Viac entít
Otvárame 2. okno

UINF/PAZ1c

6. prednáška



Vzťahy tried a entít

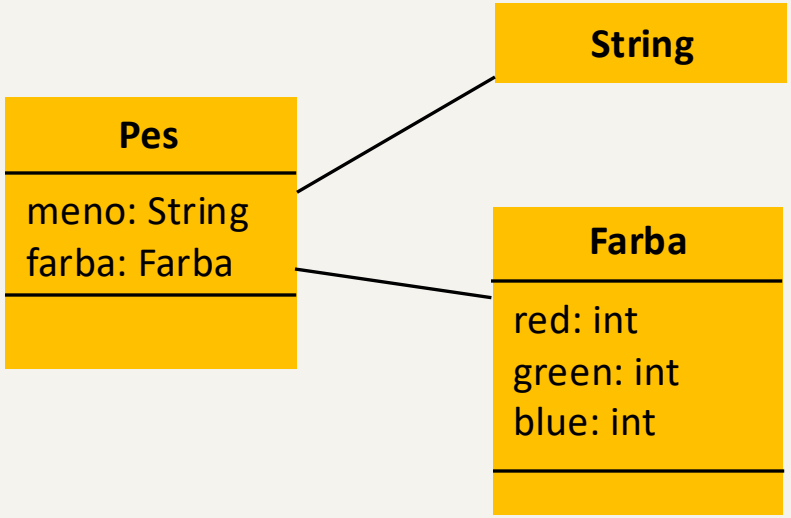


- Veľa druhov vzťahov
  - má vlastnosť, potrebuje, dedí (rozširuje), obsahuje / je časťou, súvisí, ...
- Veľa spôsobov modelovania (kreslenia)
  - UML, ontológie, ...
- Reprezentácia v rôznych jazykoch
  - OOP, SQL, OWL, ...

Vzťah „má vlastnosť“ / „má stav“

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

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



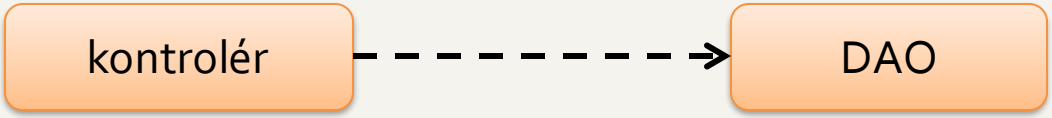

```
class Pes {
   String meno
   Color farba
}
```

```
class String {
    …
}
```

Vzťah „závisí na“ (dependency)

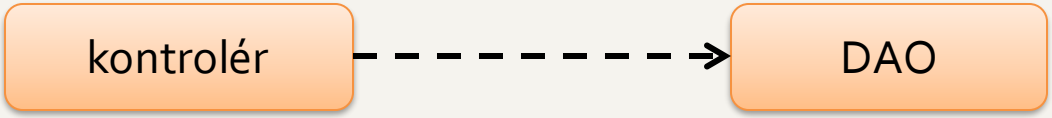


- typické pre moduly – nie pre entity
- trieda **potrebuje** pre svoje **fungovanie** inú triedu
- **typicky:** metóda objektu nebude fungovať, ak nemá k dispozícii príslušný objekt
- **príklad:** kontrolér pre ukladanie objektu v používateľskom rozhraní potrebuje objekt DAO pre prístup k databáze



Vzťah „závisí na“ (dependency)

- realizácia = mám referenciu na závislosť
  - typicky v inštančnej premennej
- získanie referencie
  - Aktívne
    - objekt A si sám vytvorí potrebný objekt B
    - alebo si A opýta závislosť od továrne
  - Pasívne - dependency injection
    - dostane závislosť cez konštruktor
    - dostane závislosť cez setter
    - dostane závislosť cez anotáciu+DI framework



Vzťah „je časťou" (asociácia)

- Delí sa na agregáciu a kompozíciu
  - Agregácia - Entita **A** môže existovať aj bez **B**
  - Kompozícia - Entita **A** nedokáže bez **B** existovať
- Reprezentovanie
  - zdroják: cez inštančné premenné (podobne ako dependency, ale opačne)
  - databáza: cez FOREIGN KEY

