Informatika 2

O predmete

doc. Ing. Ján Janech, PhD.
19. 2. 2024

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Vyučujúci

- Prednáša:
  - doc. Ing. Ján Janech, PhD., A114, jan.janech@fri.uniza.sk
- Cvičí:
  - Ing. Michal Ďuračík, PhD., RA113, michal.duracik@fri.uniza.sk
  - doc. Ing. Ján Janech, PhD., RA114, jan.janech@fri.uniza.sk
  - Ing. Pavol Kysela, Scheidt & Bachmann, kysela.pavol@scheidt-bachmann.sk
  - Ing. Matej Meško, PhD., RA124, matej.mesko@fri.uniza.sk
  - Ing. Milan Straka, PhD., RA216, milan.straka@fri.uniza.sk
  - Ing. Monika Václavková, PhD., RA211, monika.vaclavkova@fri.uniza.sk

Konzultácie

Ing. Michal Ďuračík, PhD.	Streda 12:00 - 13:00 a Štvrtok 8:00 - 9:00
doc. Ing. Ján Janech, PhD.	Pondelok 12:00 – 15:00
Ing. Pavol Kysela	?
Ing. Matej Meško, PhD.	Pondelok 10:00 – 14:00
Ing. Milan Straka, PhD.	Streda 9:00 – 11:00
Ing. Monika Václavková, PhD.	Utorok 10:00 – 12:00

Cieľ predmetu

- rozšírenie vedomostí a skúseností v oblasti programovania
- pokročilé princípy objektového programovania
- využívanie polymorfizmu v objektovom programovaní a algoritmizácii
- programovací jazyk Java – iba nástroj

Hodnotenie predmetu

Položka hodnotenia	Potrebný počet bodov	Max. počet bodov
Hodnotenie práce cez semester	25	50
Hodnotenie praktickou skúškou	25	50
Spolu	**61 (na „E“)**	**100**

Bodové hodnotenie	Hodnotenie známkou
<93, 100>	A – výborne
<85, 93)	B – veľmi dobre
<77, 85)	C – dobre
<69, 77)	D – uspokojivo
<61, 69)	E – dostatočne
<0, 61)	Fx – nevyhovel

Priebežné hodnotenie

Položka hodnotenia	Potrebný počet bodov	Max. počet bodov
Prvý test (~4. týždeň)	0	5
Druhý test (~8. týždeň)	0	5
Tretí test (~12. týždeň)	0	5
Semestrálna práca	**5**	20
Checkpointy	0	3
Domáce úlohy	0	6
Aktivity na cvičení	0	6
Spolu	**25**	**50**

Semestrálna práca

Položka hodnotenia	Očakávaný počet bodov	Max. počet bodov
1. Checkpoint (8. týždeň)	1	1
2. Checkpoint (10. týždeň)	2	2
Odovzdaná a obhájená semestrálna práca	10	20

- Semestrálna práca musí implementovať polymorfizmus
  - Musí byť zrejmý v Checkpointe 1
  - Musí byť implementovaný v Checkpointe 2

Dochádzka

- prednášky – nepovinné
- cvičenia – nepovinné
  - POZOR! mizivá šanca spraviť predmet bez navštevovania cvičení (skúsenosti)
  - viac ako 2 absencie = vylúčenie z cvičenia
  - možnosť získavať body aj bez účasti na cvičeniach

Skúška

- max. 50 bodov
- potrebných aspoň 25 bodov
- praktická
- dostanete zadanie
- 120 minút
- posledný týždeň – príprava

Podvádzanie

- plagiáty semestrálnych prác
- plagiáty domácich úloh
- automatická kontrola na konci semestra

- podvody na skúške

- disciplinárne konanie
  - podmienečné vylúčenie zo štúdia
  - vylúčenie zo štúdia

Informatika 2

Prednáška 1 – Písanie udržiavateľného kódu

doc. Ing. Ján Janech, PhD.
19. 2. 2024

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Cieľ prednášky

- charakteristiky dobrého návrhu tried
  - nízka implementačná závislosť
  - vysoká súdržnosť
  - duplicita kódu – zlá vlastnosť kódu

- refaktoring – zlepšenie kódu
- balíčky
- zaujímavé koncepty v jazyku Java na zlepšenie čitateľnosti

