Пројектовање софтвера

Вежбе, први део градива

Задатак 1

- Посматра се систем за кафетерију. Потребно је имплементирати део система задужен за прављење топлих напитака. Сваки топли напитак се прави тако што се прво загреје вода, затим се вода обрађује са самим напитком, онда се напитак сипа у чашу и на крају се додају додаци. Кафа се прави тако што се загрејана вода сипа преко филтера са кафом, и на крају се додају шећер и млеко. Чај се прави тако што се у загрејану воду стави кесица са чајем и на крају додаје лимун.
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 1 - решење

- Део система за прављење топлих напитака има јасно дефинисане кораке, који варирају у зависности од тога које се пиће креира.
- Кораци:
  - Загревање воде
  - Обрађивање воде са самим напитком
  - Сипање напитка у чашу
  - Додавање додатака

Задатак 1 - решење

- Део система за прављење топлих напитака има јасно дефинисане кораке, који варирају у зависности од тога које се пиће креира.
- Кораци:
  - Загревање воде
  - ~~Обрађивање воде са самим напитком~~ → Сипање воде преко филтера са кафом
  - Сипање напитка у чашу
  - ~~Додавање додатака~~ → Додавање млека и шећера

У случају прављења кафе:

Задатак 1 - решење

- Део система за прављење топлих напитака има јасно дефинисане кораке, који варирају у зависности од тога које се пиће креира.
- Кораци:
  - Загревање воде
  - ~~Обрађивање воде са самим напитком~~ → Стављање кесице са чајем у загрејану воду
  - Сипање напитка у чашу
  - ~~Додавање додатака~~ → Додавање лимуна

У случају прављења чаја:

Задатак 1 - решење

- У овој ситуацији имамо пример система где су јасно дефинисани кораци који се извршавају у неком процесу, као и њихов редослед, али неки кораци варирају у зависности од тога које пиће се креира.
- Ово представља пример у коме се користи пројектни узорак

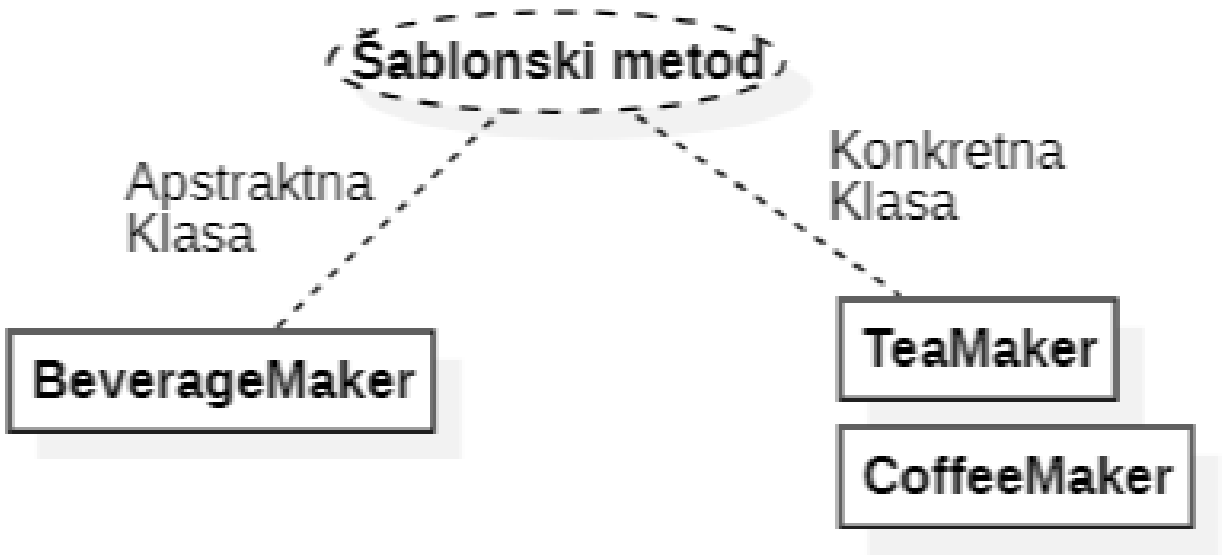
ШАБЛОНСКИ МЕТОД

Задатак 1 - решење

```
public abstract class BeverageMaker {
    public void makeBeverage() {
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }

    protected abstract void brew();
    protected abstract void addCondiments();

    protected void boilWater() {
        System.out.println("Boiling water");
    }

    protected void pourInCup() {
        System.out.println("Pouring into cup");
    }
}
```

```
public class CoffeeMaker extends BeverageMaker {

    @Override
    protected void brew() {
        System.out.println("Dripping coffee through filter");
    }

    @Override
    protected void addCondiments() {
        System.out.println("Adding sugar and milk");
    }
}
```

```
public class TeaMaker extends BeverageMaker {
    @Override
    protected void brew() {
        System.out.println("Steeping the tea");
    }

    @Override
    protected void addCondiments() {
        System.out.println("Adding lemon");
    }
}
```



Zadatak 1 - saradnja

Задатак 1 - дискусија

- Када би постојале класе кафа и чај са својим посебно дефинисаним понашањем, овај систем би могао да се имплементира коришћењем пројектног узорка Градитељ.

Задатак 2

- Посматра се систем за кафетерију. Потребно је имплементирати део система који је везан за понашање различих врста кафе. Кафи може да се одреди цена и опис. Кафа може бити *Single Origin* или *Blend* типа. Без обзира на тип, свакој кафи је могуће додати произвољан број прилога, где сваки утиче на различит начин на цену и опис кафе. Прилози могу бити млеко, шећер и чоколада.
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 2 – решење

- Први приступ – Правимо интерфејс Coffee, у којем се налазе методе за дохватање цене и описа. Као класе које имплементирају дати интерфејс имамо све могуће комбинације – BlendWithSugar, BlendWithMilk, BlendWithChocolate, SingleOriginWithSugar, SingleOriginWithMilk, SingleOriginWithChocolate, BlendWithSugarAndMilk…
- Проблем 1 – Велики број класа. Са већ постојећим врстама кафе и додацима већ постоји доста класа. Када би се систем проширивао са новом врстом кафе, потребно би било додати n нових класа за све комбинације са различитим прилозима (слично и за нови прилог).
- Проблем 2 – Како моделовати додавање више истих прилога у исту кафу?

Задатак 2 – решење

- Други приступ – Пројектни узорак Декоратер. Све врсте кафе као и све врсте прилога се изводе из интерфејса Coffee, при чему ће прилози бити декорације. Омогућено је додавање произвољног броја прилога (чак и истих), и систем постаје лако проширив, јер се за додавање нове врсте прилога или кафе додаје по једна нова класа.

Задатак 2 - решење

```
public abstract class Coffee {
   public abstract int price();
   public abstract String description();
}
```

Задатак 2 - решење

```
public abstract class Coffee {
   public abstract int price();
   public abstract String description();
}
```

```
public class BlendCoffee extends Coffee {

   private static final int BLENDCOFFEE_PRICE = 150;
   @Override
   public int price() {
      return BLEND_COFFEE_PRICE;
   }

   @Override
   public String description() {
      return "Blend Coffee";
   }
}
```

```
public class SingleOriginCoffee extends Coffee {

   private static final int SINGLE_ORIGIN_COFFEE_PRICE =
200;
   @Override
   public int price() {
      return SINGLE_ORIGIN_COFFEE_PRICE;
   }

   @Override
   public String description() {
      return "Single Origin Coffee";
   }
}
```

Задатак 2 - решење

```
public abstract class Coffee {
    public abstract int price();
    public abstract String description();
}
```

```
public class BlendCoffee extends Coffee {

    private static final int BLENDCOFFEE_PRICE = 150;
    @Override
    public int price() {
        return BLEND_COFFEE_PRICE;
    }

    @Override
    public String description() {
        return "Blend Coffee";
    }
}
```

```
public class SingleOriginCoffee extends Coffee {

    private static final int SINGLE_ORIGIN_COFFEE_PRICE =
200;
    @Override
    public int price() {
        return SINGLE_ORIGIN_COFFEE_PRICE;
    }

    @Override
    public String description() {
        return "Single Origin Coffee";
    }
}
```

```
public abstract class CoffeeAddition extends Coffee {
    private Coffee coffee;

    public CoffeeAddition(Coffee coffee) {
        this.coffee = coffee;
    }

    @Override
    public int price() {
        return coffee.price();
    }

    @Override
    public String description() {
        return coffee.description();
    }
}
```

Задатак 2 - решење

```
public abstract class Coffee {
   public abstract int price();
   public abstract String description();
}
```

```
public class BlendCoffee extends Coffee {

   private static final int BLENDCOFFEE_PRICE = 150;
   @Override
   public int price() {
      return BLEND_COFFEE_PRICE;
   }

   @Override
   public String description() {
      return "Blend Coffee";
   }
}
```