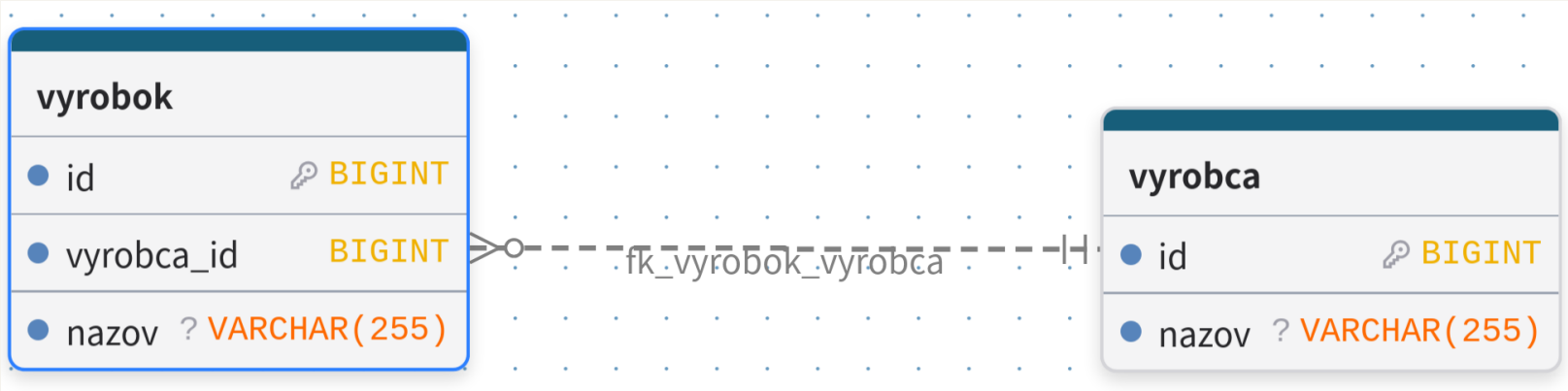
Agregácia vs kompozícia

- 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

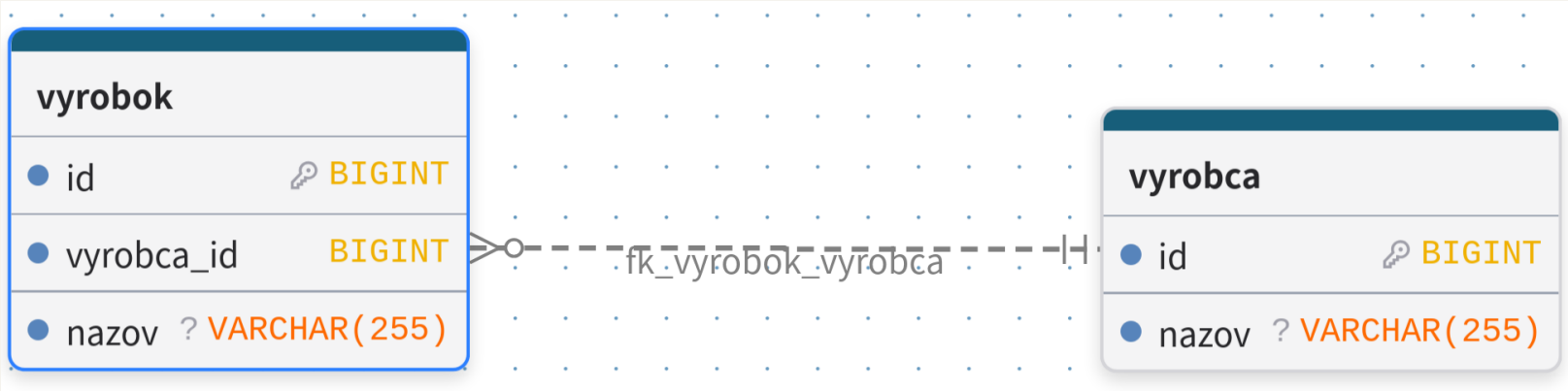
V kóde: asociácia 1:M

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

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



V databáze: asociácia 1:M

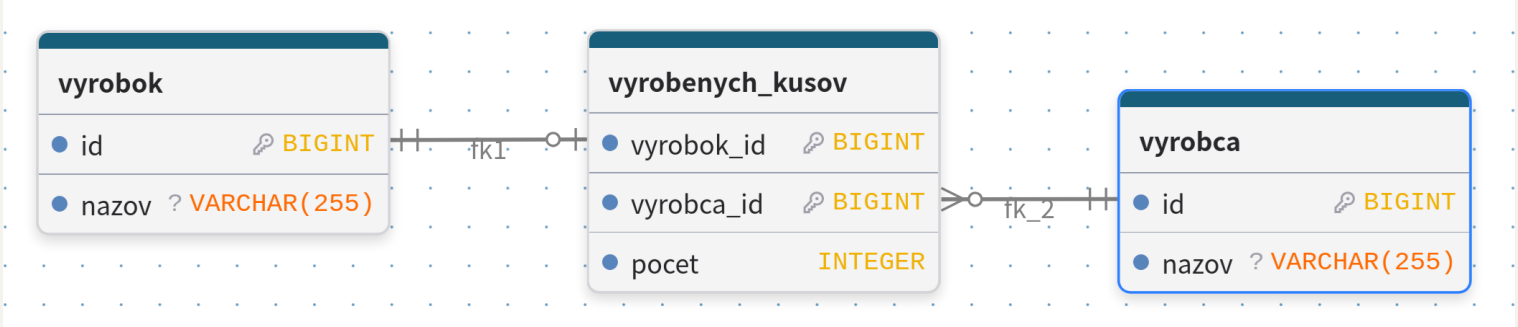



```
CREATE TABLE vyrobok (
 `id` INT NOT NULL AUTO_INCREMENT,
 `id_vyrobca` INT NOT NULL,
 `nazov` VARCHAR(80) NOT NULL,
 PRIMARY KEY (`id`),
 CONSTRAINT `fk_vyrobok_vyrobca`
    FOREIGN KEY (`id_vyrobok`)  REFERENCES vyrobca(id)
)
```

