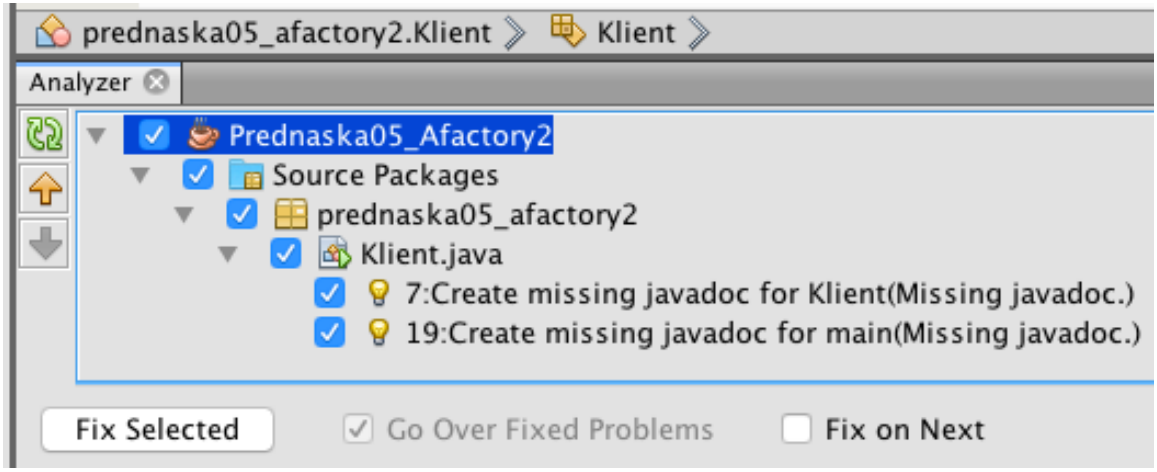
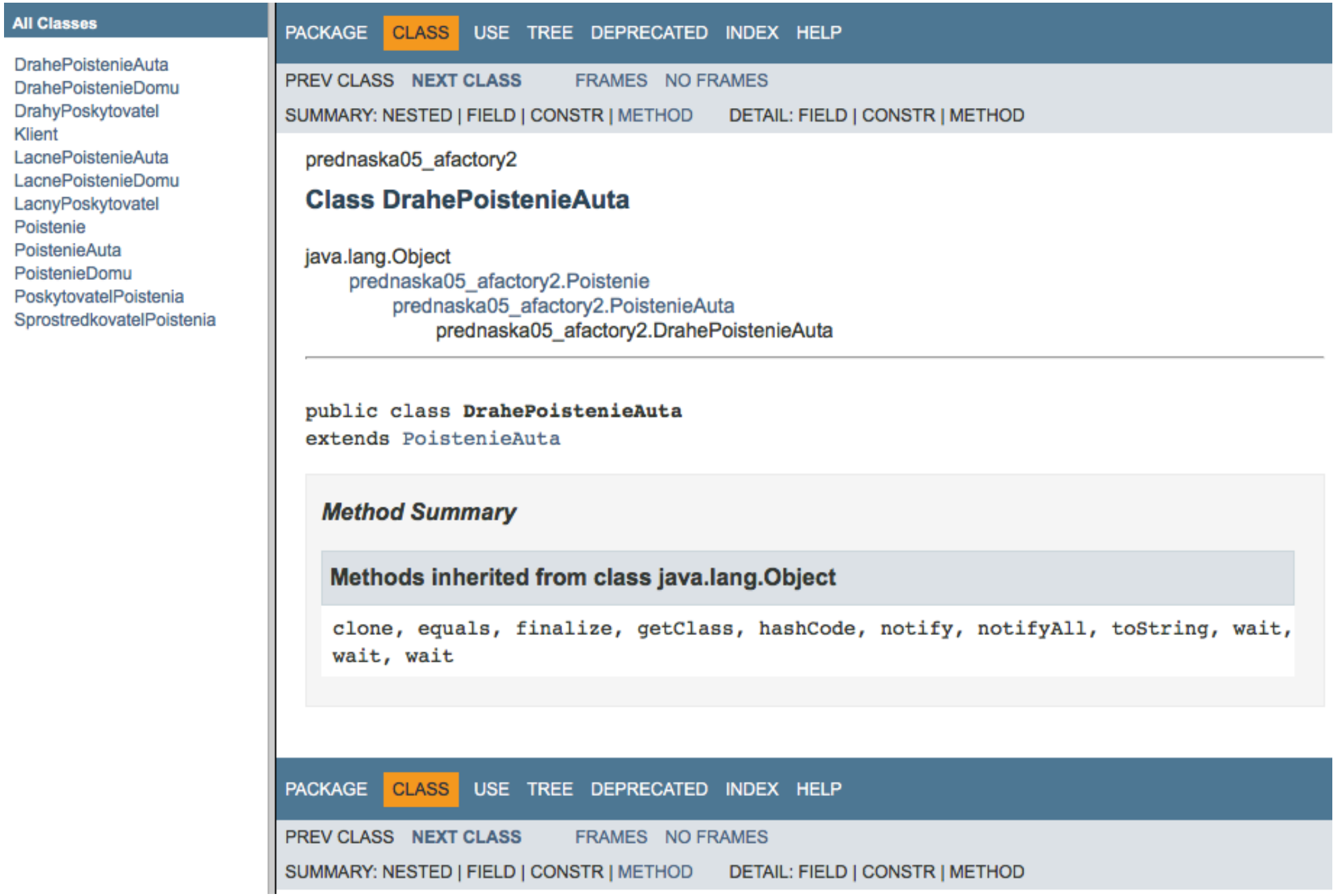
SW vývoj – dokumentácia kódu