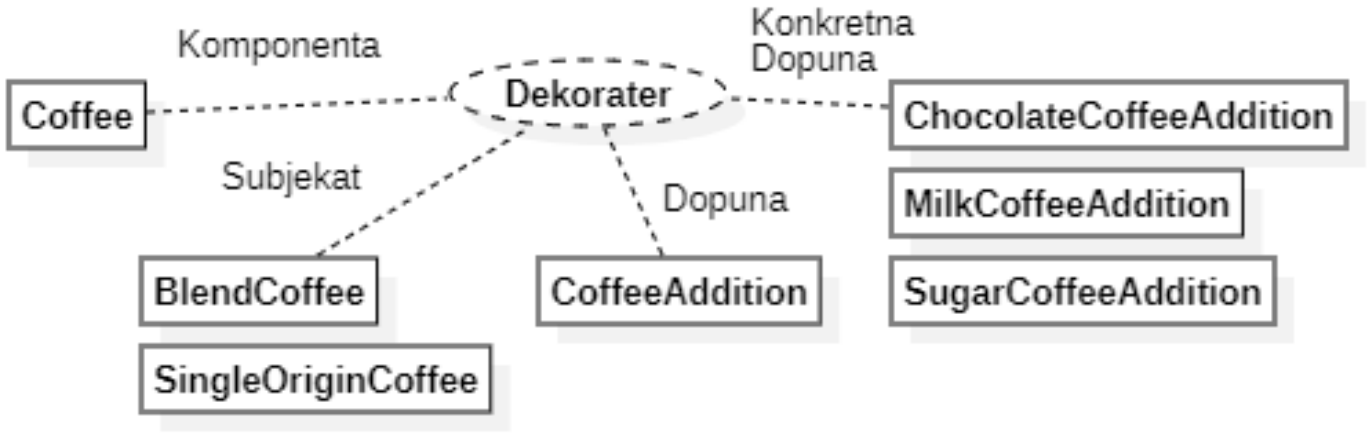
```
public class SingleOriginCoffee extends Coffee {

   private static final int SINGLE_ORIGIN_COFFEE_PRICE =
200;
   @Override
   public int price() {
      return SINGLE_ORIGIN_COFFEE_PRICE;
   }

   @Override
   public String description() {
      return "Single Origin Coffee";
   }
}
```

```
public abstract class CoffeeAddition extends Coffee {
   private Coffee coffee;

   public CoffeeAddition(Coffee coffee) {
      this.coffee = coffee;
   }

   @Override
   public int price() {
      return coffee.price();
   }

   @Override
   public String description() {
      return coffee.description();
   }
}
```

```
public class ChocolateCoffeeAddition extends CoffeeAddition {

   public ChocolateCoffeeAddition(Coffee coffee) {
      super(coffee);
   }

   private static final int CHOCOLATE_PRICE = 80;
   @Override
   public int price() {
      return super.price() + CHOCOLATE_PRICE;
   }

   @Override
   public String description() {
      return super.description() + " with Chocolate";
   }
}
```

```
public class MilkCoffeeAddition extends CoffeeAddition {

   private static final int MILK_PRICE = 100;

   public MilkCoffeeAddition(Coffee coffee) {
      super(coffee);
   }

   @Override
   public int price() {
      return super.price() + MILK_PRICE;
   }

   @Override
   public String description() {
      return super.description() + " with Milk";
   }
}
```

```
public class SugarCoffeeAddition extends CoffeeAddition {

   public SugarCoffeeAddition(Coffee coffee) {
      super(coffee);
   }

   @Override
   public int price() {
      return super.price();
   }

   @Override
   public String description() {
      return super.description() + " with Sugar";
   }
}
```



Задатак 2 - сарадња

Задатак 2 – решење

- Пример коришћења – Прављење Blend кафе са шећером, млеком и чоколадом.

```
Coffee coffee = new ChocolateCoffeeAddition(new MilkCoffeeAddition(new SugarCoffeeAddition(new BlendCoffee())));
System.out.println("Coffee price is: " + coffee.price());
System.out.println("You are currently drinking: " + coffee.description());
```

Coffee price is: 330
You are currently drinking: Blend Coffee with Sugar with Milk with Chocolate

Задатак 3

- Имплементирати пројектни узорак Уникат тако да омогућава приступ произвољном броју инстанци, где се приступ ради преко имена инстанце.

Задатак 3 - решење

- Традиционална имплементација Униката

```
public class Singleton {
   private Singleton() {}
   private static Singleton instance = null;
   public static Singleton getInstance() {
      if (instance == null) {
         instance = new Singleton();
      }
      return instance;
   }
}
```

Задатак 3 - решење

- ~~Традиционална имплементација Униката~~

Имплементација са више инстанци

```
public class Singleton {
   private Singleton() {}
   private static Singleton instance = null;
   public static Singleton getInstance() {
      if (instance == null) {
         instance = new Singleton();
      }
      return instance;
   }
}
```

```
public class Multiton {
   private static Map<String, Multiton> instances = new HashMap<>();
   private String name;

   private Multiton(String name) {
      this.name = name;
   }

   public static Multiton getInstance(String name) {
      if (!instances.containsKey(name)) {
         instances.put(name, new Multiton(name));
      }
      return instances.get(name);
   }
}
```

Задатак 4

- Развија се *Chat Bot*. Има предефинисан скуп питања који може да постави и где сва питања могу да се прикажу. Редослед приказивања питања може бити или насумичан или тако што се приказују оним редом којим су и предефинисана.
- Унапредити задату имплементацију и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 4

```
public class Chatbot {
    private List<String> questions = new ArrayList<>() {
        {
            add("What is 2 + 2?");
            add("What is 3 * 3?");
            add("What is 9 / 3?");
            add("What time is it?");
        }
    };

    public void displayQuestions(boolean showFirst) {
        Random random = new Random();
        while (!questions.isEmpty()) {
            int index;
            if (showFirst) {
                index = 0;
            } else {
                index = random.nextInt(0, questions.size());
            }
            String question = questions.get(index);
            questions.remove(index);
            System.out.println(question);
        }
    }
}
```

Задатак 4 – решење

```
public class Chatbot {
    private List<String> questions = new ArrayList<>() {
        {
            add("What is 2 + 2?");
            add("What is 3 * 3?");
            add("What is 9 / 3?");
            add("What time is it?");
        }
    };

    public void displayQuestions(boolean showFirst) {
        Random random = new Random();
        while (!questions.isEmpty()) {
            int index;
            if (showFirst) {
                index = 0;
            } else {
                index = random.nextInt(0, questions.size());
            }
            String question = questions.get(index);
            questions.remove(index);
            System.out.println(question);
        }
    }
}
```

Тренутно решење пати од непрегледног кода, који се састоји из непотребне if-else структуре. Затрпава логику функције за приказ питања, и у случају додавања још питања, додавале би се нове else гране и додатно компликовале код.

У оваквој ситуацији се код може унапредити коришћењем пројектног узорка Стратегија.

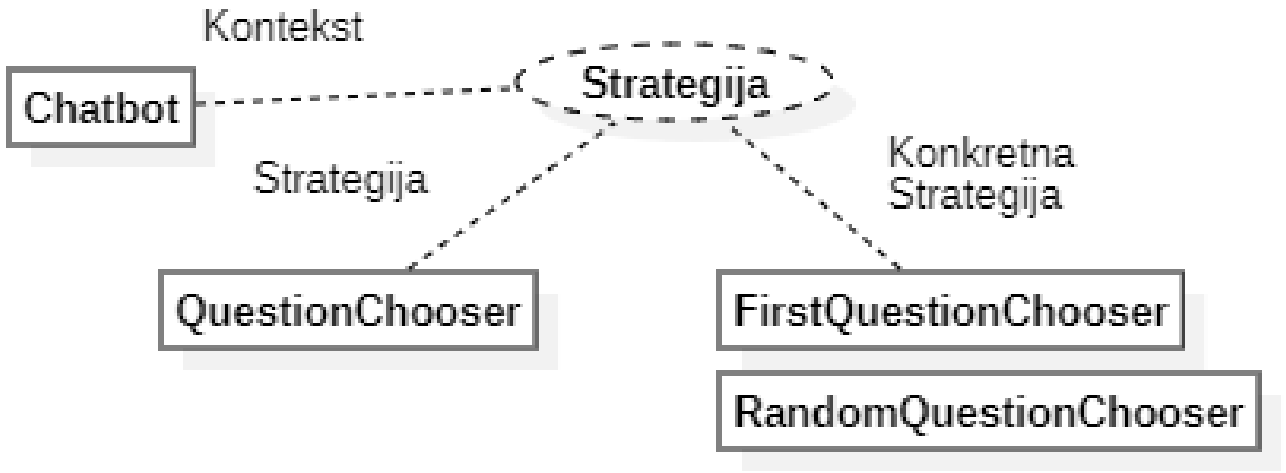
Задатак 4 - решење

```
public class Chatbot {
    private List<String> questions = new ArrayList<>() {
        {
            add("What is 2 + 2?");
            add("What is 3 * 3?");
            add("What is 9 / 3?");
            add("What time is it?");
        }
    };

    private QuestionChooser questionChooser;

    public Chatbot(QuestionChooser questionChooser) {
        this.questionChooser = questionChooser;
    }

    public void setQuestionChooser(QuestionChooser questionChooser) {
        this.questionChooser = questionChooser;
    }

    public void displayQuestions() {
        while (!questions.isEmpty()) {
            int index = questionChooser.getQuestion(questions.size());
            String question = questions.get(index);
            questions.remove(index);
            System.out.println(question);
        }
    }
}
```

```
public abstract class QuestionChooser {
    public abstract int getQuestion(int len);
}
```

```
public class FirstQuestionChooser extends QuestionChooser {

    @Override
    public int getQuestion(int len) {
        return 0;
    }
}
```

```
public class RandomQuestionChooser extends QuestionChooser {

    Random random = new Random();
    @Override
    public int getQuestion(int len) {
        int index = random.nextInt(0, len);
        return index;
    }
}
```



Задатак 4 - сарадња

Задатак 5

- Имплементирати узорак Апстрактна Фабрика коришћењем пројектног узорка Прототип. Приказати имплементацију система и улоге класа дијаграмом сарадње.

