Haaga-Helia

Ohjelmointi (Java)

Tietokantojen käyttö Java-ohjelmissa
DAO-luokat (Data Access Object)

Haaga-Helia

Oppitunti 1: JDBC (Java Database Connectivity)

Tietokantojen käyttö Java-ohjelmissa

Haaga-Helia

Tietokannat Javassa

- Kaksi vakiintunutta tapaa relaatiotietokantojen käyttöön:
  - **JDBC** (Java Database Connectivity)
    Yhtenäinen tapa tehdä SQL-kyselyitä lähes mihin tahansa tietokantoihin ja käsitellä kyselyiden tuloksia.
  - **JPA** (Java Persistence API)
    Yhtenäinen ORM (Object Relational Mapping) -ratkaisu, jolla Javaluokkia saadaan automaattisesti kytkettyä tietokannan tauluihin ilman SQL:n kirjoittamista.
- Tällä kurssilla käytämme JDBC:tä
  - Yksinkertaisempi ottaa käyttöön
  - Taustalla vähemmän automaatiota, eli saamme paremmin käsityksen siitä, miten kirjoittamamme SQL-kyselyt konkreettisesti suoritetaan

Haaga-Helia

JDBC – Java Database Connectivity

- JDBC toimii lähes minkä tahansa (SQL) tietokantojen kanssa, kunhan Java-projektiin on lisätty käytettävän tietokannan ajuri
- Tällä kurssilla käytetään SQLite-tietokantaa, joka on paikallinen muisti- tai tiedostopohjainen tietokanta
  - Ei erillistä tietokantapalvelinta
  - Ei salasanoja yms. "ylimääräistä"
- Samat Java-koodit toimisivat myös esim. MySQL tai MariaDB –tietokantoja hyödyntäen
  - Käyttäisimme tällöin vain eri ajuria

Haaga-Helia

SQLiten komentorivikäyttö (valinnainen vaihe)

- Tietokannan komentorivikäyttö ei ole tällä kurssilla välttämätöntä, mutta kyselyitä on helpompi suunnitella ja kokeilla Java-ohjelman ulkopuolella.

- Lataa itsellesi sqlite3-komentorivityökalu osoitteesta: https://sqlite.org/download.html
  → `sqlite-tools-win32-x86-VERSIO.zip` → `sqlite3.exe`

- Tallenna tiedosto esim. samaan kansioon Chinook-tietokannan kanssa

- Ohjeita komentorivityökalun käyttöön löydät osoitteesta: https://sqlite.org/cli.html tai videosta https://video.haaga-helia.fi/media/SQLite+tools/0_pez4r54j

Haaga-Helia

JDBC:n SQLite-ajuri

- Tietokannan käyttämiseksi Javasta käsin tarvitsemme erillisen ajurin
- Erilaiset riippuvuudet asennetaan pääsääntöisesti automaatiotyökalujen avulla
  - Kirjastot jaellaan tyypillisesti JAR-tiedostoina (Java Archive)
  - Suosittuja automaatiotyökaluja Javalle ovat mm. Maven ja Gradle
  - Automaatiotyökalujen avulla monimutkaistenkin riippuvuuksien hallinta on yksinkertaista

- Tässä tapauksessa haemme tarvittavan ajurin manuaalisesti Mavenin tietovarastosta:

  https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc
  → Valitse uusin versio → Download (JAR)
- Siirrä tallentamasi tietokanta-ajuri Eclipseen projektin alle uuteen hakemistoon nimeltä *"lib"*
- lisää lib-hakemisto projektisi projektisi polkuun tämän ohjeen mukaisesti: https://stackoverflow.com/a/23420543.

Haaga-Helia

Tietokantojen käyttö

Tietokantaan yhdistäminen Javasta

Haaga-Helia

Tietokantaan yhdistäminen

```
import java.sql.Connection;
import java.sql.DriverManager;

public class ChinookDatabase {
    private static final String URL = "jdbc:sqlite:M:\\sqlite\\Chinook_Sqlite.sqlite";

    public Connection connect() throws SQLException, ClassNotFoundException {
        Class.forName("org.sqlite.JDBC");
        return DriverManager.getConnection(URL);
    }
}
```

JDBC-polku pitää sisällään käytettävän ajurin nimen ja tietokannan sijainnin.
SQLiten tapauksessa sijainti on polku levyllä.