- „Dobrý programátor píše programy, ktorým rozumie nielen počítač ale aj človek“
  - ak budete postupovať podľa myšlienky „poriadok je pre blbcov, inteligent zvláda chaos“ – skôr, či neskôr na to doplatíte
  - v programe sa potrebujete vyznať nielen teraz, ale aj o rok VY SAMI !
- Druhy komentárov
  - Riadkový komentár //
  - Obecný komentár /* tu môže byť viac riadkov textu ... */
  - Dokumentačný komentár
    - určený pre program „javadoc“, ktorý je súčasťou JDK
    - otvára sa postupnosťou znakov /**
    - Používaju sa rôzne doplnkové značky: @param, @return, ...
  - Netbeans pomáha pomocou ďalších nástrojov:
    - Tools -> Analyse Javadoc
    - Run -> Generate Javadoc
  - alebo pomocou commandline nastroja „javadoc“ (napr. ak zlyhá netbeans)
    - napr. „javadoc -d myjavadoc src/docudemo/DocuDemo.java“



Javadoc – demo vygenerovaná dokumentácia (bez pridaných značiek)



All Classes

DrahePoistenieAuta
DrahePoistenieDomu
DrahyPoskytovatel
Klient
LacnePoistenieAuta
LacnePoistenieDomu
LacnyPoskytovatel
Poistenie
PoistenieAuta
PoistenieDomu
PoskytovatelPoistenia
SprostredkovatelPoistenia

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

prednaska05_afactory2

Class DrahePoistenieAuta

java.lang.Object
  prednaska05_afactory2.Poistenie
    prednaska05_afactory2.PoistenieAuta
      prednaska05_afactory2.DrahePoistenieAuta

```
public class DrahePoistenieAuta
extends PoistenieAuta
```

Method Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

JavaDoc značky

- @author – povinné pre triedy a rozhrania, za značkou sa zapisuje meno autora, autorov sa môže napísať viac, alebo pre každého sa použije nový riadok s novou značkou. Pole author sa negeneruje do dokumentácie, je viditeľné len v zdrojovom kóde.
- @version – povinné pre triedy a rozhrania, za značkou sa zapisuje číslo verzie, pre formát nie je žiadny predpis
- @param – pre metódy a konštruktory – uvádza sa presný názov parametru z hlavičky a zaň potom popis, pre každý parameter sa pridáva nový riadok.
- @return – pre metódy ktoré vracajú nejakú hodnotu, za značkou sa uvádza popis návratovej hodnoty
- @exception, resp throws – popisuje dôvody, prečo metóda vyhadzuje výnimku
- @see – ukazuje niekam (na iný balík, triedu, jej atribút, metódu)
- @since – popisuje sa od akej verzie je trieda, metóda dostupná
- @deprecated – zdôvodňuje od kedy a prečo je trieda, metóda nepodporovaná

Príklady:

```java
/**
 * Toto je nová pekná metóda, účelovo vytvorena len pre dokumenačné účely
 * @param pp parameter 1 - dummy
 * @param ff parameter 2 - dummy
 * @return vždy vracia null, lebo je to len demo :-)
 */
public PoskytovatelPoistenia getFactory2(String pp, String ff) {
    return null;
}
```

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`PoskytovatelPoistenia`	`getFactory(java.lang.String pp)` Generovanie "továrne" poskytovateľom poistenia
`PoskytovatelPoistenia`	`getFactory2(java.lang.String pp, java.lang.String ff)` Toto je nová pekná metóda, účelovo vytvorena len pre dokumenačné účely

getFactory2

```
public PoskytovatelPoistenia getFactory2(java.lang.String pp,
                                         java.lang.String ff)
```

Toto je nová pekná metóda, účelovo vytvorena len pre dokumenačné účely

Parameters:

`pp - parameter 1 - dummy`

`ff - parameter 2 - dummy`

Returns:

`vždy vracia null, lebo je to len demo :-)`

Javadoc a @see

Zdrojový kód:

```
* @see LacnePoistenieAuta
* @see LacnePoistenieAuta#druhPoistenia
```

Dokumentácia:

See Also:

`LacnePoistenieAuta, Poistenie.druhPoistenia`

Kam môže see ukazovať:

```
@see #field
@see #Constructor(Type, Type...)
@see #Constructor(Type id, Type id...)
@see #method(Type, Type,...)
@see #method(Type id, Type, id...)
@see Class
@see Class#field
@see Class#Constructor(Type, Type...)
@see Class#Constructor(Type id, Type id)
@see Class#method(Type, Type,...)
@see Class#method(Type id, Type id,...)
@see package.Class
@see package.Class#field
@see package.Class#Constructor(Type, Type...)
@see package.Class#Constructor(Type id, Type id)
@see package.Class#method(Type, Type,...)
@see package
```

