slušalicu; nazovi broj; telefon na drugoj strani zvoni; osoba na drugoj strani podiže slušalicu)
- Nakon nekoliko primjera korištenja, oblikuje se sustav i testira kako funkcionira u ostalim slučajevima (iterativni proces) – pokrivanje cca 80% glavnih situacija i neke izuzetke
- Neka bitna svojstva sustava (pouzdanost, performanse, provjera ispravnosti ne mogu se reprezentirati klasama

# 2) Određivanje operacija

- Kako se objekt neke klase konstruira, uništava, kopira
- Koji je minimalni skup operacija potreban za funkcioniranje klase
- Određivanje standardnih naziva i funkcionalnosti u svim klasama unutar komponente
- Operacije:
  - stvaranje i uništavanje
  - nadzor (ne mijenjaju stanje objekta)
  - promjena (mijenjaju stanje objekta)
  - konverzija (generiranje objekta drugog tipa ovisno o stanju originalnog objekta)
  - iterator (pristup i korištenje niza sadržanih objekata)

# 3) Određivanje zavisnosti

- Zavisnosti
  - parametrizacija
  - nasljeđivanje
  - korištenje relacija
- Ne pretjerivati u korištenju zavisnosti

# 4) Određivanje sučelja

- Privatne funkcije obično ne treba razmatrati u fazi dizajna. Ove funkcije bi trebale biti izvedive, neovisno o implementaciji
- Javne funkcije – specificiraju se točni tipovi argumenata

# Korištenje modela

- Uvijek kada je moguće, dizajn i programiranje trebaju biti temeljeni na prethodnom radu
- Odabir modela pogodnog za prilagodbu
- Korištenje obrazaca u programiranju – korištenje konkretnih primjera za ilustraciju općeg principa
- Svaki uspješan veliki sustav nastao je redizajnom nekog manjeg upogonjenog sustava

# Eksperimentiranje, analiza, testiranje

- Prototip – umanjena verzija sustava ili nekog njegovog dijela
- Kako testirati, što testirati? Situacije koje mogu imati kobne posljedice i situacije koje se često javljaju. Testiranju se obično ne posvećuje dovoljno pažnje

# Održavanje softvera

- Redizajn, ponovna implementacija – tj. ciklus razvoja programa
- Obično vezano za istu razvojnu skupinu (dokumentacija bolje pomaže razumijevanju detalja nego u razumijevanju ključnih ideja i principa)

# Učinkovitost

- Čist i jednostavan dizajn
- Optimizacija u kasnijoj fazi, na temelju mjerenja performansi i određivanja uskih grla
- U fazi dizajna treba nastojati izbjegavati rješenja koja će se kasnije teško optimizirati

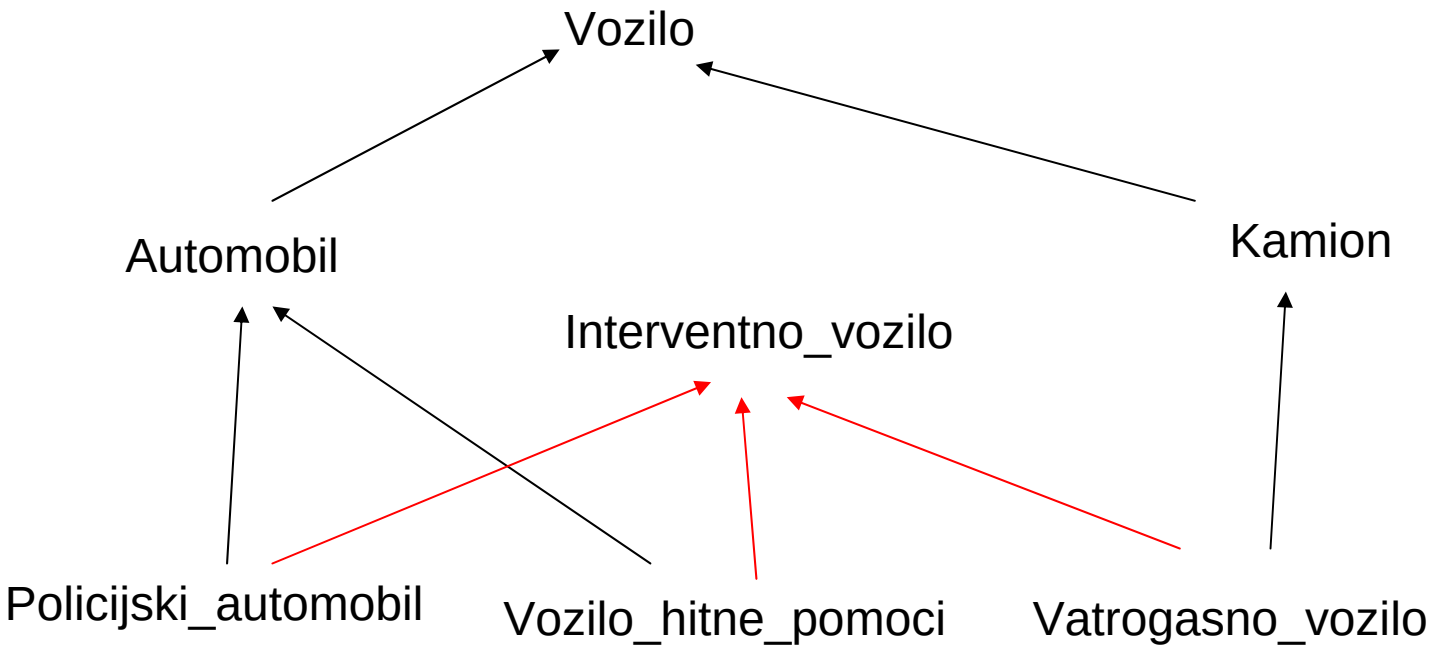
# I još...

- Upravljanje softverskim projektom
- Ponovna upotrebljivost koda
- Mogućnost primjene rješenja na veće probleme
- Izvedivost s postojećim ljudskim resursima
- Hibridni dizajn

# Uloga klasa – razine apstrakcije

- Kuća:
  - atomi
  - molekule
  - cigle i druga građa
  - podovi, zidovi, stropovi
  - sobe
- odgovarajuća razina detalja i odgovarajuća razina apstrakcije

# Hijerarhija klasa




- usmjereni aciklički graf klasa

# Implementacije hijerarhije klasa

- class Vozilo { /* ... */ };
- class Interventno_vozilo { /* ... */ };
- class Automobil : public Vozilo { /* ... */ };
- class Kamion : public Vozilo { /* ... */ };
- class Policijski_automobil : public Automobil, protected Interventno_vozilo { /* ... */ };
- class Vozilo_hitne_pomoci : public Automobil, protected Interventno_vozilo { /* ... */ };
- class Vatrogasno_vozilo: public Kamion, protected Interventno_vozilo { /* ... */ };

# protected, private, public

- private – ime se može koristiti samo u članskim funkcijama
- protected – ime se može koristiti samo u članskim funkcijama klase u kojoj je deklarirano, te u klasama izvedenim iz te klase
- public – ime se može koristiti u svim funkcijama

# izvedene klase