SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Lagatoria

Objektovo-orientované programovanie

Róbert Junas

FIIT STU

Cvičenie: streda 14:00

Cvičiaci: Ing. Anna Považanová

11.4.2021

Id: 102970

1. Rámcové zadanie

Knihy sú dôležitou súčasťou každého z nás. Preto ich plánovanie a vydávanie je veľmi dôležité. Môj projekt sa zaoberá vydávaním kníh do stánkov a vydavateľovej predajne. Teda výsledkom plánovania bude kniha predávajúca sa v kníhkupectve.

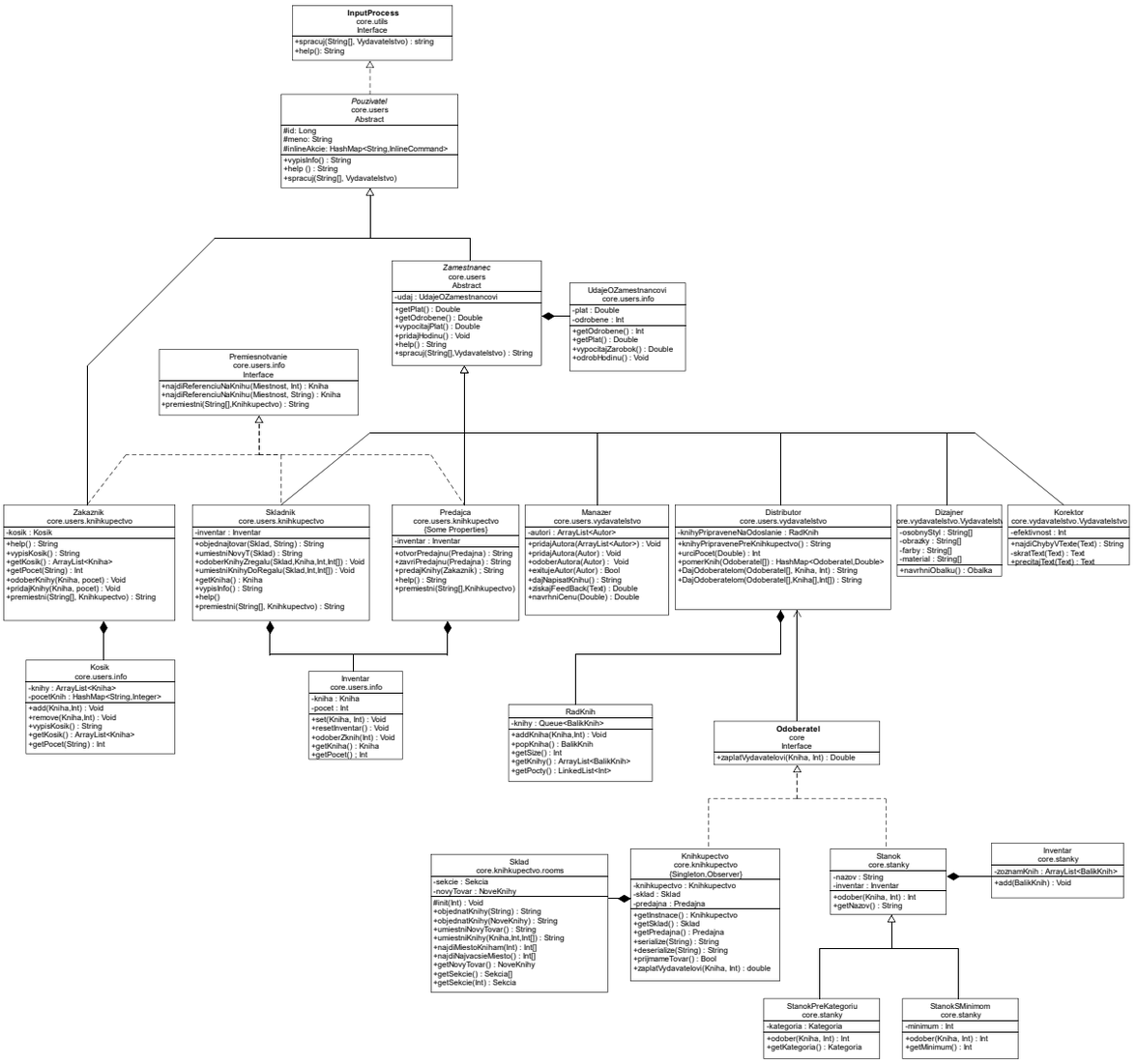
Autor musí najprv knihu napísať – dostane nápad, napíše ju, vymyslí názov a určí žáner knihy. Potom dá svoju knihu vydavateľstvu.

