Zavod za elektroniku, mikroelektroniku
računalne i inteligentne sustave

22. travnja 2021.

# Oblikovni obrasci u programiranju

međuspit

1. Razmotrite sljedeći kod u programskom jeziku C++ i ponudite odgovarajući kod u C-u. Uputa: svaki razred izvedite odgovarajućom strukturom, te svaku metodu - odgovarajućom funkcijom.

```
class Base{
  int m_;
public:
  Base(){
    m_=0;
  }
  virtual void setm(int x){m_=x;}
};
class Worker{
public:
  virtual int b(){return 42;}
};
```

```
class Client: public Base{
public:
  Client(Worker* pw){
    setm(pw->b());
  }
};

int main(){
  Worker w;
  Client c(&w);
}
```

2. U okviru 4. zadatka 2. laboratorijske vježbe oblikovali ste komponentu DistributionTester koja je na nekoliko načina mogla generirati cijele brojeve te na temelju generiranih brojeva računati određene statistike. Sada je potrebno proširiti funkcionalnost te komponente tako da se, neovisno o odabranom načinu generiranja, podrže još i sljedeće funkcionalnosti:

   - ograničavanje broja elemenata koji se povlače iz izvora
   - modificiranje brojeva koje izvor generira na neki proizvoljan način (primjerice, kvadriranjem, uvećavanjem za 5, ...)

   Navedene funkcionalnosti trebaju se moći proizvoljno kombinirati. Očekujemo da bi se u budućnosti mogli pojaviti i drugi takvi zahtjevi. Vaši zadatci su:

   (a) Predložite prikladnu programsku organizaciju i skicirajte kod izvedbe najvažnijih elemenata.

   (b) Navedite koje oblikovne obrasce ste koristili, nacrtajte strukturni dijagram te povežite sudionike obrazaca s elementima svoje implementacije.

3. Razmatramo programski sustav koji obrađuje podatke o tvrtkama, zaposlenicima i vozilima. Svaka tvrtka ima više zaposlenika i više poslovnih vozila. Poslovna vozila mogu biti kamioni, automobili i kombiji. Svaki zaposlenik može imati osobno vozilo. Osobna vozila mogu biti bicikli, motocikli, romobili i automobili.

   Skicirajte izvedbu funkcije koja određuje potrebni broj parkirnih mjesta za sva poslovna i sva osobna vozila. Pretpostavite da su sva vozila odgovorna za čuvanje i pružanje informacije o tome koliko parkirnih mjesta zauzimaju. Pokušajte izvesti funkciju bez korištenja naredbe **if**.

   Skicirajte dijagram razreda koji bi podržao opisanu domenu. Pojasnite s kojim je oblikovnim načelima usklađeno vaše rješenje.

4. Razvijamo informatički sustav za javno parkiralište s 500 parkirnih mjesta. Na parkiralištu je postavljen senzor koji detektira ulaze i izlaze vozila te ekran za prikaz proizvoljnog teksta. Ekran prikazuje isti tekst dok ne dobije novu poruku. Postojeća programska podrška komponente Senzor referencira objekt tipa `IParkingStanje` te osigurava da se nakon svakog ulaza odnosno izlaza vozila poziva metoda `IParkingStanje.ulaz()`, odnosno `IParkingStanje.izlaz()`. Ekran i Rampa imaju sljedeće gotove implementacije:

```
public class Ekran {
    public void prikazi(String tekst) {...}
}
```

```
public class Rampa {
    public void blokiraj_ulaz() {...}
    public void odblokiraj_ulaz() {...}
}
```

Zahtjevi informatičkog sustava su sljedeći:

- Ekran mora prikazivati trenutni broj vozila na parkingu.
- Svaki ulazak/izlazak mora biti evidentiran u internom logu "Untitled.txt", gdje svaki redak sadrži trenutno vrijeme i trenutni broj vozila na parkingu.
- Ukoliko je parking popunjen potrebno je blokirati rampu dok se ne oslobodi barem jedno mjesto.

