

NAPOMENA: Vežbe podrazumevaju da je student ovladao teorijom iz dela “Design Patterns” (Design Patterns.ppt).

Odabrani paterni

Paterni ili šabloni služe za rešavanje tipskih problema i proistekli su iz iskustva programera pri rešavanju raznih tipova problema. Postoji više vrsta šablona koji se koriste za kreiranje klase, strukture klase i ponašanja klase. U nastavku su dati neki od odabranih šablona.

Singleton – singleton šablon za kreiranje

Garantuje da će postojati samo jedna instanca neke klase postavljanjem *private* vidljivosti konstruktoru. Time se postiže globalni pristup instanci objekta koja se nalazi u datoj klasi. Koristi se za čuvanje kolekcija i drugih referenci kojima se često pristupa i kada je potrebno znati da ne postoji više takvih instanci.

Primer 1. Upotreba singleton klase u primeru studentske službe za rad sa entitetima Student.

```
public class UIManager {
    private static UIManager instance = new UIManager();
    private static Scanner sc = new Scanner(System.in);
    public static ArrayList<Object> lista = new ArrayList<Object>();
    private UIManager() { System.out.println("const"); }

    public static UIManager getInstance() {
        if (instance == null) {
            instance = new UIManager();
        }
        return instance;
    }

    public boolean readFromFile(String filename) {
        sc.nextLine();
        return true;
    }

    public boolean writeToFile(String filename) {
        return true;
    }
}

public class Test {
    @SuppressWarnings({ "unchecked", "rawtypes" })
    public static void main(String[] args) {
        ArrayList<Student> listaStudenata = new ArrayList<Student>();

        UIManager.lista.addAll(listaStudenata);
        UIManager.getInstance().writeToFile("student.txt");
        UIManager.getInstance().readFromFile("student.txt");
        listaStudenata.addAll((ArrayList) UIManager.lista);

        for (Student s : listaStudenata) {
            System.out.println(s);
        }
    }
}
```

Composite – kompozitni strukturni šablon

Predstavlja kompozitnu komponentu koja ima rekurzivnu hijerarhijsku internu strukturu. To znači da je interna hijerarhijska struktura tipa stabla – svaka komponenta je list ili grana (sadrži svoje podkomponente).

Primer 2. Upotreba kopozitnog strukturnog šablona u primeru kategorije artikla. Obezbediti ispis koji prati stepen interne hijerarhijske strukture.

```
public class Kategorija {
    private String naziv;
    private int stepen = 1;
    private List<Kategorija> podkategorije;

    Kategorija(String naziv) {
        this.naziv = naziv;
        this.podkategorije = new ArrayList<Kategorija>();
    }

    public void add(Kategorija pkat){
        pkat.stepen++;
        this.podkategorije.add(pkat);
    }

    public void remove(Kategorija pkat){
        this.podkategorije.remove(pkat);
    }

    public List<Kategorija> getPodkategorije() {
        return podkategorije;
    }

    @Override
    public String toString() {
        StringBuilder pkat = new StringBuilder();
        if (podkategorije.size()>0) {
            pkat.append(", sadrži podkategorije:\n");
            String tabs = new String(new char[stepen]).replace("\0", "\t");
            for (Kategorija k : podkategorije) {
                pkat.append(tabs+k+"\n");
            }
            pkat.replace(pkat.length()-1, pkat.length(), "");
        }
        return "Kategorija: " + naziv + pkat.toString();
    }
}
```

Odabrani paterni

```
public class Test {  
    public static void main(String[] args) {  
        //kategorije štampača  
        Kategorija k1 = new Kategorija("Štampači");  
  
        Kategorija k11 = new Kategorija("Matrični");  
        Kategorija k12 = new Kategorija("Laserski");  
        Kategorija k13 = new Kategorija("Ploteri");  
        Kategorija k14 = new Kategorija("Potrošni materijal");  
  
        Kategorija k121 = new Kategorija("Linijski");  
        Kategorija k122 = new Kategorija("Prenosni");  
        Kategorija k123 = new Kategorija("Termalni");  
  
        Kategorija k141 = new Kategorija("Toneri");  
        Kategorija k142 = new Kategorija("Ketridži");  
        Kategorija k143 = new Kategorija("Papir");  
  
        k1.add(k11); k1.add(k12); k1.add(k13); k1.add(k14);  
        k12.add(k121); k12.add(k122); k12.add(k123);  
        k14.add(k141); k14.add(k142); k14.add(k143);  
  
        System.out.println(k1);  
    }  
}
```

Odgovarajući ispis:

```
Kategorija: Štampači, sadrži podkategorije:  
    Kategorija: Matrični  
    Kategorija: Laserski, sadrži podkategorije:  
        Kategorija: Linijski  
        Kategorija: Prenosni  
        Kategorija: Termalni  
    Kategorija: Ploteri  
    Kategorija: Potrošni materijal, sadrži podkategorije:  
        Kategorija: Toneri  
        Kategorija: Ketridži  
        Kategorija: Papir
```

Observer – šablon za modelovanje ponašanja po tipu posmatrača

Koristi se za nadgledanje promena stanja određenog objekta. Klase posmatrači se registruju za nadgledanje određenog objekta čije promene stanja rezultuju obaveštavanjem pretplaćenih klasa posmatrača. Objekat koji se nadgleda nasledi klasu *Observable* i sadrži metode koje obaveštavaju posmatrače o izmeni stanja. Dok objekat koji posmatra implementira *Observer* interfejs i sadrži metodu koja se poziva kada dođe do promene stanja posmatranog objekta.

