



Architektúra internetových a intranetových systémov:

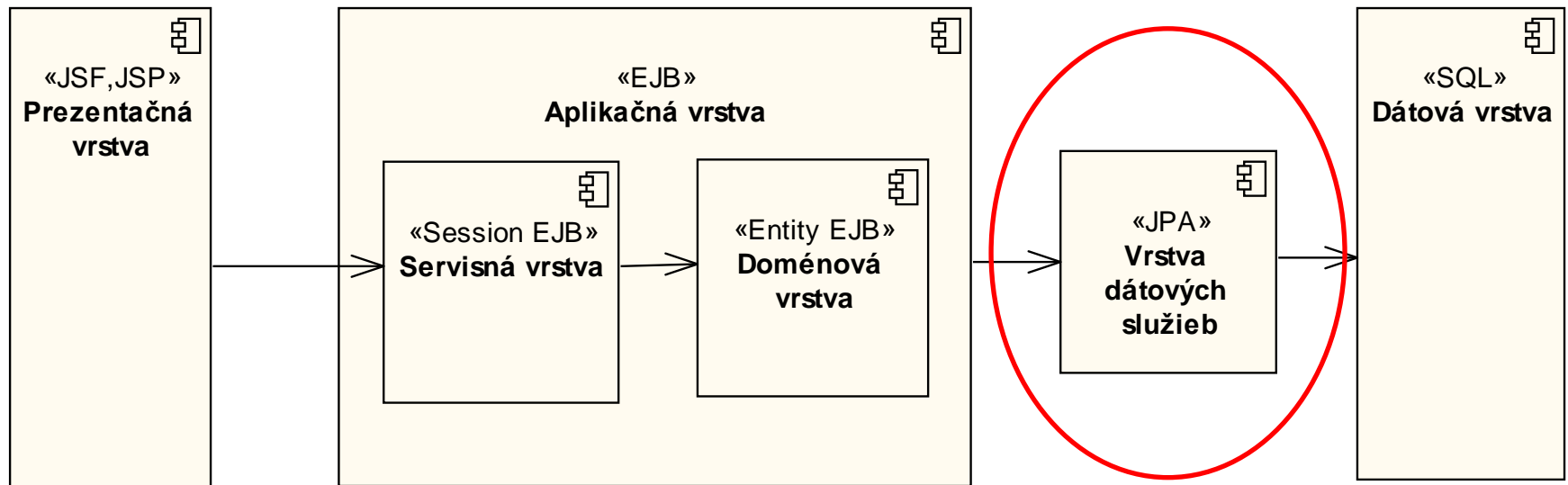
Vrstva dátových služieb

Ľubor Šešera



Vrstvy internetového systému

cmp 3.5 Vrstvy v JEE - alt. 3



Odlišnosti OO modelu a R-modelu

- Tradičný dátový zdroj – relačná databáza
 - objektový a relačný svet štrukturalizujú dáta rôznym spôsobom

Hlavné rozdiely OO a RDB modelu

1. Spájanie súvisiacich dát:

- Komplexné objekty.
 - RDB iba n-tice (záznamy) a z nich multimnožiny (tabuľky)
 - OO model aj zoznamy (list), polia,...
 - OO komplexné objekty je možné skladať
- Identita objektov.
 - RDB: záznam je určený svojím primárnym kľúčom
 - OO model: objekt má identifikátor

Odlišnosti OO modelu a R-modelu

2. Spájanie dát s funkciami

- Zapúzdrenie dát
 - RDB: procedúry pristupujú priamo k dátovým štruktúram (databázovým tabuľkám)
 - OO model: ukrývanie dát
 - Zmena vnútornej dátovej štruktúry tak štatisticky zasiahne iba menšiu časť kódu
- Typy objektov
 - RDB: preddefinovaná množina atomických typov
 - čísla, bitové a znakové reťazce, dátum a čas
 - Komplexné objekty, t. j. záznamy sú beztypové -> run-time chyby
 - OO model: deklarácia typov aj pre komplexné objekty

Odlišnosti OO modelu a R-modelu

- Rozšíriteľnosť základných typov
 - RDB: tabuľka je „zabudovaná“ štruktúra údajov s pevnou množinou príkazov
 - OO model: vývojár si definuje vlastné typy spolu s operáciami
- Dedenie
 - RDB: nie
 - OO model: možná hierarchia typov s dedením atribútov a metód
- Polymorfizmus
 - RDB: nie
 - OO model: prekrývanie + inkluzívny polymorfizmus
 - Inkluzívny polym. = všade tam, kde sa vyžaduje objekt nadtypu je možné použiť objekt podtypu

Odlišnosti OO modelu a R-modelu

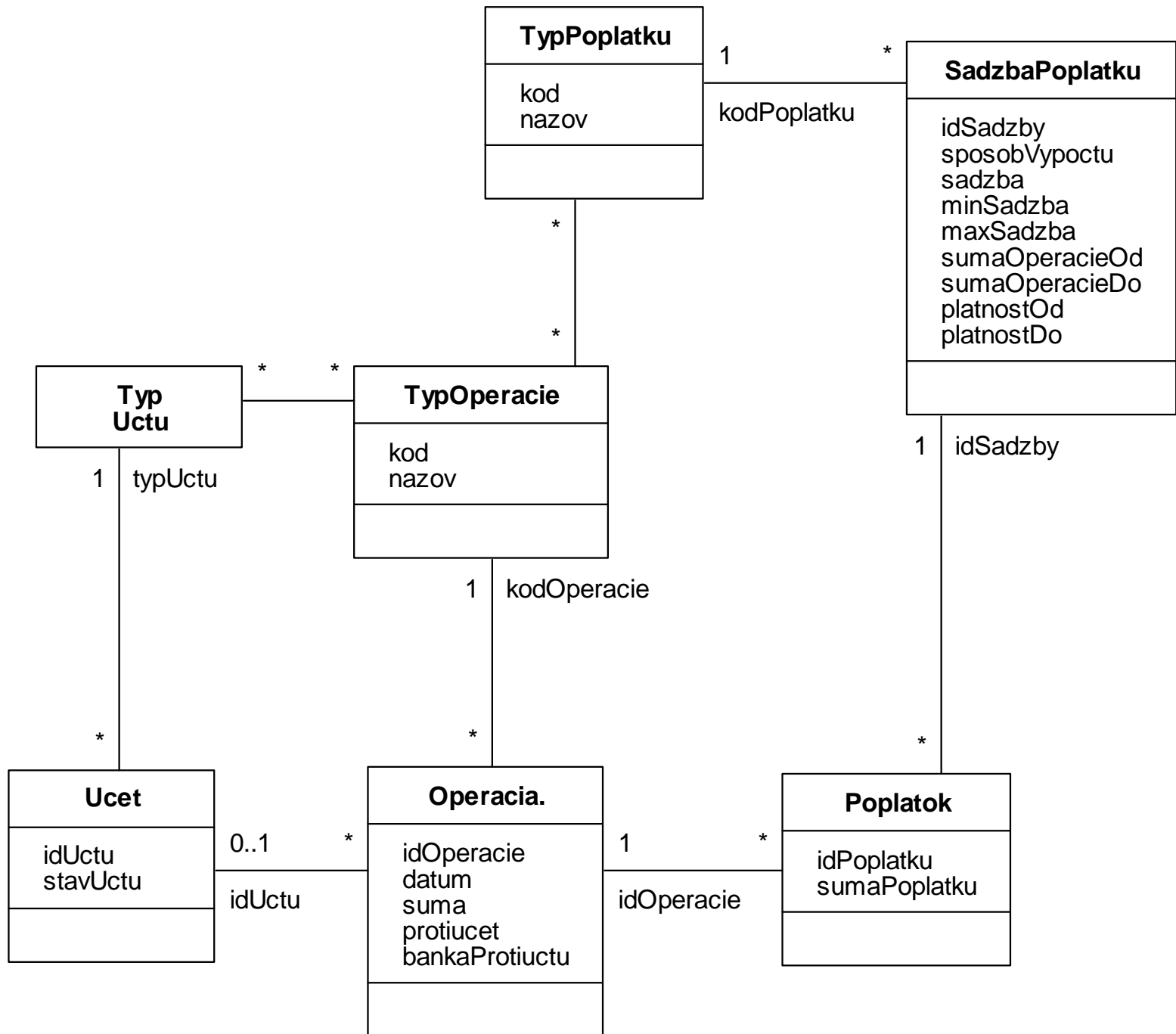
3. Manipulácia s dátami

- Sekvenčný verzus množinový prístup
 - RDB: množinové operácia (SQL)
 - Spájanie dát: operácia Table join
 - OO model: sekvenčný prístup
 - Spájanie dát: operácia navigácie (prechádzanie cez smerníky)
 - Otázka: OO dopytovacieho jazyka
 - napr. JPQL