Unit testy a TDD

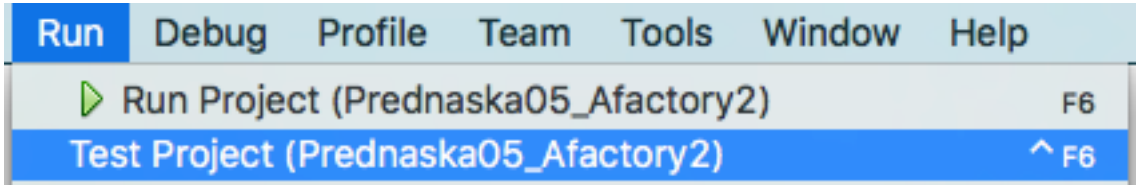
- Aký je mechanizmus vývoja a testovania pri väčších projektoch:
  - Na začiatku máme zoznam požiadaviek
  - Na základe požiadaviek nezávisle:
    - Prebehne návrh a vývoj SW
    - Prebehne definovanie Testovacích scenárov
- TDD- Test Driven Development = najprv definovať sadu testov a až potom písať testovaný program
  - V priebehu písania testov si programátor ujasní, čo vlastne program má robiť, aký má byť robustný, …
  - V priebehu vývoja si priebežne môže predpripravenou sadou testov kontrolovať koľko ich úspešne zbehlo a koľko práce mu ešte zostáva
  - V okamihu, keď úspešne zbehnú všetky testy je vývojár s prácou hotový a nemusí premýšľať či v jeho programe nie sú chyby
- Môžeme vytvárať špeciálne testovacie triedy, ktoré budú testovať naše riadne triedy, ideálne v pomere 1:1
  - Zaujímavé je percento kódu, ktoré je pokryté unit testami (test coverage)
  - Tieto testovacie triedy sa dávajú do separátnych adresárov
- Najpoužívanejšie knižnice pre Javu sú JUnit a TestNG.
  - My budeme používať junit: t.j. “import org.junit”

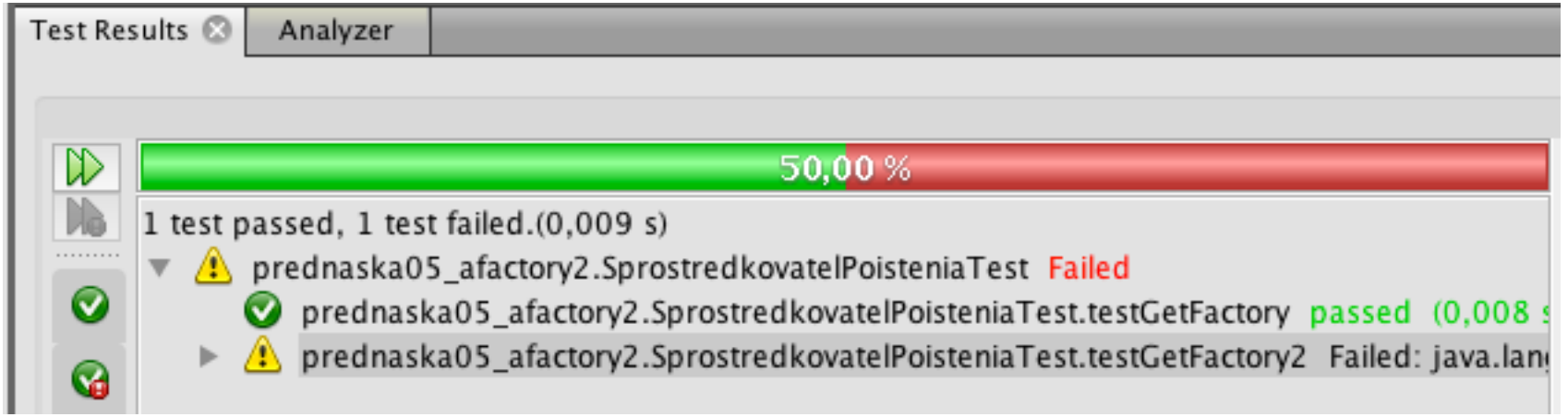
Knižnica junit – popis

- Predpokladá že testovacie metódy môžu byť volané v ľubovoľnom poradí
- Anotácie
  - @Test – označuje metódu, ktorá je testovaná
  - @Test (expected = Exception.class) – test zlyhá ak metóda nehodí očakávanú výnimku
  - @Test(timeout=100) – test zlyhá ak metóda trvá dlhšie ako 100ms
  - @Before metoda() / @After metoda() – metóda sa vykoná pred, resp. po KAŽDOM teste
  - @BeforeClass static metoda() / @AfterClass static metoda() – metóda sa vykoná pred štartom všetkých testov, resp. po konci všetkých testov
- Assert metódy
  - assertEquals() – porovnáva 2 objekty, či sú rovnaké
  - assertTrue() / assertFalse()
  - assertNull() / assertNotNull()
  - assertSame() / assertNotSame()
  - assertArrayEquals()
- Fail metóda
  - Spôsobí zlyhanie testu, vkladá sa na miesta kódu, kde sa program nesmie dostať.

Netbeans a Junit

- Umožňuje vytvárať testovacie triedy „trieda.java“->(klik myšou)->Tools->Create/update tests
- Nasleduje úprava vygenerovaných vzorových testovacích metód [príklad ...]...
- Následuje spustenie ...