Задатак 5 – Производи

```
abstract class Proizvod1{
    abstract public Proizvod1 clone();

}
```

```
abstract class Proizvod2{
    abstract public Proizvod2 clone();

}
```

```
class Proizvod1A extends Proizvod1{
    int A;

    public Proizvod1A clone(){
        Proizvod1A copy = new Proizvod1A();
        copy.A = this.A;
        return copy;
    }

}
```

```
class Proizvod1B extends Proizvod1{
    int B;

    public Proizvod1B clone(){
        Proizvod1B copy = new Proizvod1B();
        copy.B = this.B;
        return copy;
    }

}
```

```
class Proizvod2A extends Proizvod2{
    int A;

    public Proizvod2A clone(){
        Proizvod2A copy = new Proizvod2A();
        copy.A = this.A;
        return copy;
    }

}
```

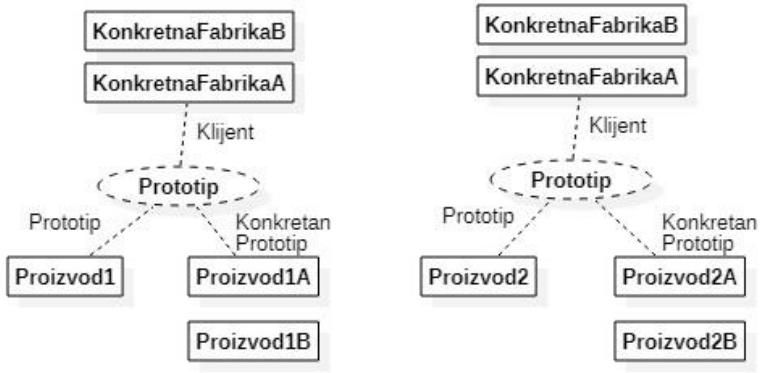
```
class Proizvod2B extends Proizvod2{
    int B;

    public Proizvod2B clone(){
        Proizvod2B copy = new Proizvod2B();
        copy.B = this.B;
        return copy;
    }

}
```

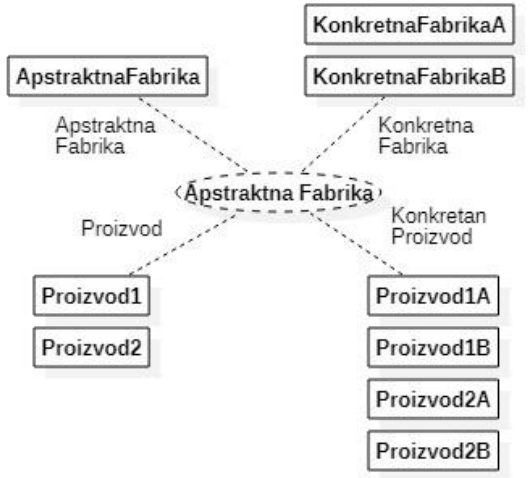
Задатак 5 – Фабрике

```
abstract class ApstraktnaFabrika{
    protected Proizvod1 proizvod1;
    protected Proizvod2 proizvod2;

    public ApstraktnaFabrika(Proizvod1 proizvod1, Proizvod2
proizvod2) {
        this.proizvod1 = proizvod1;
        this.proizvod2 = proizvod2;
    }

    abstract public Proizvod1 napraviProizvod1();

    abstract public Proizvod2 napraviProizvod2();

}
```

```
class KonkretnaFabrikaA extends ApstraktnaFabrika{

    public KonkretnaFabrikaA(Proizvod1A proizvod1, Proizvod2A proizvod2) {
        super(proizvod1, proizvod2);
    }

    public Proizvod1 napraviProizvod1(){
        return this.proizvod1.clone();
    }

    public Proizvod2 napraviProizvod2(){
        return this.proizvod2.clone();
    }

}
```

```
class KonkretnaFabrikaB extends ApstraktnaFabrika{

    public KonkretnaFabrikaB(Proizvod1B proizvod1, Proizvod2B proizvod2) {
        super(proizvod1, proizvod2);
    }

    public Proizvod1 napraviProizvod1(){
        return this.proizvod1.clone();
    }

    public Proizvod2 napraviProizvod2(){
        return this.proizvod2.clone();
    }

}
```





Задатак 5 - сарадња

Задатак 6

- Претпоставимо да је уређај са којим је неопходно успоставити комуникацију мотор са једносмерним драјвером и уграђеним мерачем броја обртаја и да клијентска апликација у себи позива методу за постављање брзине којој се брзина задаје као цео број, а да је гашење мотора предвиђено позивом исте те методе са вредношћу параметра једнаком 0. Произвођачи мотора су обезбедили драјвере за рад са сваком од компоненти и ХАЛ (Hardware Abstraction Layer) који апстракује позиве метода драјверских класа. Мотор је намење за серијску комуникацију преко серијског порта, па ХАЛ пружа методу за постављање брзине мотора која прима пакет бита који представљају брзину кодиран у формату пријема са серијског порта. У оквиру те методе обавља се претварање улаза, упис у једносмерни драјвер, ресетовање бројача обртаја и покретање мотора.
- Приказати имплементацију система уколико је познато да се користе пројектни узорци Фасада, Адаптер и Уникат. Приказати дијаграме колаборације за сваки узорак.

Задатак 6 - решење

- Задати интерфејс коме је потребно одрадити прилагођавање (Циљ):

```
public interface WithSpeed {

    public void setSpeed(int num);
}
```

Задатак 6 - решење

- Класа коју је потребно прилагодити (Адаптирана):

```
public interface WithSpeed {

    public void setSpeed(int num);
}
```

Уједно и Фасада и Уникат!

```
public class HAL {

    private static HAL instance = null;
    private DriverCounter driverCounter;
    private DriverDCDrive driverDCDrive;
    private DriverMotor driverMotor;

    private HAL(DriverCounter driverCounter, DriverDCDrive driverDCDrive, DriverMotor
driverMotor) {
        this.driverCounter = driverCounter;
        this.driverDCDrive = driverDCDrive;
        this.driverMotor = driverMotor;
    }

    public static HAL getInstance() {
        if (instance == null) {
            instance = new HAL(new DriverCounter(), new DriverDCDrive(), new
DriverMotor());
        }
        return instance;
    }

    public void setSpeed(int sp) {
        driverDCDrive.write(sp);
        driverCounter.reset();
        driverMotor.turnOn();
    }
}
```

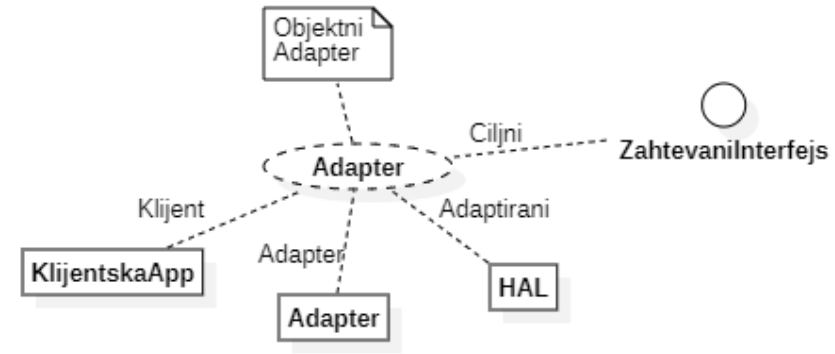
Задатак 6 - решење

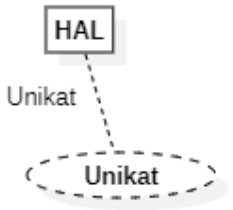
- Адаптер:

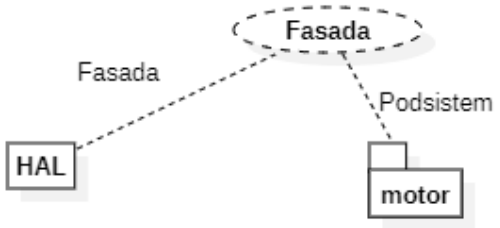
```
public interface WithSpeed {

    public void setSpeed(int num);
}
```

```
public class SpeedAdapter implements WithSpeed {
    private HAL hal = HAL.getInstance();
    @Override
    public void setSpeed(int num) {
        int sp = convert(num);
        hal.setSpeed(sp);
    }
}
```

```
public class HAL {

    private static HAL instance = null;
    private DriverCounter driverCounter;
    private DriverDCDrive driverDCDrive;
    private DriverMotor driverMotor;

    private HAL(DriverCounter driverCounter, DriverDCDrive driverDCDrive, DriverMotor
driverMotor) {
        this.driverCounter = driverCounter;
        this.driverDCDrive = driverDCDrive;
        this.driverMotor = driverMotor;
    }

    public static HAL getInstance() {
        if (instance == null) {
            instance = new HAL(new DriverCounter(), new DriverDCDrive(), new
DriverMotor());
        }
        return instance;
    }

    public void setSpeed(int sp) {
        driverDCDrive.write(sp);
        driverCounter.reset();
        driverMotor.turnOn();
    }
}
```







Задатак 6 - сарадња

Задатак 7

- Посматра се игрица где је потребно имплементирати део система задужен за рад са текстурама. Постоје текстуре за воду и копно. Текстуре се чувају у галерији текстура. Постоје земаљска и марсовска варијанта сваке текстуре, а у једном тренутку све текстуре у галерији треба да буду или земаљске или марсовске. Свака конкретна текстура се црта увек на исти начин, независно од контекста цртања.
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 7 - решење

- Потребно је направити два различита типа текстура, које могу варирати у зависности од планете. У једном тренутку се захтевају текстуре само са једне планете.
- У оваквим ситуацијама је применљив пројектни узорак Апстрактна фабрика

Задатак 7 - решење

- Потребно је направити два различита типа текстура, које могу варирати у зависности од планете. У једном тренутку се захтевају текстуре само са једне планете.
- У оваквим ситуацијама је применљив пројектни узорак Апстрактна фабрика

```
public abstract class TextureGallery {
   public abstract TextureWater createTextureWater();
   public abstract TextureLand createTextureLand();
}
```

Задатак 7 - решење

- Потребно је направити два различита типа текстура, које могу варирати у зависности од планете. У једном тренутку се захтевају текстуре само са једне планете.
- У оваквим ситуацијама је применљив пројектни узорак Апстрактна фабрика

```
public abstract class TextureGallery {
   public abstract TextureWater createTextureWater();
   public abstract TextureLand createTextureLand();
}
```

```
public class TextureGalleryEarth extends TextureGallery{


   @Override
   public TextureWater createTextureWater() {
      return new TextureWaterEarth();
   }

   @Override
   public TextureLand createTextureLand() {
      return new TextureLandEarth();
   }
}
```

```
public class TextureGalleryMars extends TextureGallery {

   @Override
   public TextureWater createTextureWater() {
      return new TextureWaterMars();
   }

   @Override
   public TextureLand createTextureLand() {
      return new TextureLandMars();
   }
}
```