Glavni program sustava izgleda ovako:

```
public class ParkingSustav {
    public static void main(String[] args) {
        Rampa rampa = poveziRampu();
        Ekran ekran = poveziEkran();
        int kapacitet = 500;
        IParkingStanje stanje = ParkingStanje(kapacitet)
        Senzor senzor = poveziSenzor(stanje)
        // Vas kod ovdje!
        radiZauvijek(senzor, rampa, ekran);
    }
}
```

Implementirajte djelove sustava koji nedostaju, dovrsite glavnu metodu, navedite korištene obrasce te povežite sudionike s elementima svojeg rješenja. Nije potrebno pisati kod za klase Rampa, Ekran i Senzor, te metode poveziRampu, poveziEkran, poveziSenzor i radiZauvijek.

5. Razmatramo sustav strojnog učenja za raspoznavanje slika. Korisnik je napisao petlju učenja koja učitava slike metodom `ucitajSliku(int indeks)` razreda `SkupPodataka`.

```
SkupPodataka skup_podataka("/gdje/su/slike/")
for(int epoch=1; epoch <= num_epochs; epoch++){
    for(int i=0; i < broj_slika; i++){
        slika = skup_podataka.ucitajSliku(i);
        korak_ucenja(slika);
    }
}
```

Potrebno je omogućiti umjetno uvećanje raznolikosti skupa podataka slučajnim transformiranjem slika nakon učitavanja. Korisnik bi trebao moći jednostavno zadavati razne slikovne transformacije (npr. zumiranje, izrezivanje, promjena kontrasta) kao i njihove proizvoljne kombinacije. Predložite rješenje koje bi omogućilo provedbu navedenih zahtjeva u skladu s načelima oblikovanja i bez izmjene predložene petlje učenja.

Zadaci:

(a) Nacrtajte dijagram razreda predloženog rješenja te skicirajte novu izvedbu razreda SkupPodataka.

(b) Navedite obrazac koji ste koristili i povežite njegove generičke sudionike s Vašim konkretnim rješenjem.

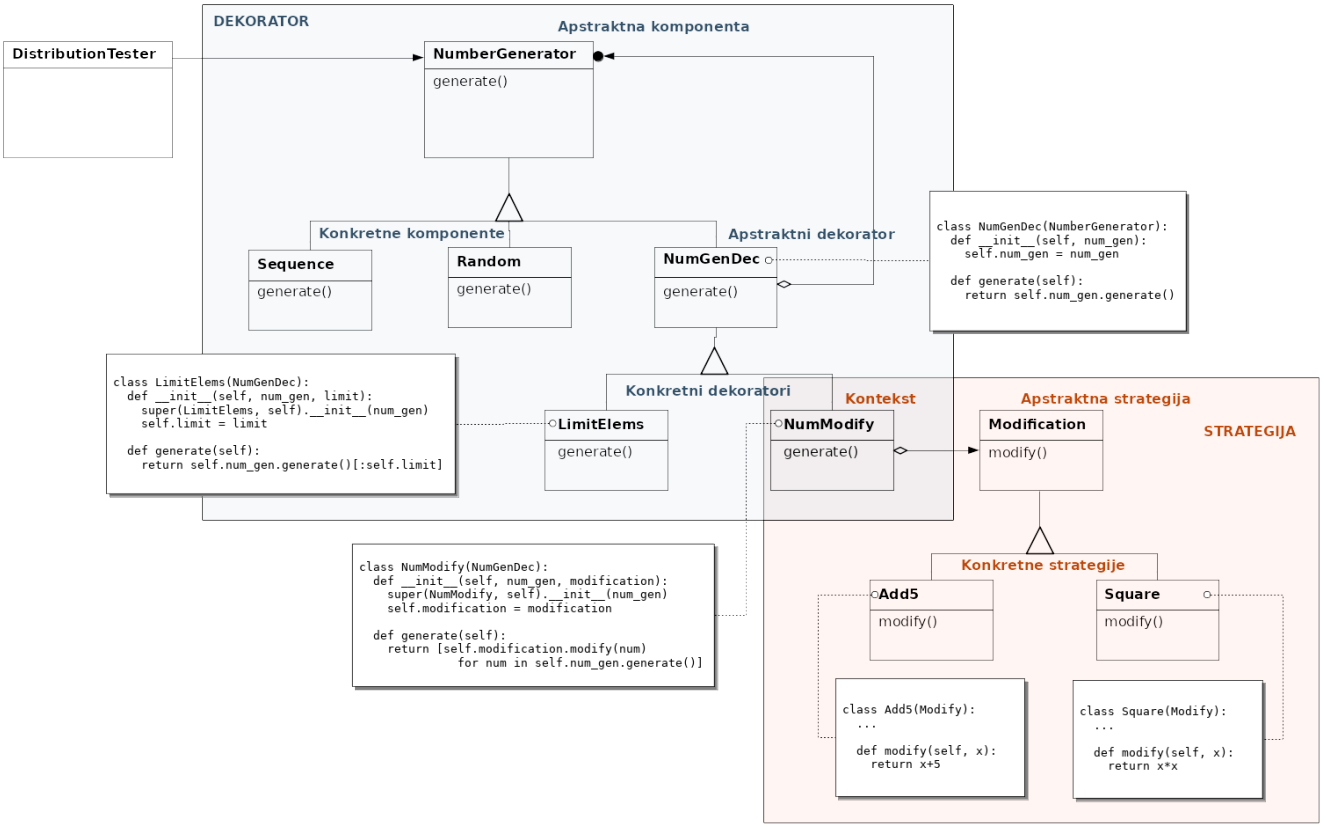
(c) Skicirajte kod koji stvara objekt nove izvedbe razreda SkupPodataka, a čija metoda `ucitajSliku` vraća slučajno zumiranu sliku sa slučajno promijenjenim kontrastom.