Archit. vzory vrstvy dátových služieb

■ Alternatívy

- Brána do tabuľky (Table Data Gateway)
- Brána do riadku tabuľky (Row Data Gateway)
- Aktívny záznam (Active Record) –
- Prostriedok na objektovo-relačné mapovanie (Object-Relational Mapper, ORM)



Brána do tabuľky

- Objekt vo vrstve dátových služieb, ktorý zapuzdruje všetok prístup k jednej DB tabuľke
- **Jeden objekt** slúži pre **všetky záznamy tabuľky**, s ktorými sa pracuje
- Úlohy:
 - Zapuzdriť operácie SELECT, INSERT, UPDATE, DELETE
 - Dosadiť do SQL príkazu potrebné argumenty
 - Vyvolať vykonanie SQL príkazu
 - Vrátiť výsledok vo forme **blízkej relačnému modelu** (typ ResultSet)

Brána do tabuľky - príklad






```
public class SadzbaPoplatkuTableGateway {  
    private static final String sadzbaPoplatkuSelect =  
        "SELECT " + STLPCE +  
        "FROM SadzbyPoplatkov " +  
        "WHERE kodPoplatku = ? " +  
        "AND ? >= platnostOd AND ? <= platnostDo " +  
        "AND (sumaOperacieOd IS NULL  
            OR sumaOperacieOd <= ?) "  
        "AND (sumaOperacieDo IS NULL  
            OR sumaOperacieDo > ?)";
```

SQL príkaz v
tvare reťazec
znakov

```
    private static final String STLPCE  
        = "idSadzby, sposobVypoctu, sadzba, "  
        "platnostOd, platnostDo, minSadzba, "  
        "maxSadzba, sumaOperacieOd, sumaOperacieDo";
```

* SadzbaPoplatku	
idSadzby sposobVypoctu sadzba minSadzba maxSadzba sumaOperacieOd sumaOperacieDo platnostOd platnostDo	
1	idSadzby

Brána do tabuľky – príklad (pokr.)

```
public static ResultSet najdiSadzbu (String kodPoplatku,  
                                     Date datum, BigDecimal sumaOperacie) {  
    Connection dbconnection =  Získanie spojenia na DB  
                                DriverManager.getConnection(...);  
    PreparedStatement prikaz =  Kompilácia SQL príkazu  
                                dbconnection.prepareStatement(sadzbaPoplatkuSelect);  
    prikaz.setString(1, kodPoplatku);  
    prikaz.setDate(2, datum);  Dosadenie parametrov  
    prikaz.setDate(3, datum);  do SQL príkazu  
    prikaz.setBigDecimal(4, sumaOperacie);  
    prikaz.setBigDecimal(5, sumaOperacie);  
    ResultSet vysledok = prikaz.executeQuery();  Vykonanie SQL príkazu  
    return vysledok;  
}
```

Brána do tabuľky – príklad (pokr.)

```
ResultSet vysledokSQL =
    sadzbaPoplatkuTableGateway.najdiSadzbu(
        kodPoplatku, new Date(), sumaOperacie);
while (vysledokSQL.next()) {
    Long idSadzby = new Long (vysledokSQL.getLong(1));
    String sposobVypoctu = vysledokSQL.getString(2);
    BigDecimal sadzba = vysledokSQL.getBigDecimal(3);
    BigDecimal minSadzba = vysledokSQL.getBigDecimal(4);
    BigDecimal maxSadzba = vysledokSQL.getBigDecimal(5);
    ...
    BigDecimal vyskaPoplatku =
        sluzbyPoplatkov.vypocitajPoplatok
            (sposobVypoctu, sadzba, minSadzba,
             maxSadzba, sumaOperacie);
    ...
}
```

Aplikačná vrstva
spracováva de facto
„neobjektovo“

Brána do riadku tabuľky

- Objekt vo vrstve dátových služieb, ktorý zapúzdruje všetok prístup k *jednému záznamu* (riadku) DB tabuľky
- Existuje **jeden** objekt pre **jeden** spracovávaný záznam tabuľky
- Hlavný rozdiel oproti Bráne do tabuľky -nevracia dátovú štruktúru, ktorá odzrkadľuje dátové štruktúry v relačnej databáze, ale vykoná **pretypovanie** výsledku DB dopytu na typy používané v aplikačnej vrstve – OO model

Brána do riadku tabuľky - Príklad

```
public class SadzbaPoplatkuRowGateway {  
    private static final String sadzbaPoplatkuSelect =  
        "SELECT " + STLPCE +  
        "FROM SadzbyPoplatkov " +  
        "WHERE kodPoplatku = ? " +  
            "AND ? >= platnostOd AND ? <= platnostDo " +  
            "AND (sumaOperacieOd IS NULL  
                OR sumaOperacieOd <= ?) " +  
            "AND (sumaOperacieDo IS NULL  
                OR sumaOperacieDo > ?)";  
  
    private static final String STLPCE =  
        "idSadzby, sposobVypoctu, sadzba, platnostOd, " +  
        "platnostDo, minSadzba, maxSadzba, " +  
        "sumaOperacieOd,sumaOperacieDo";  
}
```

Rovnako ako v
Bráne do tabuľky

Brána do riadku tab. – Príklad (pokr.)

```
public static SadzbaPoplatku najdiSadzbu (String kodPoplatku,  
                                         Date datum, BigDecimal sumaOperacie) {  
    Connection dbconnection = DriverManager.getConnection(...);  
    PreparedStatement prikaz =  
        dbconnection.prepareStatement(sadzbaPoplatkuSelect);  
    prikaz.setString(1, kodPoplatku);  
    prikaz.setDate(2, datum);  
    prikaz.setDate(3, datum);  
    prikaz.setDate(4, sumaOperacie);  
    prikaz.setDate(5, sumaOperacie);  
    ResultSet vysledokSQL = prikaz.executeQuery();  
    vysledokSQL.next();  
    SadzbaPoplatku vysledok = SadzbaPoplatkuRowGateway.  
        naplnSadzbu (vysledokSQL);  
    return vysledok; }
```

Rovnaký kód ako v
Bráne do tabuľky

↑
Tu je rozdiel: Výsledok je objekt
požadovaného typu

Brána do riadku tab. – Príklad (pokr.)

```
public static SadzbaPoplatku naplnSadzbu
                                (ResultSet vysledokSQL) {
    Long idSadzby = new Long (vysledokSQL.getLong(1));
    String sposobVypoctu = vysledokSQL.getString(2);
    BigDecimal sadzba = vysledokSQL.getBigDecimal(3);
    BigDecimal minSadzba = vysledokSQL.getBigDecimal(4);
    BigDecimal maxSadzba = vysledokSQL.getBigDecimal(5);
    BigDecimal sumaOperacieOd =
                                vysledokSQL.getBigDecimal(6);
    BigDecimal sumaOperacieDo =
                                vysledokSQL.getBigDecimal(7);
    Date platnostOd = getDate(8); Date platnostDo = getDate(9);
    SadzbaPoplatku vysledok = new SadzbaPoplatku (
        idSadzby, sposobVypoctu, sadzba, minSadzba,
        maxSadzba, sumaOperacieOd, ...);
    return vysledok; } ...}
```

Podobný kód ako v
Bráne do tabuľky

Tu je rozdiel:
Vytvorí
objekt

Brána do riadku tab. – Príklad (pokr.)