Задатак 7 - решење

- Конкретне текстуре исте врсте су реупотребљиви објекти, с обзиром на то да се увек цртају на исти начин, независно од контекста.
- Из тог разлога, нема потребе да се увек креира нова текстура по захтевању исте, већ је могуће вратити постојећу инстанцу текстуре одговарајућег типа.
- За ове потребе је могуће искористити пројектни узорак Мува

Задатак 7 - решење

- Конкретне текстуре исте врсте су реупотребљиви објекти, с обзиром на то да се увек цртају на исти начин, независно од контекста.
- Из тог разлога, нема потребе да се увек креира нова текстура по захтевању исте, већ је могуће вратити постојећу инстанцу текстуре одговарајућег типа.
- За ове потребе је могуће искористити пројектни узорак Мува

```
public abstract class TextureGallery {
   public enum FieldType {LAND, WATER};

   private Map<FieldType, Texture> textures = new
HashMap<>();
   protected abstract TextureWater createTextureWater();
   protected abstract TextureLand createTextureLand();

   public Texture getTexture(FieldType type) {
      if (textures.get(type) == null) {
         if (type == FieldType.LAND) {
            textures.put(type, createTextureLand());
         } else if (type == FieldType.WATER) {
            textures.put(type, createTextureWater());
         }
      }
      return textures.get(type);
   }
}
```

Задатак 7 - решење

- Конкретне текстуре исте врсте су реупотребљиви објекти, с обзиром на то да се увек цртају на исти начин, независно од контекста.
- Из тог разлога, нема потребе да се увек креира нова текстура по захтевању исте, већ је могуће вратити постојећу инстанцу текстуре одговарајућег типа.
- За ове потребе је могуће искористити пројектни узорак Мува

Имплементација остаје иста у подкласама

```
public abstract class TextureGallery {
   public enum FieldType {LAND, WATER};

   private Map<FieldType, Texture> textures = new
HashMap<>();
   protected abstract TextureWater createTextureWater();
   protected abstract TextureLand createTextureLand();

   public Texture getTexture(FieldType type) {
      if (textures.get(type) == null) {
         if (type == FieldType.LAND) {
            textures.put(type, createTextureLand());
         } else if (type == FieldType.WATER) {
            textures.put(type, createTextureWater());
         }
      }
      return textures.get(type);
   }
}
```

Задатак 7 - решење

- Конкретне текстуре исте врсте су реупотребљиви објекти, с обзиром на то да се увек цртају на исти начин, независно од контекста.
- Из тог разлога, нема потребе да се увек креира нова текстура по захтевању исте, већ је могуће вратити постојећу инстанцу текстуре одговарајућег типа.
- За ове потребе је могуће искористити пројектни узорак Мува

Метода за дохватање конкретног објекта, на основу прослеђеног типа текстуре

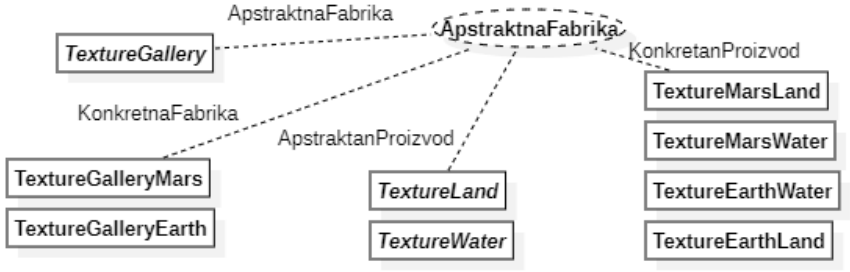
```
public abstract class TextureGallery {
   public enum FieldType {LAND, WATER};

   private Map<FieldType, Texture> textures = new
HashMap<>();
   protected abstract TextureWater createTextureWater();
   protected abstract TextureLand createTextureLand();

   public Texture getTexture(FieldType type) {
      if (textures.get(type) == null) {
         if (type == FieldType.LAND) {
            textures.put(type, createTextureLand());
         } else if (type == FieldType.WATER) {
            textures.put(type, createTextureWater());
         }
      }
      return textures.get(type);
   }
}
```

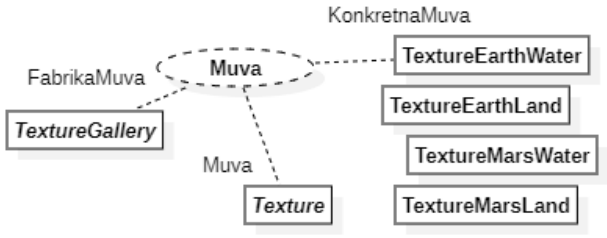
Задатак 7 - решење

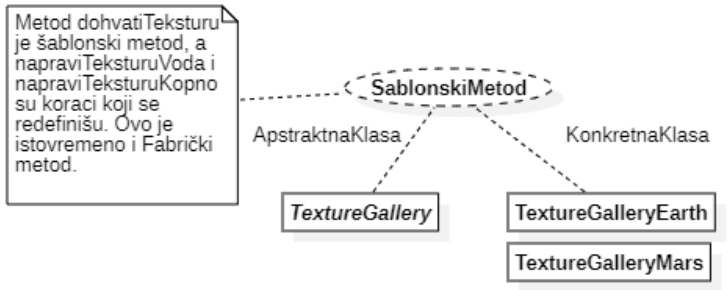
- Конкретне текстуре исте врсте су реупотребљиви објекти, с обзиром на то да се увек цртају на исти начин, независно од контекста.
- Из тог разлога, нема потребе да се увек креира нова текстура по захтевању исте, већ је могуће вратити постојећу инстанцу текстуре одговарајућег типа.
- За ове потребе је могуће искористити пројектни узорак Мува

Овако имплементирана функција истовремено представља и пројектни узорак Шаблонски метод

```
public abstract class TextureGallery {
   public enum FieldType {LAND, WATER};

   private Map<FieldType, Texture> textures = new
HashMap<>();
   protected abstract TextureWater createTextureWater();
   protected abstract TextureLand createTextureLand();

   public Texture getTexture(FieldType type) {
      if (textures.get(type) == null) {
         if (type == FieldType.LAND) {
            textures.put(type, createTextureLand());
         } else if (type == FieldType.WATER) {
            textures.put(type, createTextureWater());
         }
      }
      return textures.get(type);
   }
}
```







Задатак 7 - сарадња

Задатак 7 – дискусија

- Галерије могу да буду стратегије креирања текстура. Ако би се увела класа која представља ниво, где сваки ниво може да има текстуре једног типа, сваком нивоу би могла да се при креирању постави стратегија.
- Дохватање текстуре на основу типа је тренутно урађено као Шаблонски метод. У зависности од контекста коришћења, потенцијално могуће и друге имплементације
- Ако би се даље разрађивао овај систем и увела класа поље, које садржи текстуру, нема потребе исцртавати, а ни учитавати текстуру поља, осим ако није у видном пољу играча – пројектни узорак Заступник

Задатак 8

- Посматра се имплементација за део оперативног система. Дата је једна могућа имплементација меморијског фрагмента који имплементира интерфејс у којем се налазе методе за читање са адресе, упис на адресу и извршавање инструкције са дате адресе, као и метода за копирање самог фрагмента. Процеси користе дати интерфејс за меморијске фрагменте у којима се налазе код, стек и подаци. За процесе је такође имплементирана метода fork, којом се прави процес дете.

Задатак 8

```
abstract class MemFragment{
   public abstract Byte read(int address);

   public abstract void write(int address, Byte data);

   public abstract void execute(int address);

   public abstract MemFragment clone();

}
```

```
class MemFragmentImpl extends MemFragment{
   private Byte[] data;

   public MemFragmentImpl(int size) {
      this.data = new Byte[size];
   }

   public Byte read(int address){
      return this.data[address];
   }

   public void write(int address, Byte data){
      this.data[address] = data;
   }

   public void execute(int address){
      //Poziv izvrsavanja instrukcije na datoj adresi
   }

   public MemFragmentImpl clone(){
      MemFragmentImpl copy = new MemFragmentImpl(this.data.length);

      for(int i=0; i < this.data.length; i++){
         copy.data[i] = this.data[i];
      }

      return copy;
   }
}
```

Задатак 8

```
abstract class MemFragment{
    public abstract Byte read(int address);

    public abstract void write(int address, Byte data);

    public abstract void execute(int address);

    public abstract MemFragment clone();

}
```

```
class Process{
    private MemFragment code;
    private MemFragment stack;
    private MemFragment data;

    //...
    public int fork(){
        Process child = new Process();
        //...
        child.code = code.clone();
        child.stack = stack.clone();
        child.data = data.clone();
        //...

    }
    //...
}
```

```
class MemFragmentImpl extends MemFragment{
    private Byte[] data;

    public MemFragmentImpl(int size) {
        this.data = new Byte[size];
    }

    public Byte read(int address){
        return this.data[address];
    }

    public void write(int address, Byte data){
        this.data[address] = data;
    }

    public void execute(int address){
        //Poziv izvrsavanja instrukcije na datoj adresi
    }

    public MemFragmentImpl clone(){
        MemFragmentImpl copy = new MemFragmentImpl(this.data.length);

        for(int i=0; i < this.data.length; i++){
            copy.data[i] = this.data[i];
        }

        return copy;
    }
}
```

Задатак 8

```
abstract class MemFragment{
    public abstract Byte read(int address);

    public abstract void write(int address, Byte data);

    public abstract void execute(int address);

    public abstract MemFragment clone();

}
```

```
class Process{
    private MemFragment code;
    private MemFragment stack;
    private MemFragment data;

    //...
    public int fork(){
        Process child = new Process();
        //...
        child.code = code.clone();
        child.stack = stack.clone();
        child.data = data.clone();
        //...

    }
    //...
}
```

```
public class OS {

    Process createProcess() {

        Process p = new Process();

        p.code = new MemFragmentImpl(codeSize);
        p.data = new MemFragmentImpl(dataSize);
        p.stack = new MemFragmentImpl(stackSize);

        return p;
    }

}
```

```
class MemFragmentImpl extends MemFragment{
    private Byte[] data;

    public MemFragmentImpl(int size) {
        this.data = new Byte[size];
    }

    public Byte read(int address){
        return this.data[address];
    }

    public void write(int address, Byte data){
        this.data[address] = data;
    }

    public void execute(int address){
        //Poziv izvrsavanja instrukcije na datoj adresi
    }

    public MemFragmentImpl clone(){
        MemFragmentImpl copy = new MemFragmentImpl(this.data.length);

        for(int i=0; i < this.data.length; i++){
            copy.data[i] = this.data[i];
        }

        return copy;
    }
}
```