Uputa: Pretpostavite da je na raspolaganju biblioteka ImgUtil čija funkcionalnost pojednostavnjuje izvedbu potrebnih slikovnih transformacija.

# 1 Rješenje zadatka s OOP u C-u

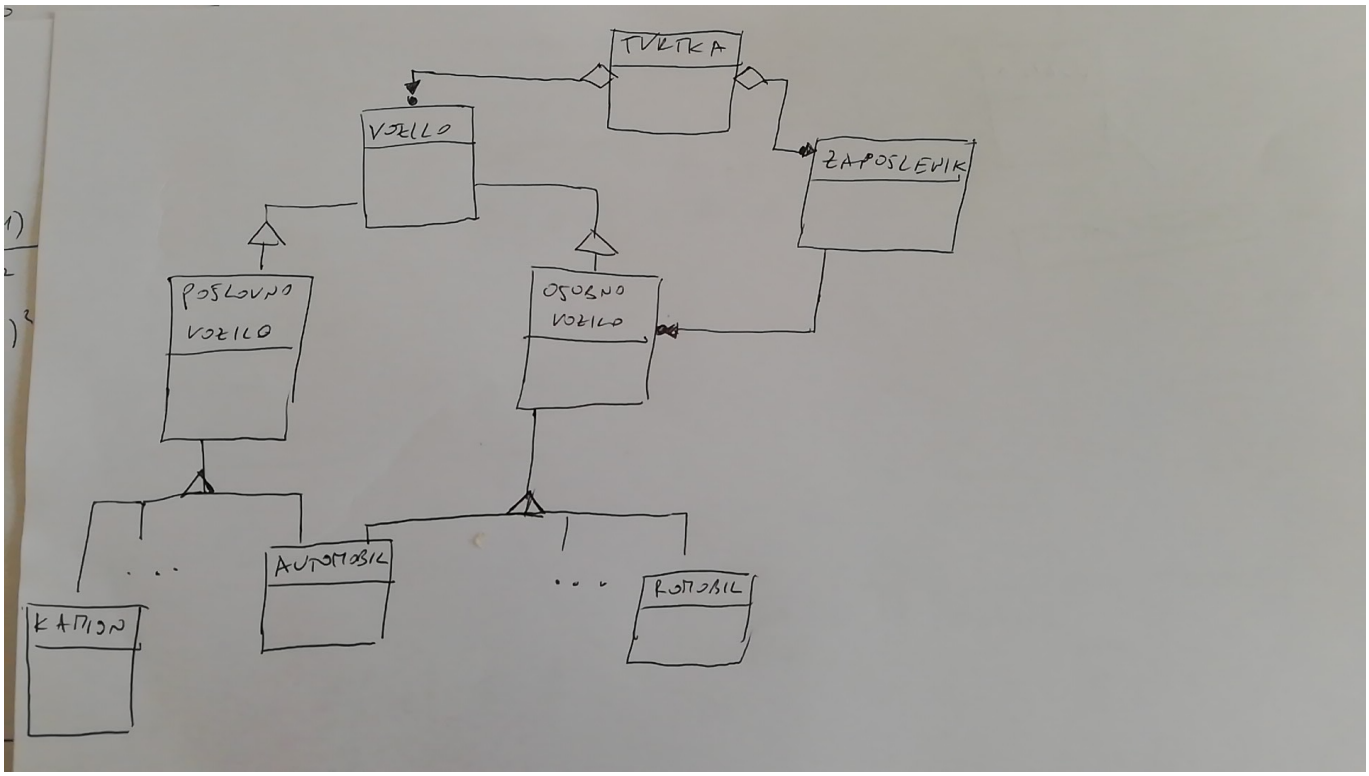
```
typedef int (*PTRFUN)();

/*================================*/
struct StrBase{
  PTRFUN* vptr;
  int m;
};
typedef struct StrBase Base;

int BaseSetM(Base* this, int x){
  this->m=x;
}
PTRFUN BaseVT[1] = {BaseSetM};

void BaseCtr(Base* this){
  this->vptr = &BaseVT[0];
  this->m=0;
}

/*================================*/
struct StrWorker{
  PTRFUN* vptr;
};
typedef struct StrWorker Worker;

int WorkerB(Worker *this){
  return 42;
}
PTRFUN WorkerVT[1] = {WorkerB};

void WorkerCtr(Worker* this){
  this->vptr = WorkerVT;
}

/*================================*/
void ClientCtr(Base* this, Worker* pw){
  BaseCtr(this);
  this->vptr = &BaseVT[0];
  int newm = pw->vptr[0](pw);
  this->vptr[0](this, newm);
}

/*================================*/
int main(){
  Worker w;
  WorkerCtr(&w);

  Base c;
  ClientCtr(&c,&w);
}
```