Aplikačná vrstva
dostane objekt
požadovaného typu



```
SadzbaPoplatku sadzbaPoplatku =  
    sadzbaPoplatkuRowGateway.najdiSadzbu (  
        kodPoplatku, new Date(), sumaOperacie);  
BigDecimal vyskaPoplatku =  
    sluzbyPoplatkov.vypocitajPoplatok  
        (sadzbaPoplatku, sumaOperacie);  
...
```

Aktívny záznam

- Objekt ktorý zapúzdruje všetok prístup k jednému *záznamu* databázovej tabuľky
- V porovnaní s Bránou do riadku tabuľky navyše *pridáva* k tomuto záznamu *doménovú logiku*.
 - => aktívny záznam sa umiestňuje do aplikačnej vrstvy, t. j. dochádza k spojeniu aplikačnej vrstvy a vrstvy dátových služieb
 - Nemalo by sa príliš používať dedenie a ďalšie pokročilé črty objektovo orientovaného návrhu

Aktívny záznam – Príklad

```
public class SadzbaPoplatku {  
    private static final String sadzbaPoplatkuSelect =  
        "SELECT " + STLPCE +  
        ...  
    public static SadzbaPoplatku najdiSadzbu (. . .) {  
        ...  
        SadzbaPoplatku vysledok =  
            SadzbaPoplatkuRowGateway.  
                naplnSadzbu (vysledokSQL);  
        ...}  
    public void vypocitajPoplatok (Operacia operacia) {  
        BigDecimal sumaPoplatku =  
            getPoplatkovaFunkcia().vypocitajPoplatok(. . .)  
        ...  
    }
```

Metóda vrstvy dátových služieb

Metóda aplikačnej vrstvy

Prostriedok na ORM

- RDB a OO model: podobnosť aj odlišnosť
- Podobné:
 - Trieda \leftrightarrow Tabuľka, Atribút \leftrightarrow Stĺpec
- Odlišné
 - Identifikácia objektov, komplexné objekty, dedenie, ...
- Prostriedok pre ORM
 - Umožňuje deklaratívne (meta-jazyk) definovať určité transformácie OO modelu do RDB

- Najprv samostatné nástroje
 - TopLink pre Smalltalk (The Object People)
 - TopLink pre Java (1996-1998)
 - Od roku 2002 patrí firme Oracle
 - Hibernate (2001, open source)
 - → JBoss → Red Hat
- Súčasť aplikačných serverov:
 - EJB 1.x a 2.x - vlastný mechanizmus mapovania OO na RDB (ťažkopádny)
 - EJB 3.0 obsahuje JPA (Java Persistence API)
 - Vychádza z Hibernate a TopLink

Deployment descriptor v JPA

```
<entity name="Ucet " class="Ucet"> <table name="Ucty" />
  <attributes>
    <id name="idUctu"> <column name="idUctu"> </id>
    <basic name="stavUctu"> <column name="stavUctu"> </basic>
  </attributes>
</entity>
<entity name="SadzbaPoplatku" class="SadzbaPoplatku">
  <table name="SadzbyPoplatkov" />
  <attributes>
    <id name="idSadzby"> <column name=" idSadzby"> </id>
    <basic name="sposobVypoctu">
      <column name="sposobVypoctu"> </basic>
    <basic name="sadzba"> <column name="sadzba"> </basic>
    ...
  </attributes>
</entity>
</entity-mappings>
```

Deployment descriptor v JPA

```
<named-native-query name="sadzbaPoplatkuSelect">  
  <query>  
    SELECT idSadzby, sposobVypoctu, sadzba, platnostOd,  
          platnostDo, minSadzba, maxSadzba,  
          sumaOperacieOd,sumaOperacieDo  
    FROM SadzbyPoplatkov  
    WHERE kodPoplatku = ?  
          AND ? >= platnostOd AND ? <= platnostDo  
          AND (sumaOperacieOd IS NULL  
              OR sumaOperacieOd <= ?)  
          AND (sumaOperacieDo IS NULL  
              OR sumaOperacieDo > ?)  
  </query>  
</named-native-query>
```

V deployment descriptore
vieme zapísať aj dopyt

Anotácie v JPA (pre atribúty)

@Entity

@Table (name="SadzbyPoplatkov")

```
public class SadzbaPoplatku {
```

```
    @Id
```

```
    @Column (name="idSadzby")
```

```
    public Long idSadzby;
```

```
    @Column (name=" sposobVypoctu")
```

```
    String sposobVypoctu;
```

```
    @Column (name="sadzba")
```

```
    BigDecimal sadzba;
```

```
    ...
```

```
}
```


Anotácie v JPA (pre atribúty) +defaults

@Entity

```
public class SadzbaPoplatku {
```

```
    @Id
```

```
    public Long idSadzby;
```

```
    String sposobVypoctu;
```

```
    BigDecimal sadzba;
```

```
    BigDecimal minSadzba;
```

```
    BigDecimal maxSadzba;
```

```
    BigDecimal sumaOperacieOd;
```

```
    BigDecimal sumaOperacieDo;
```

```
    Date platnostOd;
```

```
    Date platnostDo;
```

```
    ...
```

```
}
```

Anotácie v JPA (pre get metódy)

@Entity

@Table (name="Ucet")

```
public class Ucet {  
    private String idUctu;  
    private String stavUctu;
```

...

@Id

@Column (name="idSadzby")

```
public String getIdUctu() {return idUctu;}
```

```
public void setIdUctu (String idUctu) {this.idUctu = idUctu;}
```

@Column (name="stavUctu")

```
public String getStavUctu() {return stavUctu;}
```

```
public void setStavUctu (String stavUctu) {this.stavUctu =  
stavUctu;}
```

```
}
```

Anotácie vs. Deployment descriptor

- Výhody anotácií
 - Všetko na jednom mieste (pri malých modeloch)
 - Kontroly pri kompilácii
- Nevýhody anotácií
 - Drôtovanie schémy DB do aplikačného kódu
 - Zneprehľadňuje kód (pri väčších modeloch)
 - Rekompilácia pri zmenách

JPA EntityManager

- ORM zabezpečuje objekt typu EntityManager
- public void **persist** (Object entity)
 - uloží objekt do databázy a zoberie ho do svojej správy;
- public <T> T merge (T entity)
 - pridá existujúci objekt medzi objekty, ktoré spravuje daný EntityManager a následne tak zabezpečí jeho zápis do databázy;
- public void **remove** (Object entity)
 - vymaže objekt z databázy;
- public void refresh (Object entity)
 - obnoví objekt hodnotami z databázy;

JPA EntityManager

- `public <T> T find (Class<T> entityClass, Object primaryKey)`
 - vyhledá objekt v databáze podľa primárneho kľúča;
- `public Query createQuery (String jpqlString)`
 - vytvorí dynamický dopyt v tvare JPQL;
- `public Query createNamedQuery (String name)`
 - vytvorí inštanciu dopytu na základe definovaného pomenovaného dopytu;
- `public Query createNativeQuery (String sqlString)`
 - vytvorí dynamický dopyt v tvare SQL;

JPA EntityManager

- Neobsahuje metódu update
 - Ak je objekt v správe, EntityManager dokáže automaticky rozpoznať, že sa objekt zmenil a pred ukončením transakcie aktualizovať zodpovedajúci záznam v databáze

Vytvorenie účtu:

```
Ucet ucet = new Ucet();  
ucet.setIdUctu ("1234567890");  
ucet.setStavUctu ("otvoreny");  
entityManager.persist (ucet);
```

Vyhľadanie podľa primárneho kľúča:

```
String cisloUctu = "1234567890";  
Ucet ucet = entityManager.find (Ucet.class, cisloUctu);
```

Vymazanie účtu:

```
entityManager.remove (ucet);  
entityManager.remove (entityManager.merge(ucet));
```

Ak nie je v správe

Vytvorenie EntityManagera v EJB

- Anotáciou s Injektážou závislostí:

@Stateless

```
public class UcetService {
```

```
    @PersistenceContext
```

```
    private EntityManager entityManager;
```

```
...}
```


Štrukturálne vzory pre ORM

- Okruhy problémov:
 1. Identita objektov
 2. Relácie medzi objektmi
 3. Triedne dedenie a polymorfizmus
 4. Dátová navigácia a objektovo orientované dopyty

