cphbusiness

JDBC og databaser

SQL fra Java

- Med JDBC blander vi to sprog; SQL og Java
- Kompileren kan kun tjekke Java og opfatter SQL som ren tekst – der kan stå hvad som helst
- Hvis vi vil have vores SQL tjekket, kan vi derfor starte i pgAdmin
- Senere kan vi så tilføje vores SQL statements til Javakoden og køre forespørgslerne programmatisk.
- Til sidst kan vi lægge noget lækkert front-end på og så har vi et fuldt system ☺

JDBC

- Java Database Connectivity er et API som består af interfaces og abstrakte klasser
- De konkrete klasser er implementeret så de passer til forskellige databaser og kaldes drivere
- På den måde kan vi som udviklere altid kode op mod samme API og driveren oversætter så til noget, som kan forstås af Postgres, MySQL etc

Maven og dependencies

- Vi kan selv hente driveren eller vi kan lade Maven gøre det
- Maven er et byggeværktøj som kan håndtere dependencies, organisere koden, køre test mv.

```xml
<dependencies>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.6.0</version>
    </dependency>
</dependencies>
```

```
src/
├── main/
│   ├── java/          ← Alt Java-kildekode
│   └── resources/     ← Konfigurationsfiler, properties, etc.
└── test/
    ├── java/          ← Test-klasser
    └── resources/     ← Test-ressourcer
```

Mapping mellem Java og database

- Klasser/objekter i Java *kan* svare til tabeller/rækker i databasen
- Hvad kan skabe mismatch mellem Java og databasen?
- Hvilke konstruktioner er unikke for Java og hvilke er for databaser?

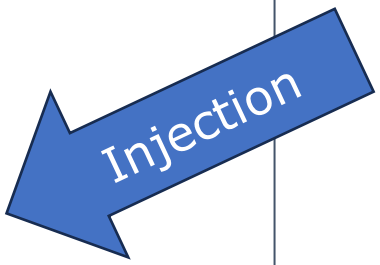
Dependency Injection (DI)

- Giver løs kobling mellem databasen og javakoden
- Smart når vi skal lave integrationstest
- Også smart hvis vi skifter vores database ud (fx fra MySQL til Postgres)

```
public class BorrowerMapper {  no usages

    private final DatabaseConnector connector;  1 usage

    public BorrowerMapper(DatabaseConnector connector) {  no usages
        this.connector = connector;
    }
}
```



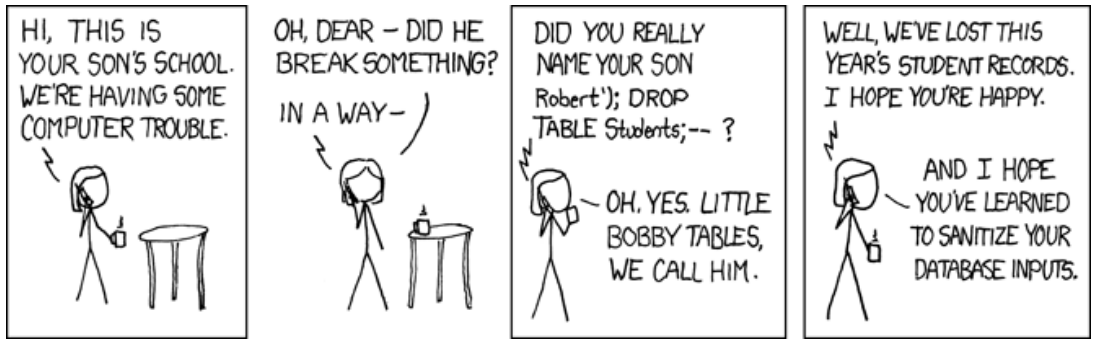
Statement og PreparedStatement

- Statement er ren SQL:

```
String sql = "SELECT * FROM laaner WHERE laaner_id = " + id;
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(sql);
```

- PreparedStatement lader os give argumenter med (og er mere sikkert fordi vi undgår SQL injection):

```
String sql = "SELECT * FROM laaner WHERE laaner_id = ?";
PreparedStatement statement = connection.prepareStatement(sql);
statement.setInt( parameterIndex: 1, id);
ResultSet resultSet = statement.executeQuery();
```




```
INSERT INTO Students (name) VALUES ('Robert');
DROP TABLE Students;
```

Generated keys

```
try (PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS))
```

```
}
ResultSet idResultset = ps.getGeneratedKeys();
if (idResultset.next()) {
    newId = idResultset.getInt( columnIndex: 1);
    member.setMemberId(newId);
} else {
    member = null;
}
```

Vi beder PreparedStatement give os den genererede nøgle

Vi fortæller, at PreparedStatement skal kunne give os de genererede nøgler

DTO – Data Transfer Object

- En måde at oversætte mellem ResultSet og resten af vores kode
- Vi bliver ofte nødt til at pakke vores data ind i et objekt for at kunne sende det rundt i programmet
- Hvorfor sender vi ikke bare ResultSet rundt?
- Hvad er mon forskellen på entiteter/objekter og DTO’er?