- príklad: knižnica

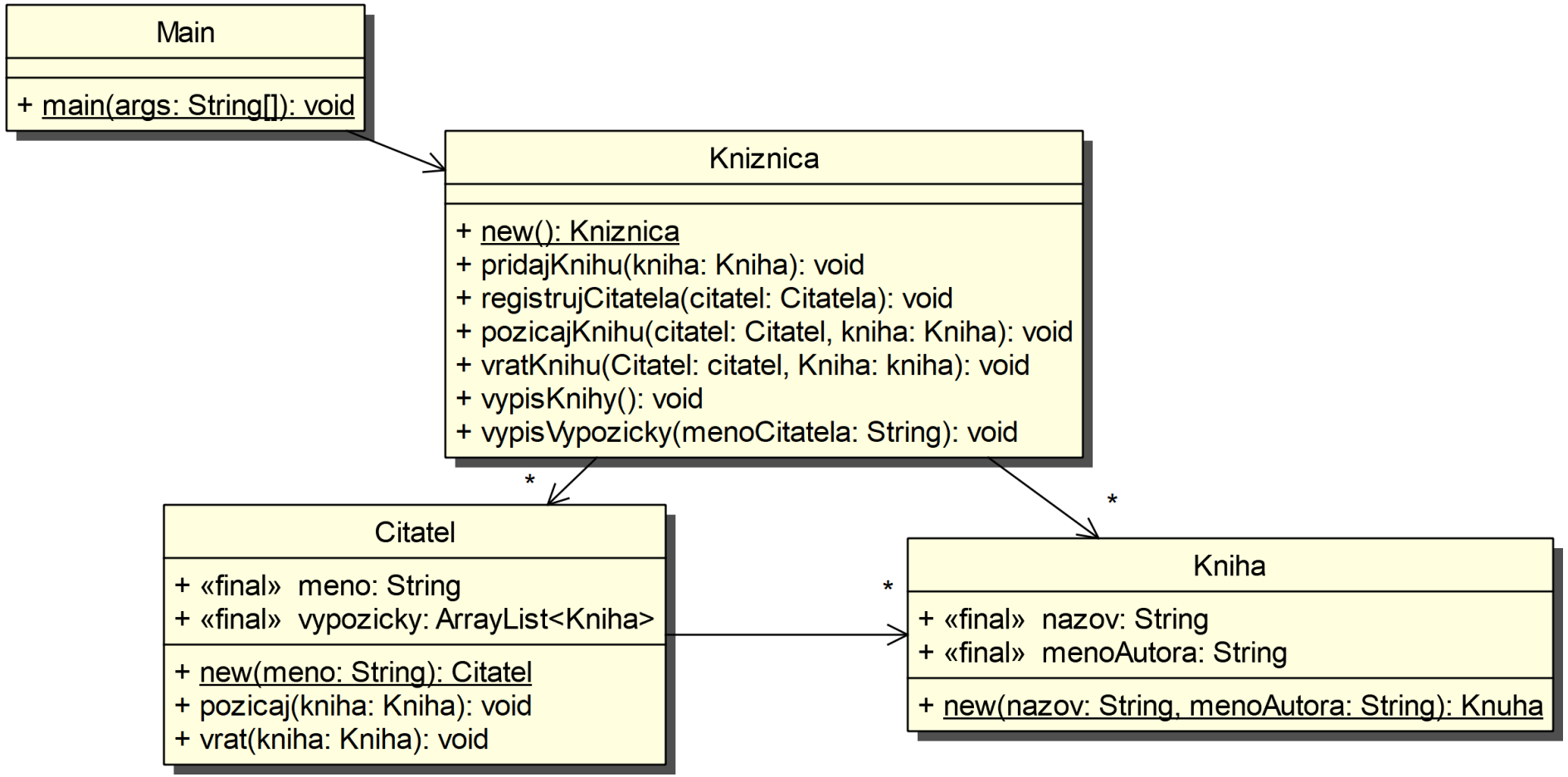
Písanie udržovateľného kódu

- udržovateľný = čitateľný kód
- už sme spomínali
  - konvencie
  - samopopisné identifikátory
  - komentáre v zložitejších miestach algoritmu
  - dokumentačné komentáre
- súdržnosť (cohesion) – max.
- implementačná závislosť (coupling) – min.
- duplicity – min.

Projekt Knižnica

- IS na správu výpožičiek
- Knižnica vedie
  - zoznam kníh
  - zoznam čitateľov
  - zoznam výpožičiek pre jednotlivých čitateľov

UML diagram riešenia



Priamy prístup k atribútom

- priamy prístup hry k východom – atribútom miestnosti

- porušenie zapuzdrenia – základný princíp
- zvyšuje implementačnú závislosť

Implementačná závislosť (1)

- coupling
- úroveň vzájomného prepojenia tried
- zmeny v implementácii jednej triedy si vynútia zmeny v implementácii druhej, závislej triedy
- snažíme sa minimalizovať

Implementačná závislosť (2)

- vonkajší pohľad – rozhranie = „čo objekt robí“
- vnútorný pohľad – implementácia = „ako to robí“

- minimálna závislosť – používa iba dobre navrhnuté rozhranie
- vysoká závislosť – požíva „ako to robí“