Identita objektov

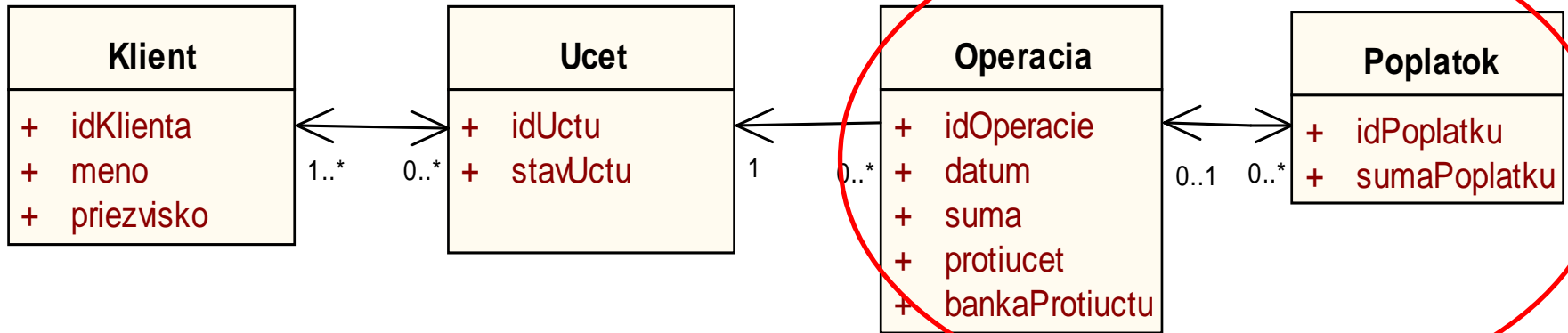
- OO model
 - Vnútro pamäťový objekt má svoj identifikátor
 - Zodpovedá jeho adrese v pamäti
- RDB
 - Záznam jednoznačne určený svojím primárnym kľúčom, ktorý definuje vývojár, resp. používateľ
 - Môže byť tvorený hodnotami viacerých stĺpcov
- JPA
 - Jednoduchý primárny kľúč: anotácia @Id
 - Zložený primárny kľúč: + metóda equals

Relácie medzi objektmi

- OO model
 - reprezentované formou referencií / smerníkov
- RDB
 - reprezentované pomocou cudzích kľúčov
- Prostriedok ORM
 - Automatická transformácia
- Násobnosti
 - Jeden-jeden, mnoho-jeden
 - Priamočiara transformácia
 - Jeden-mnoho
 - Opačne umiestnený cudzí kľúč + SELECT
 - Mnoho-mnoho
 - Vytvorenie spojovacej tabuľky

Relácie medzi objektmi - príklad

class 3.20 Mapovanie asociácii



Relácie medzi objektmi - príklad

@Entity

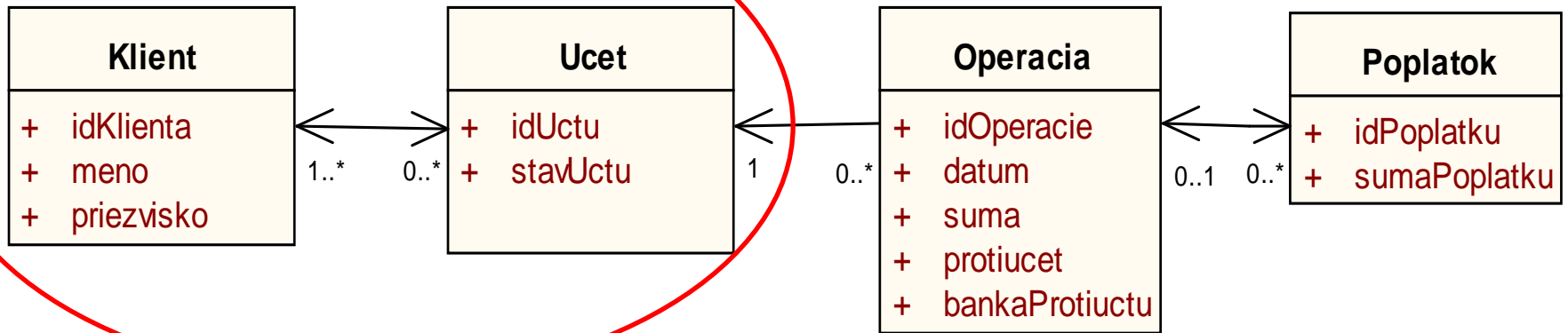
```
public class Poplatok {  
    @Id  
    public Long idPoplatku;  
    public BigDecimal sumaPoplatku;  
    @ManyToOne  
    public Operacia operacia;  
}
```

@Entity

```
public class Operacia {  
    @Id  
    public Long idOperacie;  
    ...  
    @OneToMany (cascade=CascadeType.ALL, mappedBy="operacia")  
    public Set<Poplatok> poplatky;  
}
```

Relácie medzi objektmi - príklad

class 3.20 Mapovanie asociacii



Relácie medzi objektmi - príklad

@Entity

```
public class Ucet {
```

```
    . . .
```

```
    @ManyToMany (mappedBy="ucty")
```

```
    public Set<Klient> klienti;
```

```
}
```

@Entity

```
public class Klient {
```

```
    . . .
```

```
    @ManyToMany
```

```
    @JoinTable (name="KlientUcet"
```

```
                joinColumns= @JoinColumn (name="klient"
```

```
                referencedColumnName="idKlienta")
```

```
                inverseJoinColumns= @JoinColumn
```

```
                (name="ucet" referencedColumnName="idUctu")
```

```
    public Set<Ucet> ucty;
```

