

Viac entít
Otvárame 2. okno

UINF/PAZ_{1c}
6. prednáška



Vzťahy tried a entít

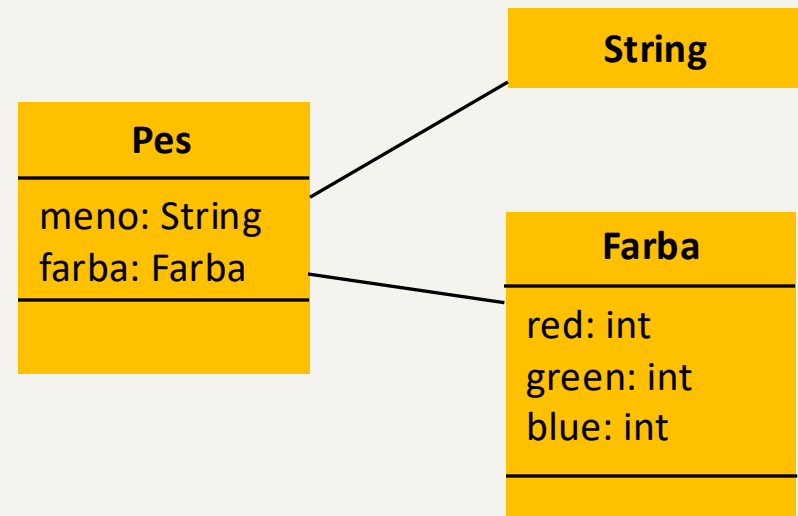


- Veľa druhov vzťahov
 - má vlastnosť, potrebuje, dedí (rozširuje), obsahuje / je časťou, súvisí, ...
- Veľa spôsobov modelovania (kreslenia)
 - UML, ontológie, ...
- Reprezentácia v rôznych jazykoch
 - OOP, SQL, OWL, ...

Vzťah „má vlastnosť“ / „má stav“

Pes **MÁ** meno (typ String)
Pes **MÁ** farbu (typ Color)

```
class Farba {  
    int red  
    int green  
    int blue  
}
```



```
class Pes {  
    String meno  
    Color farba  
}
```

```
class String {  
    ...  
}
```

Vzťah „závisí na“ (dependency)



- typické pre moduly – nie pre entity
- trieda **potrebuje** pre svoje **fungovanie** inú triedu
- **typicky**: metóda objektu nebude fungovať, ak nemá k dispozícii príslušný objekt
- **príklad**: kontrolér pre ukladanie objektu v používateľskom rozhraní potrebuje objekt DAO pre prístup k databáze



Vzťah „závisí na“ (dependency)

- realizácia = mám referenciu na závislosť
 - typicky v inštančnej premennej
- získanie referencie
 - aktívne
 - objekt si vytvorí závislosť - vytvorí potrebný objekt
 - opýta si závislosť od továrne
 - pasívne (dependency injection)
 - dostane závislosť cez konštruktor
 - dostane závislosť cez setter
 - dostane závislosť cez anotáciu+DI framework



Factory

- trieda závisí na inštancii továrne
- jednoduché vytváranie inštancie - stačí prázdny konštruktor
- pre používateľa je trieda samostatnejšia
 - vytvoríme inštanciu a používame

Dependency injection

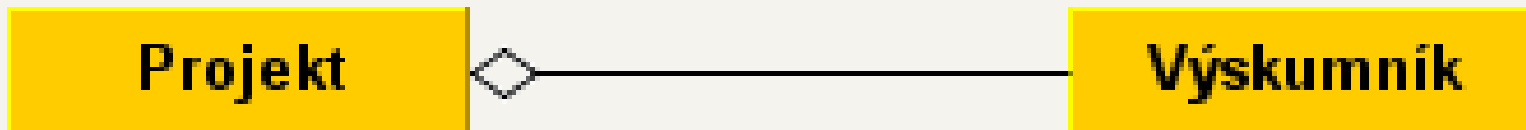
- trieda nezávisí na ničom
- všetky závislosti dáme ako parametre do konštruktora...
- ... alebo cez settre
- ... alebo použijeme `@Autowired` cez Spring
- pre používateľa je trieda flexibilnejšia
 - môžeme nastaviť závislosti zvonku

Oba prístupy sa dajú kombinovať

Vzťah „je časťou“ (agregácia)