Asociácia 1:M s dodatočnými údajmi

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

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

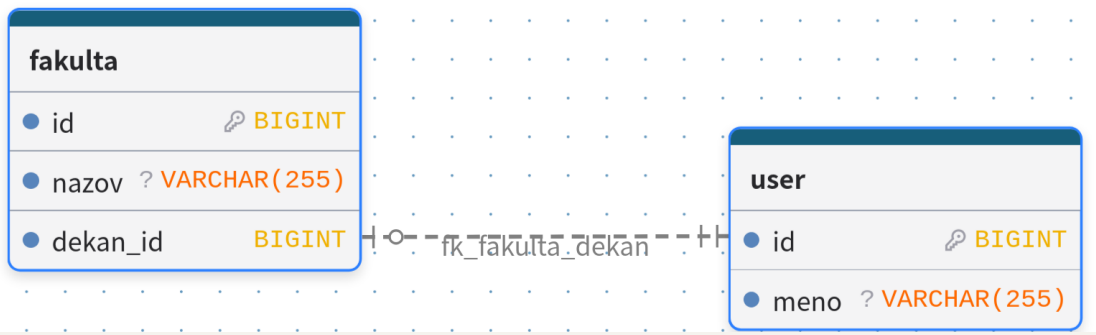


V kóde: asociácia 1:1

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

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

V databáze: asociácia 1:1

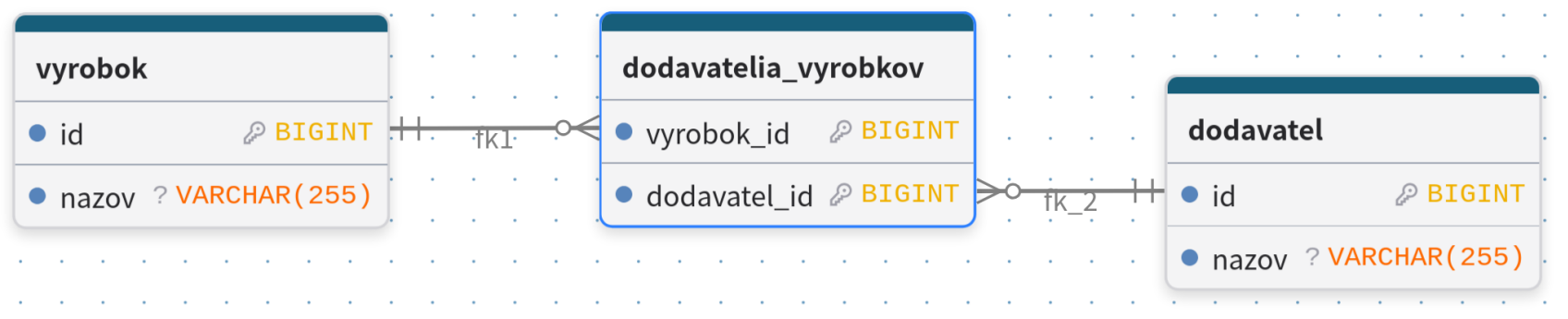



```
CREATE TABLE fakulta (
  `id` INT NOT NULL AUTO_INCREMENT,
  `id_dekan` INT NOT NULL UNIQUE,
  `nazov` VARCHAR(80) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_fakulta_dekan`  FOREIGN KEY (`id_dekan`)
      REFERENCES user (id)
)
```

Asociácia M:N

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

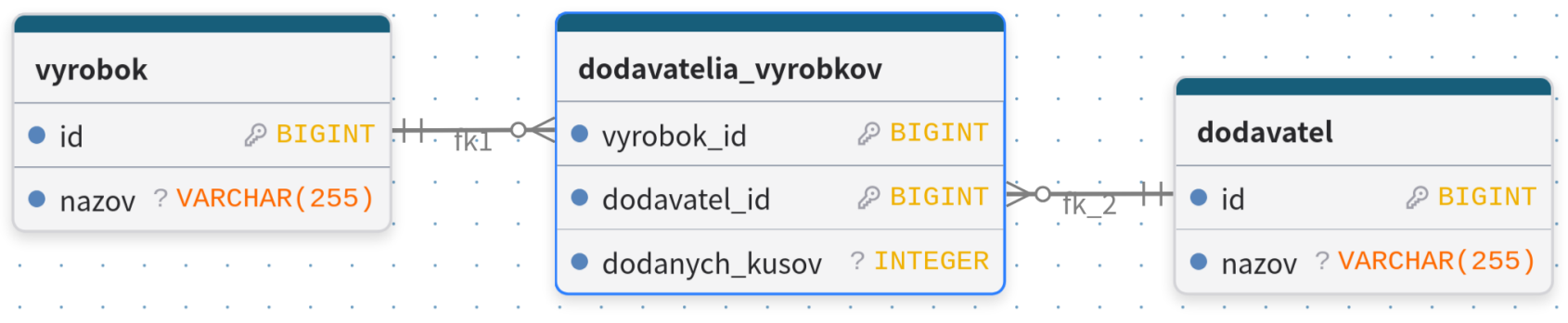
```
public class Dodávateľ {
  private List<Výrobok> výrobky;
 /*...*/
}
```