Testovanie pomocou dvojníkov objektov pri TDD a unit testoch

- Ak testujeme nejaký modul (napr. triedu), označujeme ho SUT (System Under Test)
- Aby testovanie SUT mohlo prebehnúť úspešne, používajú sa v jeho okolí rôzne Druhy dvojníkov (definície velmi rôznorodé v literatúre)
  - **Dummy** – poskytujú referencie na Dummy objekty aby program mohol bežať dalej, ale objekty samotné sa nepoužívajú
  - **Fake** – objekty majú fungujúcu implementáciu, ale obsahujú kadejaké skratky, kvôli ktorým nie sú vhodné do produkcie (napr. pamäťová DB namiesto reálnej s perzistenciou)
  - **Stub** – poskytujú odpovede na volania počas testu, poskytnú nejaký preddefinovaný želaný vstup. Môžu uchovávať informácie z volaní a využívať ich. T.j. mať STAV.
  - **Mock** – predpogramované objekty, vedia ošetriť volania ktoré očakávajú, dá sa testovať kompletná interakcia za kontrolovaných podmienok. Simulujú SPRÁVANIE objektu, napr. po zavolaní metódy X, samotný mock má zavolať inú metódu inde ..., alebo testujú či ich metódy boli volané v očakávanom poradí, atď ... Používajú sa rôzne frameworky (EasyMock, jMock, Mockito, ...)

Poznámky:

- Zjednodušene Mock= inteligentnejší Stub, vie overiť či bol volaný vtedy keď mal byť volaný, resp. Mosk spúšťa ďalšie aktivity, ...
- Stuby nemôžu spraviť „fail“, mocky áno.

Scenár testovania so Stubmi a Mockmi

Stuby	**Mocky**
Setup – nastav objekt (SUT)	Setup – nastav objekt (SUT)
Nastav okolité stuby.	**Nastav očakávania** (predkonfiguruj okolité mockované objekty)
Vykonaj funkcionalitu	Vykonaj funkcionalitu
	Skontroluj očakávania – boli volané správne metódy na mockovaných objektoch?
Over výsledný stav na objekte (na SUTe)	Over výsledný stav na objekte (na SUTe)
Uvoľni testovacie zdroje	Uvoľni testovacie zdroje