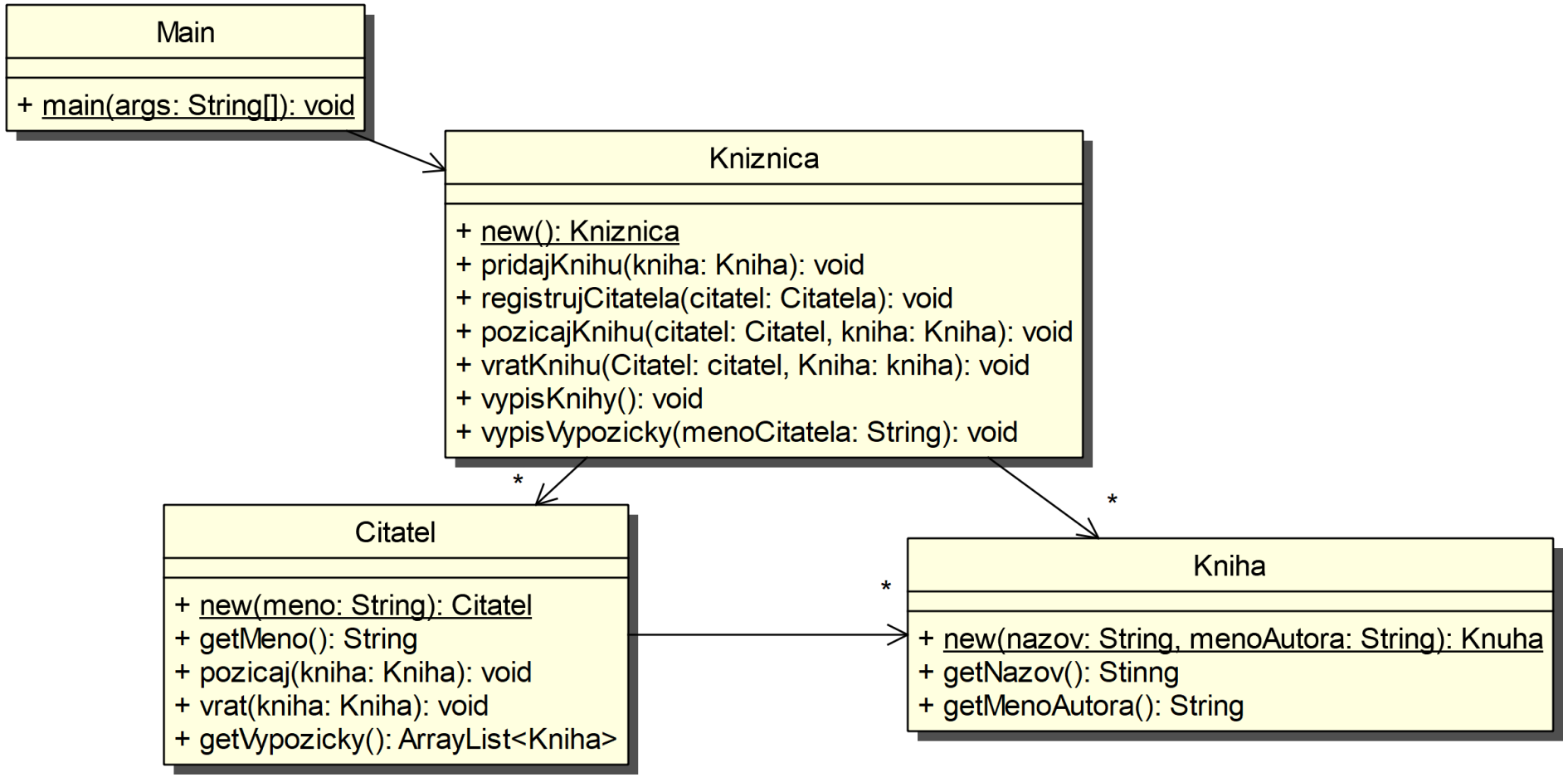
Znaky minimálnej závislosti

- pochopenie triedy bez nutnosti skúmať triedy, na ktorých je závislá
- zmena implementácie jednej triedy nevyžaduje zmeny iných tried

Príklady implementačnej závislosti

- závislosť na implementácii algoritmu
- závislosť na atribútoch (aj prenesene cez get/set)
- závislosť na poradí posielania správ
- porušenie zapuzdrenia
- ...

UML diagram riešenia



Metóda Kniznica.pozicajKnihu (1)

```
public void pozicajKnihu(String menoCitatela, String nazovKnihy) {
    ...
    citatel.pozicaj(kniha);
}
```

Metóda Kniznica.pozicajKnihu (2)

```
    Citatel citatel = null;
    for (Citatel c : this.citatelia) {
        if (c.getMeno().equals(menoCitatela)) {
            citatel = c;
            break;
        }
    }
    if (citatel == null) {
        System.out.println("Čitateľ nenájdený");
        return;
    }
```

Metóda Kniznica.pozicajKnihu (3)

```
    Kniha kniha = null;
    for (Kniha k : this.knihy) {
        if (k.getNazov().equals(nazovKnihy)) {
            kniha = k;
            break;
        }
    }
    if (kniha == null) {
        System.out.println("Kniha nenájdená");
        return;
    }
```

Metóda Kniznica.pozicajKnihu – vyhodnotenie

- metóda toho robí veľa
  - nájdenie čitateľa podľa mena
  - nájdenie knihy podľa názvu
  - požičanie knihy

Súdržnosť kódu

- cohesion
- počet a rôznosť úloh jednej jednotky kódu
- vysoká súdržnosť – jednotka má jedinú logickú úlohu

- metóda – práve jedna presne definovaná úloha – operácia.
- trieda – jedna presne definovaná logická entita

- cieľ – maximalizácia súdržnosti

Vysoká súdržnosť uľahčuje

- pochopenie úloh triedy a metód
- opakované použitie (reuse) triedy alebo metód v iných častiach kódu, v inom softvéri

Aká veľká má byť

- trieda?
- metóda?

- metóda je príliš dlhá, ak vykonáva viac ako jednu úlohu – operáciu
- trieda je príliš zložitá, ak spája viac ako jednu logickú entitu
- rešpektovanie týchto pravidiel necháva ešte stále dostatočný priestor programátorovi

Iná metóda – Kniznica.vratKnihu (1)

```
public void vratKnihu(String menoCitatela, String nazovKnihy) {
    ...
    citatel.vrat(kniha);
}
```

Iná metóda – Kniznica.vratKnihu (2)

```
    Citatel citatel = null;
    for (Citatel c : this.citatelia) {
        if (c.getMeno().equals(menoCitatela)) {
            citatel = c;
            break;
        }
    }
    if (citatel == null) {
        System.out.println("Čitateľ nenájdený");
        return;
    }
```

Iná metóda – Kniznica.vratKnihu (3)

```
    Kniha kniha = null;
    for (Kniha k : this.knihy) {
        if (k.getNazov().equals(nazovKnihy)) {
            kniha = k;
            break;
        }
    }
    if (kniha == null) {
        System.out.println("Kniha nenájdená");
        return;
    }
```

Duplicita kódu (1)

- nežiaduci jav
- pri modifikáciách nutnosť úpravy kódu na viacerých miestach
- zväčšuje pravdepodobnosť vzniku chyby
- znižuje čitateľnosť a zrozumiteľnosť kódu

Duplicita kódu (2)

- indikuje zlý návrh – znižuje súdržnosť
- odstraňovanie duplicít – jedna zo zásad udržovateľnosti kódu
- =>
- refaktoring: samostatná metóda pre duplicitný kód

