JDBC

Adam Łagoda

Agenda

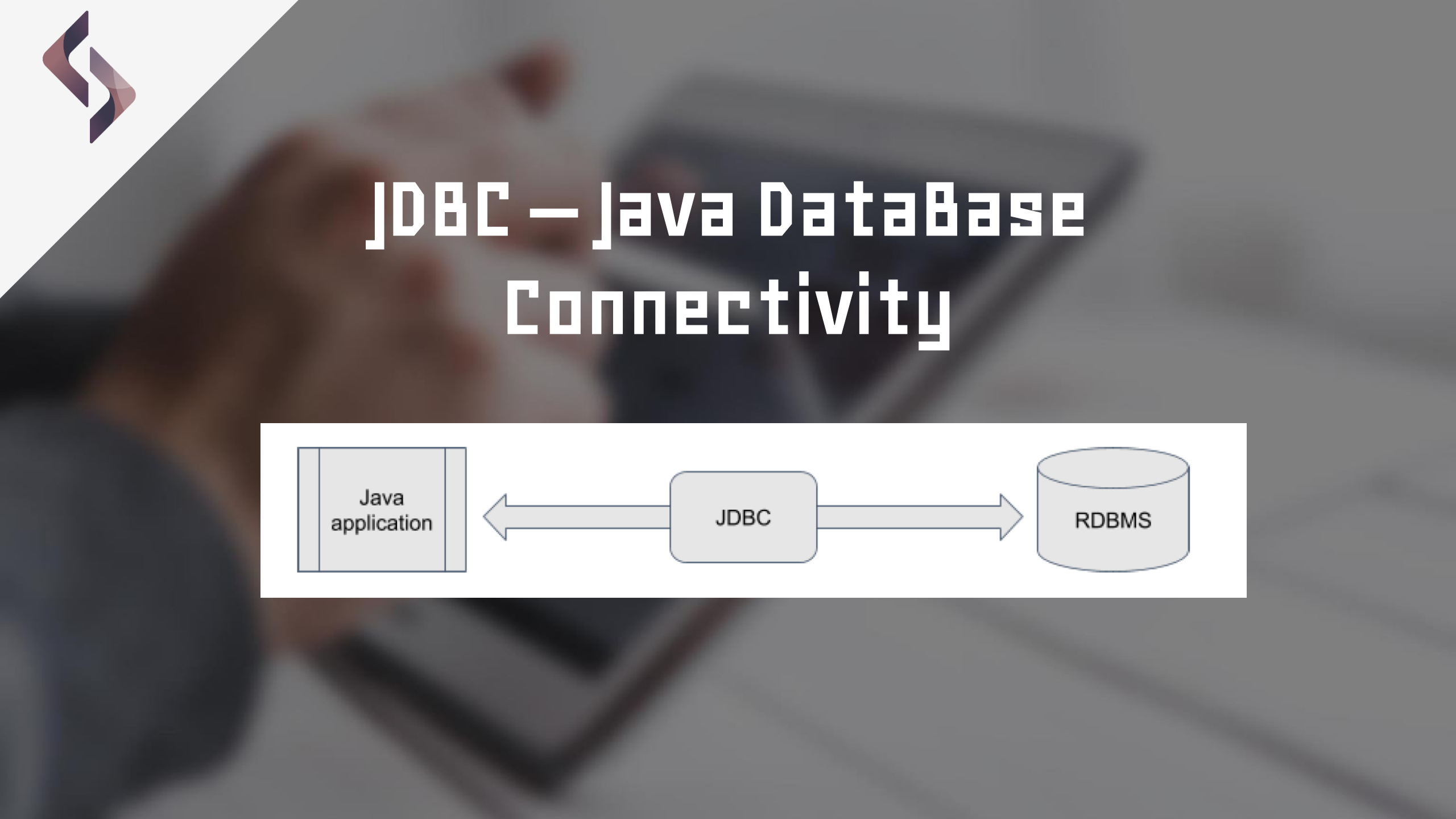
1. Czym jest JDBC
2. Tworzenie połączenia za pomocą DriverManager
3. Tworzenie połączenia za pomocą DataSource
4. Connection
5. JDBC - DQL
6. JDBC - DDL
7. JDBC - DML
8. CRUD