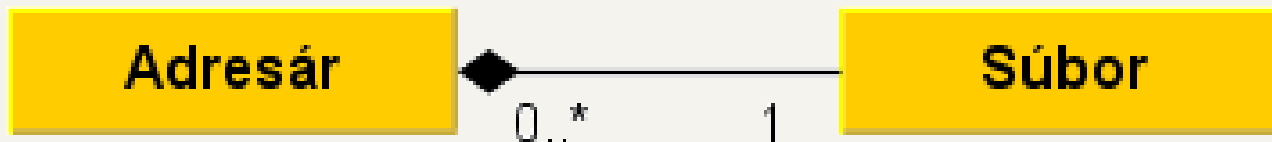
- v UML: agregácia a.k.a. zdieľaná kompozícia
- objekt je **súčasťou** iného objektu
 - dokáže však fungovať samostatne a nezávisle
 - môže byť súčasťou viacerých objektov naraz

```
public class Projekt {  
    private List<Výskumník> riešitelia = new ArrayList<>();  
    ...  
    public void pridajRiešiteľa(Výskumník výskumník) {  
    }  
}
```



Kompozícia

- **kompozícia** a.k.a. silná agregácia
- objekty majú spriahnuté životné cykly
- ak sa vymaže adresár, majú sa vymazať aj podsúbory
- súbor **musí** byť v **nejakom** adresári



Agregácia alebo kompozícia?

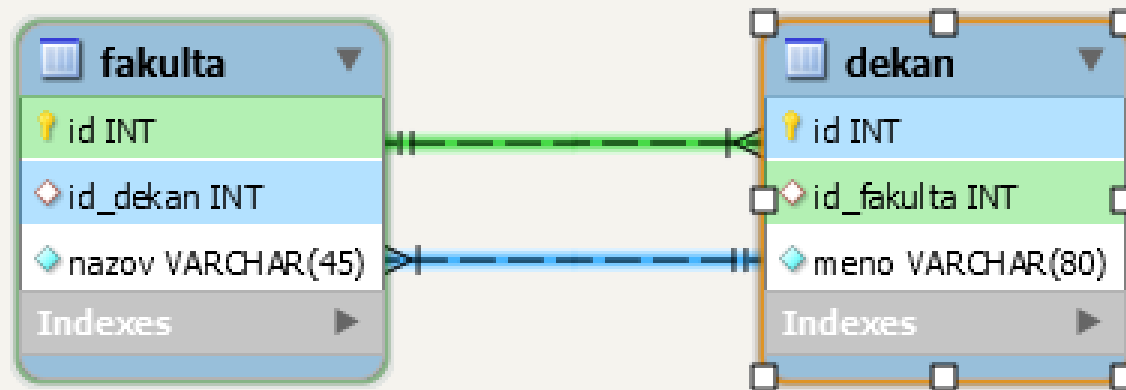
- vec aplikačnej domény
- rozdiel iba v integritných obmedzeniach
 - v databáze môžeme kompozíciu čiastočne ustrážiť cez NOT NULL cudzie kľúče
 - výrobok musí mať odkaz na svojho výrobcu
 - `manufacturer_id NOT NULL FOREIGN KEY`
 - nevieme: výrobca musí mať aspoň 1 výrobok
 - v kóde si všetko musíme ustrážiť sami v metódach
- oba typy vzťahov reprezentované rovnako
 - zdroják: cez inštančné premenné
 - databáza: cez FOREIGN KEY, NOT NULL, UNIQUE ...
- spoločný názov: **asociácie** (vzťahy)

V kóde: asociácia 1:1

```
public class Dekan {  
    private Fakulta fakulta;  
    /*...*/  
}
```

```
public class Fakulta {  
    private Dekan dekan;  
    /*...*/  
}
```

V databáze: asociácia 1:1

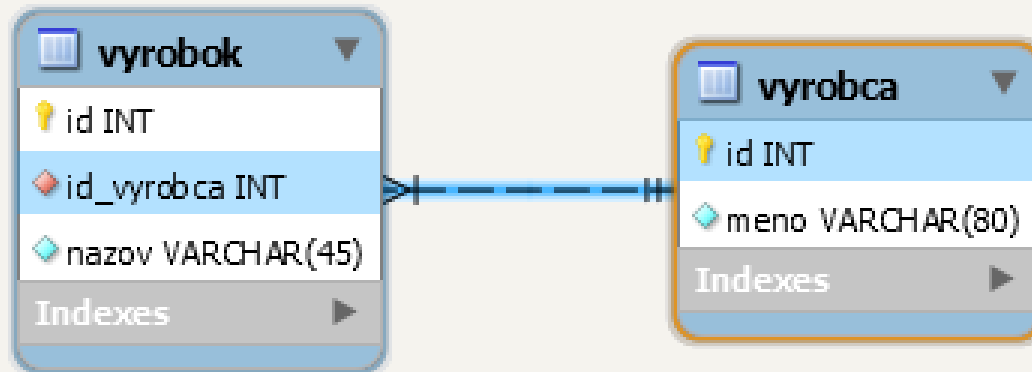


```
CREATE TABLE dekan (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `id_fakulta` INT NOT NULL,  
  `meno` VARCHAR(80) NOT NULL,  
  PRIMARY KEY (`id`),  
  CONSTRAINT `fk_dekan_fakulta` FOREIGN KEY (`id_fakulta`)  
    REFERENCES fakulta (id)  
)
```