Primer 3. Primena Šablona osmatrača na primeru sistema E-obrazovanja i studenata. Studenti su priključeni određenom kursu za čije promene (npr. postavljanje rezultata ispita) dobijaju obaveštenja na svojim nalogu.

```
public class Student implements Observer
{
    private String name;
    private String surname;
    private String index;
    private ArrayList<Course> courses = new ArrayList<Course>();

    Student(String name, String surname, String index) {
        this.name = name;
        this.surname = surname;
        this.index = index;
    }

    @Override
    public void update(Observable o, Object arg) {
        String courseName = ((Course) o).getName();
        System.out.println("Student profile: "+index+" "+name+" "+surname);
        System.out.println("*New notification in " + courseName + " course:");
        System.out.println("**" + arg.toString());
    }

    public void addCourse(Course c){
        courses.add(c);
    }

    public void listCourses(){
        System.out.println("Student "+index+" "+name+" "+surname+
            " "+" is involved in: ");
        for (Course c : courses) {
            System.out.println(c.getName());
        }
    }
}
```

Odabrani paterni

```
public class Course extends Observable {
    private String name;
    private ArrayList<String> notifications = new ArrayList<String>();

    public Course(String name) {
        this.name = name;
    }

    public void postAnnouncement(String announcement) {
        notifications.add(announcement);
        // markira objekt da je izmenjen
        setChanged();
        // ako je objekat markiran kao izmenjen
        // obavesti sve posmatrace
        notifyObservers(announcement);
        // marker je postavljen na false
        System.out.println(hasChanged());
    }

    public String getName() {
        return name;
    }
}

public class Enastava {

    public static void main(String[] args) {
        Course c1 = new Course("OOP");
        Course c2 = new Course("NANS");

        Student s1 = new Student("Marko", "Markovic", "sw12345");
        Student s2 = new Student("Janko", "Jankovic", "sw34567");
        Student s3 = new Student("Branko", "Brankovic", "sw87654");

        c1.addObserver(s1); s1.addCourse(c1);
        c1.addObserver(s2); s2.addCourse(c1);
        c2.addObserver(s2); s2.addCourse(c2);
        c2.addObserver(s3); s3.addCourse(c2);

        c1.postAnnouncement("Rezultati ispita održanog...");
        c2.postAnnouncement("Teorijski deo ispita biće održan...");
        s2.listCourses();
    }
}
```

Odabrani paterni

- Primer 4. Primena Šablona osmatrača na primeru korisnika mikro-blogging platforme *Tweeter*-a. Korisnici pomenute platforme mogu uzajamno da se prate (posmatraju) i dobijaju obaveštenja o objavljenim *tweet*-ovima (promeni stanja).

```
public class TwitterUser extends Observable implements Observer {
    private ArrayList<String> tweets = new ArrayList<String>();
    private String name;

    public TwitterUser(String name) {
        this.name = name;
    }

    public void postTweet(String tweet) {
        tweets.add(tweet);
        setChanged();
        notifyObservers(tweet);
    }

    public void countFollowers() {
        System.out.println("User " + name + " has " +
            countObservers() + " followers!");
    }

    @Override
    public void update(Observable o, Object arg) {
        System.out.println("User " + name + " update:");
        String followedPerson = ((TwitterUser) o).name;
        System.out.println("'" + followedPerson + " post:\n" +
            "'" + arg.toString());
    }
}
```

Odabrani paterni

```
public class Twitter {  
    public static void main(String[] args) {  
        TwitterUser user = new TwitterUser("James Gosling");  
  
        TwitterUser s1 = new TwitterUser("OOP Student1");  
        TwitterUser s2 = new TwitterUser("OOP Student2");  
        TwitterUser s3 = new TwitterUser("OOP Student3");  
  
        user.addObserver(s1);  
        user.addObserver(s2);  
        user.addObserver(s3);  
  
        user.countFollowers();  
        user.postTweet("Java is C++ without the guns, knives, and clubs!");  
    }  
}
```

Zadaci

- Zadatak 1. Napisati program koji proširuje zadatak iz prethodnih vežbi (VEZBE_07) tako što se primenjuje bar jedan od odabranih paterni sa vežbi ili neki drugi po izboru.