Przydatne informacje

1. Repozytorium jest dostępne pod https://github.com/adamlagoda/javadb-starter
2. Rozwiązania zadań znajdują się na branchach nazywanych wg schematu ***jdbc_zadanie[NR_ZADANIA]_solution*** np. *jdbc_zadanie2_solution*
3. Rozwiązania zadanań są niezależne od poprzednich, tzn. program może nie uruchomić się prawidłowo, gdy przełączymy się z *master* na branch z rozwiązaniem.
4. Prezentacja jest dostępna w formie PDF-a w repozytorium

Dlaczego nie serializacja

- Java udostępnia mechanizm serializacji
- Dane z pamięci aplikacji można zapisać do pliku
- Modyfikacja kodu uniemożliwia odczytanie danych
- Konieczność pobierania całości danych, by zmodyfikować fragment
- Brak języka zapytań
- I wiele innych…



JDBC – Java DataBase Connectivity

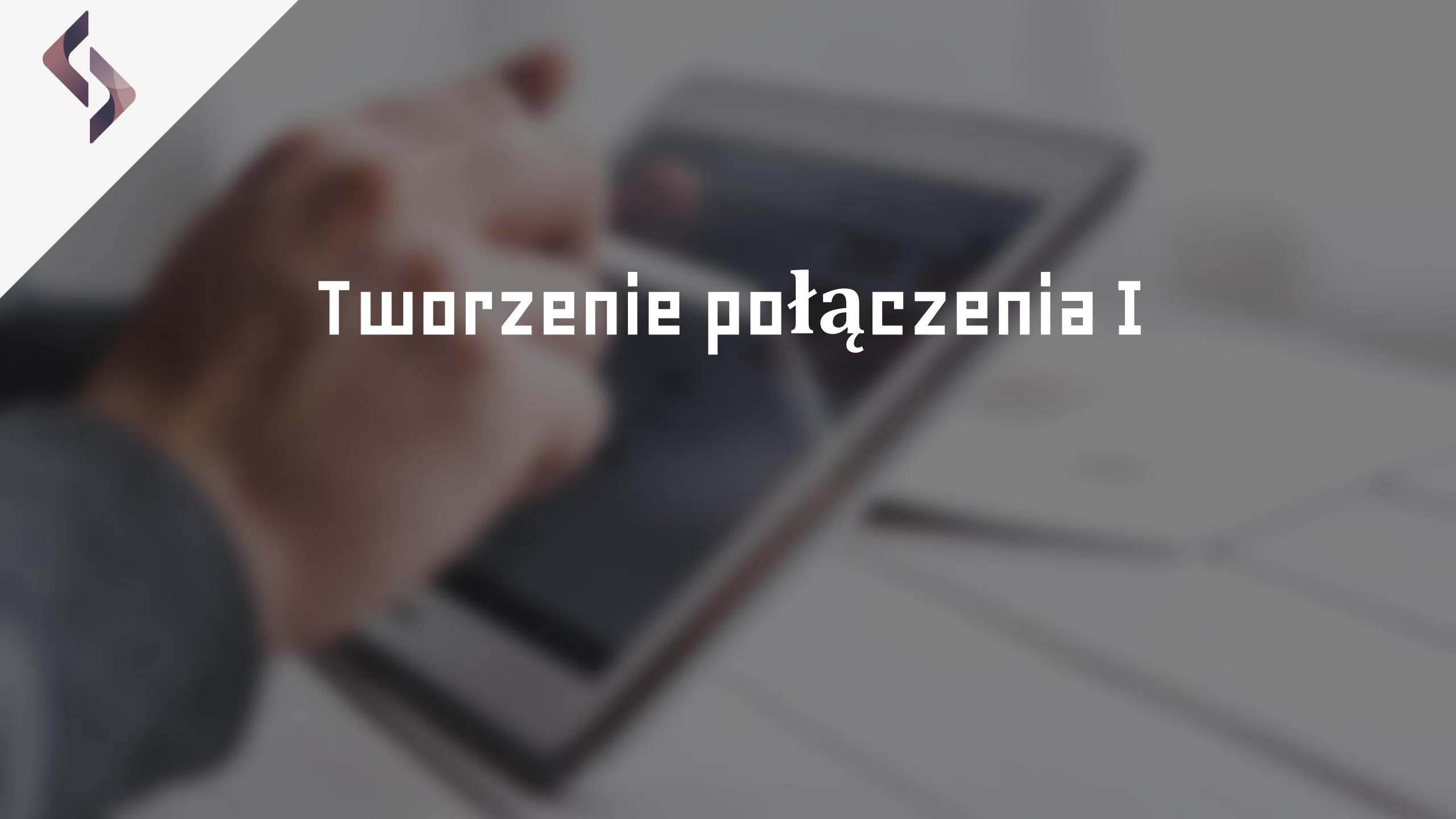
JDBC

- interfejs pozwalający na ustanowienie połączenia do bazy danych z poziomu aplikacji napisanej w Javie