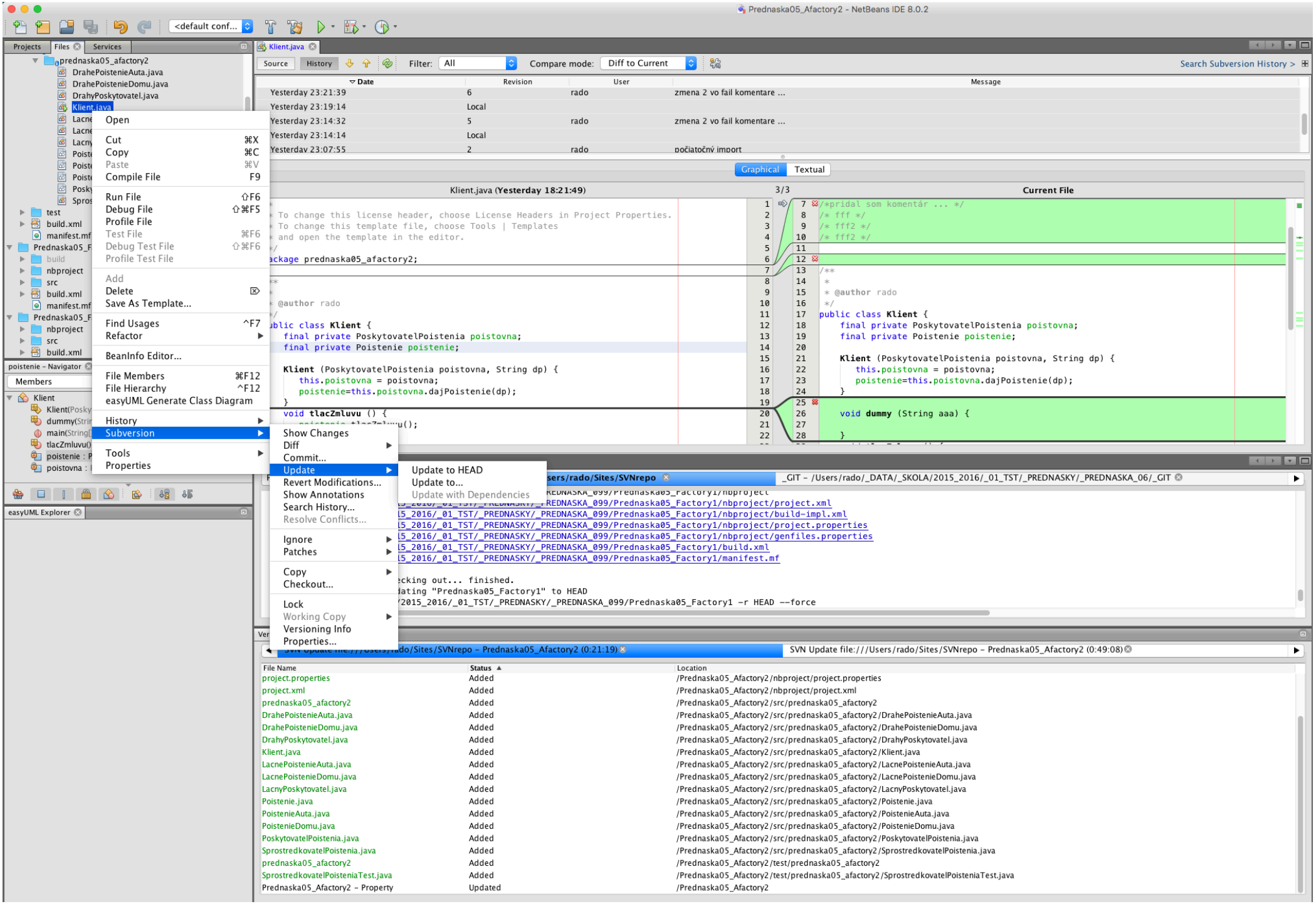
Systémy na kontrolu verzií

- Systémy na kontrolu verzií (VCS = version control system) spravujú súbory, adresáre, ktoré sa v čase menia, typycky umožňujú
  - prechod na predchádzajúce verzie
  - sledovať históriu zmien
  - je to taký „stroj času“ ...
- jadrom systému je tzv. repository (repozitár) – úložisko všetkých verzií
- užívateľ má k dispozícii „pracovnú verziu“
- VCS riešia sieťový prístup mnohými užívateľmi
  - problém zdieľania súborov a paralelného prístupu
  - **riešenie vzorom: zamknutie-modifikácia-odomknutie (POUŽÍVA SA ZRIEDKAVO)**
    - problémy,
      - keď užívateľ zabudne súbor odomknúť
      - nechcená serializácia (každý rieši inú časť v súbore – zbytočné zamykanie)
      - dealock (UserX edituje FileA a UserY edituje FileB, UserX ale zrazu potrebuje zeditovať aj FileB, aby zodpovedalo zmenám vo FileA. Nemôže – FileB je zamknuté)
    - príliš reštriktívne ....
  - **riešenie vzorom (kopíruj-modifikuj-spoj) (copy-modify-merge)** (**POUŽÍVA SA ČASTO**)
    - každý si robí vlastnú kópiu
    - keď niekto chce prepísať v úložisku novšiu verziu verziou, ktorá je odvodená od staršej, tak je vyzvaný na spojenie verzií (merge).

Poznámka: vzor zamknutie.modifikácia-odomknutie sa používa skôr pri binárnych (obraz, zvuk, ...) a nie textových súboroch (tam je ťažké spraviť merge ...).

VCS

- Existuje množstvo voľne šíriteľných aj komerčných VCS ... (git, cvs, svn, ...)
- My budeme používať SVN („subversion“)
  - Svn klient komunikuje zmeny – potvrdzuje (commituje) pre ľubovoľnýu počet súborov a adresárov ako jednu atomickú transakciu.
  - Zakaždým keď repozitár akceptuje commit vytvára nový stav, revíziu – jedinečné celé číslo, zakaždým o 1 vačsie. (úplne prvá je verzia 0, ktorá bola prázdna), pričom:
    - HEAD = najnovšia revízia v repozitári
    - BASE = revízia položky v pracovnej kópii
- Prístup k repozitáru cez URL
  - Buď na lokálnom disku file:///
  - cez WEBDAV: http://..., https://
  - cez vlastný protokol: svn://, resp. svn+ssh:// ... (cez SSH tunel)
- Pracovná kópia užívateľa môže byť v 4 stavoch (kombinácie nižšieuvedených):
  - Lokálne nezmenená/zmenená
  - Aktuálna/neaktuálna (niekto medzitým nezmenil/zmenil repozitár)
- Akcie
  - získanie (vytvorenie) pracovnej kópie z repozitára kópie (CHECKOUT)
  - aktualizácia pracovnej kópie z repozitára (UPDATE)
  - publikovanie pracovnej kópie (COMMIT)
  - vypublikovanie (vytvorenie) prvej verzie v repozitári (IMPORT)
  - vylistovanie repozitára (LIST)