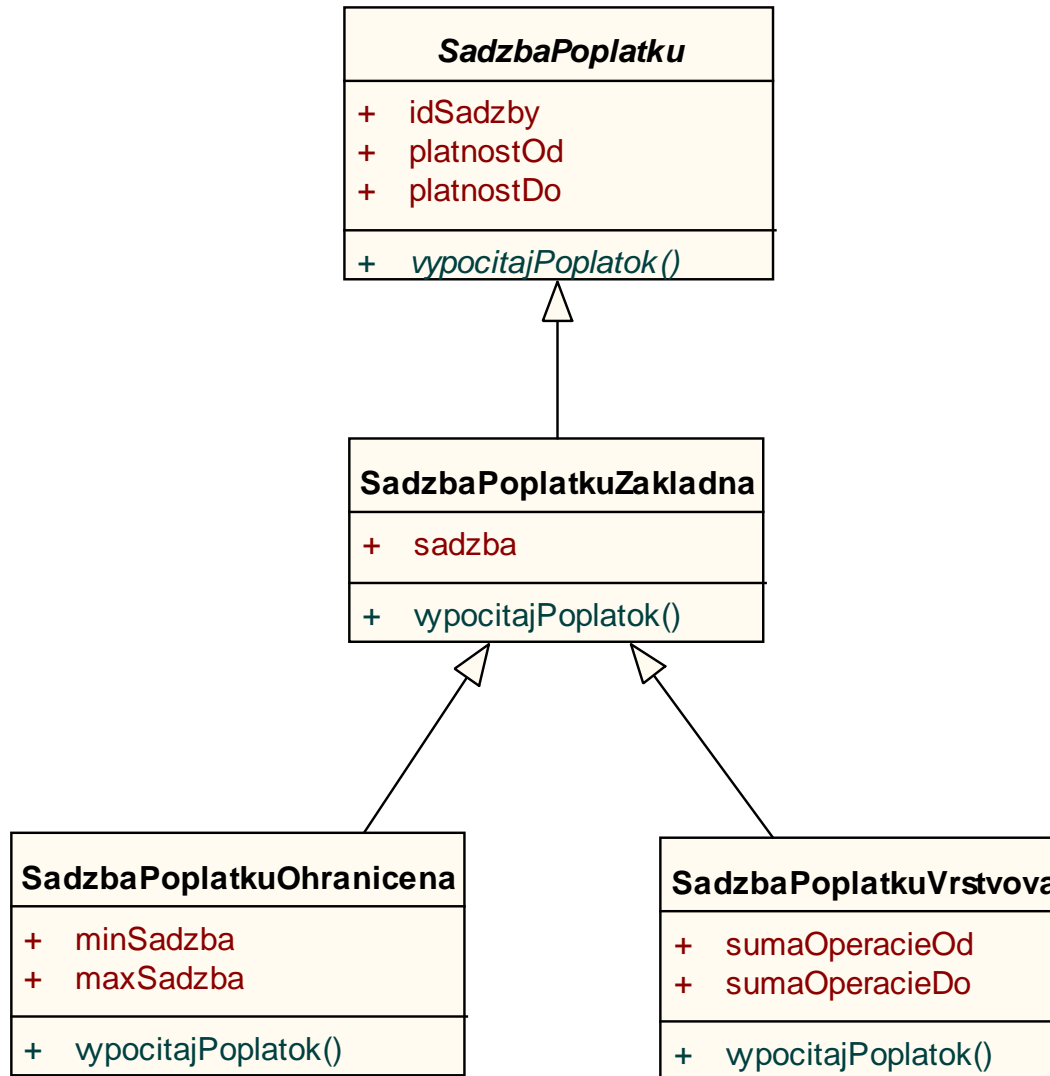
```
}
```

Triedne dedenie a polymorfizmus

1. Jedna tabuľka pre hierarchiu dedenia (Single Table Inheritance)
 - Všetky triedy v danej hierarchii dedenia sa mapujú do jedinej databázovej tabuľky
2. Tabuľka pre každú triedu v hierarchii dedenia (Class Table Inheritance)
 - Každá trieda v danej hierarchii dedenia sa mapuje do samostatnej databázovej tabuľky.
3. Tabuľka pre každú konkrétnu triedu v hierarchii (Concrete Table Inheritance)
 - DB tabuľka obsahuje stĺpce pre atribúty konkrétnej triedy plus stĺpce pre atribúty všetkých nadtried tejto triedy

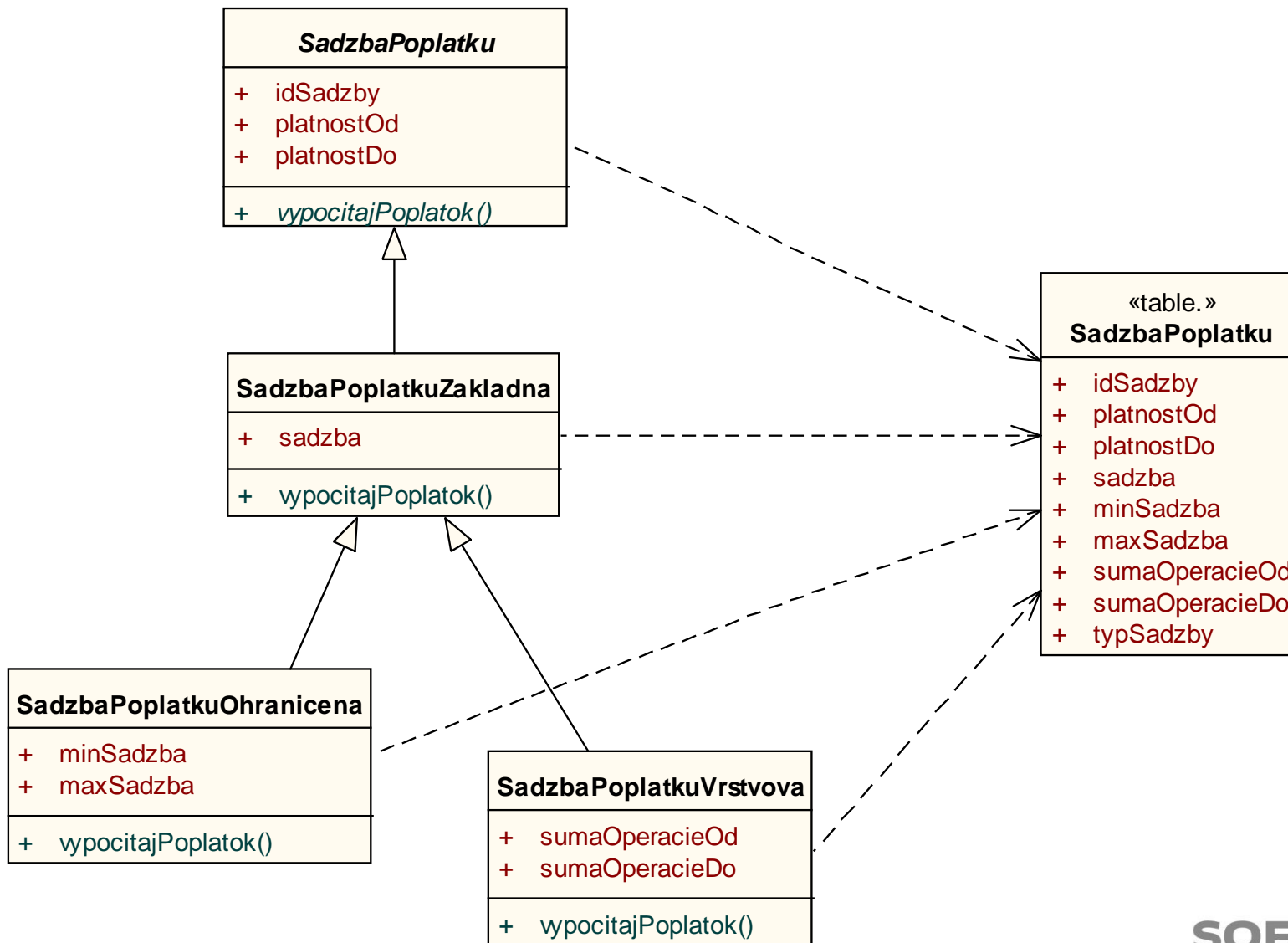
Dedenie - príklad

class 3.13 Príklad pre spôsoby dedenia



Jedna tabuľka pre hierarchiu dedenia

class 3.14 Jedna tabuľka pre hierarchiu dedenia



Jedna tabuľka pre hierarchiu dedenia

- Výhody
 - Efektívnosť nakladania dát
- Nevýhody
 - Neefektívna reprezentácia v databáze
 - Nemožnosť definovať v databáze určité ohraničenia pre dátovú integritu
 - Databázová tabuľka môže obsahovať priveľa objektov
 - Mentálna transformácia pri vynechaní ORM

Jedna tabuľka pre hierarchiu dedenia

@Entity

@Table (name="SadzbaPoplatku")

@Inheritance (strategy=InheritanceType.SINGLE_TABLE)

@DiscriminatorColumn (name="typSadzby",
discriminatorType=DiscriminatorType.STRING, length=4)

public class **SadzbaPoplatku** {...}

@Entity

@DiscriminatorValue (value="zakl")

