Projektovanje i arhitektura računarskih sistema

Dizajn šabloni ponašanja

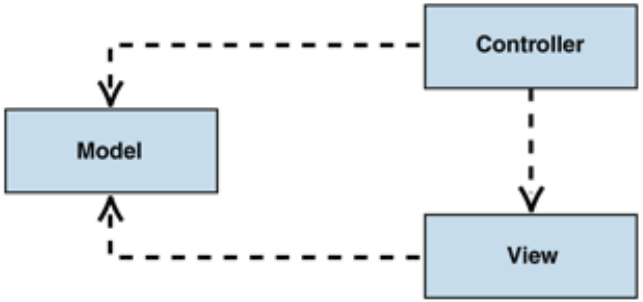
Odsek za računarsku tehniku i računarske komunikacije

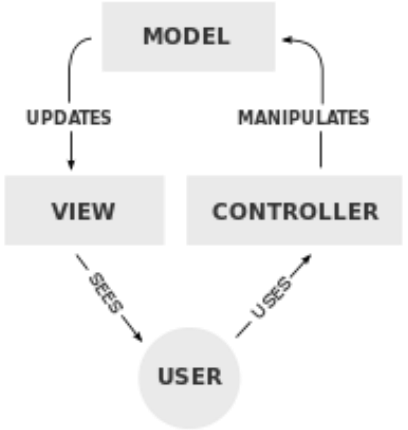
Tipični dizajn šabloni

- Stvaralački šabloni
  - Singleton
  - Factory
  - Factory Method
  - Abstract Factory
  - Graditelj (Builder)
  - Prototip
- Strukturni šabloni
  - Dekorator
  - Adapter
  - Fasada
  - Most
  - Kompozit
  - Proxy
- **Šabloni ponašanja**
  - **MVC/MVP**
  - **Sinhrono/asinhrono**
  - **Posmatrač**
  - Request/respond
  - Lanac odgovornosti
  - Komanda
  - **Interpreter**
  - **Iterator,** mediator
  - Memento
  - **Strategy**
  - State
  - Visitor

Model-View-Controller

- Koristi se u mnogim GUI aplikacijama
- Razdvajanje odgovornosti pojedinih komponenti
  - Model – čuva podatke sa kojima aplikacija barata.
  - View – služi za prikaz podataka iz modela i prosleđuje korisničke aktivnosti kontroleru
  - Controller – interpretira aktivnosti od View, utiče na Model, kontroliše tok podataka između Model i View, i po potrebi ažurira View.





MVC – Prednosti i mane

- Prednosti
  - Smanjuje kompleksnost pojedinih komponenti
  - Jasne zavisnosti i granice odgovornosti
  - Čistija i razumljivija implementacija
  - Zamenljivost komponenti
  - Ponovno iskorišćenje komponenti bez modifikacije koda
  - Lakše održavanje
  - Implementacija vođena pravilima MVC – dileme „gde pripada“ se rešavaju principom MVC
- Mane
  - Možda malo više koda, ali je preglednije
  - Zahteva disciplinu u dizajnu i razvoju

MVC – Školski primer modela

```
public class Student {
   private String rollNo;
   private String name;

   public String getRollNo() {
      return rollNo;
   }

   public void setRollNo(String rollNo) {
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

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

MVC – Školski primer prikaza

```java
public class StudentView {
   public void printStudentDetails(String studentName, String studentRollNo){
      System.out.println("Student: ");
      System.out.println("Name: " + studentName);
      System.out.println("Roll No: " + studentRollNo);
   }
}
```

MVC – Školski primer kontrolera

```
public class StudentController {
   private Student model;
   private StudentView view;

   public StudentController(Student model, StudentView view){
      this.model = model;
      this.view = view;
   }

   public void setStudentName(String name){
      model.setName(name);
   }

   public String getStudentName(){
      return model.getName();
   }

   public void setStudentRollNo(String rollNo){
      model.setRollNo(rollNo);
   }

   public String getStudentRollNo(){
      return model.getRollNo();
   }

   public void updateView(){
      view.printStudentDetails(model.getName(), model.getRollNo());
   }
}
```

MVC – Primer glavnog programa