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;
  /*...*/
}
```

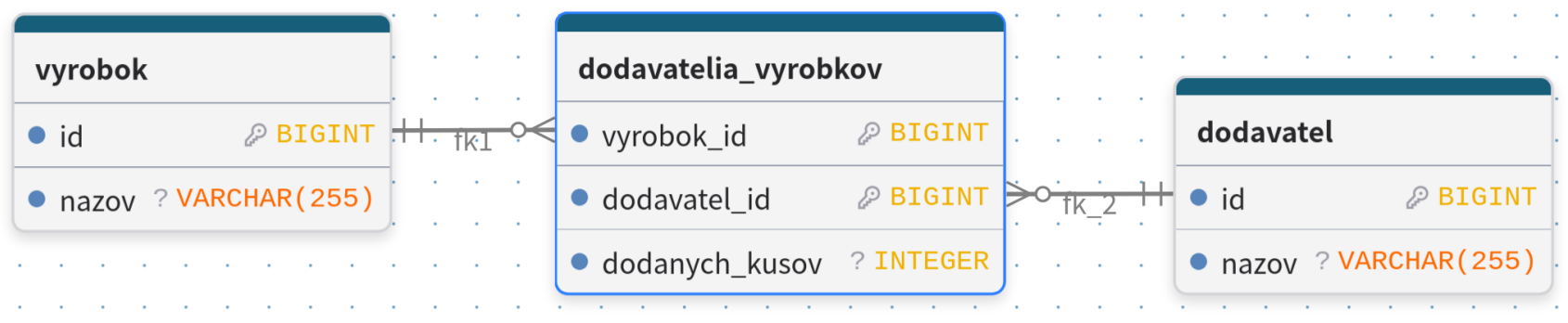


Asociácia M:N s dodatočnými údajmi

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

Vhodný kandidát na novú samostatnú entitu.

1. **Stačí vhodne pomenovať.**
2. **Dať vlastné ID.**




Modelujme

Základná asociácia (1:M)

TABLE **X** FOREIGN KEY y_id REFERENCES **Y**(id)

- Vytvorí vzťah 1:M medzi **X** a **Y**
  - Každý riadok z **X môže** mať v sebe **jedno** id z **Y**
  - Id z **Y** sa **môžu** nachádzať **viackrát** krát v tabuľke **X**

- Kardinalita **jeden** (pri **Y**) **až mnoho** (pri **X**)
- Účasť je **obojstranne voliteľná**



Asociácia (1:M) s povinnou účasťou ("polo"-kompozícia)

TABLE **X** FOREIGN KEY y_id REFERENCES **Y**(id),
kde y_id je **NOT NULL**

- Vytvorí vzťah 1:M medzi **X** a **Y**
  - Každý riadok z **X** **musí** mať v sebe **práve jedno** id z **Y**
  - Id z **Y** sa **môžu** nachádzať **viackrát** krát v tabuľke **X**

- Kardinalita **jeden** (pri **Y**) **až mnoho** (pri **X**)
- Účasť je **jednostranne povinná** pri **Y**



Jedinečná asociácia (1:1)

TABLE **X** FOREIGN KEY y_id REFERENCES **Y**(id),
kde y_id je **UNIQUE**

- Vytvorí vzťah 1:1 medzi **X** a **Y**
  - Každý riadok z **X môže** mať v sebe **jedno** id z **Y**
  - Id z **Y** sa **môžu** nachádzať **jedenkrát** krát v tabuľke **X**

- Kardinalita **jeden** (pri **Y**) **až jeden** (pri **X**)
- Účasť **voliteľná**



Jedinečná asociácia (1:1) s povinnou účasťou

TABLE **X** FOREIGN KEY y_id REFERENCES **Y**(id),
kde y_id je **UNIQUE** aj **NOT NULL**

- Vytvorí vzťah 1:1 medzi **X** a **Y**
  - Každý riadok z **X** **musí** mať v sebe **práve jedno** id z **Y**
