Dve kategórie tried

- triedy možno zadeliť do dvoch kategórií
- **entity** (entities) – zodpovedajú skutočným objektom z aplikačnej domény
  - osoba, zápisný list, kniha, faktúra,...
- **služby** (services) – poskytujú operácie nad entitami
  - **prihlásenie na skúšku**: zoznam termínov, pridanie študenta, výpis študentov na termíne...

Priebežný príklad – online internetový obchod

- Zákazník: „*Chceme si postaviť internetový obchod. Mal by ponúkať knihy, umožňovať nákup, kategorizovať knihy blablablabla*“.
- Zákazník často nemá jasnú predstavu o tom, čo chce. Dôležité je, že to chce čo najlacnejšie a čo najskôr.
- Potrebujeme z neho vydolovať **používateľské požiadavky** (user requirements)
- Informačný systém musí spĺňať všetky požiadavky

Možnosti budovania IS

- rôzne metodológie budovania IS
- jeden „extrém“: vodopádový model (waterfall)
  1. zozbierame všetky požiadavky
  2. analýza
  3. vývoj
  4. implementácia
  5. testovanie
  6. nasadenie
- používaný pre stabilné systémy, kde sa požiadavky nemenia
- v praxi: požiadavky prichádzajú nepretržite
- Často už máme pol IS hotového a príde radikálna požiadavka, z ktorej vyplynie prekopanie.

Možnosti budovania IS

- druhý „extrém“: extreme programming (XP)
  - časté vypúšťanie verzií
  - jednoduchý návrh
  - refactor
  - testovanie
- používaný pre dynamické systémy, kde požiadavky nie sú známe na začiatku
- alebo v prípadoch, kde zákazník de facto neexistuje (rôzne open-source systémy)
- nevýhoda: vyžaduje zbehlých programátorov s disciplínou
  - inak sa systém zvrhne na veľkú guľu blata (Big Ball of Mud)

Priebežný príklad – online internetový obchod

- Zákazník: „*Chceme si postaviť internetový obchod. Mal by ponúkať knihy, umožňovať nákup, kategorizovať knihy blablablabla*“.
- Skúsme si čo najskôr vybudovať funkčný systém.
- Pozrime sa na minimálne požiadavky.
- Identifikujme entity a služby.

Priebežný príklad – online internetový obchod

Entity

- kniha
  - autor
  - názov
  - ISBN
  - počet exemplárov na sklade
  - komentáre používateľov
  - ….

Služby

- BookService
  - pridaj knihu
  - aktualizuj knihu
  - odober knihu
  - daj zoznam všetkých kníh
  - nájdi knihu podľa ID
  - …

Priebežný príklad – online internetový obchod

Služby

- BookService
  - pridaj knihu
  - aktualizuj knihu
  - odober knihu
  - daj zoznam všetkých kníh
  - nájdi knihu podľa ID

- služba pracuje nad entitami
- typický prípad operácií: CRUD
  - C – create – vytvorenie
  - R – read – získanie entity
  - U – update – aktualizácia
  - D – delete – odstránenie
- CRUD sa notoricky opakujú v mnohých IS

Po minimalizácii

Entity

- kniha
  - autor: String
  - názov: String
  - ISBN: String

Služby

- BookService
  - pridaj knihu: add(Book b)
  - daj zoznam všetkých kníh: list(): List<Book>

Priebežný príklad – online internetový obchod

- vyrobíme dve triedy:
  - Book.java
    - atribúty = inštančné premenné
    - vyrobíme gettre a settre
    - nejaký vhodný konštruktor
    - prázdny konštruktor
  - BookService.java
    - knihy evidované v pamäti
    - máme zoznam kníh
    - pridanie pridá knihu do interného zoznamu
    - list() vráti interný zoznam

Entity v Jave – Book.java

- atribúty = inštančné premenné = privátne
- vyrobíme gettre a settre
- nejaký vhodný konštruktor (napr. berúci autora, titul a ISBM)
- prázdny konštruktor
  - ak máme neprázdny konštruktor, musíme prázdny konštruktor explicitne dodať!
- vhodné prekryť metódu toString() – získame zmysluplnú textovú reprezentáciu

```
public String toString() {
    return “autor” + “ “ + titul + “ “ + isbn;
}
```

Code Review

- code review je procesom, kde nadriadený kontroluje kvalitu kódu a vracia pripomienky
- programátor sa vie z nich poučiť
- code review na cvičení:
  - inštančné premenné sú private! (úplné zapúzdrenie)
  - nemusíme ich inicializovať: primitívy sa inicializujú na 0, false, ... Triedy sa inicializujú na null

Code Review

- zoznamy VŽDY nainicializujeme
- ArrayList<String> mená = new ArrayList<String>();
- inak máme mená = null
- a teda máme je to opruz
- lebo prázdnosť zoznamu znamená test
- if(mená != null && mená.isEmpty())...
- v 99% prípadoch sa pýtame, či je zoznam prázdny alebo nie a netestujeme, či vôbec zoznam existuje

Priebežný príklad – online internetový obchod

- tretia trieda: tester
  - má metódu *main()*
  - v nej vytvoríme inštanciu BookService
  - dve inštancie kníh
  - pridáme ich do služby cez add()
  - otestujeme list()
- máme beživý IS
  - síce neinteraktívny, žiadne užívateľské rozhranie, ale bežíme!
  - môžeme ísť na pivo (= release!)