Vydavateľstvo sa skladá z ľudí – manažéra, ktorý je v kontakte s autorom, dizajnér, ktorý navrhne obálku a vyberie väzbu knihy; a korektor opravujúci chyby v knihe. Manažér navrhne cenu knihy a nakoniec distribútor povie koľko kníh sa má vytlačiť a rozdelí medzi odoberateľov. Po tomto procese sa knihy dajú na tlač, kde sa vytlačia strany, pripevnia sa k väzbe a pošlú sa odoberateľom.

Kníhkupectvo sa skladá z predajne a skladu. Predajňa môže byť organizovaná do viacerých kategórií, ako je žáner/druh. V sklade sa ale ukladajú knihy podľa toho, na ktoré miesto, prípadne viac priľahlých miest, sa zmestia. V kníhkupectve pracuje predajca a skladník. Predajca obsluhuje zákazníka a dopĺňa knihy do predajne. Skladník pridáva knihy do skladu a premiestňuje ich do iných sektorov skladu, aby mohol urobiť miesto pre nové knihy.

Zákazník môže nájsť knihu v predajni, vidieť informácie o knihe.

2. Štruktúra



3. Kritéria

3.1. Hlavné kritéria

3.1.1 Dedenie

3.1.2 Polymorfizmus

3.1.3 Rozhrania

3.1.4 Zapuzdrenie

3.1.5 Agregácia

3.1.6 Oddelenie aplikačnej logiky od používateľského rozhrania

3.1.7 Organizácia do balíkov

3.2. Ďalšie kritéria

3.2.1 Návrhové vzory

Observer

Model Observer sme využili naprieč celým projektom, či už na komunikáciu ovládačov s View alebo Modelom, ale aj medzi triedami Manažér a Autor, kde Manazer upovedomí svojich autorov, že chce aby napísali knihu.

```
public void dajNapisatKnihu(){
    observer.notify( caller: this, msg: "Manazer rozposiela ziadosti o knihu\n");
    for (Autor autor: autori) {
        autor.notify( caller: this, msg: "");
    }
}
```

Observer bol využitý medzi autormi a vydavateľstvom, kde autor upovedomuje vydavateľstvo o svojom postupe v písaní textu.

```
public String vymysliKnihu() {
    observer.notify( caller: this, msg: "Autor ["+getMeno()+" "+getPrievzisko()+"] vymyslel Knihu");
    int targetStringLength = 10;
    Random random = new Random();

    return random.ints( randomNumberOrigin: 97,  randomNumberBound: 123)
            .limit(targetStringLength)
            .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)
            .toString();
}

  public synchronized void odosliVydavatelovi(Text text){
      observer.notify( caller: this,text);
      piseKnihu = false;
  }
```

Máme definované aj rozhranie Observer, kde je definovaný protyp funkcie notify(Object, Object)