Refaktoring

- úprava fungujúceho kódu so zachovaním jeho funkčnosti

- nemení „čo objekt robí“
- mení „ako to objekt robí“

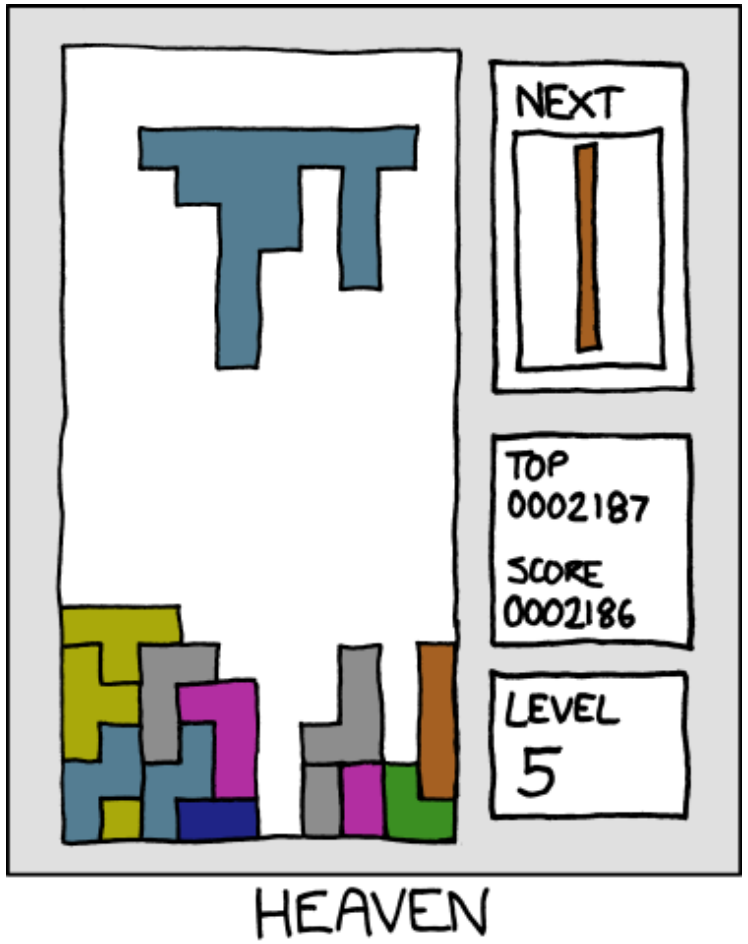
- zlepšenie udržovateľnosti kódu

Refaktoring – poznámky

- !!! nezabudnúť na regresné testovanie
  - vždy spustiť znovu všetky unit testy

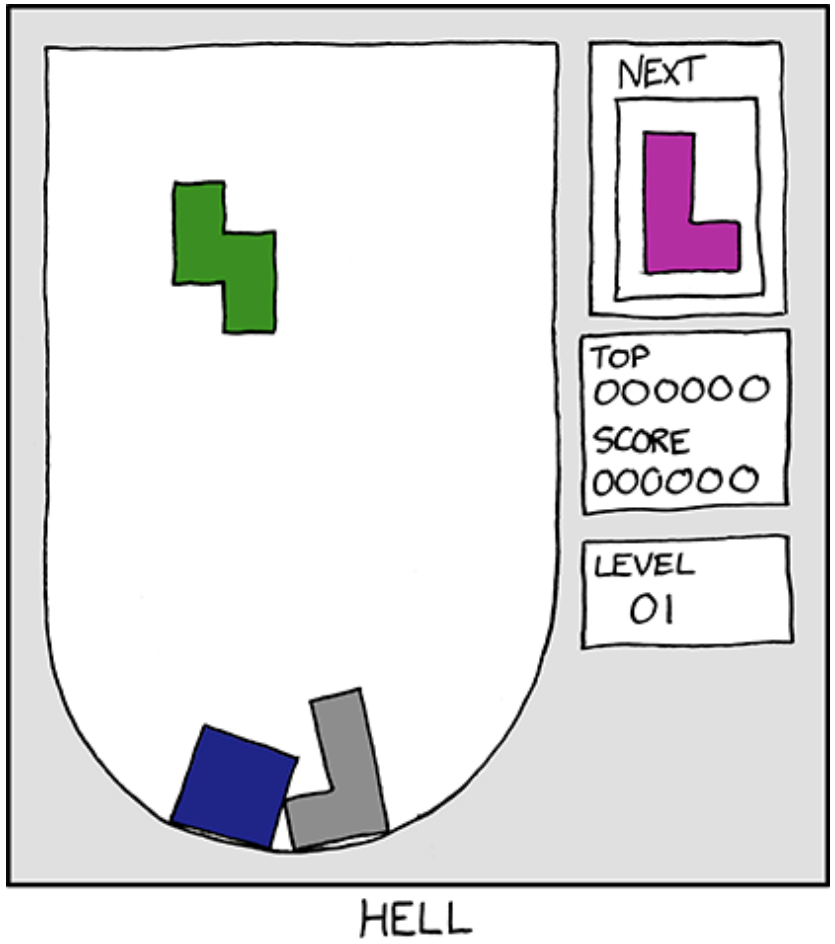
- refaktoring treba robiť často
  - hromadenie technického dlhu

Ako sa úpravy programu tvária



http://xkcd.com/888/

Aké sú v skutočnosti



http://xkcd.com/724/

Nová verzia metódy Kniznica.pozicajKnihu

```
public void pozicajKnihu(String menoCitatela, String nazovKnihy) {
    Citatel citatel = this.najdiCitatela(menoCitatela);
    Kniha kniha = this.najdiKnihu(nazovKnihy);
    citatel.pozicaj(kniha);
}
```

Problém v opravenej metóde

- nie sú ošetrené situácie, keď je zlý názov knihy, alebo meno čitateľa
- preložiť ide
- s nesprávnymi vstupmi padne na behovú chybu

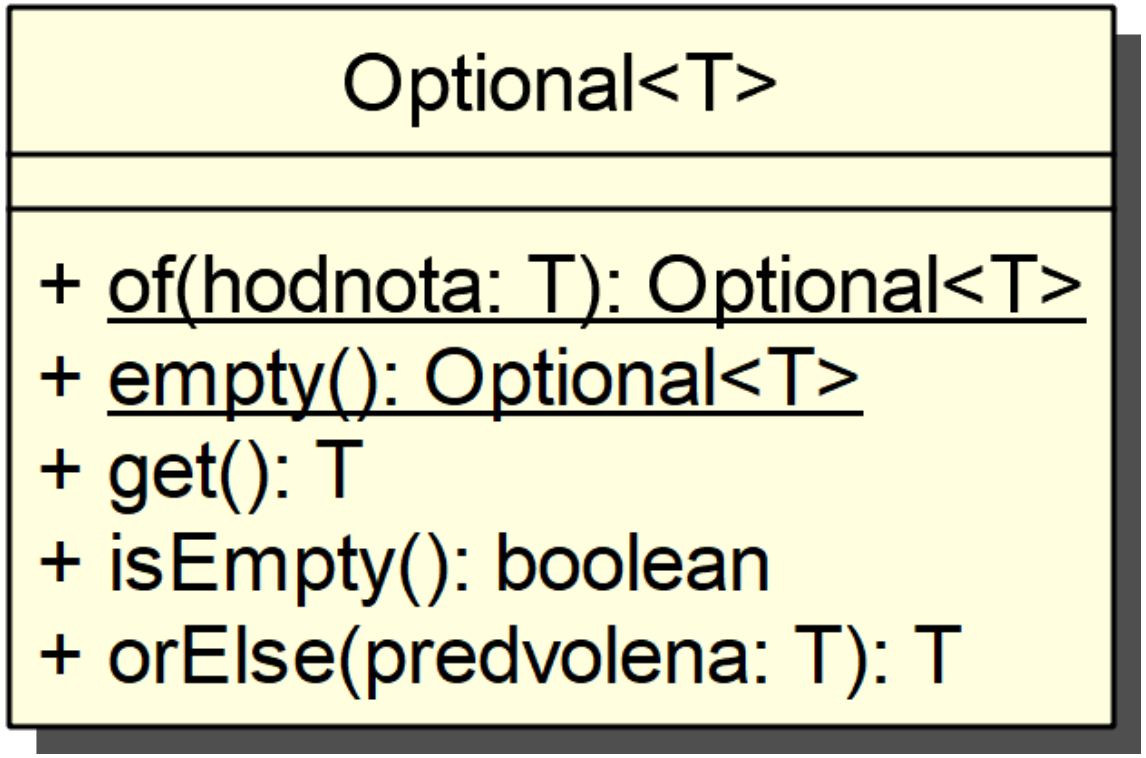
- kiežby takúto situáciu našlo prostredie/prekladač
  - dá sa to – generický typ Optional<T>