```
public class MVCPatternDemo {
   public static void main(String[] args) {

      //fetch student record based on his roll no from the database
      Student model  = retriveStudentFromDatabase();

      //Create a view : to write student details on console
      StudentView view = new StudentView();

      StudentController controller = new StudentController(model, view);

      controller.updateView();

      //update model data
      controller.setStudentName("John");

      controller.updateView();
   }

   private static Student retriveStudentFromDatabase(){
      Student student = new Student();
      student.setName("Robert");
      student.setRollNo("10");
      return student;
   }
}
```

MVC – Diskusija

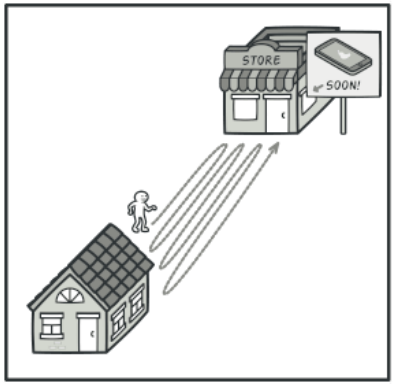
- Gde se čuva stanje programa?
  - Da li je na osnovu modela moguće rekonstruisati stanje programa?
  - Isključivo u modelu
- Kako to olakšava razvoj i testiranje?
  - Stanje softvera se lakše može snimati i ponoviti
  - Prikupljanje testnih ulaza
  - Lakše modelovanje stanja programa
- Modularnost olakšava ponovnu iskoristivost i zamenu pojedinih komponenti (npr. modela, ili prikaza ili kontrolne logike)

Zadatak

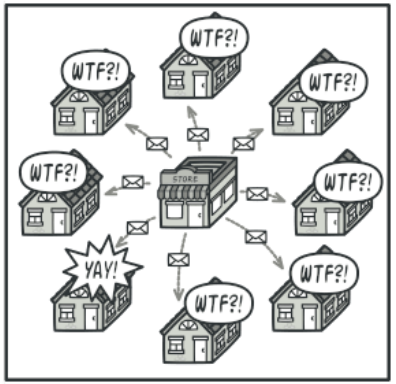
- Omogućiti efikasnu 1-na-N komunikaciju
  - 1 predajnik (izvor informacija), više prijemnika
  - Obaveštavanje o događaju ili stanju
  - Postoji sloboda u definisanju ko može da reaguje na poruku
- Primeri iz realnog života
  - Praćenje pošiljke
  - Internet of Things
  - Sistemski softver za distribuciju sistemskih događaja

Posmatrač (observer)

- Konstantno propitivanje je neefikasno
- Broadcast (objava) je neefikasno trošenje resursa
- Rešenje je observer pattern koji podrazumeva dve uloge:
  - Publisher (objavljivač)
  - Subscriber (pretplatnik)
- Objavljivač vodi računa o pretplatnicima i slanju poruka na efikasan način
- Koji su sve interfejsi potrebni? Ko ih definiše, a ko ih koristi?



Propitivanje

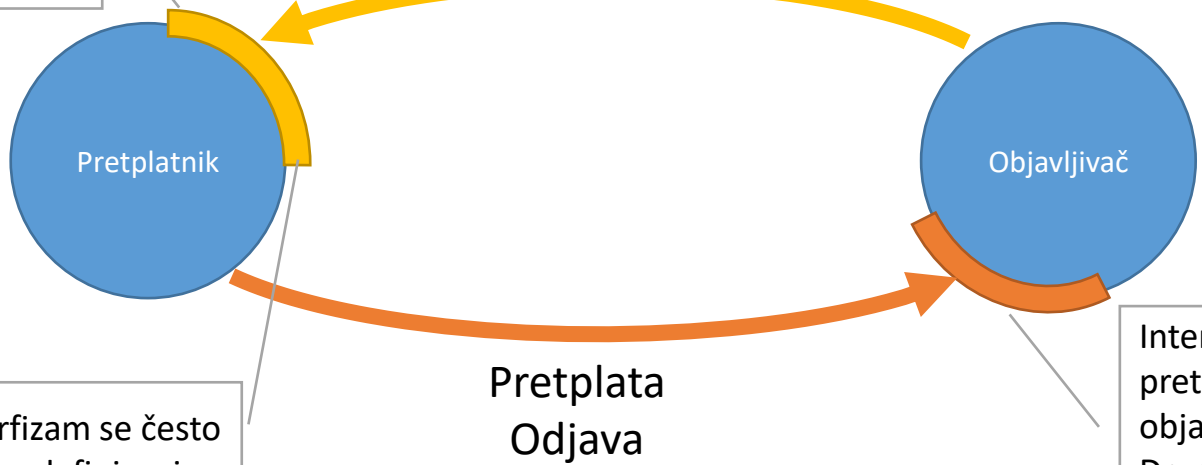


Objava

Observer šablon

Interfejs preko kojeg objavljivač vidi pretplatnika
Druga strana mora imati neku referencu na objekat

Obaveštenje je najčešće asinhrono



Polimorfizam se često koristi uz definisani interfejs