Composite

Využívaný spojení s knihami, kde kniha sa skladá z textov a obálky a nakoniec aj knihy sa balia do balíkov kde je ich väčší počet.

```
/**
 * zakladny inteface pre navrhovy vzor Composite pre Knihy
 */
public interface CastiKnihy extends Serializable {
    /**
     * vypisuje informacie o knihe
     */
    String getInfo();
}
```

```java
public class Kniha implements CastiKnihy {
    private String  isbn, vydavatelstvo;
    private int rok, predaneKusy;
    private double cena;
    private Boolean bestseller;
    private ArrayList<CastiKnihy> casti;

    public Kniha(String isbn, String vydavatelstvo, int rok, double cena){...}

    @Override
    public String getInfo() {
        String res = "";
        for(CastiKnihy c : casti){
            c.getInfo();
        }
        res += "\tISBN: " + isbn + '\n';
        res += "\tcena: " + String.format("%.2f",cena) + "€ - vydavatel: " + vydavatelstvo + " " + rok +"\n";

        return res;
    }

    public void pridajSucast(CastiKnihy cast){
        if(cast instanceof Obalka){
            for(CastiKnihy c : casti){
                if(c instanceof Obalka){
                    return;
                }
            }
        }

        casti.add(cast);
    }

    public CastiKnihy getSucast(Class t){
        for(CastiKnihy c : casti){
            if(t == c.getClass()){
                return c;
            }
        }
        return null;
    }
```

```java
/**
 * Obalka a cast knihy
 */
public abstract class Obalka implements CastiKnihy {
    private String dizajn;
    private String farba;
    protected String typObalky;

    public Obalka(String dizajn, String farba){
        this.dizajn = dizajn;
        this.farba = farba;
    }

    @Override
    public String getInfo() {
        String res = "";
        res+="\tObalka:\n";
        res+="\t\tdizajn: " + dizajn + ", farba: " + farba + "\n";

        return res;
    }
}
```

```java
public class Text implements CastiKnihy {
    private String autor, nazov, jazyk;
    private int dlzka;
    private Boolean opravene;
    private Kategoria kategoria;

    public Text(String nazov,String autor, String jazyk, int dlzka, Kategoria kategoria, Boolean opravene){
        this.opravene = opravene;
        this.autor = autor;
        this.nazov = nazov;
        this.jazyk = jazyk;
        this.dlzka = dlzka;
        this.kategoria = kategoria;
    }
    public Text(String nazov,String autor, String jazyk, int dlzka, Kategoria kategoria){
        this.opravene = false;
        this.autor = autor;
        this.nazov = nazov;
        this.jazyk = jazyk;
        this.dlzka = dlzka;
        this.kategoria = kategoria;
    }
    @Override
    public String getInfo() { return "\t["+kategoria.toString()+"] "+autor + ": " + nazov + " [" + jazyk+ "]\n"; }
```

Visitor

V projekte sa vyskytuje aj model Visitor a to spôsobom, že každý druh autora píše iným spôsobom, teda trvá mu to kratšie alebo dlhšie, vzhľadom koľko práce musí dať do samotného písania. Máme definované dva spôsoby písania: Normálne a rýchle písanie. Kde sa odráža počet strán od rýchlosti akou autor písal.

```java
public interface Pisanie {

    Text visit(HistoryAutor autor) throws InterruptedException;
    Text visit(FantasyAutor autor) throws InterruptedException;
    Text visit(PoetryAutor autor) throws InterruptedException;
}
```

```java
public class NormalnePisanie implements Pisanie{

    @Override
    public Text visit(HistoryAutor autor) throws InterruptedException {
        Thread.sleep( millis: 8000);
        int pocetStran = (int)(Math.random()*(1200-300+1)+300);
        String meno = autor.getMeno() + " " + autor.getPrievzisko();
        return new Text( nazov: "Nazov Historickej knihy: ",meno, jazyk: "Slovensky",pocetStran, Kategoria.HISTORIA);
    }

    @Override
    public Text visit(FantasyAutor autor) throws InterruptedException {
        Thread.sleep( millis: 5000);
        int pocetStran = (int)(Math.random()*(700-300+1)+300);
        String meno = autor.getMeno() + " " + autor.getPrievzisko();
        return new Text( nazov: "Nazov romanu: ",meno, jazyk: "Slovensky",pocetStran, Kategoria.FANTASY);
    }

    @Override
    public Text visit(PoetryAutor autor) throws InterruptedException {
        Thread.sleep( millis: 7000);
        int pocetStran = (int)(Math.random()*(200-50+1)+50);
        String meno = autor.getMeno() + " " + autor.getPrievzisko();
        return new Text( nazov: "Nazov basniciek: ",meno, jazyk: "Slovensky",pocetStran, Kategoria.POEZIA);
    }
}
```

```java
public class RychlePisanie implements Pisanie {
    @Override
    public Text visit(HistoryAutor autor) throws InterruptedException {
        Thread.sleep( millis: 5000);
        int pocetStran = (int)(Math.random()*(900-300+1)+300);
        String meno = autor.getMeno() + " " + autor.getPrievzisko();
        return new Text( nazov: "Nazov Historickej knihy: ",meno, jazyk: "Slovensky",pocetStran, Kategoria.HISTORIA);
    }

    @Override
    public Text visit(FantasyAutor autor) throws InterruptedException {
        Thread.sleep( millis: 2500);
        int pocetStran = (int)(Math.random()*(500-300+1)+300);
        String meno = autor.getMeno() + " " + autor.getPrievzisko();
        return new Text( nazov: "Nazov romanu: ",meno, jazyk: "Slovensky",pocetStran, Kategoria.FANTASY);
    }

    @Override
    public Text visit(PoetryAutor autor) throws InterruptedException {
        Thread.sleep( millis: 3000);
        int pocetStran = (int)(Math.random()*(100-50+1)+50);
        String meno = autor.getMeno() + " " + autor.getPrievzisko();
        return new Text( nazov: "Nazov basniciek: ",meno, jazyk: "Slovensky",pocetStran, Kategoria.POEZIA);
    }
}
```

```java
@Override
public Text accept(Pisanie pisanie) throws InterruptedException {
    return pisanie.visit( autor: this);
}
```