- umożliwia komunikację z bazami danych za pomocą języka SQL, konfigurację połączenia z bazą oraz uzyskanie informacji o samej bazie
- opracowany w 1996 r. przez Sun Microsystems
- **Java 8 JDBC 4.2**
- Java ≥ 9 JDBC 4.3
- to tylko interfejs. Implementacja jest dostarczana przez poszczególnych producentów baz danych (MySQL, PostgreSQL, MSSQL, Oracle itp.),
- MySQL Connector/J - sterownik implementujący standard JDBC.

https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

Zadanie 0

1. Ściągnij projekt *https:github.com/lagodaadam/javadb-starter*
2. Sprawdź zależność do implementacji JDBC w pliku pom.xml modułu *jdbc-starter* i spróbuj podnieść jej wersję do najbardziej aktualnej



Tworzenie połączenia I

Tworzenie połączenia I

JDBC API

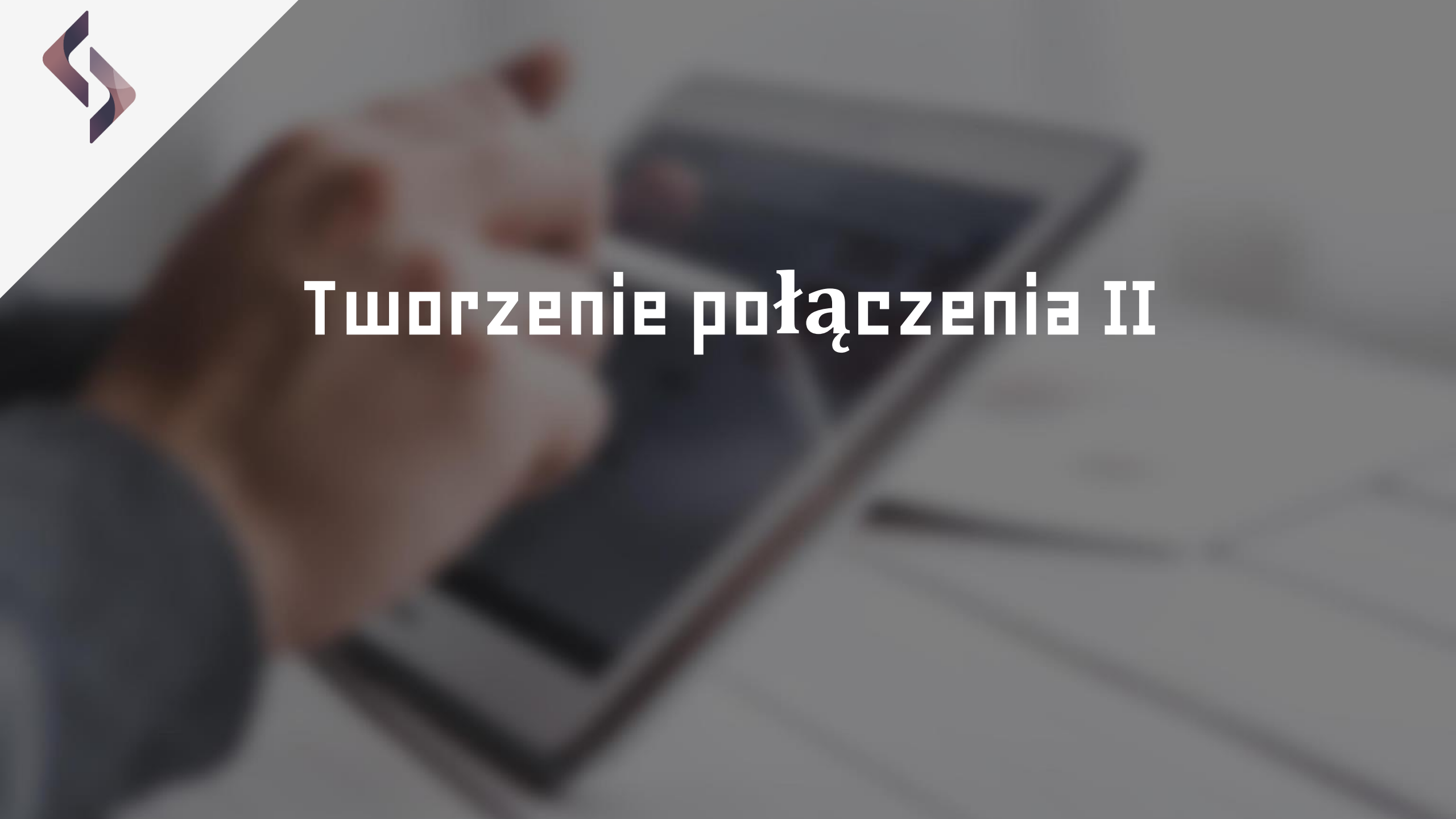
java.sql.Driver
(sterownik do łączenia aplikacji z bazą danych)