Interfejs preko kojeg pretplatnik vidi objavljivača
Druga strana mora imati neku referencu na objekat

Primer

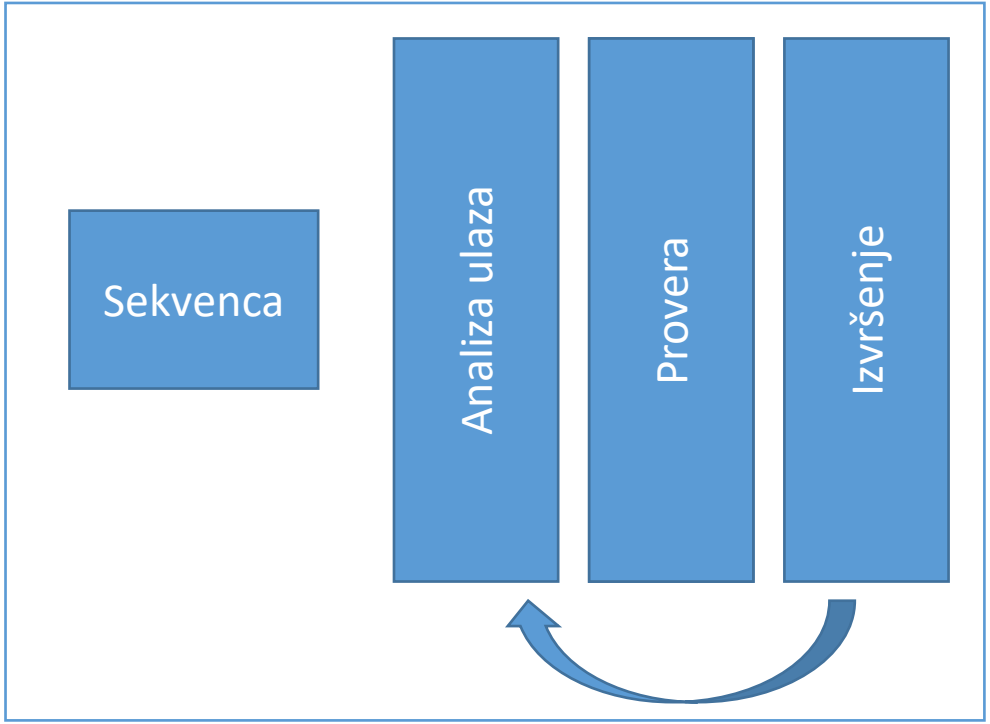
- Primer u Java kodu je u IDE

Zadatak

- Napravite okruženje koji redom izvršava neki niz komandi za upravljanje robotom
- Postoji 20 osnovnih komandi koji se proizvoljno mogu kombinovati, uz proveru parametara i proveru sigurnosti da robot ne može da se ošteti
- Sekvencu komandi je moguće pokrenuti, zaustaviti i zameniti u toku rada robotskog sistema

Interpreter dizajn šablon

- Ovo su osnovni elementi svakog interpretera



Zadatak

- Kako bi napravili sistem koji omogućuje da se tokom izvršavanja bira obrada slike?
- Nekada se koristi crno-beli filtar, nekada analiza ivica, nekada segmentacija slike
- Dodatno:
  - Kako bi omogućili da se obrade mogu dodavati bez izmene postojećeg koda?

Strategija

- Algoritam se implementira kao nezavisan modul
- Sve strategije (moduli, algoritmi) imaju iste ulaze i izlaze
- Algoritmi (moduli) su međusobno izmenljivi
- U Javi, Strategy dizajn šablon se može implementirati korišćenjem interfejsa koji definiše zajedničko ponašanje koje će biti implementirano od strane svake konkretne strategije.
- Npr konkretni algoritmi se implementiraju u klasama koje implementiraju ovaj interfejs.
- Kontekst koji koristi ove strategije obično sadrži referencu na trenutnu strategiju koja se primenjuje i poziva odgovarajuću metodu koja se nalazi u konkretnoj strategiji.

Primer

- Primer u Java kodu je u IDE

Zadatak

- Postoji struktura podataka sa velikim brojem elemenata
- Kako bi napravili mehanizam koji omogućuje da se na svaki element primeni neka obrada? Potencijalno obrada može da zavisi od nekih uslova.
- Ne moramo nužno poznavati način organizacije podataka, a obrada ni ne treba da zna organizaciju podataka.
- Dovoljno je poznavati način da se poseti svaki element.

Iterator

- Iterator sakriva način organizacije podataka
- Ideja je da se odvoji način pristupa od skladištenja
- Poseduje metod za pristup i prelaz na sledeći element – ili varijacije
- Poznata metoda iz STL, nećemo ulaziti u detalje
- Bitno je prepoznati da će se nad datom strukturom podataka obavljati operacija kojoj Iterator pogoduje.