Typ Optional<T>

- náhrada za null hodnoty
- explicitné vyjadrenie toho, že hodnota nemusí byť k dispozícii

- potrebný import – java.util.Optional

Optional<T> – rozhranie



- of – vráti inštanciu Optional s hodnotou
- empty – vráti inštnaciu Optional bez hodnoty
- get – vráti hodnotu, alebo spadne na behovú chybu, ak hodnota nie je
- isEmpty – vráti true, ak má Optional hodnotu
- orElse – vráti hodnotu v Optional, alebo hodnotu danú parametrom

Metóda Kniznica.najdiCitatela – bez Optional

```
private Citatel najdiCitatela(String menoCitatela) {
    for (Citatel citatel : this.citatelia) {
        if (citatel.getMeno().equals(menoCitatela)) {
            return citatel;
        }
    }
    return null;
}
```

Metóda Kniznica.najdiCitatela – s Optional

```
private Optional<Citatel> najdiCitatela(String menoCitatela) {
    for (Citatel citatel : this.citatelia) {
        if (citatel.getMeno().equals(menoCitatela)) {
            return Optional.of(citatel);
        }
    }
    return Optional.empty();
}
```

Metóda Kniznica.pozicajKnihu s využitím Optional

```
public void pozicajKnihu(String menoCitatela, String nazovKnihy) {
    Optional<Citatel> citatel = this.najdiCitatela(menoCitatela);
    Optional<Kniha> kniha = this.najdiKnihu(nazovKnihy);
    if (citatel.isEmpty() || kniha.isEmpty()) {
        System.out.println("Nesprávne vstupy");
    } else {
        citatel.get().pozicaj(kniha.get());
    }
}
```

Výhody/nevýhody Optional

- nevýhoda – viac písania
- výhoda – jednoduchšie čítanie
- výhoda – nezabudnete ošetriť null
  - niečo skontroluje prekladač
  - niečo skontroluje prostredie

Drobná pomôcka nie len pre Optional – špeciálny typ var

- použiteľný len ako typ lokálnej premennej
- inštrukcia pre prekladač – odvodí typ premennej sám na základe priradenej hodnoty
- musí byť inicializovaná v rovnakom príkaze

```
var premenna = 0; // bude int
var premenna2 = "ahoj"; // bude String
var premenna3 = new ArrayList<Citatel>();
```

Metóda Kniznica.pozicajKnihu s využitím var

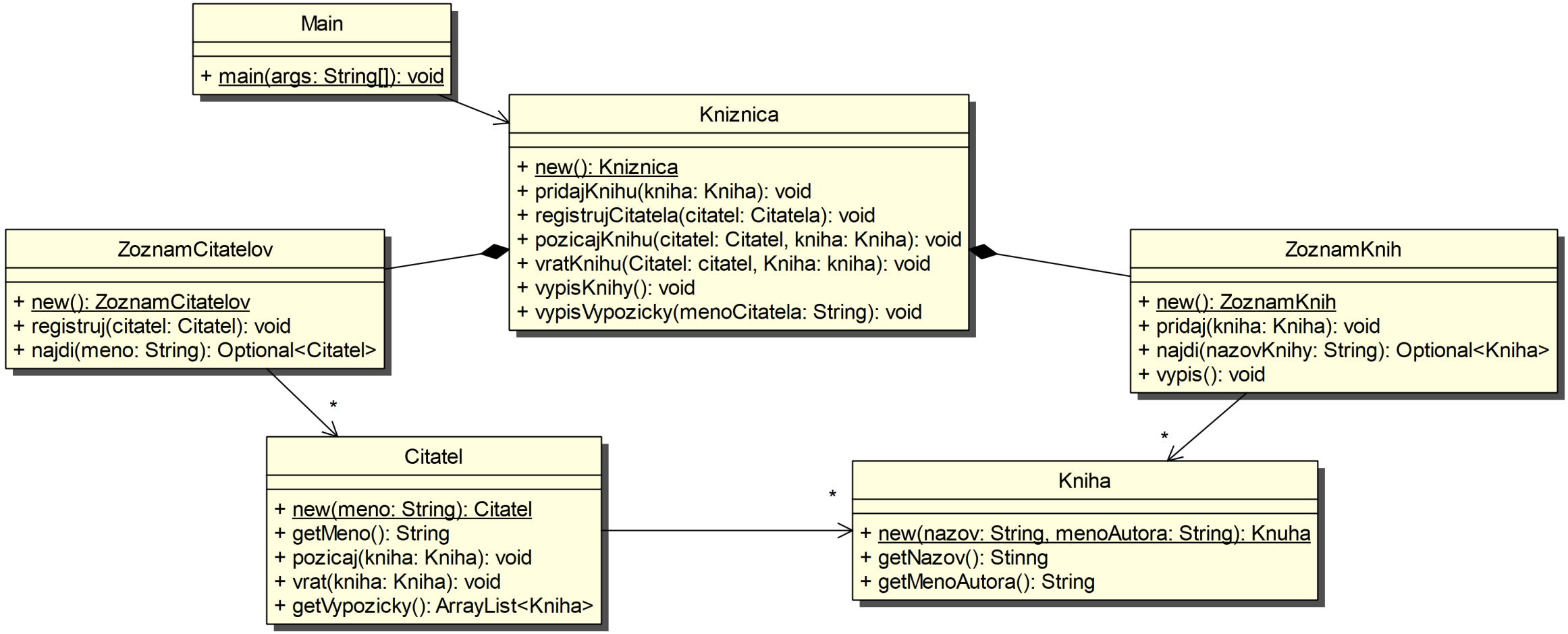
```
public void pozicajKnihu(String menoCitatela, String nazovKnihy) {
    var citatel = this.najdiCitatela(menoCitatela);
    var kniha = this.najdiKnihu(nazovKnihy);
    if (citatel.isEmpty() || kniha.isEmpty()) {
        System.out.println("Nesprávne vstupy");
    } else {
        citatel.get().pozicaj(kniha.get());
    }
}
```