  - Id z **Y** sa **môžu** nachádzať **jedenkrát** krát v tabuľke **X**

- Kardinalita **jeden** (pri **Y**) **až** **jeden** (pri **X**)
- Účasť je **jednostranne povinná** pri **Y**

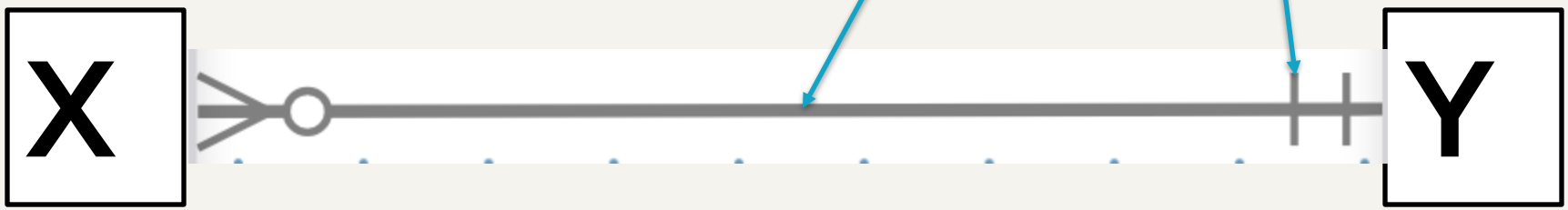


Identifikujúca asociácia 1:M

X FOREIGN KEY y_id REFERENCES **Y**(id),
kde y_id je časťou PRIMARY KEY

- Vytvorí vzťah 1:M medzi **X** a **Y**
  - Každý riadok z **X** **musí** mať v sebe **práve jedno** id z **Y**
  - Id z **Y** sa **môžu** nachádzať **viackrát** krát v tabuľke **X**

- Kardinalita **jeden** (pri **Y**) **až mnoho** (pri **X**)
- Účasť je **jednostranne povinná** pri **Y** (y_id aj NOT NULL)

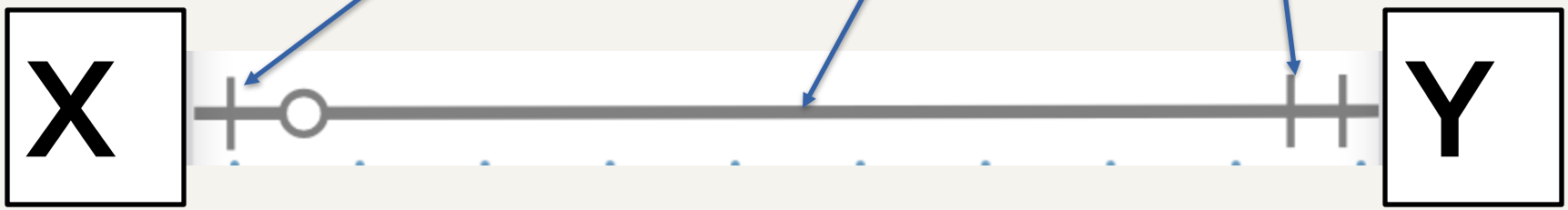


Identifikujúca asociácia 1:1

X FOREIGN KEY y_id REFERENCES **Y**(id),
kde y_id je PRIMARY KEY

- Vytvorí vzťah 1:1 medzi **X** a **Y**
  - Každý riadok z **X** **musí** mať v sebe **práve jedno** id z **Y**
  - Id z **Y** sa **môžu** nachádzať **jedenkrát** krát v tabuľke **X**

- Kardinalita **jeden** (pri **Y**) **až** **jeden** (pri **X**)
- Účasť je **jednostranne povinná** pri **Y** (y_id aj NOT NULL)



Limity integritných obmedzení

- Cez integritné obmedzenia nevieme meniť
  - Účasť pri **X** (z 0 na 1)
  - Kardinalitu pri **Y** (z 1 na mnoho)
- Účasť pri **X** nutné riešiť v kóde, alebo cez TRIGGER
- Kardinalitu **mnoho** pri **Y** riešiť vzťahom M:N cez pomocnú tabuľku



Pozor na zacyklenie entít

- napr. **Dodávateľ** si pamätá svoje **Výrobky**, a **Výrobok** svojich **Dodávateľov**
- mnohé automatické serializácie zlyhávajú, je potrebné ich štelovať, alebo to robiť ručne

Pozor na zacyklenie entít

- vyberieme si 1 **"dominantnú"** stranu vzťahu a druhú ani nenačítavame z DB, ...