# 2 Rješenje zadatka iz labosa




# 3 Rješenje zadatka s tvrtkom i zaposlenicima

```
def koliko_treba_mjesta(tvrtka):
  nm = 0
  for v in tvrtka.poslovna_vozila:
    nm += v.nmjesta()
  for z in tvrtka.zaposlenici:
    nm += z.vozilo.nmjesta()
  return nm
```




Slika 1: Caption

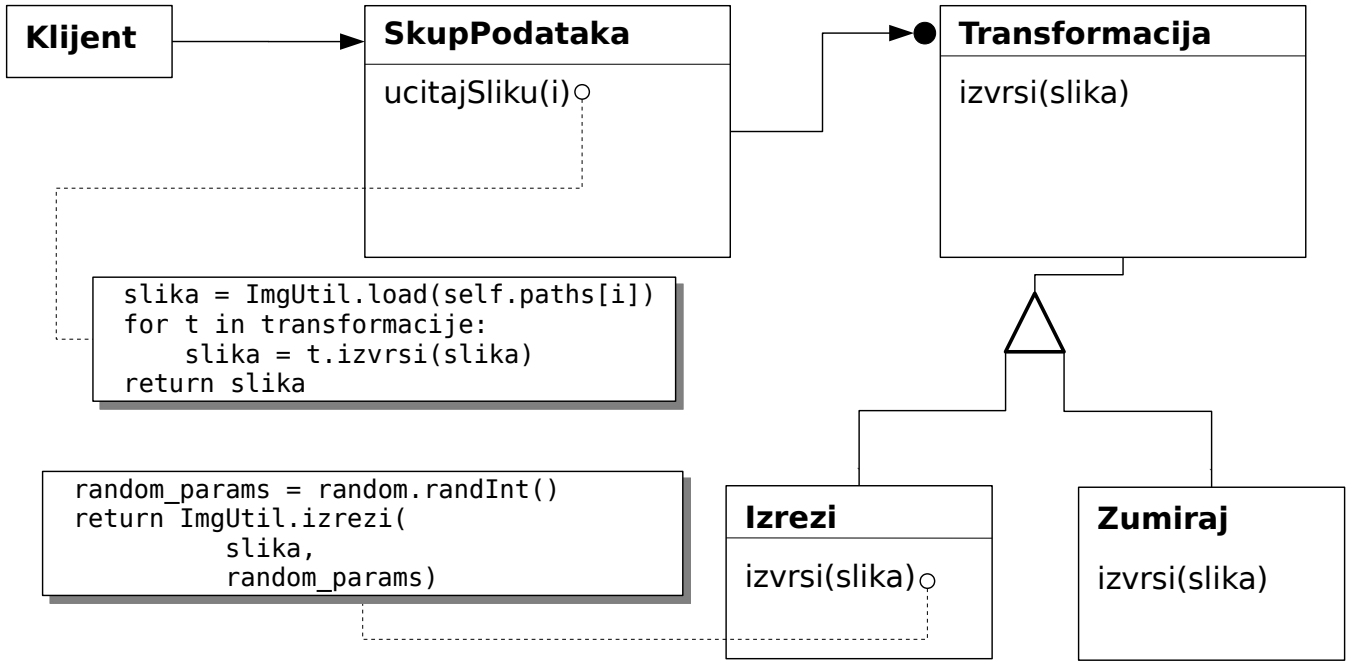
# 4 Rjesenje zadatka s Parking sustavom

```
public interface IParkingStanje {
    public void ulaz();
    public void izlaz();
    public int getBrojac();
    public int getKapacitet();
}

public class ParkingStanje implements IParkingStanje {
    public int brojac = 0;
    public int kapacitet;
    private List<IPromatrac> promatraci = new ArrayList<>()

    public ParkingStanje(int kapacitet) {
        this.kapacitet = kapacitet;
    }

    public void ulaz() {
        this.brojac++;
        this.obavijestiPromatrace();
    }

    public void izlaz() {
        this.brojac--;
        this.obavijestiPromatrace();
    }

    public void dodajPromatraca(IPromatrac p) {
        this.promatraci.add(p);
    }

    public void makniPromatraca(IPromatrac p) {
        this.promatraci.remove(p);
    }

    private void obavijestiPromatrace() {
```

```
        this.promatraci.foreach(t -> t.osvjezi(this));
    }

}

public interface IPromatrac {
    public void osvjezi(IParkingStanje);
}

public class EkranPromatrac implements IPromatrac {
    private Ekran ekran;

    public EkranPromatrac(Ekran ekran) {
        this.ekran = ekran;
    }

    public void osvjezi(IParkingStanje stanje) {
        this.ekran.prikazi(stanje.getBrojac())
    }
}

public class RampaPromatrac implements IPromatrac {
    private Rampa rampa;

    public EkranPromatrac(Rampa rampa) {
        this.rampa = rampa;
    }

    public void osvjezi(IParkingStanje stanje) {