Asociácia 1:M

```
public class Výrobok {  
    private Výrobca výrobca;  
    /*...*/  
}
```

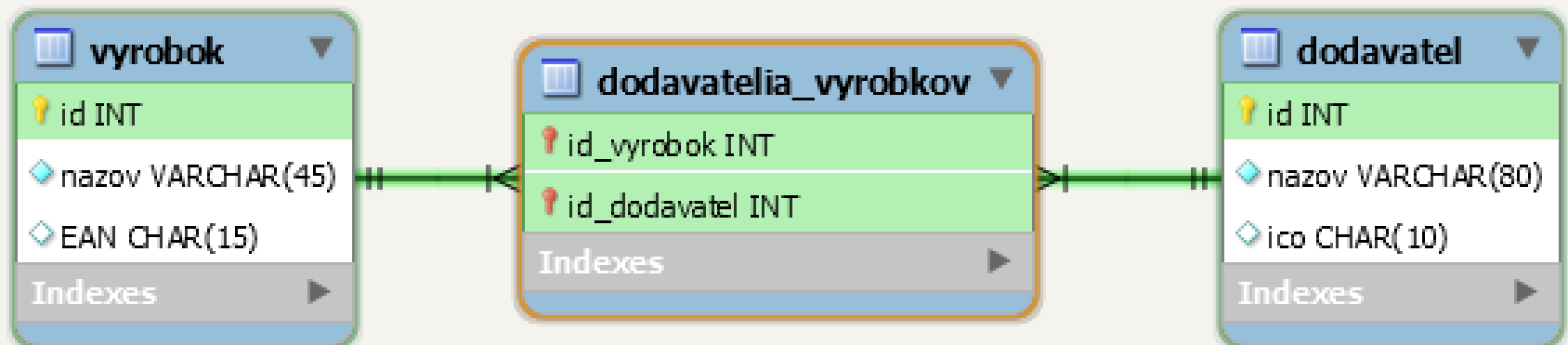
```
public class Výrobca {  
    private List<Výrobok> výrobky;  
    /*...*/  
}
```



Asociácia M:N

```
public class Výrobok {  
    private List< Dodávateľ > dodávatelia;  
    /*...*/  
}
```

```
public class Dodávateľ {  
    private List< Výrobok > výrobky;  
    /*...*/  
}
```



Asociácia M:N s dodatočnými údajmi

```
public class Výrobok {  
    private List< Dodávateľ > dodávatelia;  
    private Map< Dodávateľ, Integer > dodávateliaPočty;  
    /*...*/  
}
```

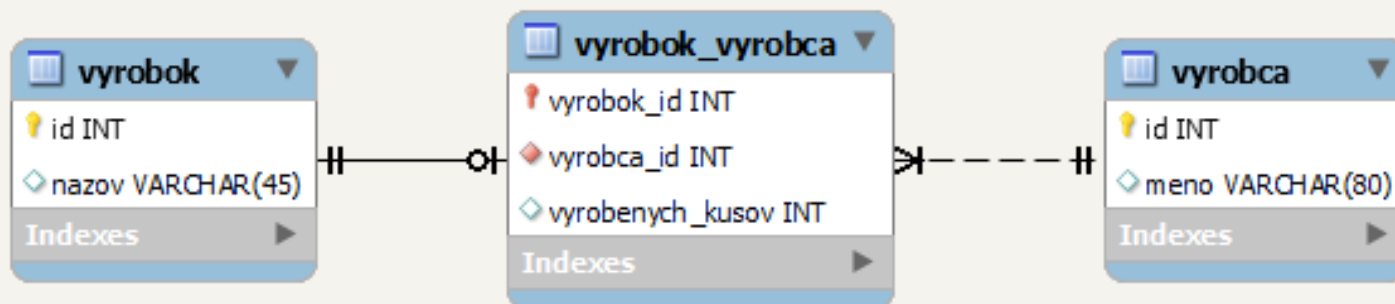
```
public class Dodávateľ {  
    private List< Výrobok > výrobky;  
    private Map< Výrobok, Integer > výrobkyPočty;  
    /*...*/  
}
```



Asociácia 1:M s dodatočnými údajmi

```
public class Výrobok {  
    private Výrobca výrobca;  
    private int vyrobenýchKusov;  
    /*...*/  
}
```

```
public class Výrobca {  
    private List<Výrobok> výrobky;  
    /*...*/  
}
```