- ... alebo možno ešte lepšie - rovno odstránime cyklus z kódu
  - napr. trieda **Výrobok** si bude držať zoznam **Dodávateľov**,
    ale z triedy **Dodávateľ** dáme preč zoznam **Výrobkov**,

Prístupy v celej budove (niekde áno, niekde nie)

- Administrátor
  - pridáva, odoberá, (de)aktivuje používateľov
  - mení používateľom karty
  - pridáva, odoberá miesta s čítačkami - dvere
  - prideľuje 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)

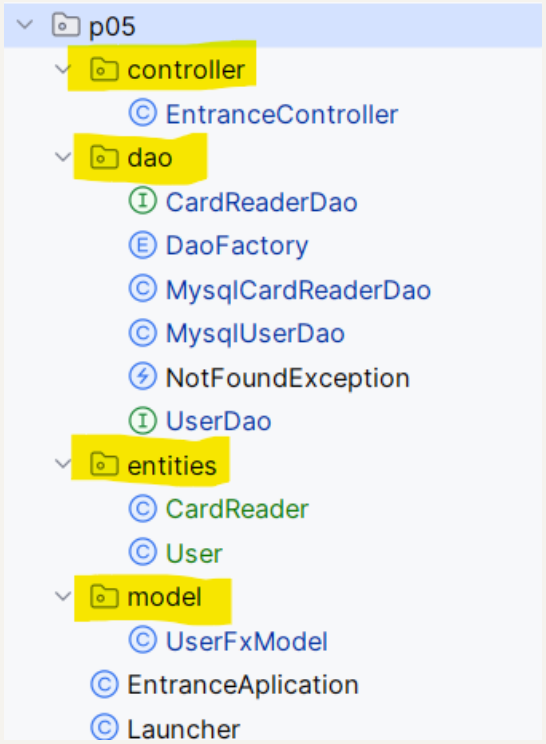
ChipReader je entita

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

Rozdelenie kódu do balíkov

Podľa schopností

(Clean/Onion Architecture)

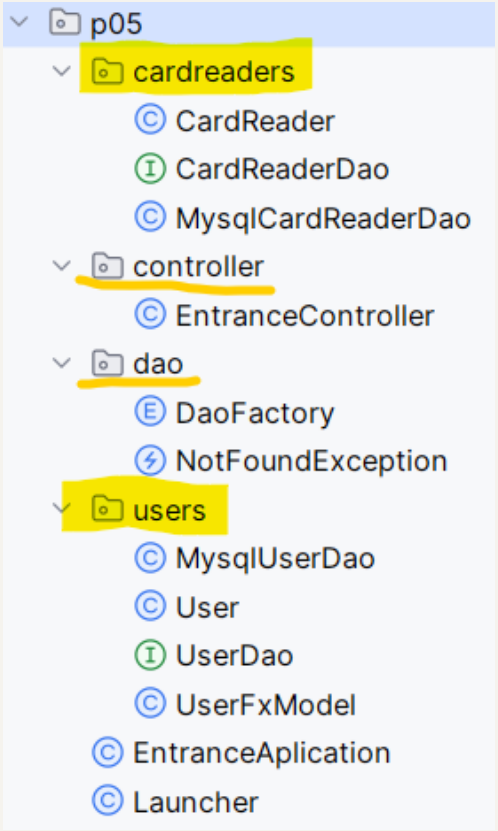


- Štandard v Java projektoch
- Neškáluje s veľkosťou projektu
  - Zopár balíkov so stovkami tried
- Java sa používa pri obrovských SW projektoch, napriek tomu Java SW architekti používajú tento prístup

Rozdelenie kódu do balíkov

Podľa entít

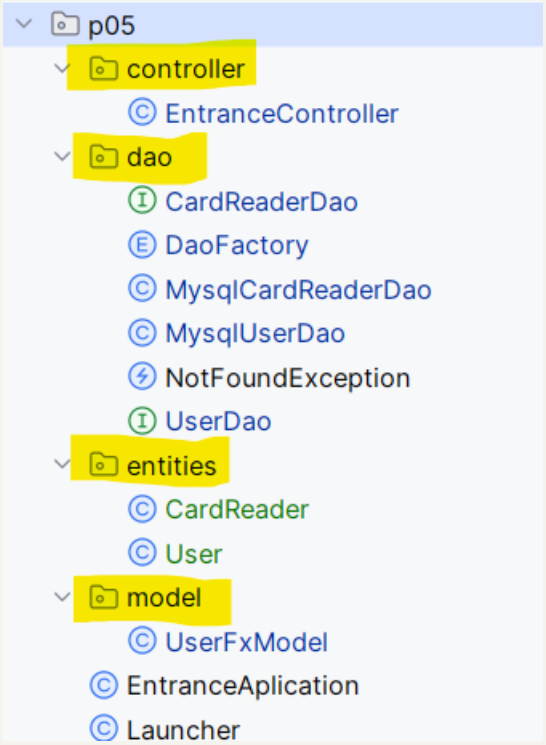
(Vertical Slice Architecture)



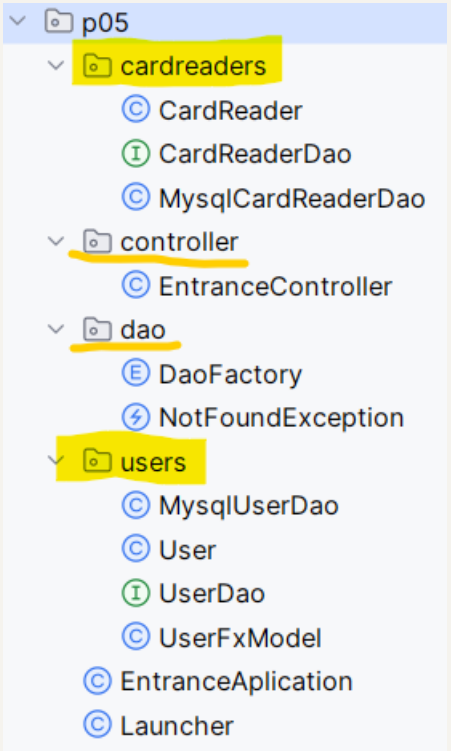
- Používanejšie v iných jazykoch
  - U mňa Go, TS (React), Rust
  - Nemá ale monopol
- Viac menších balíkov
  - Prehľadnejšia štruktúra
- Modulárnejšia appka

Rozdelenie kódu do balíkov

Podľa schopností
(Clean/Onion Architecture)



Podľa entít
(Vertical Slice Architecture)



Do akých dverí môže vstúpiť používateľ?

- Asi častá otázka
- Naťahujem 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	SA1C03
2	Fero	fe265d741a	1	1	SJ2P10
2	Fero	fe265d741a	1	2	SA1C03

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
    );

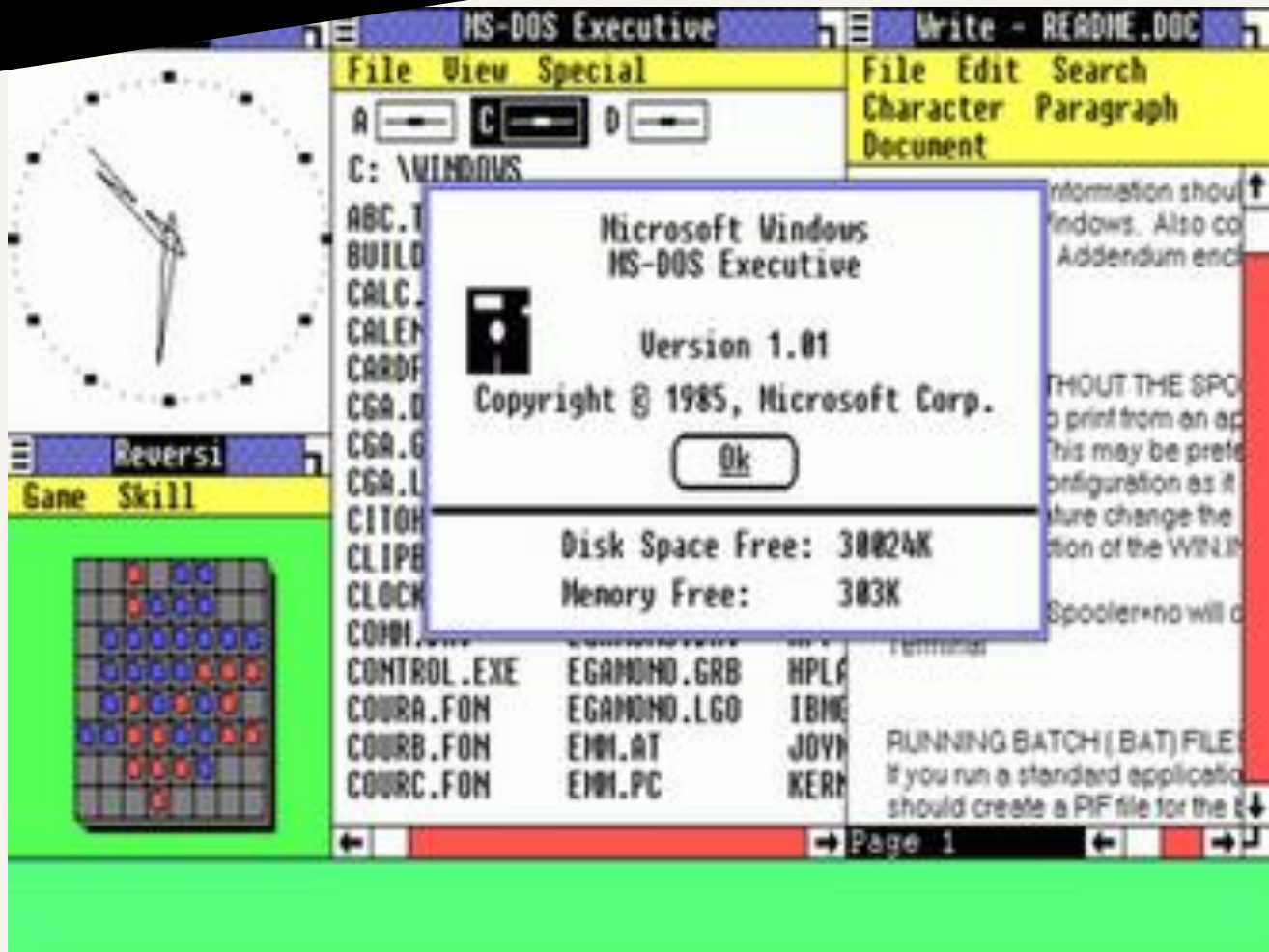
    processed.put(id, user);
    result.add(user);
}
```