java.sql.DriverManager
(zarządzanie sterownikami, rejestracja, szukanie odpowiedniego sterownika)

java.sql.Connection
(połączenie z bazą danych, informacje o bazie danych, wywoływanie zapytań do bazy danych, "ciężki" obiekt, zawsze zamykamy na koniec!)

Zadanie 1

1. Otwórz klasę `ConnectionViaDriverManager` i uzupełnij konfigurację bazy danych(parametry: **`DB_URL, DB_USER, DB_PASSWORD`**) tak żeby połączyć się ze swoją lokalną bazą MySQL - może to być baza stworzona na wcześniejszych zajęciach,
2. Uruchom metodę `main` i sprawdź czy połączenie do bazy danych się powiodło,
3. Przeanalizuj kod - zwróć uwagę na poszczególne etapy cyklu życia obiektu `Connection`,
4. Stwórz klasę `ConnectionViaDriverManager2` - użyj w niej struktury *try-with-resource* do tworzenia i zamykania obiektu `Connection`



Tworzenie połączenia II

Tworzenie połączenia II

JDBC API

javax.sql.DataSource

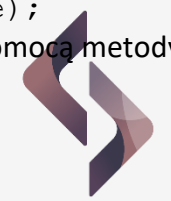
(fabryka do tworzenia połączeń z bazą danych, preferowany sposób tworzenia połączeń, “pod spodem” wykorzystuje sterownik do stworzenia połączenia)