Použitie v triede Autor.java

```java
public Text accept(Pisanie pisanie) throws InterruptedException {
    return null;
};
```

```java
final Pisanie pisanie;
if((int)(Math.random()*5) == 0){
    pisanie = new RychlePisanie();
}else{
    pisanie = new NormalnePisanie();
}
Thread thread = new Thread(new Runnable() {
    String nazov = "";
    @Override
    public void run() {
        Text text = null;

        Runnable myslienky = new Runnable() {...};

        Runnable hotovo = new Runnable() {...};

        try {...} catch (InterruptedException e) {...}
        Platform.runLater(myslienky);

        try {
            text = autor.napisText(pisanie);
            text.setNazov(nazov);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        autor.odosliVydavatelovi(text);
        Platform.runLater(hotovo);
    }
});
```

Strategy

??????

3.2.2 Výnimky

V aplikácií máme implementovaných niekoľko výnimiek: AutorExistujeException, AutorNieJeNaZozname, InvalidFormatException.

InvalidFormatException:

```
/**
 * Vyhadzuje sa prave vtedy ked citani subor obsahuje knihu v zle zadanom formate
 * format: nazov///Autor///ISBN///pocet Kusov///pocet Stran///cena///Kategoria///jazyk///vazba///vydavatel///rok
 * kontroluje sa ci pocet Kusov, Stran,rok ci su zadane ako Int a cena ci je typu float
 */
public class InvalidFormatException extends Exception{

    private int loadedRows;

    /**
     * @param message chybove hlasenie
     * @param loadedRows cislo riadku s chybou
     */
    public InvalidFormatException(String message, int loadedRows){
        super(message);
        this.loadedRows = loadedRows;
    }


    /**
     * @return vrati pocet nacitanych riadkov vratane chyboveho
     */
    public int getLoadedRows() { return loadedRows; }
}
```

Funkcia vyhadzujúca výnimku:

```
public boolean nacitajKnihy(String path) throws InvalidFormatException, FileNotFoundException {...}
```

Konštruktor chytá výnimku:

```
public NoveKnihy(String path) throws FileNotFoundException, InvalidFormatException{
    super();
    try {
        minute = !nacitajKnihy(path);
    } catch (InvalidFormatException e) {
        if(e.getLoadedRows() > 0){
            minute = false;
        }
        else{
            minute = true;
        }
        throw e;
    } catch (FileNotFoundException e){
        minute = true;
        throw e;
    }
}
```

AutorExistujeException:

```java
/**
 * Vyhadzuje sa iba vtedy ak manazer uz ma autora vo svojom zozname
 */
public class AutorExistujeException extends Exception{

    public AutorExistujeException(String msg) { super("autor ("+msg+"), uz je na manazerovom zozname"); }
}
```

Funkcia vyhadzujúca výnimku:

```java
public void pridajAutora(Autor autor) throws AutorExistujeException {
    if(autori.contains(autor))
        throw new AutorExistujeException(autor.getMeno());
    else
        this.autori.add(autor);
}
```