Prednaska05_Afactory2 - NetBeans IDE 8.0.2
Projects
Files
Services
prednaska05_afactory2
DrahePoistenieAuta.java
DrahePoistenieDomu.java
DrahyPoskytovatel.java
Klient.java
Open
Cut ⌘X
Copy ⌘C
Paste ⌘V
Compile File F9
Run File ⇧F6
Debug File ⇧⌘F5
Profile File
Test File ⌘F6
Debug Test File ⇧⌘F6
Profile Test File
Add
Delete
Save As Template...
Find Usages ^F7
Refactor
BeanInfo Editor...
File Members ⌘F12
File Hierarchy ^F12
easyUML Generate Class Diagram
History
Subversion
Tools
Properties
Show Changes
Diff
Commit...
Update
Revert Modifications...
Show Annotations
Search History...
Resolve Conflicts...
Ignore
Patches
Copy
Checkout...
Lock
Working Copy
Versioning Info
Properties...
Update to HEAD
Update to...
Update with Dependencies
Klient.java
Source
History
Filter: All
Compare mode: Diff to Current
Search Subversion History >
Date
Revision
User
Message
Yesterday 23:21:39 6 rado zmena 2 vo fail komentare ...
Yesterday 23:19:14 Local
Yesterday 23:14:32 5 rado zmena 2 vo fail komentare ...
Yesterday 23:14:14 Local
Yesterday 23:07:55 2 rado počiatočný import
Graphical
Textual
Klient.java (Yesterday 18:21:49)
3/3
Current File
/*pridal som komentár ... */
/* fff */
/* fff2 */
/* fff2 */
public class Klient {
final private PoskytovatelPoistenia poistovna;
final private Poistenie poistenie;
Klient (PoskytovatelPoistenia poistovna, String dp) {
this.poistovna = poistovna;
poistenie=this.poistovna.dajPoistenie(dp);
}
void dummy (String aaa) {
}
poistenie - Navigator
Members
Klient
easyUML Explorer
_GIT - /Users/rado/_DATA/_SKOLA/2015_2016/_01_TST/_PREDNASKY/_PREDNASKA_06/_GIT
SVN Update file:///Users/rado/Sites/SVNrepo - Prednaska05_Afactory2 (0:49:08)
File Name
Status
Location
project.properties Added /Prednaska05_Afactory2/nbproject/project.properties
project.xml Added /Prednaska05_Afactory2/nbproject/project.xml
prednaska05_afactory2 Added /Prednaska05_Afactory2/src/prednaska05_afactory2
DrahePoistenieAuta.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/DrahePoistenieAuta.java
DrahePoistenieDomu.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/DrahePoistenieDomu.java
DrahyPoskytovatel.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/DrahyPoskytovatel.java
Klient.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/Klient.java
LacnePoistenieAuta.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/LacnePoistenieAuta.java
LacnePoistenieDomu.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/LacnePoistenieDomu.java
LacnyPoskytovatel.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/LacnyPoskytovatel.java
Poistenie.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/Poistenie.java
PoistenieAuta.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/PoistenieAuta.java
PoistenieDomu.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/PoistenieDomu.java
PoskytovatelPoistenia.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/PoskytovatelPoistenia.java
SprostredkovatelPoistenia.java Added /Prednaska05_Afactory2/src/prednaska05_afactory2/SprostredkovatelPoistenia.java
prednaska05_afactory2 Added /Prednaska05_Afactory2/test/prednaska05_afactory2
SprostredkovatelPoisteniaTest.java Added /Prednaska05_Afactory2/test/prednaska05_afactory2/SprostredkovatelPoisteniaTest.java
Prednaska05_Afactory2 - Property Updated /Prednaska05_Afactory2

VCS - SVN rozhranie príkazového riadka

man svn ...

- add
- blame (praise, annotate, ann)
- cat
- changelist (cl)
- checkout (co)
- cleanup
- commit (ci)
- copy (cp)
- delete (del, remove, rm)
- diff (di)
- export
- help (?, h)
- import
- info
- list (ls)
- lock
- log
- merge
- mergeinfo
- mkdir
- move (mv, rename, ren)
- patch
- propdel (pdel, pd)
- propedit (pedit, pe)
- propget (pget, pg)
- proplist (plist, pl)
- propset (pset, ps)
- relocate
- resolve
- resolved
- revert
- status (stat, st)
- switch (sw)
- unlock
- update (up)
- upgrade

Napr:

svn ls file:///Users/rado/Sites/SVNrepo/Prednaska05_Afactory2/src/prednaska05_afactory2/

svn ls file:///Users/rado/Sites/SVNrepo/

Databázové systémy

- Databáza (báza dát) (DB) je množina štruktúrovaných údajov
- Systém riadenia bázy dát (SRBD) je systém, ktorý umožňuje vytvorenie, napĺňanie, údržbu a používanie databázy
- Databázový systém (DBS) = DB + SRDB, anglicky DBMS (DataBase Management System)
- Delenie SRDB
    - na základe údajového modelu
        - relačné (RDBMS)
        - hierarchické
        - objektovo orientované (objektová, objektovo-relačná)
        - XML
        - No SQL, ...
    - na základe počtu užívateľov
        - jednoužívateľské / viacužívateľské
    - na základe distribuovanosti
        - centralizované /distribuované
    - na zaklade perzistentnosti
        - perzistentné / pamäťové
    - na základe typu údajov
        - tradičné (textové/numerické) / multimediálne / geografické (priestorové)
    - datové sklady, realtime databázy , ...
    - ...

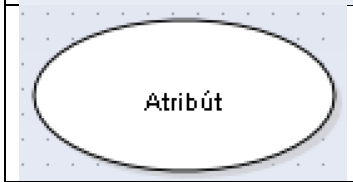
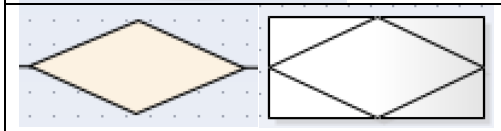
Databázové systémy