Trieda Kniznica

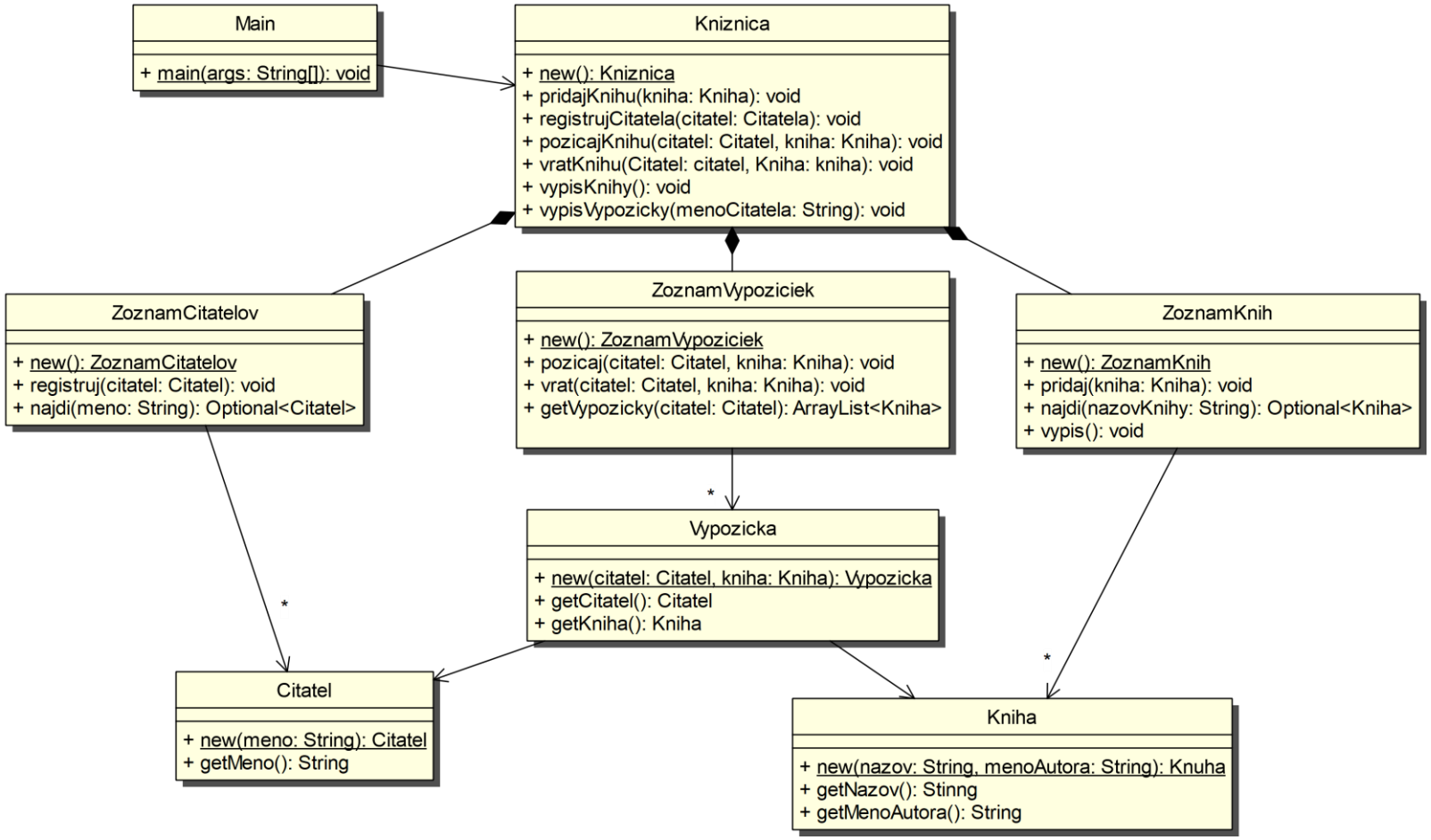
- má veľa zodpovedností
  - ovláda celú knižnicu
  - spravuje zoznam čitateľov
  - spravuje zoznam kníh