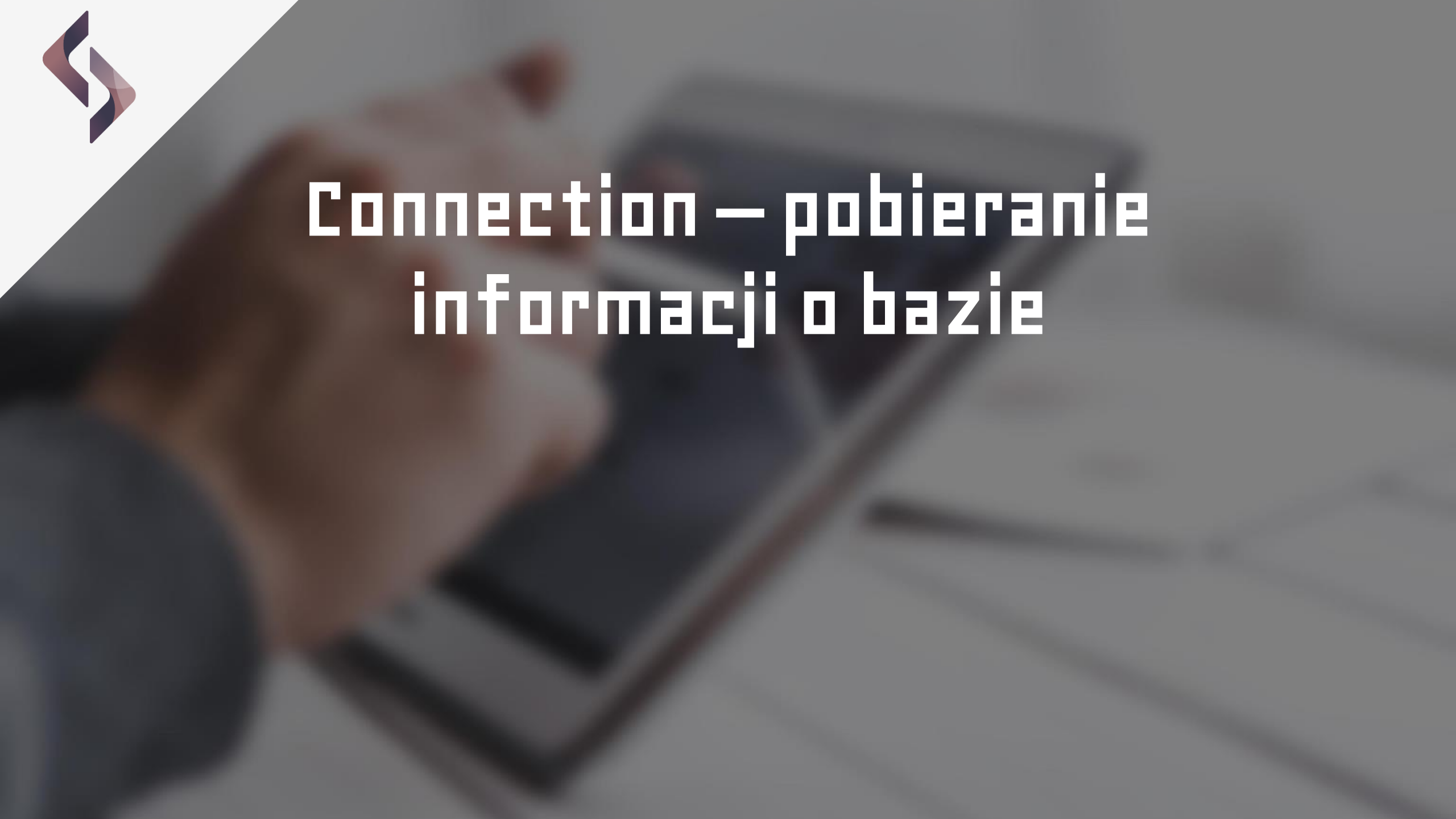
Tworzenie połączenia II

- `DataSource` umożliwia nam oddzielenie konfiguracji połączenia do bazy danych od pracy na tym połączeniu
- W przeciwieństwie do `DriverManager`, jest to interfejs, którego implementację dostarcza sterownik
- Dobrą praktyką jest przechowywanie parametrów dostępu do bazy danych w osobnym pliku (np.: `database.properties`) - tak żeby można było zmieniać konfigurację podczas wdrożeń na różne środowiska bez potrzeby zmiany kodu.

Zadanie 2

1. Otwórz klasę `ConnectionViaDataSource` i uzupełnij konfigurację bazy danych tak żeby połączyć się ze swoją lokalną bazą MySQL
2. Uruchom metodę `main` i sprawdź czy połączenie do bazy danych się powiodło
3. Przeanalizuj kod - czy widzisz różnicę między użyciem `DataSource` a `DriverManager`?
4. Otwórz klasę `ConnectionFactory` i uzupełnij kod metody `getConnection()`, wykorzystaj ją do pobrania obiektu `Connection` w klasie `ConnectionViaDataSource`
5. Przenieś parametry bazy danych do pliku '`database.properties`':
   a. Zapisz w nim pary klucz-wartość dla każdego parametru:
      ```
      org.example.jdbc.db.server={SERVER}
      org.example.jdbc.db.name={DB_NAME}
      ```
      ..... itp.
   b. Przekaż nazwę pliku jako argument przy tworzeniu obiektu klasy `ConnectionFactory`
   c. Wczytaj parametry z pliku za pomocą klasy `java.util.Properties` i metody: `Properties.load(InputStream inStream)`
   d. `inStream` można pobrać wykorzystując właściwości classloadera:
      ```
      classLoader.getResourceAsStream(name);
      ```
   e. Po wczytaniu pliku wystarczy pobrać parametry za pomocą metody:
      ```
      Properties.getProperty(String key)
      ```
   f. Uruchom i sprawdź czy program działa