Metóda chytajúca výnimku:

```java
/**
 * Snazi sa pridat vsetkych autorov vo vydavatelstve manazerovi
 * Osetruje sa vynimka ked autor uz daneho autora ma
 */
public String dajAutorovManazerovi(){
    String res = "";
    for (Autor autor: autori) {
        try {
            manazer.pridajAutora(autor);
        } catch (AutorExistujeException e) {
            res +=e.getMessage() + "\n";
        }
    }
    return  res;
}
```

AutorNieJeNaZozname:

```java
/**
 * Vyhadzuje sa vtedy keď chceme odobrať autora,
 * ktorý sa nenachádza na zozname a teda sa nedá odobrať
 */
public class AutorNieJeNaZozname extends Exception {
    public AutorNieJeNaZozname(String msg) { super("autor ("+msg+"),nie je na manazerovom zozname"); }
}
```

Funkcia vyhadzujúca výnimku:

```java
public void odoberAutora(Autor autor) throws AutorNieJeNaZozname {
    if (!autori.contains(autor)) throw new AutorNieJeNaZozname(autor.getMeno());

    autori.remove(autor);
}
```

Metóda chytajúca výnimku:

```
inlineAkcie.put("odoberA", ((args, kh, vy) -> {
    try {
        odoberAutora(vy.getAutor(Integer.valueOf(args[1])));
    } catch (AutorNieJeNaZozname autorNieJeNaZozname) {
        autorNieJeNaZozname.printStackTrace();
        return autorNieJeNaZozname.getMessage();
    }
    return "";
}));
```

3.2.3 GUI

3.2.4 Multithreading

Viacnitovosť bola využitá v Autoroch, ktorý sú schopný naraz písať texty, ktoré následne pošlú vydavateľovi.

```
public void notify(Object caller, Object msg){
    if(piseKnihu) {
        observer.notify( caller: this,  msg: "Autor neprijal poziadavku");
        return;
    }
    observer.notify( caller: this, msg: "\t" + meno + " " + prievzisko +" prijal poziadavku\n");

    piseKnihu = true;
    Autor autor = this;
    final Pisanie pisanie;
    if((int)(Math.random()*5) == 0){...}else{...}
    Thread thread = new Thread(new Runnable() {
        String nazov = "";
        @Override
        public void run() {
            Text text = null;

            Runnable myslienky = new Runnable() {...};

            Runnable hotovo = new Runnable() {...};

            try {...} catch (InterruptedException e) {...}
            Platform.runLater(myslienky);

            try {
                text = autor.napisText(pisanie);
                text.setNazov(nazov);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            autor.odosliVydavatelovi(text);
            Platform.runLater(hotovo);
        }
    });

    thread.setDaemon(true);
    thread.start();
}
```

3.2.5 Generickosť tried

3.2.6 RTTI

Použitie RTTI je v triede Distributor, vo funkciach *dajOdobertalom* a *pomerKnih*, kde sa výrazne líši správanie Stánkov a Kníhkupectva. Kníhkupectvo nie vždy musí hneď prijať knihy od Vydavatelstva a neprijaté knihy sa pridávajú na list kníh čakajúcich na prijatie. Na druhú stranu Stanky, prijímajú knihy neustále, bez zbytočného vyčkávania.

```
public String dajOdoberatelom(ArrayList<Odoberatel> odoberatelia, Kniha kniha, int pocet) {
    HashMap<Odoberatel, Double> pomer = pomerKnih(odoberatelia);
    double kapital = 0;
    int nepredane = pocet;
    String res ="";
    for (Odoberatel o : odoberatelia){
        if(o instanceof Knihkupectvo){...}
        else{
            Stanok stanok = (Stanok) o;
            if(stanok.odober(kniha,(int)(pocet*pomer.get(o))) == 1) {
                nepredane -= (int) (pocet * pomer.get(o));
                double zaplatene = stanok.zaplatVydavatelovi(kniha, (int) (pocet * pomer.get(o)));
                res += "Stanok "+ stanok.getNazov() + " prijal knihy [" + (int) (pocet * pomer.get(o)) + "] a zaplatil: "
                        + String.format("%.2f", zaplatene) + "\n";
                kapital += zaplatene;
            }else{
                res+= "Stanok " +stanok.getNazov()+ " neprijal knihy\n";
            }
        }
    }
    res += "\nCelkovo zarobene: "+ String.format("%.2f",kapital) + "€\n";
    res += "Nepredalo sa: "+ nepredane + " kusov\n";

    return res;
}
```