- malá súdržnosť => potreba refaktoringu

Zavedenie tried pre ZoznamCitatelov a ZoznamKnih



Oprava triedy Citatel – už nemá výpožičky



Zoznam kníh a čitateľov

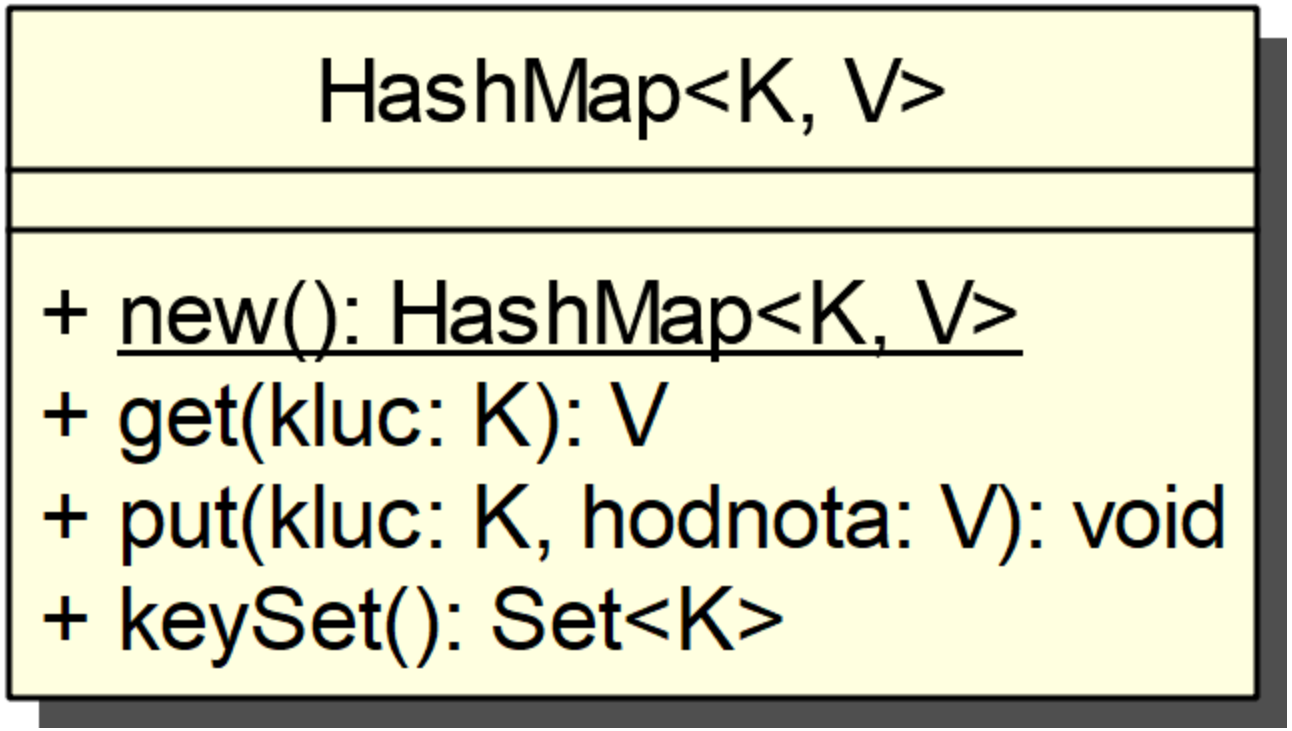
- kontajnery
- treba vyhľadávať
  - ArrayList, pole – komplikované; nutnosť vyhľadávania v cykle

- riešenie kontajner HashMap

Kontajner HashMap (1)

- neusporiadaná množina dvojíc
- dvojica = (kľúč; hodnota)
- možnosť vyhľadávania podľa kľúča
- hodnota kľúča musí byť unikátna
- generická trieda
  - možnosť meniť typ kľúča
  - možnosť meniť typ hodnoty

Kontajner HashMap (2)



Slovník

```
HashMap<String, String> slovnik = new HashMap<String, String>();
slovnik.put("hello", "ahoj");
slovnik.put("house", "dom");
slovnik.put("garden", "záhrada");

// vypise ahoj
System.out.println(slovnik.get("hello"));
```

Nová metóda Vypozicka.getDlzkaVypozicky

```
public String getDlzkaVypozicky() {
    switch (this.pocetDni) {
        case 1:
            return "jeden den";
        case 7:
            return "tyzden";
        case 14:
            return "dva tyzdne";
        case 30:
            return "mesiac";
        default:
            return String.format("%d dni", this.pocetDni);
    }
}
```

Klasický switch v metóde Vypozicka.getDlzkaVypozicky

- tento switch slúži na výpočet hodnoty
- vo všetkých vetvách ma hodnota rovnaký typ
- spracovanie hodnoty (return) je skopírované do každej vetvy

- dá sa nahradiť za rozšírený switch
  - každá vetva je výraz
  - switch je použitý ako výraz
  - nie je potrebný break

Metóda Vypozicka.getDlzkaVypozicky s rozšíreným switchom

```
public String getDlzkaVypozicky() {
    return switch (this.pocetDni) {
        case 1 -> "jeden den";
        case 7 -> "tyzden";
        case 14 -> "dva tyzdne";
        case 30 -> "mesiac";
        default -> String.format("%d dni", this.pocetDni);
    };
}
```

Rozšírený switch s blokom

```
switch (option) {
    case 1 -> {
        System.out.print("Zadaj meno čitateľa: ");
        var menoCitatela = scanner.nextLine();
        kniznica.registrujCitatela(new Citatel(menoCitatela));
        System.out.format("Čitateľ '%s' registrovaný.%n", menoCitatela);
    }
    case 2 -> {
        kniznica.vypisKnihy();
    }
    ...
```

Balíčky

- knižnice – rozširujúca funkčnosť
  - delí sa na balíčky
  - balíčky obsahujú triedy

- podobne:

- program
  - delí sa na balíčky
  - balíčky obsahujú triedy

Balíčky – Java

- môžu byť ľubovoľne vnorené
- názvy vnorených balíčkov sa oddeľujú bodkou

- príklad: java.util

- balíčky vytvárajú menné priestory

Menný priestor

- identifikátory – mená
  - premenná, trieda/enum/interface, metóda, ...
- menný priestor – situácia vyžadujúca jednoznačné identifikátory
  - blok
  - telo triedy
  - balíček
- menný priestor – kontejner identifikátorov s jednoznačným určením významu

Riešenie kolízie identifikátorov

- plne kvalifikovaný názov
- FQN – fully qualified name
- FQN identifikátor
  - postupnosť názvov balíčkov
  - bodková notácia

Využitie iného menného priestoru

- využitie FQN – ak hrozí konflikt

```
java.util.ArrayList<Integer> premenna =
                    new java.util.ArrayList<Integer>();
```

- príkaz import – ak nehrozí konflikt

```
import java.util.ArrayList;

ArrayList<Integer> premenna =
                              new ArrayList<Integer>();
```