Задатак 8

- а) Који пројектни узорак је коришћен у датом коду?
- б) Проширити систем тако да се користи copy on write механизам при коришћењу методе fork код процеса, и додати подршку за rwx бите који се користе за права приступа код меморијских фрагмената, без мењања постојећих имплементација за процесе и меморијске фрагменте.

```
class MemFragmentProxy extends MemFragment{

   private MemFragmentImpl impl;
   private int rwx;
   private MemFragmentProxy connection; //Dete sa kojim smo povezani
posle copy on write mehanizma

   public Byte read(int address){
      if((rwx & 0b100) != 0){
         return this.impl.read(address);
      }
      else{
         throw new Exception("Reading not permitted!");
      }
   }

   public void write(int address, Byte data){
      if((rwx & 0b010) != 0){
         if(this.connection){ //Ako se memorijski fragment deli, pri upisu se
pravi kopija i vise se ne dele
            unbind();
         }
         this.impl.write(address, data);
      }
      else{
         throw new Exception("Writing not permitted!");
      }
   }

   public void execute(int address){
      if((rwx & 0b001) != 0){
         this.impl.execute(address);
      }
      else{
         throw new Exception("Execution not permitted!");
      }
   }
```

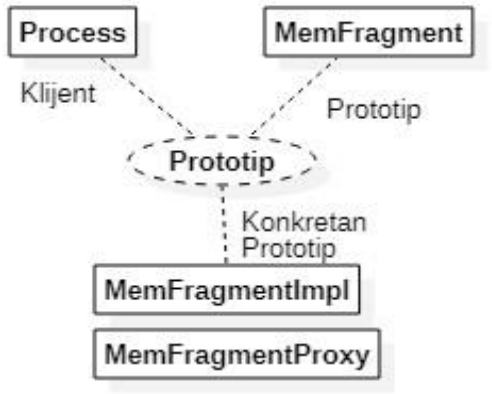
```
class MemFragmentProxy extends MemFragment{

    private MemFragmentImpl impl;
    private int rwx;
    private MemFragmentProxy connection; //Dete sa kojim smo povezani
posle copy on write mehanizma

    public Byte read(int address){
        if((rwx & 0b100) != 0){
            return this.impl.read(address);
        }
        else{
            throw new Exception("Reading not permitted!");
        }
    }

    public void write(int address, Byte data){
        if((rwx & 0b010) != 0){
            if(this.connection){ //Ako se memorijski fragment deli, pri upisu se
pravi kopija i vise se ne dele
                unbind();
            }
            this.impl.write(address, data);
        }
        else{
            throw new Exception("Writing not permitted!");
        }
    }

    public void execute(int address){
        if((rwx & 0b001) != 0){
            this.impl.execute(address);
        }
        else{
            throw new Exception("Execution not permitted!");
        }
    }
```

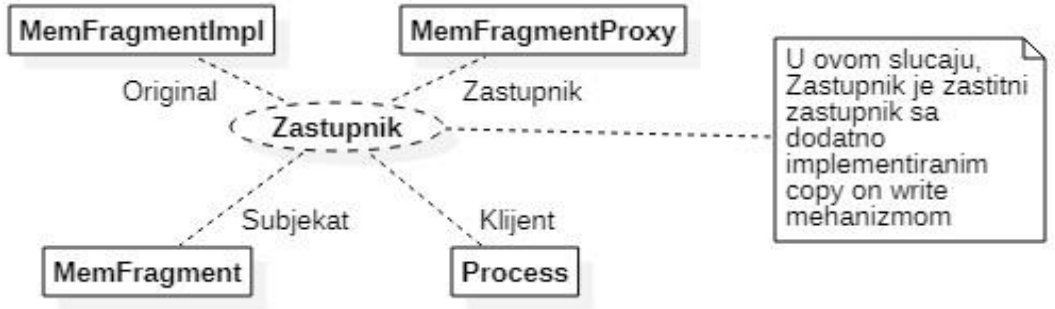
```
public MemFragmentProxy clone(){
        MemFragmentProxy child = new
MemFragmentProxy(this.rwx);

        child.impl = this.impl;
        this.connection = child;
        child.connection = this;

        return child;
    }

    private MemFragmentProxy(int rwx){
        this.rwx = rwx;
        this.connection = null;
    }

    public MemFragmentProxy(int size, int rwx){
        this(rwx);

        this.impl = new MemFragmentImpl(size);
    }

    private void unbind(){
        this.connection.connection = null;
        this.connection = null;
        this.impl = this.impl.clone();
    }

}
```

```
class MemFragmentProxy extends MemFragment{

   private MemFragmentImpl impl;
   private int rwx;
   private MemFragmentProxy connection; //Dete sa kojim smo povezani
posle copy on write mehanizma

   public Byte read(int address){
      if((rwx & 0b100) != 0){
         return this.impl.read(address);
      }
      else{
         throw new Exception("Reading not permitted!");
      }
   }

   public void write(int address, Byte data){
      if((rwx & 0b010) != 0){
         if(this.connection){ //Ako se memorijski fragment deli, pri upisu se
pravi kopija i vise se ne dele
            unbind();
         }
         this.impl.write(address, data);
      }
      else{
         throw new Exception("Writing not permitted!");
      }
   }

   public void execute(int address){
      if((rwx & 0b001) != 0){
         this.impl.execute(address);
      }
      else{
         throw new Exception("Execution not permitted!");
      }
   }
```

```
public MemFragmentProxy clone(){
      MemFragmentProxy child = new
MemFragmentProxy(this.rwx);

      child.impl = this.impl;
      this.connection = child;
      child.connection = this;

      return child;
   }

   private MemFragmentProxy(int rwx){
      this.rwx = rwx;
      this.connection = null;
   }

   public MemFragmentProxy(int size, int rwx){
      this(rwx);

      this.impl = new MemFragmentImpl(size);
   }

   private void unbind(){
      this.connection.connection = null;
      this.connection = null;
      this.impl = this.impl.clone();
   }

}
```

```
public class OS {
  Process createProcess() {

     Process p = new Process();

     p.code = new MemFragmentProxy(codeSize, 0b001);
     p.data = new MemFragmentProxy(dataSize, 0b110);
     p.stack = new MemFragmentProxy(stackSize, 0b110);

     return p;
  }
}
```





Задатак 8- сарадња

Задатак 8 - дискусија

- Код самог ОС-а за прављење процеса је морао да буде промењен тако да ОС сада прави заступнике меморијских фрагмената уместо објекте саме имплементације. Да ли је ово могло да се избегне?
- Дата имплементација заступника дозвољава да процес „родитељ“ дели меморијски фрагмент са само једним процесом „дететом“. Како имплементирати исто понашање са произвољним бројем „деце“?

Задатак 9

- Дат је интерфејс Subject који је потребно имплементирати методама које врше познату обраду. Клијентски код ће користити имплементирани интерфејс тако да ће се неки објекти над којима се врше дате обраде налазити на рачунару на којем се клијентски код извршава, а неки објекти ће се налазити на серверу са познатом IP адресом. Имплементирати овај део система тако да клијентском коду буде потпуно транспарентно да ли се објекат налази на локалном рачунару или на серверу. За слање захтева преко мреже је одлучено да ће се користити ApacheHttp библиотека, али зарад једноставности решења за комуникацију са објектима на серверу одлучено је да се користи интерфејс HttpInterface дат у наставку. Повратна вредност get методе је статусни код, а post методе је String који представља тело одговора.

Задатак 9

```
abstract class Subject{
   void f1();

   void f2();

   int f3();

   int id();

}
```

```
public interface HttpInterface{
   public int get(String address, HashMap<String, String> parameters);

   public String post(String address, HashMap<String, String> parameters);

}
```

- Узимамо претпоставку да се за void методе користи get захтев, а за методе које имају неку повратну вредност користи post захтев.

Задатак 9

```
public class ApacheAdapter implements HttpInterface{

   public int get(String address, HashMap<String, String> parameters){
      try (CloseableHttpClient httpClient = HttpClients.createDefault()) {

         URI uri = new URIBuilder(address)

         for (Map.Entry<String, String> entry : parameters.entrySet()) {
            String key = entry.getKey();
            String value = entry.getValue();
            uri.addParameter(key, value);
         }
         uri.build();
         HttpGet httpGet = new HttpGet(uri);
         try (CloseableHttpResponse response = httpClient.execute(httpGet)) {

            int statusCode = response.getCode();
            System.out.println("Status Code: " + statusCode);

            String responseBody = EntityUtils.toString(response.getEntity());
            System.out.println("Response Body: " + responseBody);

            return statusCode;
         }

      } catch (Exception e) {
         e.printStackTrace();
         return -1;
      }
   }
```

```
   public String post(String address, HashMap<String, String> parameters){
      try (CloseableHttpClient httpClient = HttpClients.createDefault()) {

         HttpPost httpPost = new HttpPost(address);

         String json = "{";

         for (Map.Entry<String, String> entry : parameters.entrySet()) {
            String key = entry.getKey();
            String value = entry.getValue();
            json.concat("\""); json.concat(key); json.concat("\":\""); json.concat(value);
json.concat("\",");
         }
         json.concat("}");
         StringEntity entity = new StringEntity(json, ContentType.APPLICATION_JSON);
         httpPost.setEntity(entity);
         httpPost.setHeader("Accept", "application/json");
         httpPost.setHeader("Content-type", "application/json");

         try (CloseableHttpResponse response = httpClient.execute(httpPost)) {
            int statusCode = response.getCode();
            System.out.println("Status Code: " + statusCode);

            String responseBody = EntityUtils.toString(response.getEntity());
            System.out.println("Response Body: " + responseBody);

            return responseBody;
         }

      } catch (Exception e) {
         return "";
         e.printStackTrace();
      }
   }
}
```