public class **SadzbaPoplatkuZakladna** {...}

@Entity

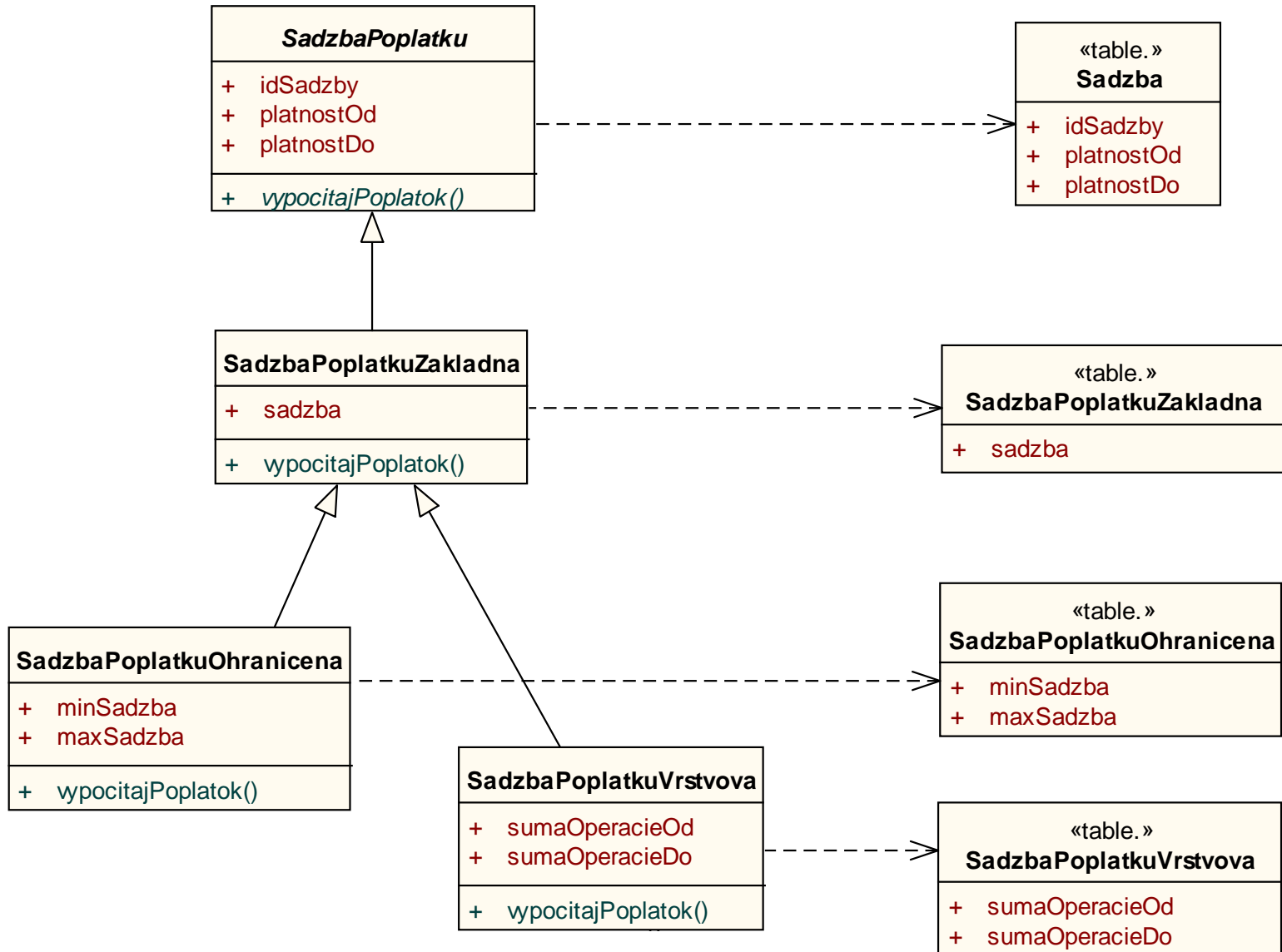
@DiscriminatorValue (value="ohrn")

public class **SadzbaPoplatkuOhranicena** {...}

...

Tabuľka pre každú triedu

class 3.15 Tabuľka pre každú triedu v hierarchii dedenia



Tabuľka pre každú triedu

- Výhody
 - Priamočiara reprezentácia
 - Efektívna dátová reprezentácia
- Nevýhody
 - Tabuľky nie sú v databáze vzájomne prepojené
 - Neefektívne získavanie dát (join)
 - Tabuľka pre nadtriedu sa môže stať úzkym miestom

Tabuľka pre každú triedu

@Entity

@Table (name="SadzbaPoplatku")

@Inheritance (strategy=InheritanceType.JOINED)

@DiscriminatorColumn (name="typSadzby",
discriminatorType=DiscriminatorType.STRING,
length=4)

public class **SadzbaPoplatku** {...}

@Entity

@Table (name="SadzbaPoplatkuZakladna")

@DiscriminatorValue (value="zakl")

@PrimaryKeyJoinColumn (name="idSadzby")

public class **SadzbaPoplatkuZakladna** {...}

Tabuľka pre každú triedu

@Entity

@Table (name="SadzbaPoplatkuOhranicena")

@DiscriminatorValue (value="ohrn")

@PrimaryKeyJoinColumn (name="idSadzby")

public class SadzbaPoplatkuOhranicena {...}

@Entity

@Table (name="SadzbaPoplatkuVrstvova")

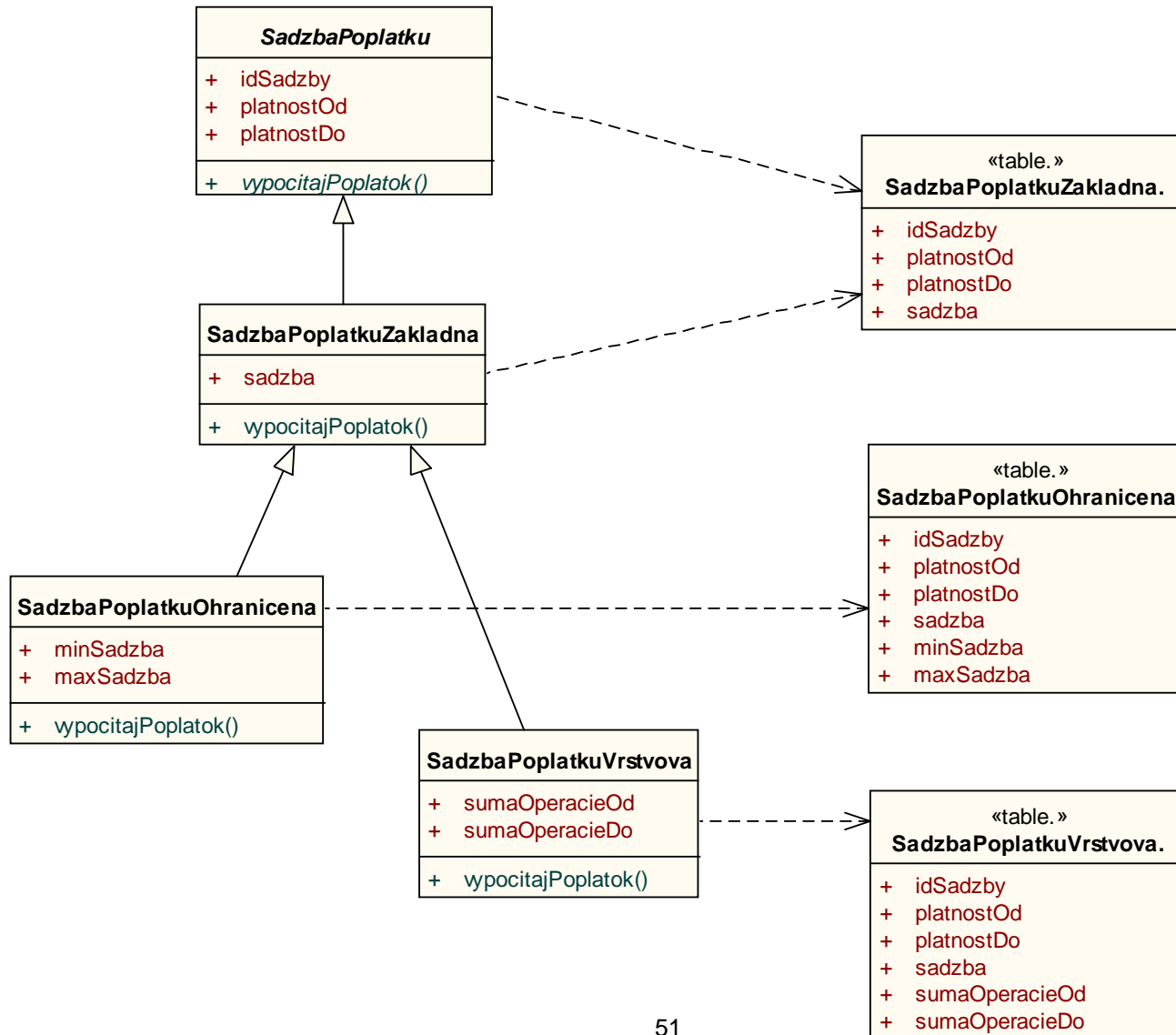
@DiscriminatorValue (value="vrst")