Tietokanta-ajurin luokan lataus
(ei välttämätöntä kaikissa tapauksissa)

Yhteyden muodostaminen tietokantaan

MySQL-tietokantaan yhdistettäisiin vastaavasti esim. osoitteella
"jdbc:mysql://127.0.0.1:3306/chinook"

Haaga-Helia

Versio 2: poikkeuksen "kääriminen" ajonaikaiseksi poikkeukseksi

```
import ...

public class ChinookDatabase {
    private static final String URL = "jdbc:sqlite:M:\\sqlite\\Chinook_Sqlite.sqlite";

    public Connection connect() {
        try {
            Class.forName("org.sqlite.JDBC");
            return DriverManager.getConnection(URL);

        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
}
```

MySQL-tietokantaan yhdistettäisiin vastaavasti esim. osoitteella
"jdbc:mysql://127.0.0.1:3306/chinook"

Haaga-Helia

Tietokantayhteyksien sulkeminen

- Tietokantayhteydet tulee myös sulkea käytön jälkeen, jotta niille varatut resurssit vapautuvat uudelleenkäytettäviksi.
- Yhteydet suljetaan kutsumalla `close()`-metodia.

```
try {
    connection.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

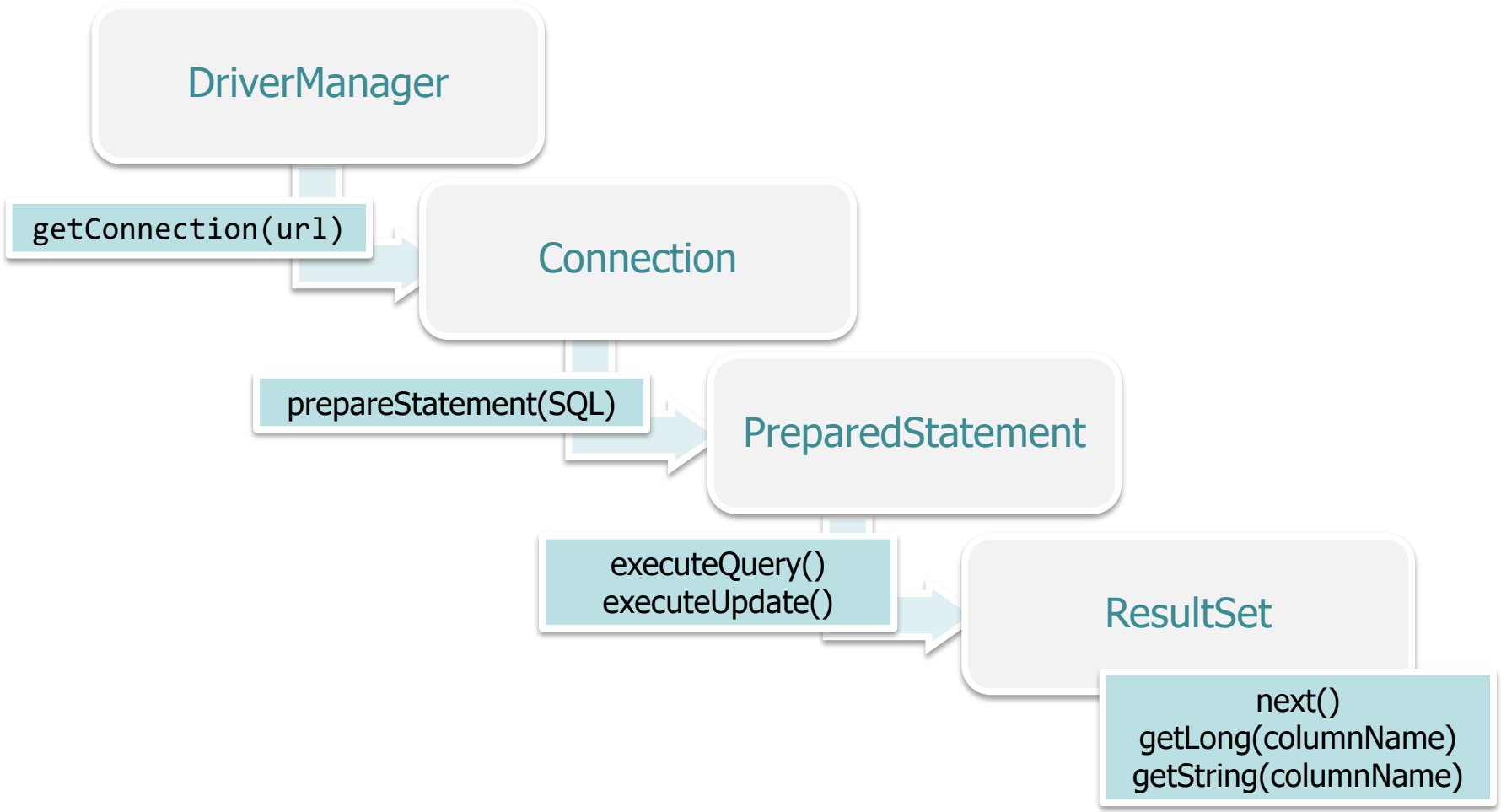
Haaga-Helia

Tietokantojen käyttö

Kyselyiden tekeminen

Haaga-Helia

Kyselyn tekeminen: luokat ja metodit



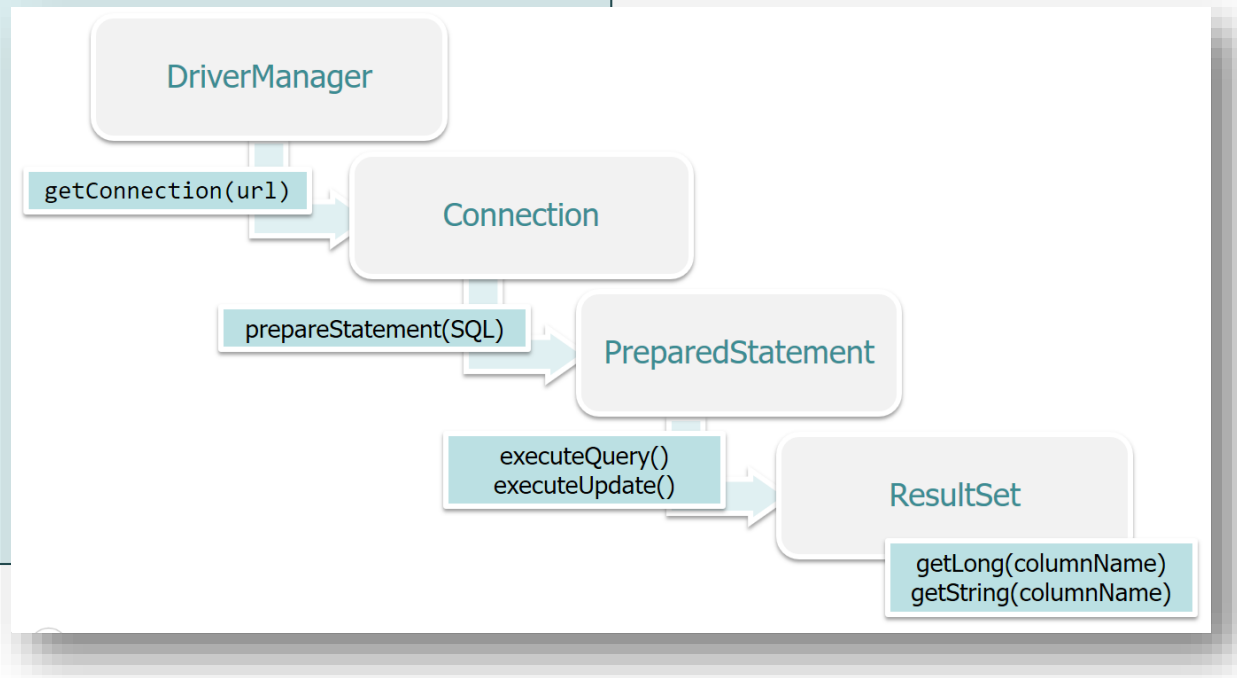
Haaga-Helia

Kyselyn tulosten läpikäynti: ResultSet, next ja get

- JDBC-kyselyiden tuloksina saadaan `ResultSet`-olioita
- `ResultSet`-olion kautta on mahdollista käsitellä **vain yhtä tulosriviä kerrallaan**
- Käsiteltävä tulosrivi voidaan vaihtaa seuraavaksi kutsumalla `next`-metodia:
  - `next` palauttaa *true*, jos käsiteltävää dataa on jäljellä
  - `next` palauttaa *false*, jos dataa ei ole enempää
- Nykyiseltä tulosriviltä voidaan kysyä eri sarakkeiden arvot get-metodeilla, esim:
  - `getString("Name")`
  - `getLong("ArtistId")`

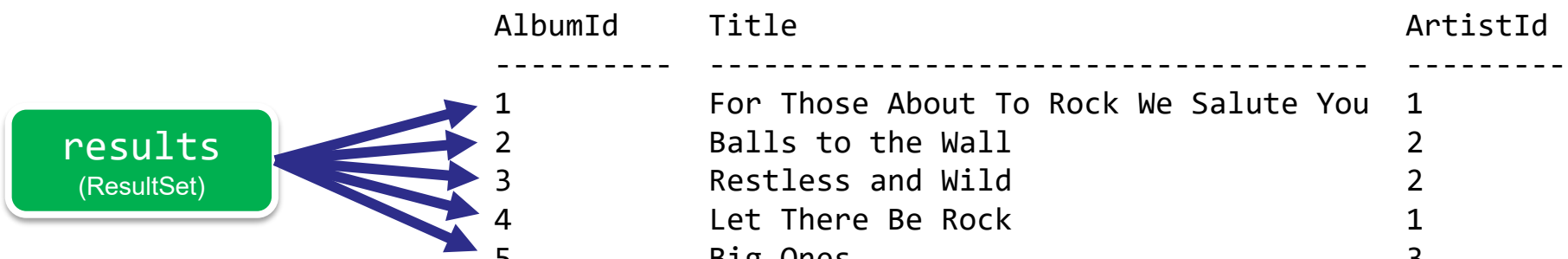
Haaga-Helia