        if stanje.getBrojac() == stanje.getKapacitet() {
            this.rampa.blokirajUlaz();
        }
        else {
            this.rampa.odblokirajUlaz();
        }
    }
}

public class LogPromatrac implements IPromatrac {
    private Writer writer;

    public LogPromatrac(String filename) {
        this.writer = new BufferedWriter(new OutputStreamWriter(
              new FileOutputStream(filename), "utf-8")));
    }

    public void osvjezi(IParkingStanje stanje) {
        writer.write(
            "Vrijeme:␣" + Utils.getVrijeme() + "␣brojac:␣" + stanje.getBrojac().toString()
        );
    }
}

public class ParkingSustav {
        public static void main(String[] args) {
                Rampa  rampa = poveziRampu();
                Ekran  ekran = poveziEkran();
                int kapacitet = 500;
                IParkingStanje  stanje = ParkingStanje(kapacitet);
                Senzor  senzor = poveziSenzor(stanje);

                stanje.dodajPromatraca(new EkranPromatrac(ekran));
                stanje.dodajPromatraca(new RampaPromatrac(rampa));
                stanje.dodajPromatraca(new LogPromatrac("Untitled.txt"));

                radiZauvijek(senzor, rampa, ekran);
        }
}
```

# 5 Rješenje zadatka s transformiranjem slika

```
class SkupPodataka:
    def __init__(self, path, transformacije):
        self.paths = glob.glob(path + "*.png")
        self.transformacije = transformacije
    def ucitajSliku(i):
        img = ImgUtil.load(self.paths[i])
        for t in self.transformacije:
            img = t.izvrsi(img)
        return img

class Transformacija:
    def __init__():
        pass
    def izvrsi(img):
        # return img
        pass

skup_podataka = SkupPodataka("/gdje/su/slike/", [Zumiraj(), PromijeniKontrast()])
```




Slika 2: Caption