@PrimaryKeyJoinColumn (name="idSadzby")

public class SadzbaPoplatkuVrstvova {...}

Tabuľka pre každú konkrétnu triedu

class 3.16 Tabuľka pre každú konkrétnu triedu v hierarchii dedenia



Tabuľka pre každú konkrétnu triedu

- Výhody:
 - Tabuľka neobsahuje nepoužívané stĺpce
 - Umožňuje efektívne nakladanie objektu, pretože nie sú potrebné drahé operácie spájania tabuliek
- Nevýhody:
 - Problém s primárnymi kľúčmi
 - Primárny kľúč nadtriedy sa kopíruje do podtried \Rightarrow musí byť jedinečný v celej hierarchii = komplikované
 - Pri zmene nadtriedy je potrebné urobiť zmeny vo všetkých databázových tabuľkách podtried
 - Nie je možné v databáze definovať určité relácie a integritné ohraničenia (relácie na abstraktné triedy)
 - Komplikovane sa vyhľadávajú objekty nadtriedy
 - Najzložitejší na realizáciu (voliteľné v JPA)

Tabuľka pre každú konkrétnu triedu

@Entity

@Inheritance (strategy=InheritanceType.TABLE_PER_CLASS)

public class SadzbaPoplatku {...}

@Entity

@Table (name="SadzbaPoplatkuZakladna")

public class SadzbaPoplatkuZakladna {...}

@Entity

@Table (name="SadzbaPoplatkuOhranicena")

public class SadzbaPoplatkuOhranicena {...}

@Entity

@Table (name="SadzbaPoplatkuVrstvova")

public class SadzbaPoplatkuVrstvova {...}

Polymorfizmus

- Inkluzívny polymorfizmus
 - Ak existuje relácia na nadtriedu, pri prechode po relácii sa očakáva, že sa naložia nielen objekty nadtriedy, ale aj jej podtried
 - Povinná črta v JPA

Dátová navigácia a OO dopyty

- RDB poskytujú SQL
- OO dopyt
 - Využitie bohatých vlastností SQL plus
 - Kolekcia typov. objektov vs. množina záznamov
 - Získavanie asociovaných objektov navigáciou (bodková notácia)
- V histórii viac pokus o OO dopytovací jazyk
 - Nedosiahli rozšírenosť SQL
- JPA: Jazyk JPQL
 - (Java Persistence Query Language)

- Plne objektovo orientovaný dopytovací jazyk
- Na rozdiel od JDBC:
 - dopyt nie je znakový reťazec, ktorému rozumie iba RDBMS, ale veta v jazyku, ktorej rozumie procesor dopytov => kontroly
- Typy dopytov:
 - Pomenovaný (named) dopyt = statický dopyt
 - Opakovaná použiteľnosť
 - Vyššia efektívnosť
 - Dynamický dopyt – v čase vykonávania programu
 - môže sa „poskladať“ ako reťazec znakov

JPQL - príklad

@Entity

@NamedQueries ({

 @NamedQuery (

 name = "najdiOperaciePodlaDatumu"

 query = "SELECT o FROM Operacia o
 WHERE ucet LIKE ?1 AND

 ?2 <= datum AND ?3 >= datum

 ORDER BY o.datum"),

 @NamedQuery (

 name = "najdiOperaciePodlaProtiuctu"

 query = "SELECT o FROM Operacia o
 WHERE ucet LIKE :ucet AND

 protiucet LIKE :protiucet

 ORDER BY o.datum") })

public class Operacia {...}

JPQL – príklad pomenovaného dopytu

@Entity

```
public class Operacia {
```

```
    @PersistenceContext
```

```
    private EntityManager entityManager;
```

```
    ...
```

```
    public List najdiOperaciePodlaDatumu
```

```
        (Ucet ucet, Date datumOd, Date datumDo) {
```

```
        Query query = entityManager.createNamedQuery  
                                ("najdiOperaciePodlaDatumu");
```

```
        query.setParameter (1, ucet);
```

```
        query.setParameter (2, datumOd);
```

```
        query.setParameter (3, datumDo);
```

```
        List operacie = query.getResultList();
```

```
    ...}
```


JPQL – príklad dynamického dopytu

```
Query query = entityManager.createQuery  
    ("SELECT o FROM Operacia  
     WHERE ucet LIKE ?1 AND  
      ?2 <= datum AND ?3 >= datum");
```

JPQL – cestné výrazy

- Namiesto operácie JOIN

```
SELECT p, p.operacia.datum  
FROM Poplatok p  
WHERE p.operacia.ucet LIKE ?1
```

- Niekedy sú však operácie JOIN potrebné
 - Ak neexistujú cudzie kľúče

Technické aspekty ORM

- **Jednotka práce (Unit of Work)**
 - Spravuje kolekciu objektov, ktoré ovplyvnila aplikačná transakcia a koordinuje zápis zmien do databázy.
- **Mapa identít (Identity Map)**
 - Zabezpečuje, aby bol každý objekt z databázy naložený iba raz a poskytuje úschovňu (angl. cache) naložených objektov vo vnútornej pamäti.
- **Lenivé nakladanie (Lazy Load)**
 - Doťahujú sa iba aktuálne potrebné hodnoty atribútov objektu a asociované objekty. Ostatné sa nakladajú neskôr, podľa potreby.

Jednotka práce

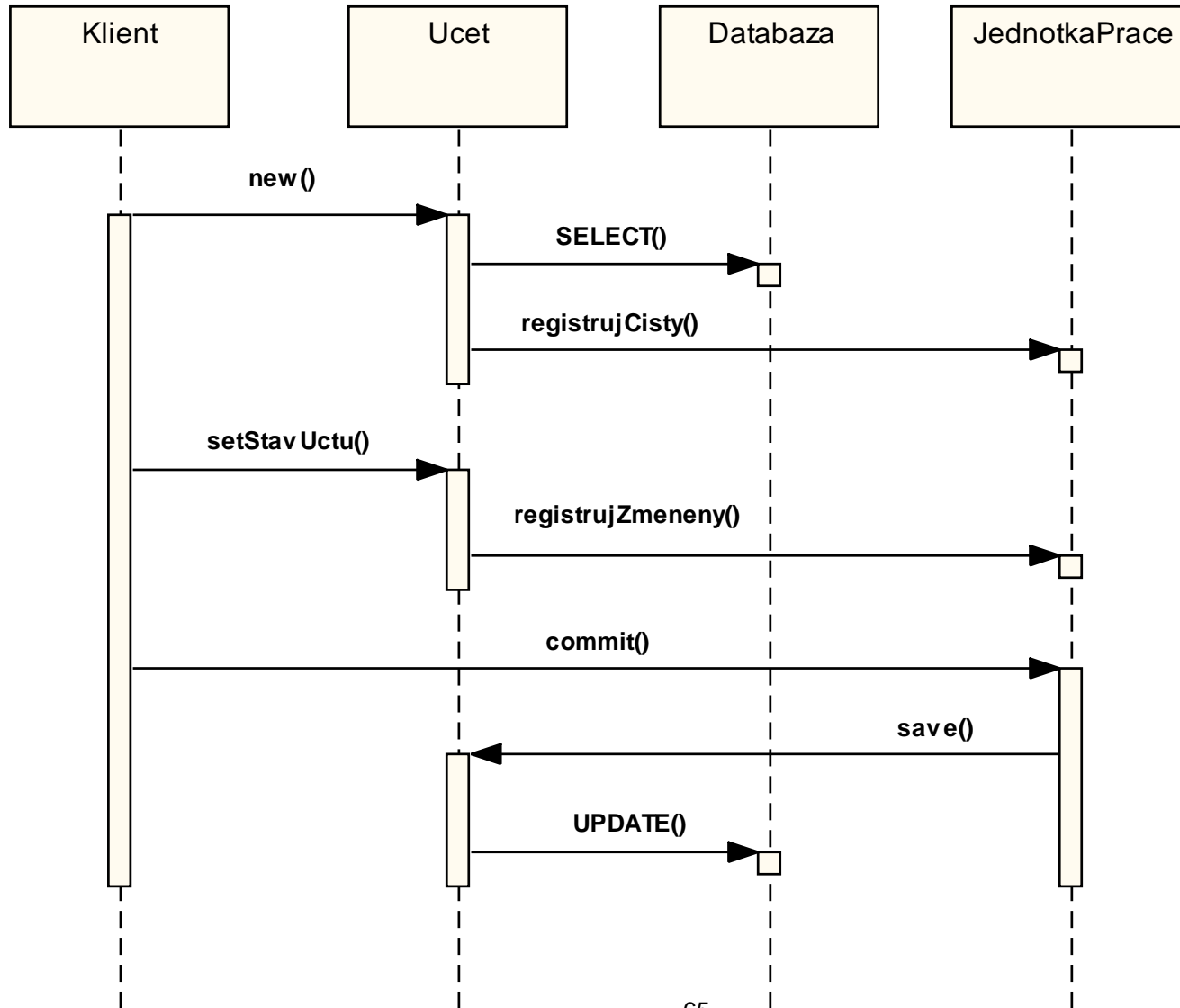
- Zmenené objekty sa nezapisujú do databázy okamžite pri každej zmene, pretože by to viedlo k veľkému množstvu malých databázových operácií
- Navyše by pri dlhých transakciách táto transakcia musela byť otvorená po celý čas a blokovala by tak systémové zdroje
- Objekty sa zapisujú zvyčajne až **na konci aplikačnej transakcie** => *pamätanie si objektov, ktoré boli zmenené* = Jednotka práce