3.2.7 Vhniezdené triedy

Vhniezdené triedy a rozhrania sa nachádzajú v triede Vydavatelstvo.java. Implementujeme tu stratégiu vydávania ako aj triedy Korektor a Dizajnér, ktoré by mali byť vždy iba súčasťou Vydavateľstva.

```
interface VydavanieStrategy{
    String vydajKnihy();
}

/**
 * Inner class Korektor sluzi na opravenie chyb v texte
 */
class Korektor extends Zamestnanec {...}

/**
 * inner class dizajner, ma na starosti vymyslenie obalky
 */
class Dizajner extends Zamestnanec {...}
```

3.2.8 Lambda výrazy

Jedno využitie lambda výrazov/refrencií na metódy sme implementovali v triede Vydavatelstvo.java, kde si ukladáme spôsob vydávania, teda vydá sa práve jedna kniha alebo všetky knihy čakajúce na zozname.

```java
/**
 * Funkcia urci strategiu akou sa budu knihy vydavat (Bud vsetky naraz alebo iba jedna)
 * Strategia ma dopad na pocet vytlackov aj cenu knihy
 * pri tlaceni vsetkych sa zhorsuje kvalita teda aj pocet vytlackov sa zhorsuje
 * @param vydajVsetko ak je parameter true tak sa vydaju vsetky knihy v rade, inak sa vydava po jednej
 */
public void typVydavania(boolean vydajVsetko){
    if(vydajVsetko){
        strategia = () -> {
            String res = "";
            ArrayList<Kniha> vytlaceneKnihy = new ArrayList<>();
            ArrayList<Integer> pocetVytlackov = new ArrayList<>();
            if(prijateTexty.isEmpty()){...}
            while(!prijateTexty.isEmpty()) {...}
            res += distributor.DajOdoberatlom(odoberatelia,vytlaceneKnihy,pocetVytlackov);
            return res;
        };
    }else{
        strategia = this::vydanie;
    }
}
```

```java
/**
 * Vyda text ktory sa nachadza na vrchu radu, do ktoreho sa text dostane cez funkciu prijmiText(Text text)
 * @return vysledok vydavania, resp. kolko aky stanok prijal a aká kniha bola vydaná a v akom mnozstve + zarobok
 */
public String vydajKnihy(){
    String result = strategia.vydajKnihy();
    model.notify( objekt: this,  msg: "001::"+vypisTexty());
    return result;
}
```

Ďalej sme lambda výrazy použili pri volaní funkcií používateľov

```java
inlineAkcie.put("otvor", (args, kh, vy) -> otvorPredajnu(kh.getPredajna()));
inlineAkcie.put("zavri", (args, kh, vy)-> zavriPredajnu(kh.getPredajna()));
inlineAkcie.put("predajna", (args, kh, vy) -> kh.getPredajna().vypisPredajnu());
inlineAkcie.put("sklad", (args, kh, vy) -> kh.getSklad().printSklad());
inlineAkcie.put("predaj", (args, kh, vy)->  predajKnihy(kh.getPredajna().getZakaznik()));
inlineAkcie.put("prines", ((args, kh, vy) -> premiestni(args,kh)));
```

3.2.9 Implicitná implementácia v rozhraní

Implicitné implementácie metód v rozhraniach sme využili napríklad v rozhraní Odoberatel:

```java
public interface Odoberatel {
    /**
     * @param kniha kniha ktoru sme prijali
     * @param pocet prijatych knih
     * @return  celkova cena za zaplatenie knih - 77% z plnej ceny knihy
     */
    default double zaplatVydavatelovi(Kniha kniha, int pocet) { return kniha.getCena()*0.77*pocet; };
}
```