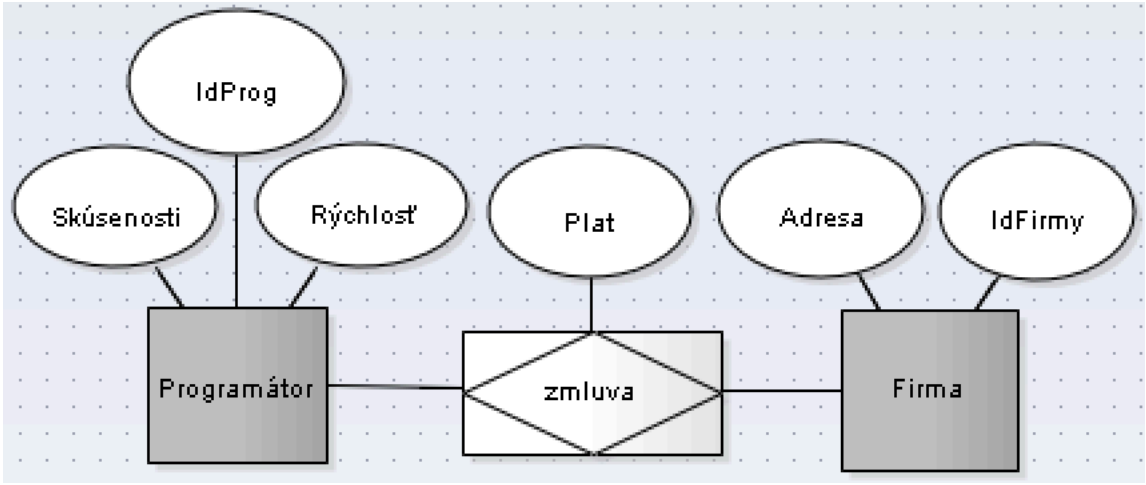
- SRBD zahŕňa prostriedky:
  - pre popis údajov – označujú sa ako **jazyk typu DDL** (Data Definition Language)
  - pre popis algoritmov – označujú sa ako **jazyk typu DML** (Data Manipulation Language)
- ANSI-SPARC trojúrovňová architektúra SRDB:
  - **Externá úroveň** reprezentuje údaje z pohľadu užívateľa. Nazýva sa aj **externá schéma**.
    - Rôzni užívateľa môžu vidieť rôzne časti údajov
  - **Koncepčná úroveň**, nazýva sa aj **logická schéma**
    - reprezentuje sémantickú úroveň (význam dát)
    - rieši bezpečnosť
    - rieši integritu (celistvosť) = sémantická korektnosť údajov, hodnoty údajov sú správne, konzistentné a aktuálne
    - atribúty, vzťahy, podmienky
  - **Interná úroveň**, nazýva sa aj **fyzická schéma**
    - fyzická reprezentácia údajov
    - optimalizuje výkon a spotrebu zdrojov
- Princíp dátovej nezávislosti
  - logická dátová nezávislosť: možnosť meniť logickú schému bez dopadu na externú schému
  - fyzická dátová nezávislosť: možnosť meniť fyzickú schému bez dopadu na logickú schému
- **My sa budeme ďalej venovať relačným databázam a jazyku SQL (zahŕňa DDL aj DML časť).**
  - Existuje veľmi veľa implementácií RDBMS ... budeme používať MySQL

Relačná databáza

Entitno-relačný (ER) diagram (Chenova notácia) – ERD model. Vzťahy:

- entita: objekt v reálnom svete (napr. „študent“, „učiteľ“) o ktorom budeme uchovávať informácie
- atribút: vlastnosť entity
  - doména atribútu – množina možných hodnôt atribútu
- relácia: vzťah medzi entitami

Grafický symbol	Význam
	Entita (obdĺžnik)
	Atribút (elipsa)
	Vzťah medzi entitami (kosoštvorec)
	Prepojenie medzi symbolmi (čiara)
0, 1, N, M, *	Násobnosť výskytu vo vzťahu (**kardinalita vzťahu** 1:N, M:N)



Relačná databáza - pojmy

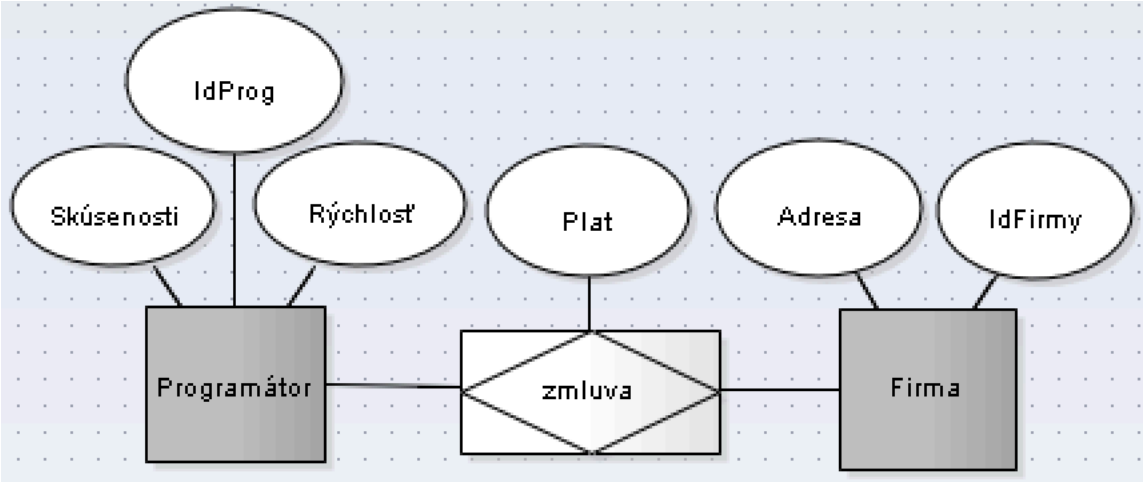
- **Relácia** stupňa n (n-árna relácia) s **kardinalitou relácie** k je množina k n-tíc, ktorá je podmnožinou karteziánskeho súčinu $R \subseteq D_1 \times D_2 \ldots \times D_n$, kde D_i- doména (množina hodnôt) atribútu i. Dá sa predstaviť ako tabuľka s k riadkami a n stĺpcami (stĺpce sú jednotlivé atribúty)
  - Pozn: Karteziánsky súčin množín A a B, je množina všetkých usporiadaných dvojíc (a, b), kde $a \in A$ a $b \in B$, t.j. platí $A \times B = \{(a, b); a \in A \ \wedge b \in B\}$
- **relačná schéma** je zápis relácie v tvare: Entita (atribút1, atribút2, .., atribút n)
  - napr: študent(osobné číslo, meno, priezvisko, výška, váha)
- n-tica = riadok = záznam = inštancia ; stĺpec = atribút
  - atribúty, ktoré pomenúvavajú jednu doménu (jeden stĺpec) sa nazývajú jednoduché, ich kombinácie sa nazývajú zložené atribúty.
- **Databáza (DB schéma)** je konečná množina relácií, ktoré sú definované nad doménami D
  - Vzťah medzi entitami v ER diagrame sa reprezentuje pomocou relácie, ktorá obsahuje kľúčové atribúty entít ktoré prepája
- **Superkľúč relácie R** je množina atribútov relácie R s vlastnosťou **Jednoznačnej identifikácia** – každá n-tica relácie R je hodnotami atribútov, ktoré tvoria superkľúč jednoznačne určená, napr. pre študenta jú superkľúče, napr. (osobné číslo, meno), (osobné číslo, priezvisko), (osobné číslo)
- **Kľúč** je minimálny superkľúč, t.j. naviac spĺňa podmienku **neredundantnosti** – ak sa vynechá nejaký atribút z kľúča K poruší sa jednoznačnosť, z vyššieuvedených superkľúcov to je (osobné číslo)
- **Kandidát na primárny kľúč** = kľúč. Vybraný kľúč relácie R sa označí ako **primárny kľúč (PK)** (bude sa podtrhovať), ostatné sú Alternatívne kľúče **(AK)**
  - Príklad na PK: študent(osobné číslo, meno, priezvisko, výška, váha)
- **cudzí kľúč** (foreign key) (FK) – kľúč, ktorý je v inej tabuľke primárnym kľúčom (podtrhuje sa čiarkovane), napr. učiteľ (meno, škola)