Obchod: verzia 0.2

- náš BookService pracuje nad zoznamom v pamäti
- zoznam neprežije beh aplikácie
- skúsme načítavanie a ukladanie do súboru
- vytvorme FileBookService (všimnime si zmysluplný názov! Žiaden BookService2, žiaden BookServiceNový...)

FileBookService

- FileBookService
  - pridaj knihu: add(Book b)
  - daj zoznam všetkých kníh: list(): List<Book>
- tie isté metódy
- ten istý zoznam
  - Ctrl-C, Ctrl-V programming ;-)
- navyše si dorobme prázdny konštruktor
  - v ňom načítame zoznam zo súboru
- metóda add() spôsobí pridanie knihy do zoznamu a do súboru

Serializácia

- ukladanie objektu do súboru vyriešime cez serializáciu
- Java mechanizmus na konverziu objektov na bajtovú reprezentáciu
- analógia: zaváranie ovocia. Ovocie zavaríme (serializujeme). V zime ho vytiahneme z kompótu (deserializácia) a zjeme

Serializácia

File *file* = new File(„zoznam.txt“);
FileOutputStream fileOut = new *FileOutputStream(file*);
ObjectOutputStream out = new ObjectOutputStream(fileOut);
fileOut.writeObject(books);

File *file* = new File(„zoznam.txt“);
FileInputStream fileOut = new *FileInputStream(file*);
ObjectInputStream out = new ObjectInputStream(fileOut);
Book book = (Book) fileOut.readObject();

Serializácia

- triedy, ktoré chceme serializovať, musia implementovať interfejs java.io.Serializable
- public class Book implements Serializable
- v opačnom prípade máme výnimku NotSerializableException pri pokuse o ukladanie
- toto musia implementovať všetky triedy v objektovom grafe, ktorý ukladáme

FileBookService

- urobíme si ďalší tester
- overíme funkčnosť FileBookService

BookService vs. FileBookService

- aký je rozdiel medzi týmito dvoma triedami?
- oba poskytujú tú istú službu
- ale každá iným spôsobom
  - jedna v pamäti
  - druhá v súbore
- ale používateľa služby nezaujíma, ako sa jeho operácie vykonajú
- zaujíma ho len výsledok
- služba funguje ako čierna skrinka

BookService vs. FileBookService

- môžeme ich schovať za interface!
- interface (čisto abstraktná trieda) predstavuje sadu schopností, ktorú môžeme čakať od inštancie
- schopnosti sú presne operácie, ktoré chceme od služby
  - add() pridaj
  - list() daj zoznam
- vytvorme teda IBookService s týmito metódami

Refactor time!

- naše služby implementujú schopnosti v interfejsi
- public class BookService implements IBookService
- public class FileBookService implements IBookService
- načo je to dobré?

Refactor time!

- ak v celom projekte používame interfejsy a implementačné triedy spomíname len pri vytváraní inštancie, môžeme zmenou jediného riadka zmeniť správanie projektu
- upravme prvý tester
  - BookService bookService = new BookService()

  na
  - IBookService bookService = new BookService()

Interfejsy

- IBookService bookService = new BookService()
- máme projekt bežiaci nad knihami v pamäti
- čo ak chceme zmigrovať na súbor?
- jediná zmena:
- IBookService bookService = new **FileBookService()**
- nič iné meniť nemusíme
- v zvyšku kódu voláme metódy na službe, ktorá je interfejsom
- a nezaujíma nás *ako* sa to spraví
- premenná bookService sa správa polymorfne!

Interfejsy

- IBookService bookService = new BookService();
- bookService.add(new Book(„Meno ruže“));
- bookService.add(new Book(„Meno ruže“));
- System.out.println(bookService.list());

// fungujeme na pamäti

po zmene

- IBookService bookService = new FileBookService();

// zrazu fungujeme nad súborom

Zásada

- zásada
- PROGRAM TO INTERFACES, NOT TO IMPLEMENTATIONS!
- vo všetkých dátových typoch používame interfejsy!
- v parametroch metód používame interfejsy!
- implementácie používame len pri vytváraní inštancií
- projekt je tak možné flexibilne meniť

Iný príklad

- chceme metódu, ktorá nájde najlacnejšiu knihu v zozname kníh
- public Book findCheapestBook(ArrayList<Book> books)
- tu sme ale povedali, že chceme zoznam kníh implementovaný nad poľom?
- čo ak mám knihy v spojovom zozname? (LinkedList)
- musím prekonvertovať spojový zoznam na poľový
- ale prečo prepánajána metódu zaujíma ako je môj zoznam implementovaný, keď potrebuje len dĺžku zoznamu a prístup k *i*-temu prvku?

Iný príklad

- zoznamu zodpovedá interfejs List
- ArrayList implements List
- LinkedList implements List
- metódu zmeňme na
- public Book findCheapestBook(List<Book> books)
- takto môžeme hľadať najlacnejšiu knihu v ľubovoľnom zozname

Iný príklad

- toto môžeme ešte viac vylepšiť
- kolekcia (Collection) je sada prvkov, naša metóda cez ňu prechádza (iteruje)
- zoznam, množina, ... : všetko sú kolekcie
- ArrayList implements Collection
- LinkedList implements Collection
- public Book findCheapestBook(Collection<Book> books)
- takto máme metódu, ktorá dokáže hľadať v ľubovoľnej sade prvkov

V praxi

- ak máme inštančnú premennú zoznam, tak píšeme LEN
- private List<Book> books = new ArrayList<Book>();
- toto je „zle“:
- private ArrayList<Book> books = new ArrayList<Book>();

Nabudúce

- unit testy!
- Spring!