Príkaz package

```
package nazov;
```

- úplne prvý príkaz v súbore
  - súbor: trieda, enum, interface
- nad ním iba komentár
- obsahuje názov balíčka
- každý súbor v balíčku !!!
- prostredie IntelliJ IDEA automaticky

Nepomenovaný balíček

- každý projekt (aplikácia)
- prvý balíček – nemá meno
- doteraz len také projekty

Reprezentácia balíčkov

- balíček == adresár (zložka, priečinok)
- podbalíčky == podadresáre
- súbory v balíčku == súbory .java v adresári

Názvy balíčkov

- konvencia: prvé písmeno malé, rovnako ako metódy

- názov balíčka musí byť unikátny !
- odstrašujúci príklad:
  - nový balíček „java“ s jednou triedou
  - prestane fungovať program
  - v starej verzii: prestane fungovať dokonca BlueJ

Unikátnosť názvov balíčkov

- v nepomenovanom balíčku len najvyšší balíček
- najvyšší balíček – vlastné meno, prezývka, názov firmy, webová doména (viacstupňové vnáranie)
- vnorený balíček – názov programu
- do neho vnorené balíčky – vlastné balíčky programu

Príklad názvov balíčkov

- nepomenovaný balíček obsahuje
  - balíček fri (správne sk.uniza.fri)
- balíček fri obsahuje
  - balíček programu kniznica
- balíček hry obsahuje
  - balíčky zaklad, citatelia, knihy, ...

Výsledné FQN tried kniznica

- fri.kniznica.zaklad.Kniznica
- fri.kniznica.knihy.ZoznamKnih
- ...

Kód pred presunom

```
public class Kniznica {

    ...

}
```

Pridanie názvu balíčka

```
package fri.kniznica.zaklad;

public class Kniznica {
    ...
}
```

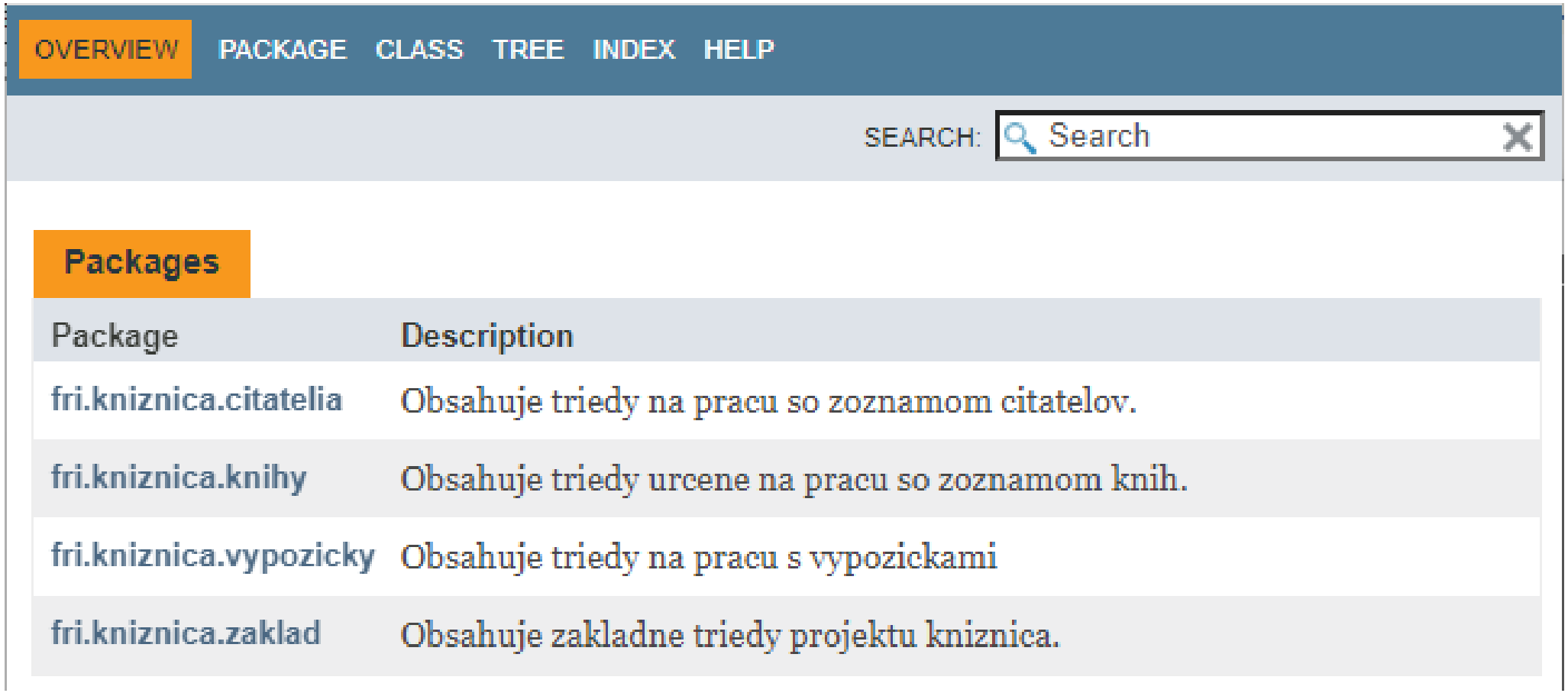
Balíček – dokumentačné komentáre

- dokumentačný komentár pre balíček – význam balíčka ako celku
- nedá sa napísať do súboru s triedou – javadoc nenájde
- špeciálny súbor package-info.java

Súbor package-info.java – príklad

```
/**
 * Obsahuje zakladne triedy projektu kniznica.
 */
package fri.kniznica.zaklad;
```

Javadoc pre balíčky



Zmeny softvéru

- Softvér nie je román
  - román sa napíše len raz
  - Krstný otec, tretie aktualizované vydanie

- úspešný softvér sa stále opravuje, rozširuje.
- neudržovaný softvér „umiera“.

Kvalita kódu

- implementačná závislosť
- súdržnosť

- kvalitný kód
  - neobsahuje duplicitu
  - dodržiava zapuzdrenie