Ale aj v rozhraní Premiestňovanie, ktoré implementujú triedy užívateľov kníhkupectva:

```java
public interface Premiestnovanie {

    default Kniha najdiReferenciuNaKnihu(Miestnost s, int i){
        return  (s.getKatalog().isEmpty() || i >= s.getKatalog().size() ) ? null : s.getKatalog().get(i);
    }

    default Kniha najdiReferenciuNaKnihu(Miestnost s, String id){
        for(Kniha kp : s.getKatalog()){
            if(kp.getISBN().toLowerCase().equals(id.toLowerCase())
                    || kp.getBasicInfo()[0].toLowerCase().equals(id.toLowerCase())){
                return kp;
            }
        }
        return null;
    }

    String premiestni(String[] args, Knihkupectvo kh);

}
```

3.2.10 Aspktovo-orientované programovanie

3.2.11 Použitie serializácie

Serializáciu používame na uloženie stavu kníhkupectva, teda na uloženie stavu predajne, skladu, sekcií, regálov, poličiek, ale aj kníh v uložených v regáloch. Výsledok serializácie sa ukladá do súboru knihkupectvo_oop.ser v priečinku /res/. Funkcie na serializáciu sú definované v Knihkupectvo.java:

```java
/**
 * Uklada sa instancia knihkupectva, knihy v nom a ich ulozenie
 * @param path cesta k suboru, kde sa ulozia informacie o knihkupectve
 * @return
 */
public static String serialize(String path){
    try{
        FileOutputStream fileOut = new FileOutputStream(path);
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(instancia);
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
        return "nepodarilo sa najst subor";
    }
    return "Knihkupecvto sa ulozilo do /res/knihkupectvo_oop.ser";
}

/**
 * metoda nacita informacie o knihkupectve do programu
 * @param path cesta k suboru odkial sa maju data o knihkupectve nacitat
 */
public static String deserialize(String path){
    try {
        FileInputStream fileIn = new FileInputStream(path);
        ObjectInputStream in = new ObjectInputStream(fileIn);
        instancia = (Knihkupectvo) in.readObject();
        in.close();
        fileIn.close();
        return "podarilo sa nacitat knihkupetvo";
    } catch (IOException e) {
        return "Nenasiel sa subor '"+path+"' -- vytvara sa nove knihkupectvo'";
    } catch (ClassNotFoundException e) {
    }
    getInstance();
    return "nepodarilo sa nacitat knihkupectvo treba vytvara sa nove";
}
```

Obrázok 1 Funkcie na serializáciu deserializaciu

Serializujú sa triedy:

```java
public class Knihkupectvo implements java.io.Serializable, Odoberatel, Observer {
```

```
public abstract class Miestnost implements java.io.Serializable{

public class Regal implements java.io.Serializable{

public class Sekcia implements java.io.Serializable{

public interface InfoKniha extends Serializable {
```

4. Verzie

Commit - 6a16c955b1a998761afc408d17bf35fd213d173b – *„rozdelenie controllera do viacerych casti“*

- Rozdelenie controllerov do viacerých tried
  - MainController – spracovanie vstupu od používateľa.
  - ViewController – spracovanie požiadaviek na úpravu view (výpis/otvorenie ďalšieho okna...)
  - ModelController – posielanie požiadaviek modelu
  - ButtonController – súčasť ModelControllera – spracovanie požiadaviek z tlačidiel.
- Prepojenie controllerov pomocou modelu observer.

Commit - 8272aa2ea6d19d04c5caf858b657488b79711e8c – *„Vytvorenie tabuliek v GUI, prepojenie niektorych tried pomocou observer“*

- Vytvorenie GUI tabuliek pre katalóg kníh v kníhkupectve, pre odoberateľov, pre texty prijaté na vydanie
- Vytvorenie tried na premieňanie String outputov na dáta vyložiteľné do tabuľky
- Vytvorenie interface Observer (implementujú Autor, Vydavatelstvo, Knihkupectvo a triedy dedené rozhraním Miestnosť)
- Prerobenie ako Controller posiela dáta View (pomocou Observer)
- Vytvorenie vlastných grafických elementov implementujúcich Observer.

Commit - ab54cc26c8f8a19a047d17de55b63fc65e33b975 – *“podokna presunuté z View do vlastných tried; vytvorenie tried BalikKnih, tried v users.info a vytvorenie viacerých stánkov”*.