Typy registrácií v Jednotky práce

- Registrácia klientom
 - Informuje klientsky program
- Registrácia objektom
 - Metódy objektu, ktoré súvisia s naložením objektu z databázy
 - Metódy, ktoré menia atribúty objektu
- Registrácia Jednotkou práce
 - Jednotka práce si pri nakladaní zapamätá kópiu objektu (vylepšenie: iba hash hodnotu)
 - Pri ukončení transakcie porovnáva objekt so zapamätanou kópiou (porovná 2 hash hodnoty)

Jednotka práce – registrácia objektom

sd Jednotka práce

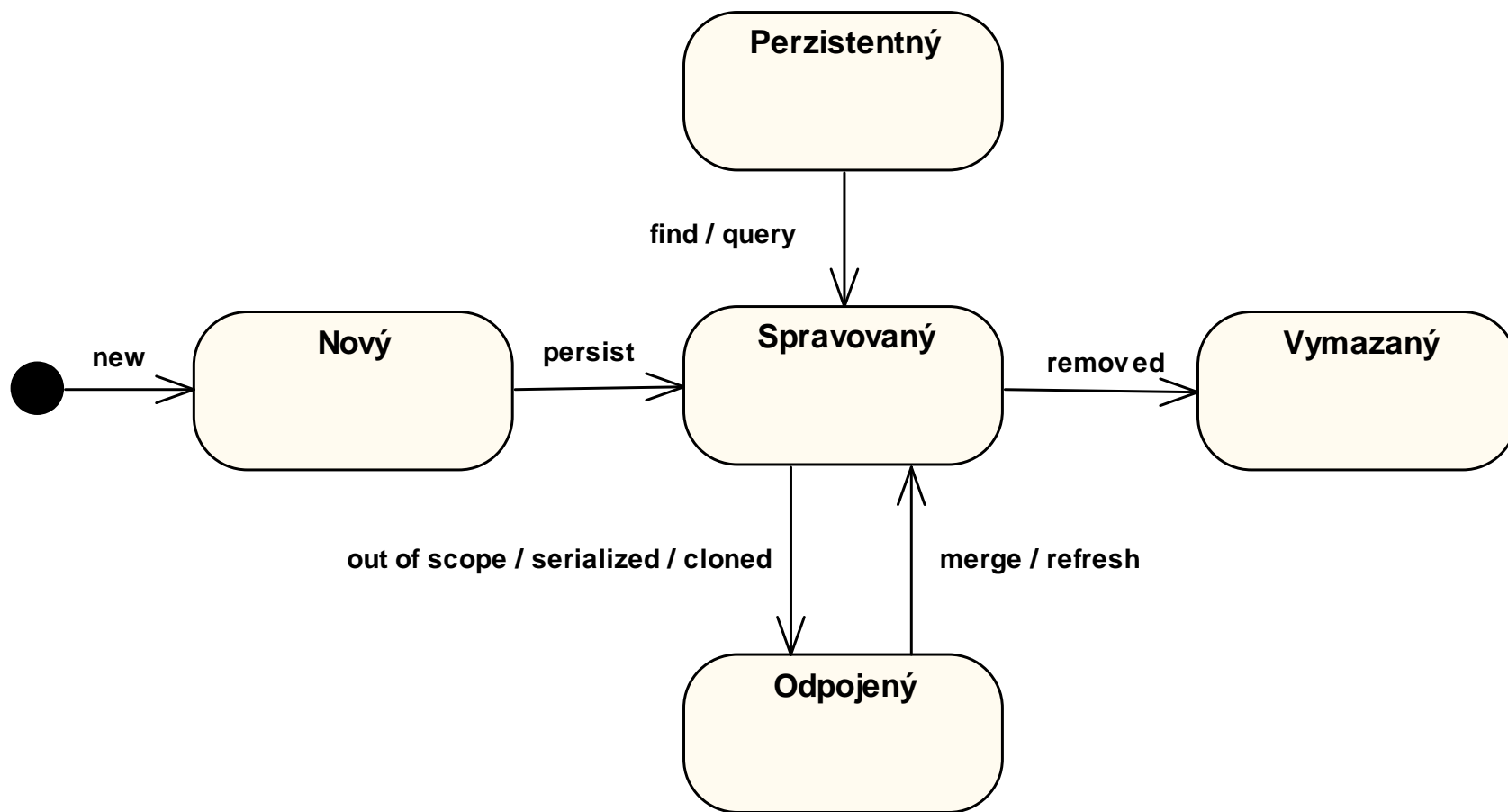


Jednotka práce v JPA

- Persistence context
 - Obsahuje kolekciu objektov, ktoré spravuje EntityManager
- Stavový diagram správy aplikačného objektu objektom typu EntityManager – ďalší slajd

Jednotka práce v JPA

stm Component Model



Mapa identít a úschovňa objektov

- Nepozornosťou pri programovaní sa môže stať, že sa z databázy opakovane načíta objekt, ktorý už bol predtým v tej istej transakcii načítaný
- Problém pri zápise do databázy
 - Vážne chyby pri vzájomnom prepisovaní záznamu
- Mapa identít - obsahuje už naložené objekty z databázy
 - Zabráneniu duplicít - pred nakladaním sa skontroluje, či už objekt nie je v Mape identít; ak áno, vráti sa jeho referencia
 - Úschovňa objektov – výber objektu z úschovne je rádovo rýchlejší ako prístup do DB na disk

Mapa identít a úschovňa objektov

- Jednoduchá Mapa identít sa realizuje relatívne ľahko, napr. HashMap
 - Pre každú DB tabuľku - jedna Mapa identít
- Ďalšie požiadavky:
 - Dedenie
 - 1 Mapa identít pre transakciu
 - 1 Mapa identít pre viacero transakcií (niekedy)
 - Výhodná pre objekty, ktoré sa využívajú na čítanie alebo sa na nich vykonáva iba málo zmien
- => Súčasťou nástrojov pre ORM

Lenivé nakladanie

- Štandardne: nedočkavé nakladanie (eager loading)
- Nakladá sa aj:
 - Hodnota atribútu je typu BLOB
 - Všetky asociované objekty 1:1, n:1
- Často stačí naloženie, až keď sa pristupuje k hodnote a asociovaným objektom
- V JPA je možné si deklaratívne nastaviť lenivé nakladanie (lazy load)

Lenivé nakladanie v JPA

- Zmena defaultového nedočkavého nakladania na lenivé nakladanie:

```
@Entity  
public class Klient {  
    ...  
    @Lob  
    @Basic (fetch=FetchType.LAZY)  
    public byte[] foto;  
    ...}
```

Lenivé nakladanie v JPA

- Zmena defaultového lenivého nakladania na nedočkavé nakladanie:

@Entity

```
public class Ucet {
```

```
    @Id
```

```
    public String idUctu;
```

```
    public String stavUctu;
```

```
    public Klient klient;
```

```
    @ManyToMany (fetch= FetchType.EAGER,  
                                                         mappedBy="ucty")
```

```
    public Set<Klient> klienti;
```

```
}
```