ResultSetExtractor – opakovane spracovanie 1 riadku

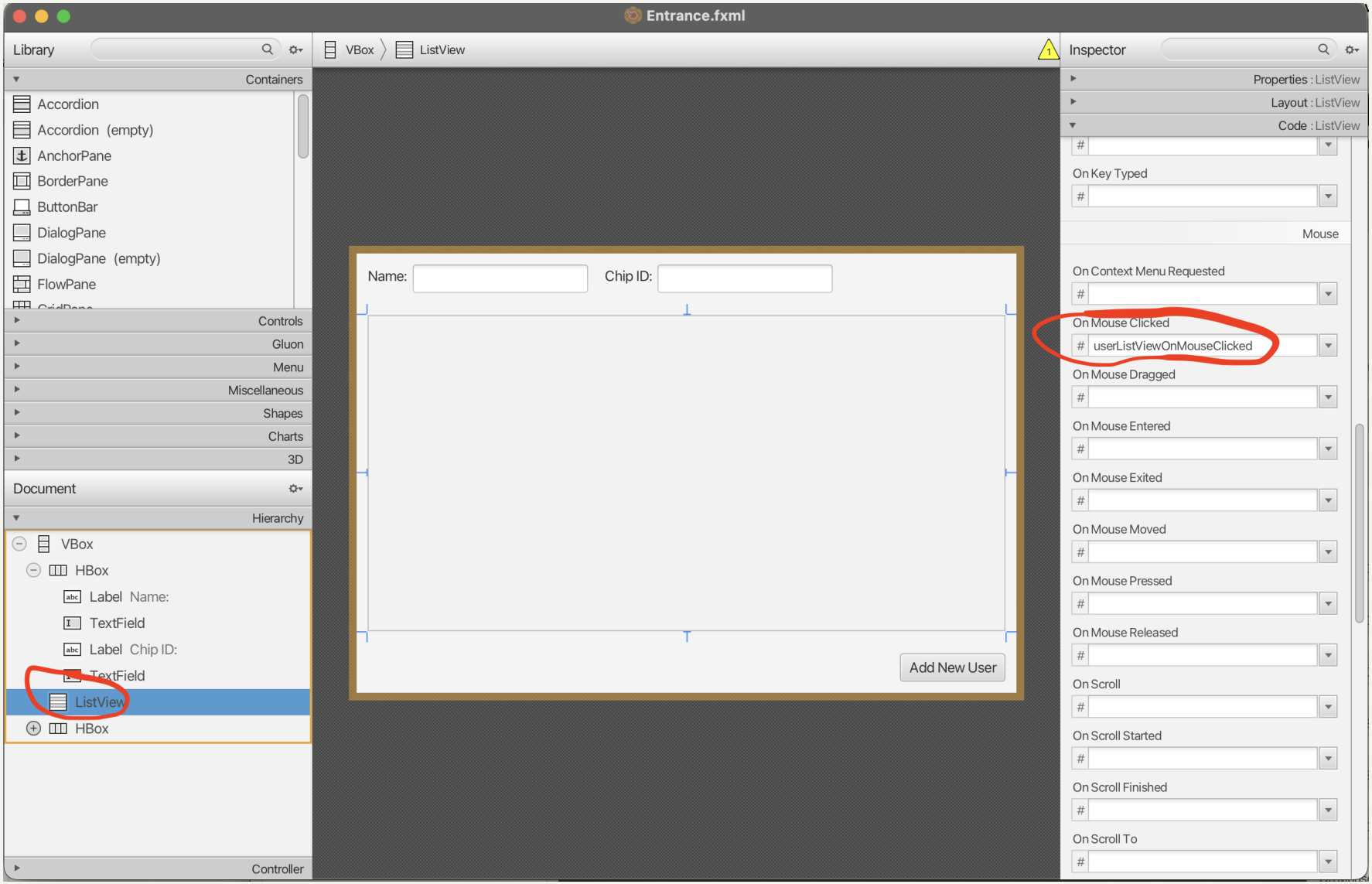
```
// do uz n-krat spracovaneho usera dodame iba novy chip_reader
var chipReaderId = rs.getLong("cr_id");
if (chipReaderId != 0){
    var chipReader = new ChipReader(
        rs.getLong("cr_id"),
        rs.getString("cr_door")
    );

    user.access().add(chipReader);
}
```