13 pravidiel pre relačné databázy (Dr. E.F. Codd 1970)

1) DBS musí byť schopný spravovať DB **len pomocou svojich relačných schopností**
2) Všetky informácie v DBS (vrátane názvov tabuliek a stĺpcov) sú reprezentované explicitne ako **hodnoty v tabuľke**
3) Ku každej hodnote uloženej v DBS sa musí dať pristúpiť pomocou:
   a. **názvu tabuľky + primárneho kľúča + názvu atribútu**
4) DBS poskytuje podporu pre **prácu s hodnotou** $\boldsymbol{null}$ (neznáme, nedefinované, mimo rozsah, chybné údaje), ktoré sú odlišné od hodnôt v akejkoľvek doméne
5) Popis databázy a jej obsahu je reprezentovaný **na logickej úrovni v tabuľkovej forme** nad ktorými sa dá dopytovať pomocou databázového jazyka
6) **Databázový jazyk musí mať dobre definovanú syntax** a musí byť ucelený. Musí podporovať definíciu údajov, manipuláciu s údajmi, pravidlá integrity, autorizáciu, transakcie.
7) **Pohľady (views)** musia byť aktualizovateľné prostredníctvom DBS
8) DBS podporuje **operácie na úrovni množín** (získavanie údajov, vkladanie, aktualizáciu, ...)
9) **Fyzická dátová nezávislosť** – zmena fyzických štruktúr nemá dopad na logické
10) **Logická dátová nezávislosť** - zmena tabuľkových štruktúr nemá mať vplyv na aplikačné programy a AHQ (ad hoc query)
11) DBS musí byť schopný **definovať pravidlá integrity**. Tieto musia byť dostupné online
12) **Distribúcia, resp. redistribúcia** údajov nemá na aplikačné programy/AHQ žiadny vplyv.
13) **Nesmie** existovať možnosť **obísť pravidlá integrity**

Príklad 1 – ER diagram a príklad jeho tabuľkovej formy

vyjadrenie väzby medzi entitami reláciou



Databáza bude obsahovať relačné schémy:

- Programátor(IdProg, Skúsenosti, Rýchlosť)
- Firma(IdFirmy , adresa)
- Zmluva(IdProg , IdFirmy, Plat) platí síce aj Zmluva(IdProg , IdFirmy, Plat)
    - (medzi programátorom a firmou ale môže byť iba jedna zmluva)

Resp. pomocou zavadenia idZmluvy (medzi programátorom a firmou môže byť viac zmluv)

- Programátor(IdProg, Skúsenosti, Rýchlosť)
- Firma(IdFirmy , adresa)
- Zmluva(idZmluvy, IdProg , IdFirmy, Plat, DefiniciaPlnenia)

Kontorla – nižšie uvedené je nevhodné prečo?

- Programátor(IdProg, Skúsenosti, Rýchlosť, IdZmluvy)
- Firma(IdFirmy , adresa, idZmluvy)
- Zmluva(idZmluvy, IdProg , IdFirmy, Plat), resp. stačí Zmluva(idZmluvy, Plat, DefiniciaPlnenia)

(Lebo údaje sú uložené efektívne iba keď programátori a firmy majú po jednej zmluve, v opačnom prípade sa začnú kopiť opakované atribúty)