Задатак 9

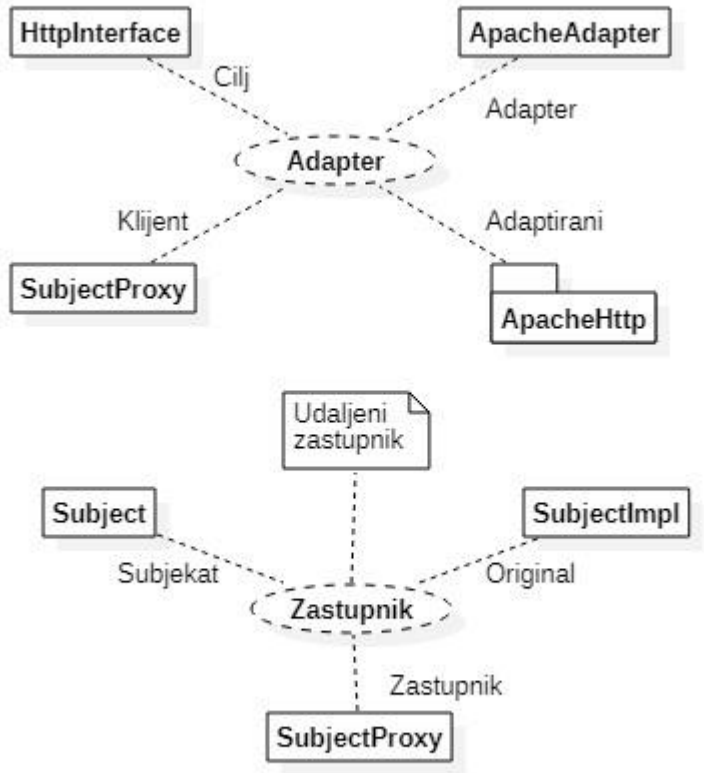
```
class SubjectProxy extends Subject{
    private String address;
    private int id;
    private HttpInterface http;

    public SubjectProxy(int id, String address, HttpInterface http){
        this.address = address;
        this.id = id;
        this.http = http;
    }

    void f1(){
        HashMap<String, String> parameters = new HashMap<>();
        parameters.put("id", String.valueOf(id));
        parameters.put("method", "f1");

        int code = this.http.get(address, parameters);

        if(code = ...){
            //...
        }
        else if(code = ...){
            //...
        }
    }
```

```
    void f2(){
        HashMap<String, String> parameters = new HashMap<>();
        parameters.put("id", String.valueOf(id));
        parameters.put("method", "f2");

        int code = this.http.get(address, parameters);

        if(code = ...){
            //...
        }
        else if(code = ...){
            //...
        }
    }

    int f3(){
        HashMap<String, String> parameters = new HashMap<>();
        parameters.put("id", String.valueOf(id));
        parameters.put("method", "f3");

        String responseBody = this.http.post(address, parameters);

        ObjectMapper objectMapper = new ObjectMapper();
        Map<String, Object> jsonResponse = objectMapper.readValue(responseBody,
Map.class);

        int returnValue = (int) jsonResponse.get("returnValue");
        return returnValue;
    }

    int id(){
        return this.id;
    }
}
```

Задатак 9

```
class SubjectImpl extends Subject{
   private int id;

   public SubjectImpl(int id){
      this.id = id;
   }

   void f1(){
      //Obrada 1...
   }

   void f2(){
      //Obrada 2...
   }

   int f3(){
      //Obrada 3...
      return ...;
   }

   int id(){
      return this.id;
   }
}
```

```
class SubjectRegistry{

   private HashMap<Integer, Subject> objects;
   private HashMap<Integer, Subject> objectsOnServer;
   private String address;

   public SubjectRegistry(){
      this.address = ...;
      this.objects = new HashMap<>();
      this.objectsOnServer = new HashMap<>();

      HttpInterface http = new ApacheAdapter();

      for(int i=0; i < ...; i++){
         if(i < ...){
            objects.put(i, new SubjectImpl(i));
         }
         else{
            objectsOnServer.put(i, new SubjectProxy(i, this.address, http));
         }
      }
   }

   public Subject getSubject(int id){
      if(objects.contains(id)){
         return objects.get(id);
      }
      else if(objectsOnServer.contains(id)){
         return objectsOnServer.get(id);
      }
      else{
         throw new Exception("Object with id " + String.valueOf(id) + " doesn't exist!");
      }
   }

}
```



Задатак 9 - сарадња

Задатак 10

- ... Чаробњак има координате на којима се тренутно налази и број живота. Чаробњаци уговарају померање за једну позицију или телепортовање за произвољан број позиција у једном од смерова, додавање или одузимање живота. Магичне печурке се стварају са случајно створеним магичним ефектом, који може да буде прост магични ефекат или чаролија. Чаробњаци могу да интерагују са магичном печурком, што резултује активирањем магичног ефекта који печурка садржи. Активација чаролије подразумева активирање магичних ефеката које она укључује. Прости магични ефекти се стварају са адресом чаробњака над којим се активира ефекат. Они могу да буду телепортовање на насумично поље, додавање или одузимање живота чаробњаку. Магичне печурке при интеракцији са чаробњаком активирају свој магични ефекат. ...
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 10

```
class Carobnjak{
    private int zivoti;
    private Coordinate xy;

    public void pomeriSe(Smer s){
        this.xy.move(s);
    }

    public void teleportujSe(Smer s, int pomeraj){
        for(int i=0; i < pomeraj; i++) this.xy.move(s);
    }

    public void dodajZivot(){
        this.zivoti++;
    }

    public void oduzmiZivot(){
        this.zivoti--;
    }

}
```

Задатак 10

```
class Carobnjak{
   private int zivoti;
   private Coordinate xy;

   public void pomeriSe(Smer s){
      this.xy.move(s);
   }

   public void teleportujSe(Smer s, int pomeraj){
      for(int i=0; i < pomeraj; i++) this.xy.move(s);
   }

   public void dodajZivot(){
      this.zivoti++;
   }

   public void oduzmiZivot(){
      this.zivoti--;
   }

}
```

```
class MagicnaPecurka{
   private MagicniEfekat efekat;
   private Coordinate xy;

   public MagicnaPecurka(Coordinate xy){
      this.xy = xy;
      this.efekat = ... //Random pravljenje efekta
   }

   public interaguj(Carobnjak c){
      this.efekat.aktiviraj();
   }

}
```

Задатак 10

```
class Carobnjak{
   private int zivoti;
   private Coordinate xy;

   public void pomeriSe(Smer s){
      this.xy.move(s);
   }

   public void teleportujSe(Smer s, int pomeraj){
      for(int i=0; i < pomeraj; i++) this.xy.move(s);
   }

   public void dodajZivot(){
      this.zivoti++;
   }

   public void oduzmiZivot(){
      this.zivoti--;
   }

}
```

```
class MagicnaPecurka{
   private MagicniEfekat efekat;
   private Coordinate xy;

   public MagicnaPecurka(Coordinate xy){
      this.xy = xy;
      this.efekat = ... //Random pravljenje efekta
   }

   public interaguj(Carobnjak c){
      this.efekat.aktiviraj();
   }

}
```

```
abstract class MagicniEfekat{
   public void aktiviraj();

   public void dodajEfekat(MagicniEfekat m){

   }

   public void izbaciEfekat(MagicniEfekat m){

   }

}
```

Задатак 10

```
class DodavanjeZivota extends MagicniEfekat{
   private Carobnjak c;

   public DodavanjeZivota(Carobnjak c){
      this.c = c;
   }

   public void aktiviraj(){
      this.c.dodajZivot();
   }

}
```

```
class OduzimanjeZivota extends MagicniEfekat{
   private Carobnjak c;

   public OduzimanjeZivota(Carobnjak c){
      this.c = c;
   }

   public void aktiviraj(){
      this.c.oduzmiZivot();
   }

}
```

```
class Teleportovanje extends MagicniEfekat{
   private Carobnjak c;
   private Smer s;
   private int pomeraj;

   public Teleportovanje(Carobnjak c){
      this.c = c;
      this.s = ...; //Random smer
      this.pomeraj = ...; //Random pomeraj
   }

   public void aktiviraj(){
      this.c.teleportujSe(this.s, this.pomeraj);
   }

}
```

Задатак 10

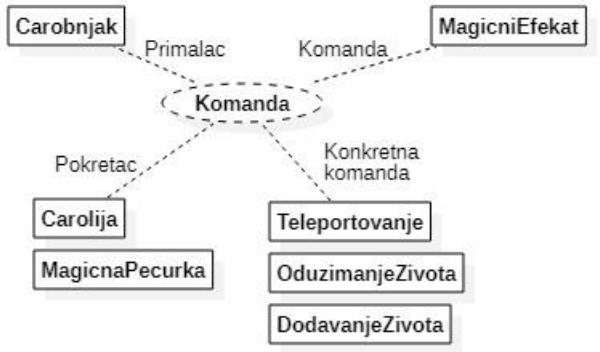
```
class DodavanjeZivota extends MagicniEfekat{
   private Carobnjak c;

   public DodavanjeZivota(Carobnjak c){
      this.c = c;
   }

   public void aktiviraj(){
      this.c.dodajZivot();
   }

}
```

```
class OduzimanjeZivota extends MagicniEfekat{
   private Carobnjak c;

   public OduzimanjeZivota(Carobnjak c){
      this.c = c;
   }

   public void aktiviraj(){
      this.c.oduzmiZivot();
   }

}
```

```
class Teleportovanje extends MagicniEfekat{
   private Carobnjak c;
   private Smer s;
   private int pomeraj;

   public Teleportovanje(Carobnjak c){
      this.c = c;
      this.s = ...; //Random smer
      this.pomeraj = ...; //Random pomeraj
   }

   public void aktiviraj(){
      this.c.teleportujSe(this.s, this.pomeraj);
   }

}
```

```
class Carolija extends MagicniEfekat{
   private ArrayList<MagicniEfekat> efekti;

   public Carolija(){
      this.efekti = new ArrayList<>();
   }

   public void aktiviraj(){
      for(MagicniEfekat m: efekti){
         m.aktiviraj();
      }
   }

   public void dodajEfekat(MagicniEfekat m){
      this.efekti.add(m);
   }

   public void izbaciEfekat(MagicniEfekat m){
      this.efekti.remove(m);
   }

}
```

Cesta kolaboracija uzorka Komanda i uzorka Sastav koja se naziva Makrokomand.