- V tomto commit-e sme presunuli podokná z View do vlastných tried a ich vytváranie spracováva Controller.
- Ďalej sme pridali triedu BalikKnih a RadKnih, ktore nahrádzaju potrebu mať v triede, kde sa kumulujú knihy mať 2 polia na uloženie Knihy a jej počtu (v Distributor.java).
- Ďalej sme vytvorili viaceré druhy stánkov a aj možnosť ich vytvárať pomocou GUI. S novými stánkami upravené funkcie DajOdoberatelom().
- Vytvorenie triedy UdajeOZamestnancovi pre uchovanie údajov o zamestnancoch (plat a odrobený čas)

- Vytvorenie triedy users.info.Inventar, ktorá slúži na uchovanie knihy u zamestnanca knihkupectva
- Vytvorenie triedy stanky.Inventar na uloženie všetkých kníh, ktoré má stánok.

Commit - 30d0cce22bc7d08e58d3fbb43f7fe17381f749d3 – *„knihkupectvo gui - v1.0“*

- Dokončenie prvej verzie GUI – podporované iba kníhkupectvo

Commit - 2376a809d83267c261df1c929bf364c0e49f2687 *– „vytvorenie vydávania“*

- Nastavenie getterov a setterov v triede Text.java
- Úprava Kníhkupectva aby dokázal prijať knihy od Vydavateľstva
- Implementované metódy zamestnancov Vydavateľstva a implementácia Vydavateľstva a Tlačiarne
- Použitie návrhového vzoru visitor pre Autor.
- Vytvorenie nite pre autorovo písanie v Autor.java

Commit - 92d987fc3f86b5866c0ee725318bf8edebf522a7 – „*gui try”*

- Úprava triedy Kniha.java na návrhový model Composite. Rozdelenie do tried Kniha, Text a Obálka.
- Úprava funkcií na načítanie kníh zo súboru, aby používali novú úpravu.
- Pridanie triedy Organizovaná sekcia (použitá v predajni)
- Vytvorenie súborov tried pre vydavateľstvo
- Pridanie atribútov triede kniha

Commit - 1cfd4a4f8210c2c375dfc52e72b6c91117de898b – „*predajňa dokončená”*

- Dokončenie funkcií predajcu a zákazníka
- Vstup zákazníka do predajne
- Funkcie na nájdenie referencie na knihy v kníhkupectve.

Commit - b9d82d1537f8754e32b7346ffd769d743c3e5*b22 – „ui overhaul”*

- Spustenie funkcií používateľov pomocou lambda funkcií.
- Implementácia niektorých funkcií predajcu

Commit - b9d82d1537f8754e32b7346ffd769d743c3e5b22 – „*serialization”*

- Implementácia serializácie, ukladanie dát knihkupectva.

Commit - 28362d3be8e9a6641e4257419f574e2214ef5277 – „*UI for skladnik”*

- Vytvorenie druhej verzie prijímania vstupu od používateľa
- Vstup mohli byť funkcie skladnika, zamestnanca alebo pouzivatela

Commit - 8f93454d7817df98b00747359f5421e019e9f382 – „*viacnásobne dedenie”*

- Vytvorenie abstraktnej triedy Zamestnanec pre lepšie rozoznanie Zamestnancov od Zakazníkov
- Umiestňovanie kníh do regálov v sklade.

Commit - 917c70f42e6126a012a89fd2a8040503cdbbd6d8 *–„ ja nechápem“*

- Prvá verzia prijímania vstupu z konzole
- Implementácia niektorých funkcií skladnika
- Jednoduché implementácie funkcií Predajcu a abstraktnej triedy Pouzivatel
- Implementácia ďalších funkcií skladu.

Commit - d44447e21aa4736dcf34876c5cb5dc85ec50ebec – „*zoop projekt”*

- Prvá verzia aplikácie
- Vytvorenie kníhkupectva a užívateľov aplikácie: (Skladník, Predajca, Zákazník)
- Vytvorenie regálov, sekcií a skladu
- Predajňa nie ešte dokončená
- Objednávanie tovaru iba zo súboru