Prístupy v celej budove (niekde áno, niekde nie)

- Administrátor

- pridáva, odoberá, (de)aktivuje používateľov

- mení používateľom karty

- pridáva, odoberá miesta s čítačkami - dvere

- prideluje používateľom práva na vstup do konkrétnych dverí

} CRUD

} CRUD

Používatelia vs. dvere

- Môžu jedny dvere pustiť viac ako jedného používateľa
 - Zjavne áno
- Môže jeden používateľ vojsť do viacerých dverí?
 - Ak nie
 - Máme vzťah 1:N
 - Používateľovi napíšeme ďalší parameter: dvere, do ktorých môže vojsť
 - Ak áno
 - Máme vzťah M:N
 - Reálnejšie (zvolíme radšej aj vtedy, keď zákazník tvrdí prvé)



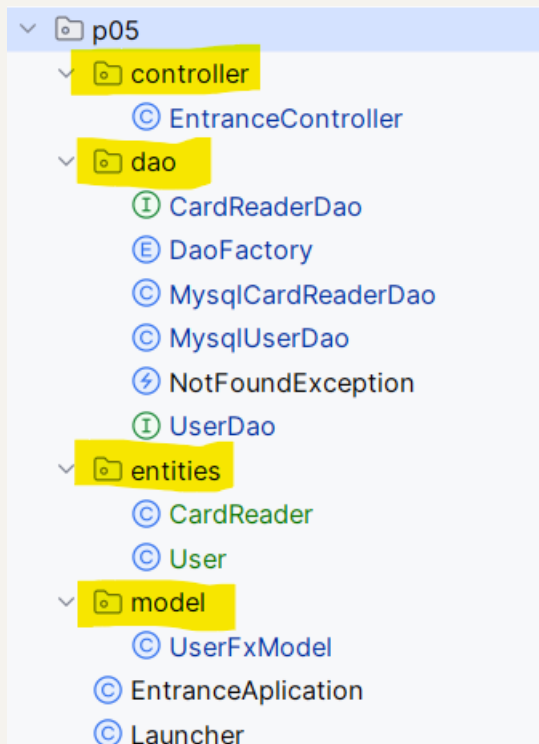
Modelujme

ChipReader je entita

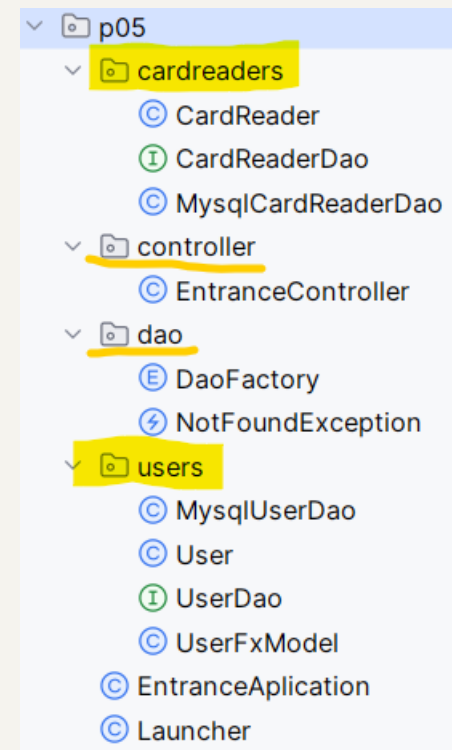
- zaslúži si vlastné ChipReaderDao
- vlastnú obrazovku na realizáciu CRUD operácií

Rozdelenie kódu do balíkov

Podľa schopností
(Clean/Onion Architecture)



Podľa entít
(Vertical Slice Architecture)



Rozdelenie kódu do balíkov

Podľa schopností (Clean/Onion Architecture)

- Štandard v Java projektoch
- Neškáluje s veľkosťou projektu
 - Zopár balíkov so stovkami tried
- Java sa používa pri obrovských SW projektoch, napriek tomu Java SW architekti používajú tento prístup

Podľa entít (Vertical Slice Architecture)

- Používanéjšie v iných jazykoch
 - U mňa Go, TS (React), Rust
 - Nemá ale monopol
- Viac menších balíkov
 - Prehľadnejšia štruktúra
- Modulárnejšia appka

Do akých dverí môže vstúpiť používateľ?