Priradzovanie čítačiek užívateľom v 2. okne



EntranceWindow



EntranceController

- Dvojitým kliknutím na užívateľa v hlavnom okne otvoríme nové okno

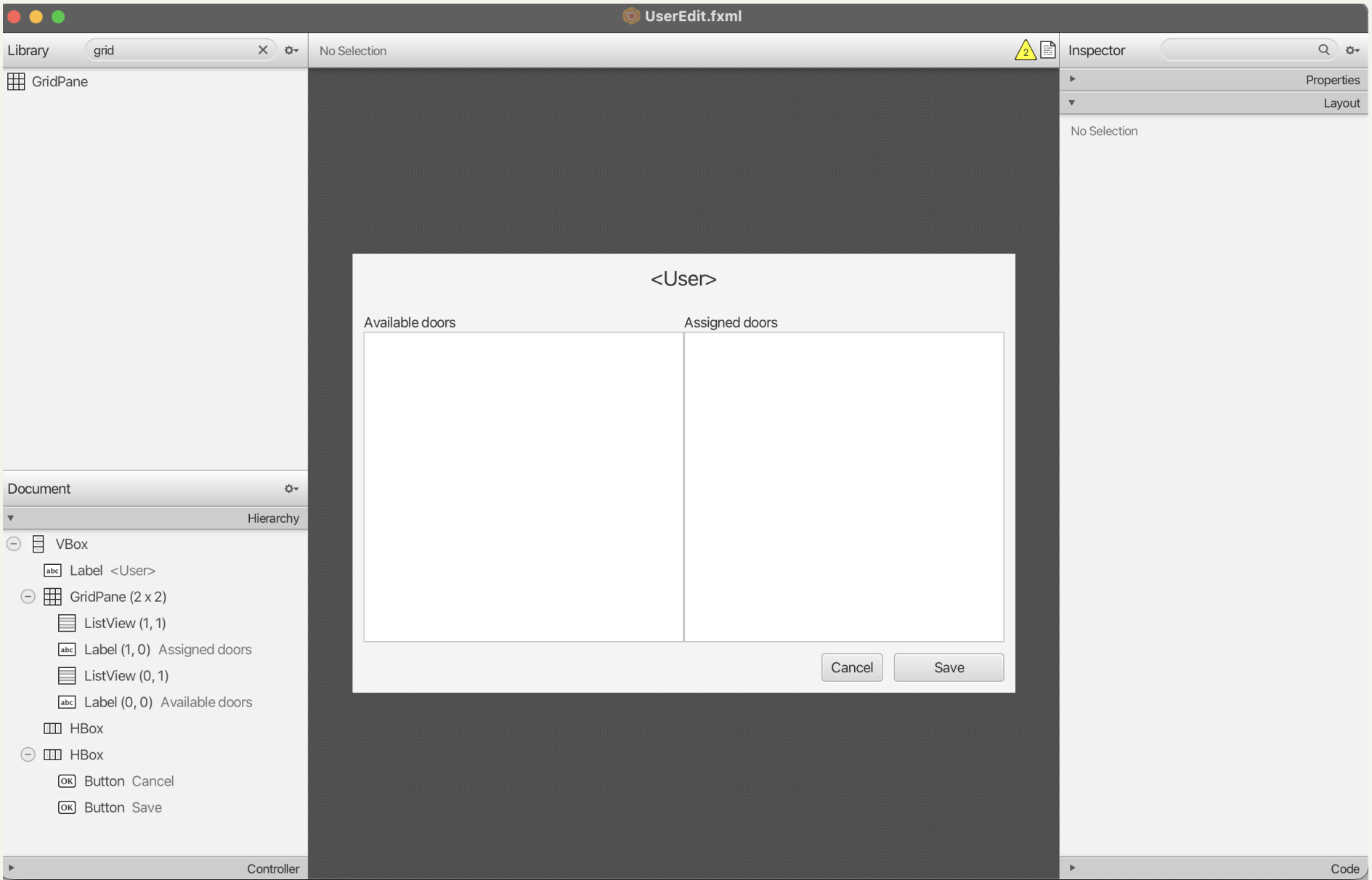
```
@FXML
void userListViewOnMouseClicked(MouseEvent event) {
   if (event.getClickCount() == 2) {
      var selectedUser = this.userListView
         .getSelectionModel()
         .getSelectedItem();
      this.openUserEditWindow();
   }
}
```

```
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());

    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();

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

}
```

Inicializujeme už **nastavené** prístupy k čítačkam

Inicializujeme **dostupné** prístupy k čítačkam

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().getSelectedItems();

   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().getSelectedItems();

   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?