Задатак 10 - сарадња

Задатак 11

- Посматра се део имплентације једног текстуалног едитора, који има различите опције у менију дугмића. У систему постоји панел, на који може да се постави текст, који може да се прикаже (логика приказивања панела кориснику није од интереса). У сваком тренутку, корисник одабиром одговарајуће ставке у менију може да сачува стање панела, тако што се чува текст који је приказан на њему. Одабиром одговарајуће ставке у менију, корисник може да врати последње сачувано стање панела, што може да ради онолико пута колико је одрађено и само чување стања.
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 11 - решење

- Честа комбинација узорака – Команда и Подсетник

Задатак 11 - решење

- Честа комбинација узорака – Команда и Подсетник

```
public class Button {

   private Command command;

   public void setCommand(Command command) {
      this.command = command;
   }

   public void execute() {
      command.execute();
   }
}
```

Задатак 11 - решење

- Честа комбинација узорака – Команда и Подсетник

```
public class Button {

   private Command command;

   public void setCommand(Command command) {
      this.command = command;
   }

   public void execute() {
      command.execute();
   }
}
```

```
public abstract class Command {
   public abstract void execute();
}
```

Задатак 11 - решење

- Честа комбинација узорака – Команда и Подсетник

```
public class Button {

   private Command command;

   public void setCommand(Command command) {
      this.command = command;
   }

   public void execute() {
      command.execute();
   }
}
```

```
public class SaveCommand extends Command {

   private Panel panel = Panel.getInstance();

   @Override
   public void execute() {
      String savedText = panel.getText();
      PanelStateLog.getInstance().push(new PanelState(savedText));
   }

}
```

```
public abstract class Command {
   public abstract void execute();
}
```

Задатак 11 - решење

- Честа комбинација узорака – Команда и Подсетник

```
public class UndoCommand extends Command {

   private Panel panel = Panel.getInstance();

   @Override
   public void execute() {
      panel.setText(PanelStateLog.getInstance().pop().getText());
   }

}
```

```
public class Button {

   private Command command;

   public void setCommand(Command command) {
      this.command = command;
   }

   public void execute() {
      command.execute();
   }
}
```

```
public class SaveCommand extends Command {

   private Panel panel = Panel.getInstance();

   @Override
   public void execute() {
      String savedText = panel.getText();
      PanelStateLog.getInstance().push(new PanelState(savedText));
   }

}
```

```
public abstract class Command {
   public abstract void execute();
}
```

Задатак 11 - решење

- Честа комбинација узорака – Команда и Подсетник

```
public class Button {

   private Command command;

   public void setCommand(Command command) {
      this.command = command;
   }

   public void execute() {
      command.execute();
   }
}
```

```
public class SaveCommand extends Command {

   private Panel panel = Panel.getInstance();

   @Override
   public void execute() {
      String savedText = panel.getText();
      PanelStateLog.getInstance().push(new PanelState(savedText));
   }

}
```

```
public class UndoCommand extends Command {

   private Panel panel = Panel.getInstance();

   @Override
   public void execute() {
      panel.setText(PanelStateLog.getInstance().pop().getText());
   }

}
```

```
public abstract class Command {
   public abstract void execute();
}
```

Уведене додатне класе, ради боље расподеле одговорности међу класама:

- `PanelState`
- `PanelStateLog`

Задатак 11 - решење

```
public class PanelStateLog {

    private static PanelStateLog instance = null;

    private PanelStateLog() {

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

    private Stack<PanelState> stateLog = new Stack<>();

    public void push(PanelState state) {
        stateLog.push(state);
    }

    public PanelState pop() {
        return stateLog.pop();
    }
}
```

Задатак 11 - решење

```
public class PanelStateLog {

    private static PanelStateLog instance = null;

    private PanelStateLog() {

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

    private Stack<PanelState> stateLog = new Stack<>();

    public void push(PanelState state) {
        stateLog.push(state);
    }

    public PanelState pop() {
        return stateLog.pop();
    }
}
```

Служи за логику памћења свих сачуваних подсетника, Чувар.

Задатак 11 - решење

```
public class PanelStateLog {

    private static PanelStateLog instance = null;

    private PanelStateLog() {

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

    private Stack<PanelState> stateLog = new Stack<>();

    public void push(PanelState state) {
        stateLog.push(state);
    }

    public PanelState pop() {
        return stateLog.pop();
    }
}
```

```
public class PanelState {
    private String text;

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public PanelState(String text) {
        this.text = text;
    }
}
```

Задатак 11 - решење

```
public class PanelStateLog {

    private static PanelStateLog instance = null;

    private PanelStateLog() {

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

    private Stack<PanelState> stateLog = new Stack<>();

    public void push(PanelState state) {
        stateLog.push(state);
    }

    public PanelState pop() {
        return stateLog.pop();
    }
}
```

```
public class PanelState {
    private String text;

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public PanelState(String text) {
        this.text = text;
    }
}
```

Служи са енкапсулацију запамћеног стања класе Panel, Подсетник.

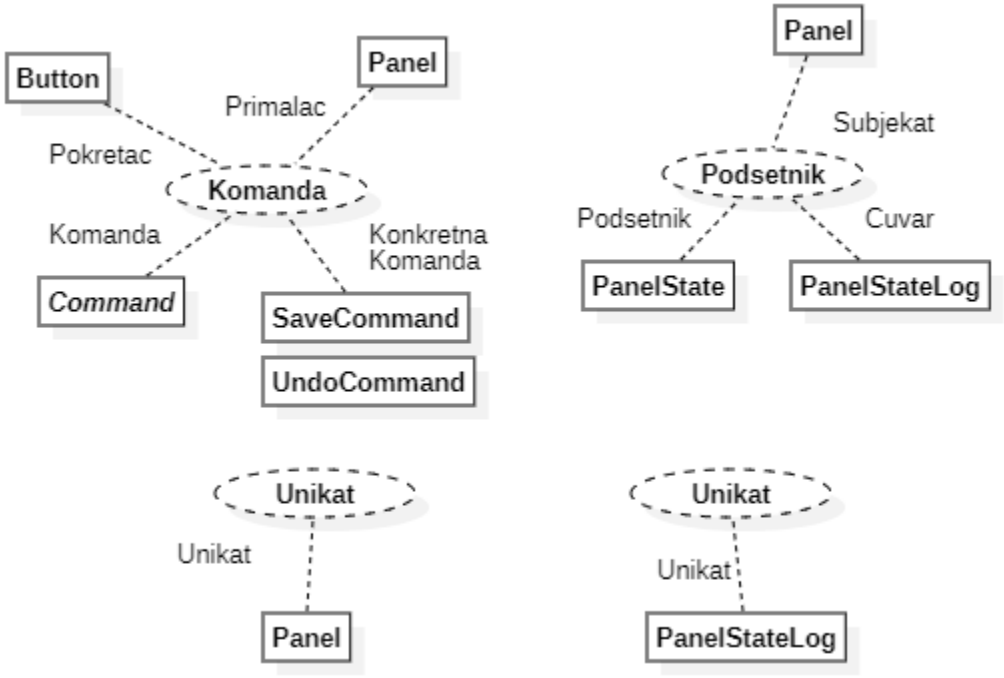
Задатак 11 - решење

```
public class PanelStateLog {

    private static PanelStateLog instance = null;

    private PanelStateLog() {

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

    private Stack<PanelState> stateLog = new Stack<>();

    public void push(PanelState state) {
        stateLog.push(state);
    }

    public PanelState pop() {
        return stateLog.pop();
    }
}
```

```
public class PanelState {
    private String text;

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public PanelState(String text) {
        this.text = text;
    }
}
```

```
public class Panel {

    private String text;
    private static Panel instance = null;
    public static Panel getInstance() {
        if (instance == null) {
            instance = new Panel();
        }
        return instance;
    }
    private Panel() {
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

Класа чије се стање памти, Субјекат.



Задатак 11 - сарадња

Задатак 11 - дискусија

- У овом решењу се памте само експлицитно сачувана стања. Како би се имплементирала варијанта где се памте све извршене команде?
- Да ли је могуће ефикасније имплементирати Подсетника, тако што се не памти комплетно стање Субјекта?

Задатак 12

- Посматра се део имплементације система за трговање на берзи. У систему постоји сервис за акције који има глобалну тачку приступа за све клијенте система. Корисници могу да се пријаве на сервис и одјаве са сервиса. Сервис прати стање свих акција, и када се цена неке акције промени јавља свим корисницима који су пријављени на сервис која акција се променила и за колико. Сервис такође омогућава куповину и продају акција. Индивидуални корисник је корисник система који сваки пут када некој акцији опадне цена купи две деонице дате акције, а када цена некој акцији порасте при чему поседује деонице дате акције, све их продаје. Компаније су корисници система које на следећи начин реагују на промене акција - када некој акцији опадне цена и цена се налази испод прага дефинисаног за дату компанију, купује пет деоница дате акције, а када цена некој акцији порасте при чему поседује деонице дате акције, све их продаје.
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 12 - решење

```
class Stock{
    private String name;
    private double price;

    public Stock(String name, double price){
        this.name = name;
        this.price = price;
    }