- Asi častá otázka
- Natahujem rovno s každým používateľom z DB

```
SELECT u.id AS idu, u.name, u.chipId, u.active,  
       cr.id AS idcr, cr.room  
FROM user u  
LEFT JOIN users_chip_readers ucr ON u.id = ucr.user_id  
LEFT JOIN chip_reader cr ON ucr.chip_reader_id = cr.id_user
```

idu	name	chipId	active	idcr	room
1	Jano	cc1f86ab86	0	2	SA1Co3
2	Fero	fe265d741a	1	1	SJ2P10
2	Fero	fe265d741a	1	2	SA1Co3

Do akých dverí môže vstúpiť používateľ?

- Alternatíva - v SELECTE mám iba ID čítačiek
 - Pre každý riadok robím dodatočný SELECT
 - Málokedy vhodné - "N+1 SELECT problem"

```
SELECT u.id AS idu, u.name, u.chipId, u.active,  
       ucr.id AS idcr, cr.room  
FROM user u JOIN users_chip_readers cur ON u.id = ucr.users_id
```

idu	name	chipId	active	idcr
1	Jano	cc1f86ab86	0	2
2	Fero	fe265d741a	1	1
2	Fero	fe265d741a	1	2

SELECT ... FROM
chip_reader
WHERE id = 2

čítačky sa často opakujú, dotiahneme ďalším selektom

ResultSetExtractor

idu	name	chipId	active	idcr	room
1	Jano	cc1f86ab86	0	2	SA1C03
2	Fero	fe265d741a	1	1	SJ2P10
2	Fero	fe265d741a	1	2	SA1C03

- RowMapper by vrátil pre každý riadok nového používateľa
- ResultSetExtractor vráti jeden objekt pre celý výsledok selektu

ResultSetExtractor

```
ResultSetExtractor<List<User>> resultSetExtractor = rs -> {  
    var result = new ArrayList<User>();  
    var processed = new HashMap<Long, User>();  
  
    while (rs.next()) {  
        // spracovanie 1 riadku  
    }  
  
    return result;  
};
```

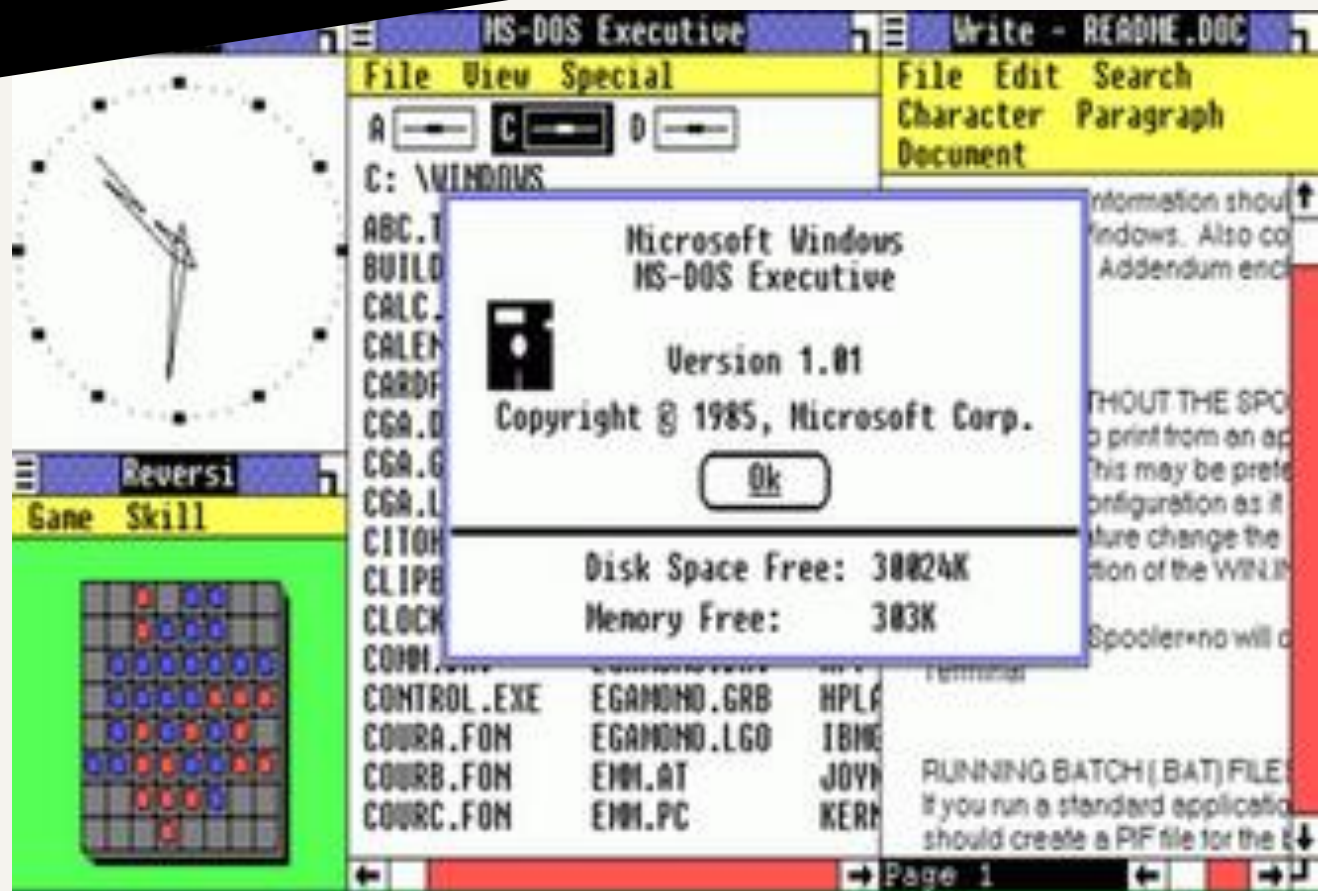
ResultSetExtractor – prvotne spracovanie 1 riadku

```
var id = rs.getLong("u_id"); // do SELECTu s JOINom davame aliasy  
na stlpce  
var user = processed.get(id);  
// ak user este nebol sprocesovany, tak vytvorime novy user objekt  
if (user == null) {  
    user = new User(  
        rs.getLong("u_id"),  
        ...,  
        new ArrayList<>() // spociatku prazdny zoznam citaciek  
    );  
  
    processed.put(id, user);  
    result.add(user);  
}
```