```
public interface ItemIterator {
    Item getNext();
    bool hasMore();
}
```

Sinhrono-asinhrono

- Sinhrono znači sekvencijalan ili taktovan rad, pri čemu se čeka završetak prethodne operacije da bi se počelo sa narednom. Postoji zavisnost.
  - Intel i9
  - Jednonitni programi
  - Muzika
- Asinhrono znači da se može započeti naredna operacija i dok se prethodna ne završi, ako postoji striktna zavisnost između operacija. Asinhrono znači i da se reakcija na događaj desi prilikom prijema obaveštenja od događaju (prekid).
  - Postoje i asinhroni eksperimentalni CPU
  - Ljudski mozak je definitivno asinhron
  - Saobraćaj
  - „Best-effort“ obrada događaja

Sinhrono-asinhrono - Primeri

- Naručivanje hrane
  - Čekanje u redu vs naručivanje telefonom
- Programiranje na zahtev šefa
  - Ili čeka iza leđa ili radi nešto korisno dok programiramo
- Učenje, nastava? ☺

- Sinhrono programiranje je lakše, ali potencijalno suboptimalno
  - Zašto? Zbog neefikasno korišćenih resursa i vremena
  - Neka to pojednostavljuje složen problem (npr. procesori i komunikacija)
- Asinhrono programiranje je složenije, ali efikasnije
  - Zahteva dodatne mehanizme obaveštavanja, pažnju da se pokriju granični slučajevi
  - Malo drugačije razmišljanje, bez pretpostavki

Rezime

- Šabloni ponašanja su prvenstveno funkcionalne prirode i nemaju nužno jedinstvenu implementacionu formu
- Navedeni su samo najtipičniji primeri: MVP, Observer, Strategija, iterator i sinhroni/asinhroni dizajn koji se često sreću u Androidu.
- Naravno, skup šablona je mnogo veći, stoga savetujem da se dodatno konsultuje literatura
- Veoma je bitno primetiti situaciju kada vam dizajn šablon može pomoći, a onda za detalje konsultovati literaturu

Dodatne reference

- Izvori
  - https://conceptf1.blogspot.rs
  - https://www.tutorialspoint.com/design_pattern
  - *Link*
  - *https://refactoring.guru*
- Dodatni materijal
  - Stephen T. Albin , „*The Art of Software Architecture: Design Methods and Techniques*“
  - Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, „*Design Patterns: Elements of Reusable Object-Oriented Software*“
  - Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, „*Head First Design Patterns: A Brain-Friendly Guide*“
  - Alexander Shvets, „Dive into design patterns“, 2019