Objektno orjentisano programiranje:

Enkapsulacija

[Prilagođeno od Almir Vuk]

Enkapsulacija

Proces kombinovanja i zaštite atributa i funkcija u jedinstvenu jedinicu zvanu klasa

rezultat enkapsulacije: podaci nisu dostupni direktno preko objekata, nego je potrebno da imamo nešto što se naziva getter i setter funkcija:

GET - dobiti vrijednost atributa iz klase,

SET - postaviti vrijednost atributa u klasi

Atributi klase se čuvaju privatno, a preko javnih gettera podatke preuzimamo dok preko settera se kroz unaprijed definisana pravila mjenja stanje u klasi.

Enkapsulacija

Primjer enkapsulacije:

za vožnju automobila nije poterbno poznavanje rada svakog dijela vozila(motor, alternator, mjenjač, itd ...).

Treba znati kako upravljati volanom, kako i kada kočiti ili ubrzati kako bi se kretali od tačke A do tačke B.

Enkapsulacija

Primjer enkapsulacije u programiranju su ugrađene funkcije:

```
int x = 712;
String s = x.ToString();
```

Nebitno koji je mehanizam ugrađen u funkciju `ToString`, važno da obavlja zadatak kako treba.

Tako i naše korisnički definisane klase pokazuju samo šta treba da bude vidljivo.

Enkapsulacija

Zašto je Enkapsulacija korisna?

Enkapsulacija pomaže da se piše čistiji kod, pri čemu omogućava da podaci i mehanzimi u klasama budu zaštićeni i dostupni samo preko javnih getter i setter funkcija.

Uvođenjem enkapsulacije klasa preuzima potpunu kontrolu nad podacima koji se smještaju u njoj.

Enkapsulacija

Obezbjeđujući samo seter ili geter metod, možemo učiniti klasu readonly ili writeonly čime obezbjeđujemo kontrolu nad podacima.

Za primjenu osnovnih pravila enkapsulacije potrebno je da klase imaju slijedeće:

- privatne (private) atribute
- javne (public) get/set metode

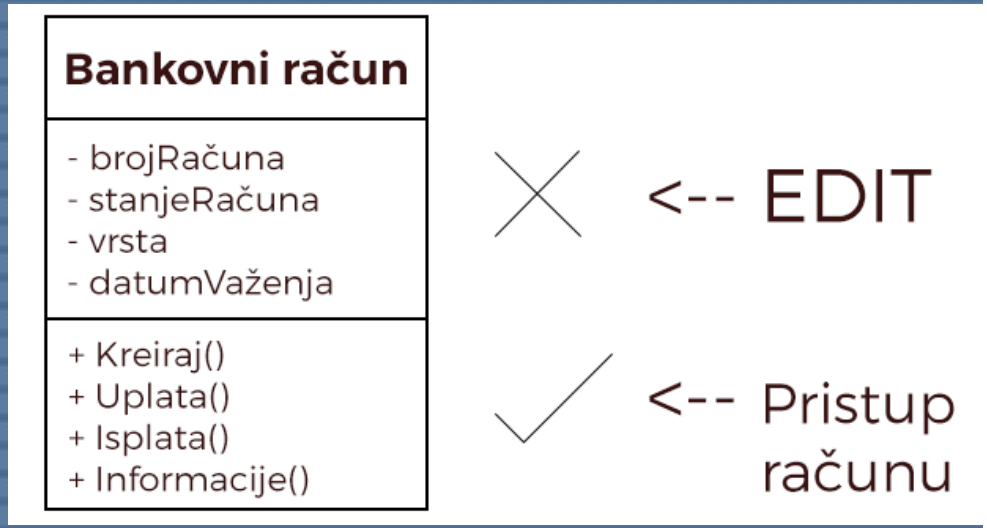
```
public class Student{
    private String name;

    public String getName(){
            return name;
    }
    public void setName(String name){
            this.name=name
    }
}
```

Enkapsulacija

Enkapsulacija je učahurivanje klase i podataka, što predstavlja na neki način i sigurnost naše aplikacije

U aplikaciju za vođenje bankovnog računa postoji klasa ’račun u banci’, koja može ima atribute i metode:

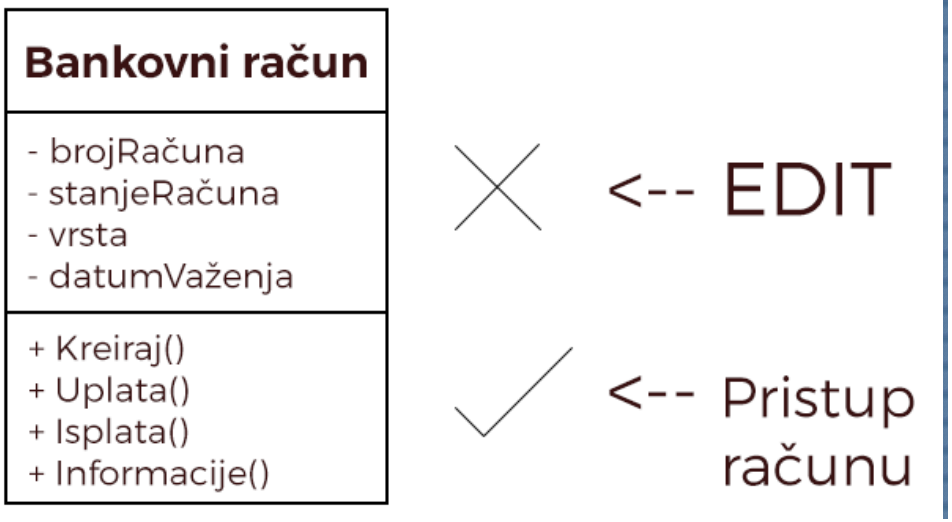


Enkapsulacija

Enkapsulacija zabranjuje direktan pristup i izmjenu atributa pristup, pregled i izmjena atiributa radi se preko odgovarajućih metoda (getter i setter)

Promjena stanja atributa u objektu bi se radila na slijedeći način:

```
BankovniRačun prvi = new BankovniRačun();
prvi.stanjeRačuna = 712;
```

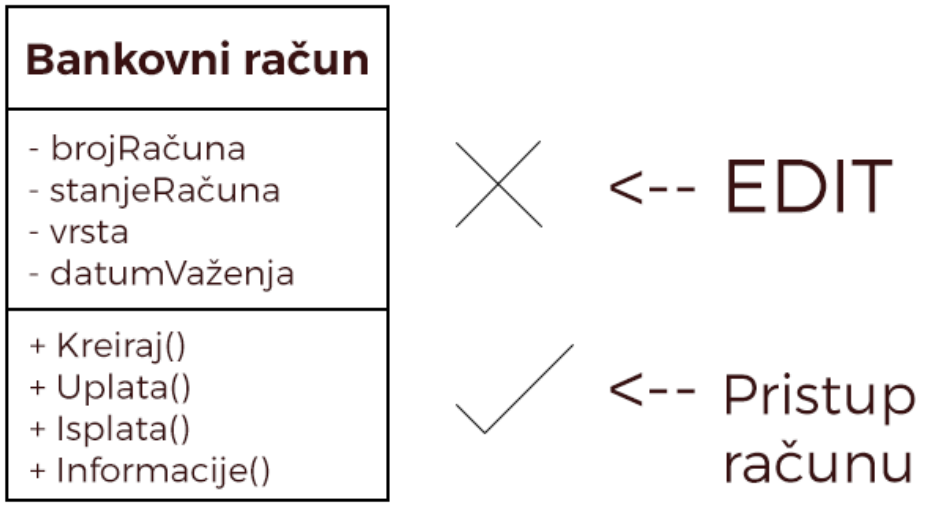


Enkapsulacija

Prvi način (direktni pristup) je označen kao nepravilan.

U drugom načinu postoje metode koje na siguran i provjeren način rade sa računima

- prije metode Isplata() može se provjeriti da li je korisnik autorizovan da mijenja stanje računa
- Unutar metode Isplata() može se provjeriti da li korisnik ima dovoljno sredstava za isplatu



Enkapsulacija

Može se reći da enkapsulacija pruža osnovni stepen sigurnosti u aplikacijama i predstavlja prvi korak ka većoj sigurnosti sistema.

Enkapsulacija

Zašto bi developer aplikacije krio podatke od samoga sebe?

Enkapsulacija nema cilj skrivati i otežati pristup do podataka.

Naprotiv, praktičnije je i pomaže modularnosti klasa i aplikacije jer izmjena jednog dijela klase je primjenjena na svim mjestima na gdje se koriste metode bez potrebe za ručnim izmjenama.

Enkapsulacija

Koliko atributa postaviti kao private?

Što više to bolje!

Enkapsulacija

```
public class Student{                    //sačuvaj kao Student.java
    private String name;

    public String getName(){
           return name;
    }
    public void setName(String name){
           this.name=name
    }
 }

package com.company;                     //sačuvaj kao Test.java
class Test{
    public static void main(String[] args){
           Student s=new Student();
           s.setName("Petar");
           System.out.println(s.getName());
    }
}
```

Enkapsulacija

```
package hr.fer.oop.topic2.example2;
public class Student {
 private String id;
 private String name;
 private String surname;
 private int noOfGrades;
 private CourseGrade[] grades;
 public void init(){ ... }
 public void addGrade(int code, String
        title, int grade){ ... }
 public double addGrade(){ ... }
 public String getId() { return id; }
 public void setId(String newId) {
         id = newId;
 }
 public  String  getName()  {  return
name; }
 public void setName(String newName) {
         name = newName;
 }
 public String getSurname(){
        return surname;}
 public void setSurname(String
newSurname) {
 surname = newSurname;
```

```
package hr.fer.oop.topic2.example2;
public class Main {
 public static void main(String[] args)
 Student s = new Student();
 s.init();
 s.setName(„Scott");
 s.setSurname(„Tiger");
 s.setId(„1936775");
 s.addGrade(105, "OOP", 9);
        ...
 System.out.println(s.averageGrade());
```