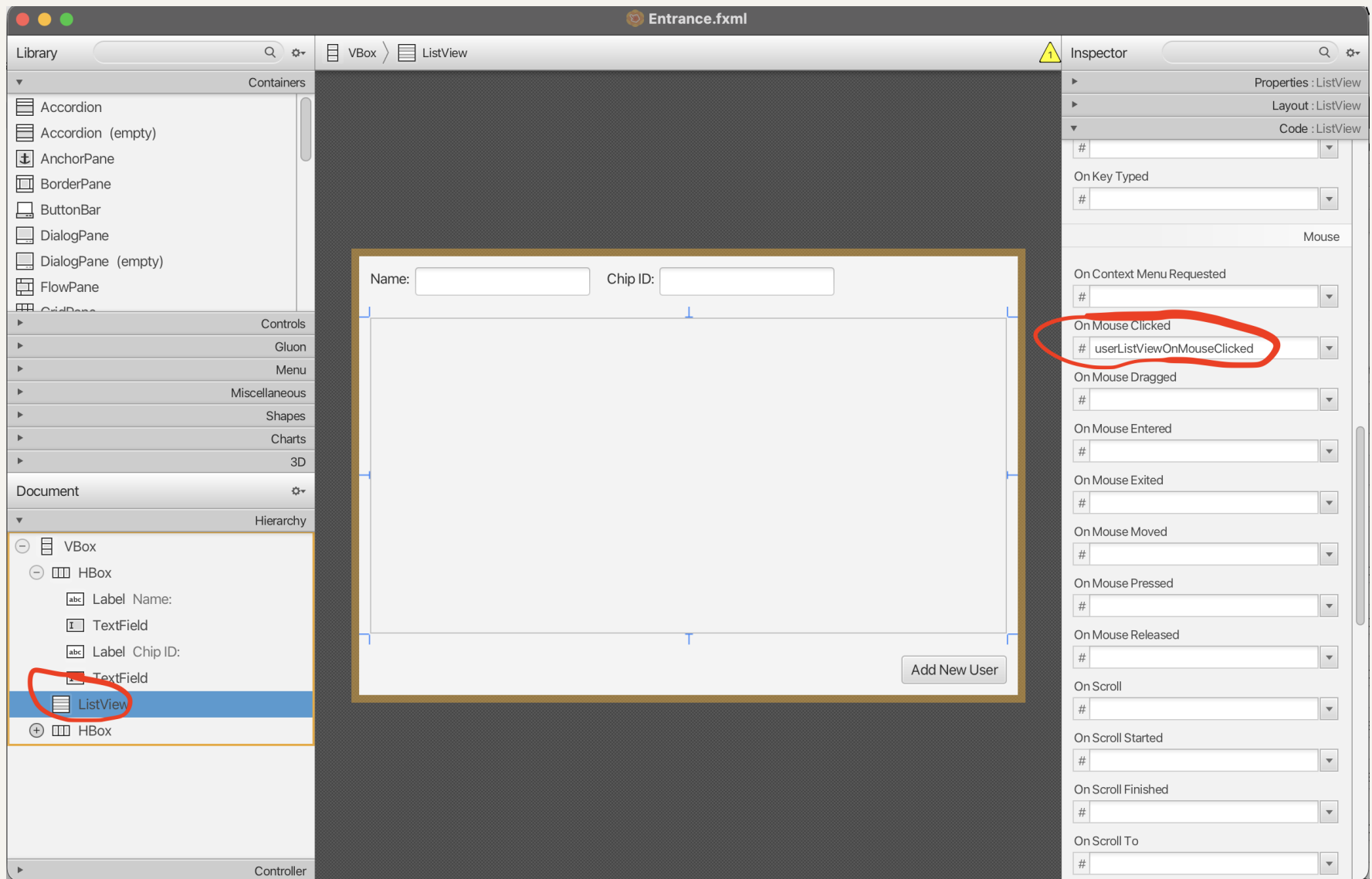
ResultSetExtractor – opakované spracovanie 1 riadku

```
// do uz n-krat spracovaneho usera dodame iba novy chip_reader  
var chipReaderId = rs.getLong("cr_id");  
if (chipReaderId != 0){  
    var chipReader = new ChipReader(  
        rs.getLong("cr_id"),  
        rs.getString("cr_door")  
    );  
  
    user.access().add(chipReader);  
}
```