```java
// Avataan yhteys
Connection connection = DriverManager.getConnection(URL);

// Muodostetaan kysely
PreparedStatement statement = connection.prepareStatement("SELECT * FROM Artist");

// Suoritetaan kysely
ResultSet results = statement.executeQuery();

// Käydään tulokset läpi
while (results.next()) {
    // Tulostetaan artistien nimet
    System.out.println(results.getString("Name"));
}

// Suljetaan kaikki resurssit
results.close();
statement.close();
connection.close();
```



Haaga-Helia

```
while (results.next()) {
    // Tulostetaan albumien nimet
    String title = results.getString("Title");
    System.out.println(title);
}
```



Alussa ResultSet ei kohdistu millekään riville.
Next-metodia on kutsuttava ennen tietojen lukemista.

Next-metodin kutsuminen siirtää ResultSetin käsittelemään aina seuraavaa "riviä" tuloksista ja palauttaa arvoksi `true`.

Kun rivit loppuvat, next palauttaa `false` ja ResultSet ei jälleen kohdistu millekään riville.

Haaga-Helia

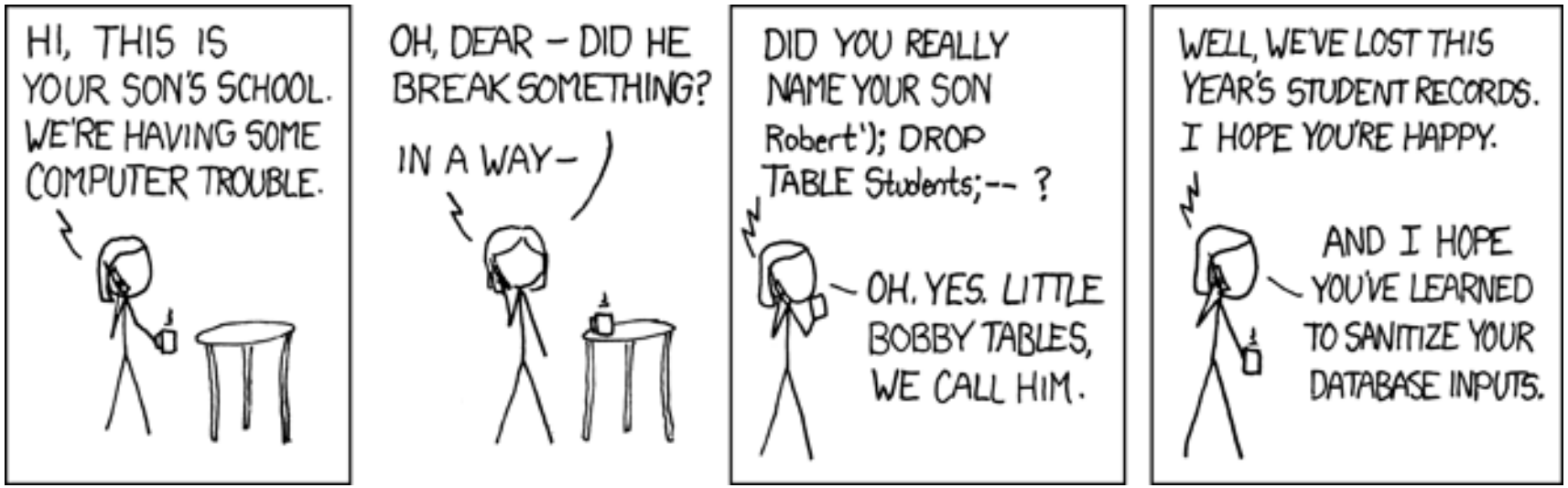
Tietokannasta saatu data käydään läpi ResultSet-olion kautta rivi kerrallaan ja sarake kerrallaan:

```
ResultSet results = statement.executeQuery();

while (results.next()) {
    long id = results.getLong("TrackId");
    String name = results.getString("Name");
    String mediaType = results.getString("MediaTypeName");
    String genre = results.getString("GenreName");
    long albumId = results.getLong("AlbumId");
    long milliseconds = results.getLong("Milliseconds");
    double unitPrice = results.getDouble("UnitPrice");

    tracks.add(new Track(id, name, albumId, mediaType, genre, milliseconds, unitPrice));
}
```

Haaga-Helia

Kyselyiden tekeminen turvallisesti

PreparedStatement



https://xkcd.com/327/

Haaga-Helia

Vaaralliset kyselyt

- Älä koskaan muodosta SQL-kyselyitä käsin yhdistelemällä merkkijonoja:

```
connection.prepareStatement("SELECT * FROM Artist WHERE Name = \"" + name + "\"");
```

- Kyselyn teko merkkijonoja yhdistelemällä aiheuttaa mm. tietoturvaongelmia

Haaga-Helia

Parametrisoidut kyselyt

- Kun kyselyissä tarvitaan ajonaikaisesti muodostettavia parametreja (id:t, nimet…), ne tulee asettaa paikalleen `PreparedStatement`-luokan metodeilla
- `PreparedStatement`-luokan SQL-kyselyihin parametrien tilalle kirjoitetaan kysymysmerkit (?), joiden kohdille asetetaan set-metodeilla arvot:

```
PreparedStatement statement =
connection.prepareStatement("SELECT * FROM Artist WHERE Name = ?");

statement.setString(1, name);
```

- setString asettaa kyselyyn merkkijonon, setLong vastaavasti kokonaisluvun.
  - set-metodien ensimmäisenä parametrina annetaan aina indeksi, joka kertoo mikä kyselyn parametreista asetetaan
  - indeksit alkavat 1:stä! 😱

Haaga-Helia

Oppitunti 2: DAO (Data Access Object)

Tietokantalogiikan eriyttäminen muusta koodista

Haaga-Helia

Tietokannan eriyttäminen muusta logiikasta

- Edellisissä esimerkeissä esiintyy hyvin paljon tietokannan käyttöön liittyviä yksityiskohtia:
  - Yhteyksien luomista ja sulkemista, SQL-kyselyitä, tulosten läpikäyntiä
- Sovelluksen kehittämisen, testauksen ja ylläpidon kannalta on iso ongelma, mikäli tietokantaa käytetään lukuisissa eri paikoissa:
  - Kun tietokantaa muutetaan, joudutaan Java-koodiin tekemään lukuisia muutoksia
- Tietokantaa käytetäänkin usein Data Access Object -luokkien (DAO) kautta
  - DAO-luokat huolehtivat yhteyksien käytöstä ja kyselyiden muodostamisesta
  - DAO-luokkien metodit palauttavat aivan tavallisia Java-olioita joko yksinään tai kokoelmina (ns. model-luokkia)

Haaga-Helia

Tiedon lisääminen ja päivittäminen

SQL:n INSERT ja UPDATE -komennot

Haaga-Helia

Datan lisääminen tietokantaan

Tiedon päivittämiseksi kutsutaan `PreparedStatement`-olion `executeUpdate`-metodia:

```
statement = connection.prepareStatement("INSERT INTO Artist (Name) VALUES (?)");

statement.setString(1, "Matti ja Teppo");
statement.executeUpdate();
```

Haaga-Helia

Edistynyttä sisältöä: insertit ja automaattisesti generoidut pääavaimet

- SQL-lauseketta luotaessa voidaan antaa SQL:n lisäksi toinen parametri: `Statement.RETURN_GENERATED_KEYS`
- Generoidut avaimet voidaan tällöin kysyä kyselyn jälkeen *getGeneratedKeys*-metodilla:
- *getGeneratedKeys* palauttaa *ResultSet*-olion, jota käytetään samankaltaisesti kuin aikaisemmassa kyselyesimerkissä.

```
statement = connection.prepareStatement(
    "INSERT INTO Artist (Name) VALUES (?)",
    Statement.RETURN_GENERATED_KEYS
);

statement.setString(1, "Matti ja Teppo");
statement.executeUpdate();

ResultSet keys = statement.getGeneratedKeys();

if (keys.next()) {
    long id = keys.getLong(1);
}
```