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

    public String getPrice(){
        return this.price;
    }

}
```

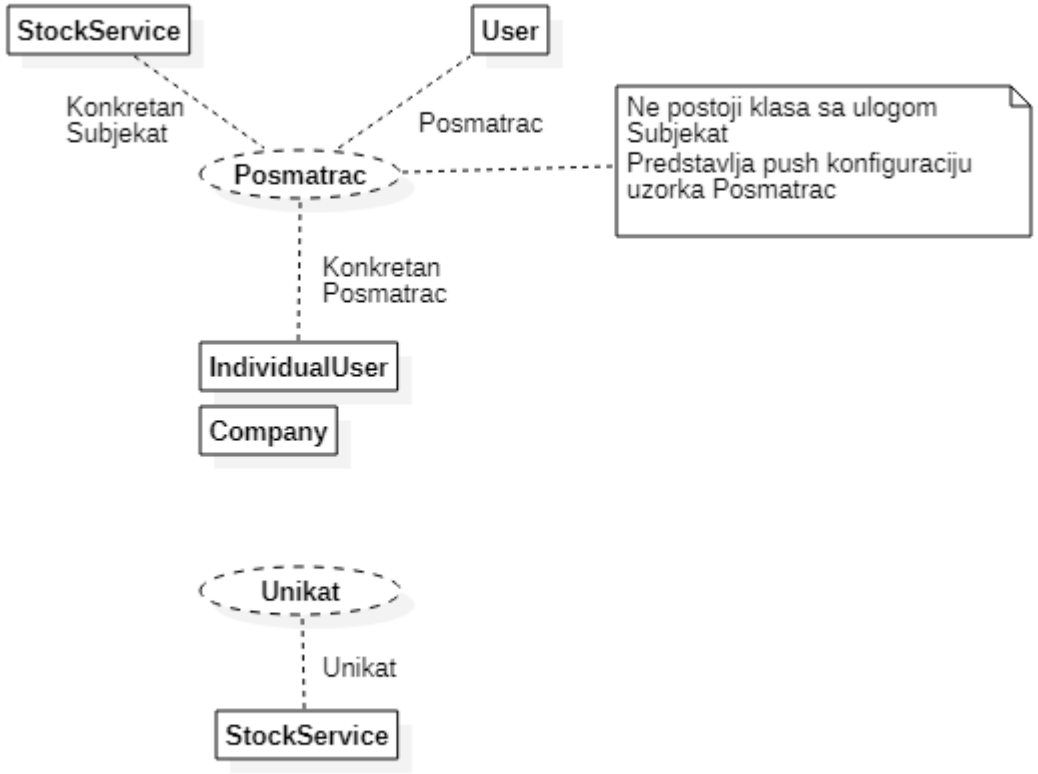
```
class StockService{
    private static StockService instance  = null;

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

    private StockService(){
        this.users = new ArrayList<>();
        this.stocks = new HashMap<>();
        //Inicijalizacija stockova...
    }
    private ArrayList<User> users;
    private HashMap<Stock, Integer> stocks;

    public void registerUser(User user){
        this.users.add(user);
    }

    public void removeUser(User user){
        this.users.remove(user);
    }

    public void buyStock(Stock stock, int count){
        stocks.put(stock, stocks.get(stock) - count);
    }

    public void sellStock(Stock stock, int count){
        stocks.put(stock, stocks.get(stock) + count);
    }

    protected void stockChange(Stock stock, double delta){
        for(User user: users){
            users.update(stock, delta);
        }
    }

    //... Logika kojom stock service saznaje koji stock se promenio i koliko ...
}
```

Задатак 12 - решење

```
interface User{
    public void update(Stock
stock, double delta);

}
```

```
class Company implements User{

    private static final int stockNumber = 5;

    private double threshold;
    private HashMap<Stock, Integer> stocks;

    public Company(double threshold){
        this.threshold = threshold;
        this.stocks = new HashMap<>();
    }

    public void update(Stock stock, double delta){
        if(delta < 0 && stock.getPrice() < threshold){
            StockService.getInstance().buyStock(stock,
stockNumber);
            stocks.set(stock, stockNumber);
        }
        if(delta > 0 && stocks.get(stock) != null){
            StockService.getInstance().sellStock(stock,
stocks.get(stock));
            stocks.set(stock, 0);
        }
    }

}
```

```
class IndividualUser implements User{
    private static final int stockNumber = 2;

    private HashMap<Stock, Integer> stocks;

    public User(){
        this.stocks = new HashMap<>();
    }

    public void update(Stock stock, double delta){
        if(delta < 0){
            StockService.getInstance().buyStock(stock,
stockNumber);
            stocks.set(stock, stockNumber);
            return;
        }
        if(delta > 0 && stocks.get(stock) != null){
            StockService.getInstance().sellStock(stock,
stocks.get(stock));
            stocks.set(stock, 0);
        }
    }

}
```



Задатак 12 - сарадња

Задатак 12 - дискусија

- Логика методе освежи код индивидуалног корисника и компаније није лако скалабилна за нове начине бирања када се купују и продају акције. Како то променити?
- У датој имплементацији је дата push конфигурација узорка Посматрач. Како би изгледала *pull* конфигурација, и која конфигурација је бољи избор у датом систему?

Задатак 13

- Посматра се део имплементације неке једноставне игре. Јунак има своју снагу, новчиће и инвентар који се састоји од ствари. Јунакова снага се рачуна тако што се на његову почетну снагу додају снаге свих ствари из његовог инвентара. Јунак може да прода неку ствар из свог инвентара, при чему добија онолико новчића колико је била вредност дате ствари. Ствар има своју снагу и вредност. Накит је ствар која нема снагу и има вредност 1000. Оружје је ствар која има снагу 5000 и вредност 50. Торба је ствар у коју се додаје произвољан број других ствари. Снага и вредност торбе се рачуна као збир снаге и вредности садржаних ствари. Непријатељ има своју снагу и јунак може да интерагује с њим. Интеракција са јунаком се своди на борбу у којој се победа рачуна у односу на снагу. Ако јунак победи, непријатељ прави награду и додељује је јунаку. Вилењак је непријатељ који као награду прави накит. Патуљак је непријатељ који као награду прави оружје. Џин је непријатељ који као награду прави торбу попуњену са 5 насумично направљених ствари.
- Приказати имплементацију система и навести одговарајуће пројектне узорке система приказане дијаграмима сарадње.

Задатак 13 - решење

```
abstract class Neprijatelj{
    protected int snaga;

    public Neprijatelj(int snaga){
        this.snaga = snaga;
    }

    public void interaguj(Junak j){
        if(j.dohvatiSnagu() > this.snaga){
            Stvar nagrada = napraviNagradu();
            j.dodajNagradu(nagrada);
        }
    }

    abstract public Stvar napraviNagradu();
}
```

```
class Junak{

    private ArrayList<Stvar> inventar;

    private int snaga;

    private int novcici;

    public Junak(int snaga){
        this.snaga = snaga;
        this.novcici = 0;
    }

    public int snaga(){
        int konacnaSnaga = this.snaga;
        for(Stvar s: inventar) konacnaSnaga +=
snaga();
        return konacnaSnaga;
    }

    public void prodaj(int index){
        Stvar s = inventar.pop(index);
        this.novcici += s.vrednost();
    }

    public void dodajNagradu(Stvar s){
        this.inventar.add(s);
    }

}
```

```
class Viljenak extends Neprijatelj{
    public Vilenjak(int snaga){
        super(snaga);
    }
    public Nakit napraviNagradu(){
        Nakit nagrada = new Nakit();
        return nagrada;
    }
}
```

```
class Patuljak extends Neprijatelj{
    public Patuljak(int snaga){
        super(snaga);
    }
    public Oruzje napraviNagradu(){
        Oruzje nagrada = new Oruzje();
        return nagrada;
    }
}
```

```
class Dzin extends Neprijatelj{
    public Dzin(int snaga){
        super(snaga);
    }
    public Torba napraviNagradu(){
        Torba nagrada = new Torba();
        for(int i=0; i<5; i++){
            torba.dodajStvar(new ...); //Dodavanje random
konkretnih stvari
        }
        return nagrada;
    }
}
```

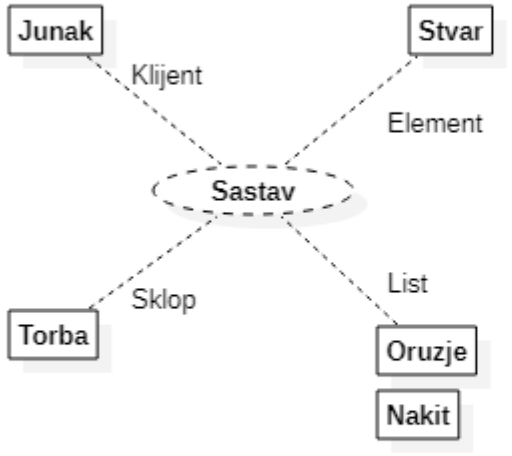
Задатак 13 - решење

```
abstract class Stvar{
    abstract public int snaga();

    abstract public int vrednost();

    public void dodajStvar(Stvar s){

    }

}
```

```
class Nakit extends Stvar{
    public int snaga(){
        return 0;
    }
    abstract public int vrednost(){
        return 2000;
    }

}
```

```
class Oruzje extends Stvar{
    public int snaga(){
        return 5000;
    }
    abstract public int vrednost(){
        return 50;
    }

}
```

```
class Torba extends Stvar{
    private ArrayList<Stvar> stvari;

    public Torba(){
        this.stvari = new ArrayList<>();
    }
    public int snaga(){
        int snaga = 0;
        for(Stvar s: stvari) snaga += s.snaga();
        return snaga;
    }

    public int vrednost(){
        int vrednost = 0;
        for(Stvar s: stvari) snaga += s.vrednost();
        return vrednost;
    }

    public void dodajStvar(Stvar s){
        this.stvari.add(s);
    }

}
```



Задатак 13 - сарадња

Задатак 13 - дискусија

- Метод за интеракцију – шаблонски метод.