Priradzovanie čítačiek užívateľom v 2. okne



EntranceWindow



EntranceController

- Dvojitým kliknutím na užívateľa v hlavnom okne otvoríme nové okno

@FXML

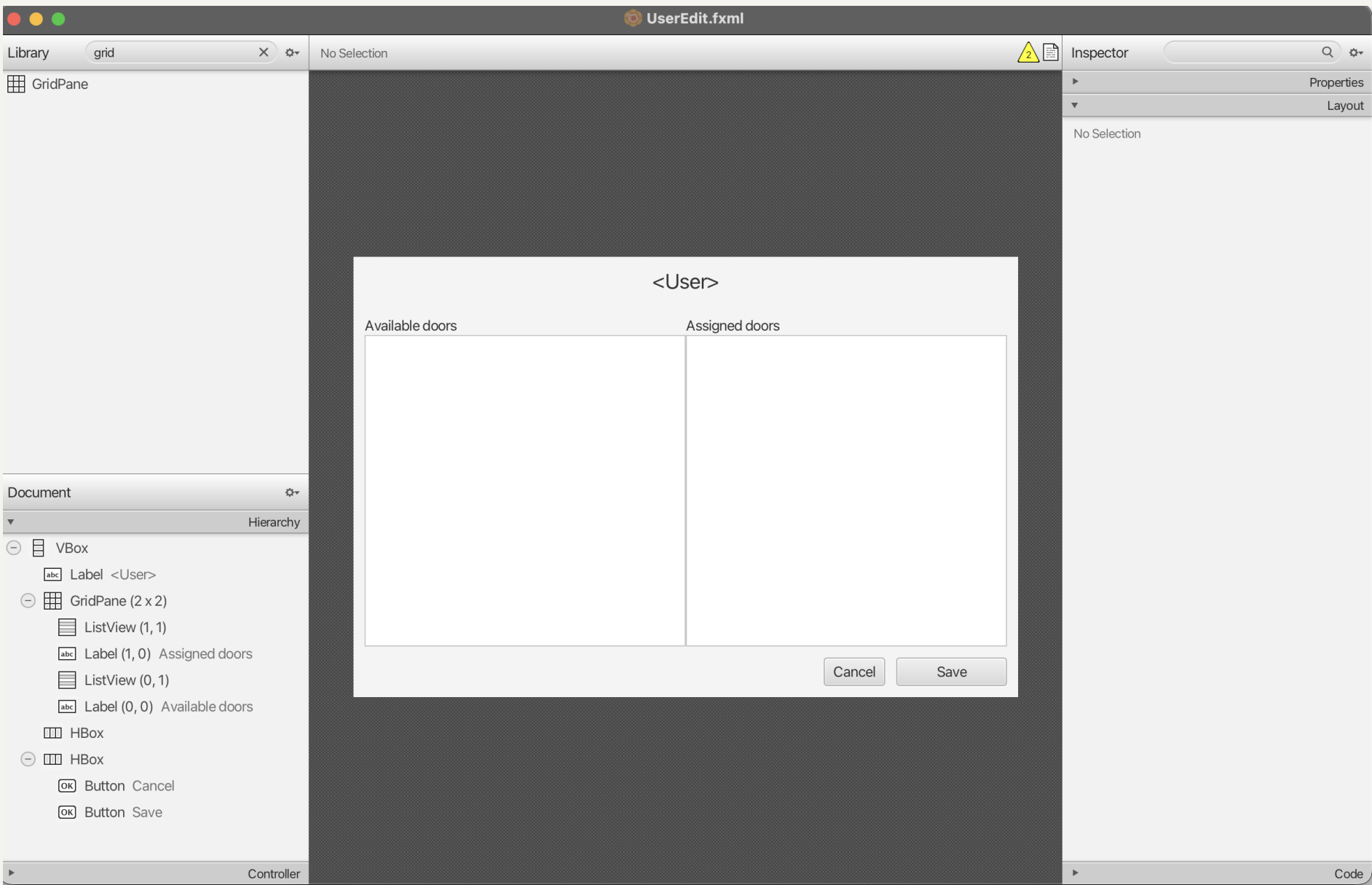
```
void userListViewOnMouseClicked(MouseEvent event) {  
    if (event.getClickCount() == 2) {  
        var selectedUser = this.userListView  
            .getSelectionModel()  
            .getSelectedItem();  
        this.openUserEditWindow();  
    }  
}
```

```
private void openUserEditWindow(User editedUser) {
    var loader = new
FXMLLoader(getClass().getResource("UserEdit.fxml"));
    Parent pane;
    try {
        pane = loader.load();
    } catch (IOException e) {
        throw new RuntimeException("Error", e);
    }
    var controller = (UserEditController) loader.getController()
Controller.setEditedUser(editedUser);
    var scene = new Scene(pane);
    var stage = new Stage();
    stage.initModality(Modality.APPLICATION_MODAL);
    stage.setTitle("Edit User " + user.name());
    stage.setScene(scene);

    stage.showAndWait();

    userListView.setItems(FXCollections.observableArrayList(userDao.findAll
()));
}
```

UserEdit.fxml



UserEditController - DAO

```
public class UserEditController {  
  
    private final ChipReaderDao chipReaderDao =  
Factory.INSTANCE.getChipReaderDao();  
  
    private final UserDao userDao = Factory.INSTANCE.getUserDao();  
  
    ....  
}
```

```
private User editedUser;
```

```
public void setEditedUser(User editedUser) {  
    this.editedUser = editedUser;  
    this.userInfoLabel.setText(editedUser.name());
```

Inicializujeme už
nastavené přístupy
k čítačkám

```
    this.assignedChipReadersListView.setItems(  
        FXCollections.observableArrayList(this.editedUser.access()  
    );
```

```
    var allChipReaders = this.chipReaderDao.findAll();  
    var availableChipReaders = allChipReaders.stream()  
        .filter(cr -> !this.editedUser.access().contains(cr))  
        .toList();
```

Inicializujeme
dostupné přístupy k
čítačkám

```
    this.availableChipReadersListView.setItems(  
        FXCollections.observableArrayList(availableChipReaders)  
    );
```

```
}
```

UserEditController -

Dvojitý klik na ľavý zoznam (dostupných)
presunie položku do pravého (priradených)

@FXML

```
void availableReadersListView(MouseEvent event) {  
    if(event.getClickCount() != 2) {  
        return;  
    }  
  
    var selectedChipReaders = this.availableChipReadersListView  
        .getSelectionModel().getSelectedItem();  
  
    this.assignedChipReadersListView.getItems().addAll(selectedChipReaders);  
    this.availableChipReadersListView.getItems().removeAll(selectedChipReaders);  
}
```

UserEditController -

Dvojitý klik na pravý zoznam (priradených)
presunie položku do ľavého (dostupných)

@FXML

```
void assignedChipReadersListView(MouseEvent event) {  
    if(event.getClickCount() != 2) {  
        return;  
    }  
  
    var selectedChipReaders = this.assignedChipReadersListView  
        .getSelectionModel().getSelectedItem();  
  
    this.availableChipReadersListView.getItems().addAll(selectedChipReaders);  
    this.assignedChipReadersListView.getItems().removeAll(selectedChipReaders);  
}
```

UserEditController -

Cancel button iba zavrie okno
(metóda a spôsob jej volania je dosť nešťastný)

```
@FXML
```

```
void cancelButtonAction(ActionEvent event) {  
    saveButton.getScene().getWindow().hide();  
}
```

UserEditController -

Save button zavolá DAO a následne zavrie okno

```
@FXML
void saveButtonAction(ActionEvent event) {
    try {
        this.userDao.update(
            this.editedUser.withAccess(this.assignedChipReadersListView.getItems())
        );
    } catch (BadRequestException e) {
        var alert = new Alert(Alert.AlertType.ERROR, "Invalid user fields: " + e.getMessage());
        alert.showAndWait();
        return;
    } catch (NotFoundException e) {
        var alert = new Alert(Alert.AlertType.ERROR, "User not found: " + e.getMessage());
        alert.showAndWait();
        return;
    }
    saveButton.getScene().getWindow().hide();
}
```

Otázky?