Connection – pobieranie informacji o bazie

Connection

- Klasa `Connection` umożliwia pobieranie danych o połączonej bazie danych za pomocą:

```
DatabaseMetaData metaData = connection.getMetaData();
```

- W metadanych bazy znajdziemy m.in. informacje z jaką bazą jesteśmy połączeni (nazwa, wersja), jakim sterownikiem połączyliśmy się itp.,
- Ponadto możemy uzyskać informację o strukturze bazy danych, najłatwiej zrobić to poprzez kod:

```
ResultSet rs = metaData.getColumns(null, null, null, null);
while(rs.next()) {
        String tableName = rs.getString("TABLE_NAME");
        logger.info("table: {}", tableName); }
```

- `ResultSet` zawiera dużo więcej informacji - nazwy wszystkich parametrów są opisane w dokumentacji metody `DatabaseMetaData.getColumns()`. Opis metody można znaleźć pod: https://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html#getColumns(java.lang.String,%20java.lang.String,%20java.lang.String,%20java.lang.String))

Zadanie 3

1. Stwórz klasę `DatabaseDiscovery`
2. W metodzie `main` pobierz obiekt `Connection` (w konstruktorze jako argument podaj plik "`database.properties`") i wykorzystując metodę `getMetaData()` pobierz z niego obiekt `DatabaseMetaData`
3. Wyświetl w konsoli (za pomocą *loggera*) informacje: nazwa i wersja SZBD, nazwa użytkownika, nazwa i wersja sterownika do bazy danych - w obiekcie `DatabaseMetaData` znajdziesz gettery do każdej z tych informacji
4. Za pomocą kodu opisanego we wstępie pobierz i wyświetl w konsoli: nazwę tabeli, nazwę kolumny, jej rozmiar i nazwę typu, który znajduje się w kolumnie dla wszystkich tabel znajdujących się w bazie danych
5. (dla chętnych) Spróbuj poeksperymentować z metodą `DatabaseMetaData.getColumns(null, null, null, null)`, sprawdź dokumentację i spróbuj wyciągnąć dane dla bazy danych „reading_room" i tabeli „books". W tym celu zamiast `null` trzeba podać odpowiednie argumenty w metodzie `getColumns()`.

JDBC – zapytania SQL

JDBC – zapytania SQL

JDBC API

java.sql.Statement
(wykonywanie prostych/statycznych zapytań SQL bez parametrów i pobranie wyników)
java.sql.ResultSet
(obiekt przechowujący zestaw danych zwróconych przez bazę po wykonaniu zapytania SQL)

JDBC – zapytania SQL

- Proste zapytanie i pobranie wyników:

```
try(Statement statement = connection.createStatement()) {
          ResultSet resultSet =
                    statement.executeQuery("SELECT id FROM table_1;");
          while (resultSet.next()) {
                    int id = resultSet.getInt("id");
                    logger.info("id: {}", id);}}
```

- `Statement.executeQuery()` - wyciąganie danych, `Statement.executeUpdate()` - aktualizowanie danych (dodawanie, aktualizacja, usuwanie),
- `Statement` i `ResultSet` również zamykamy jak tylko przestaniemy ich używać! Jak zamkniemy `Statement`, `ResultSet` zostanie automatycznie zamknięty,

http://www.java2s.com/Tutorials/Java/JDBC/0070__JDBC_Data_Types.htm

Mapowanie typów
(odpowiada za to sterownik)

SQL TYPE	Java Type
VARCHAR	Java.lang.String
INTEGER	Int
DATE	Java.sql.Date
TIMESTAMP	Java.sql.Timestamp

Zadanie 4

1. Stwórz klasę `BooksManager`
2. Zaimportuj skrypt `reading_room.sql`
3. W metodzie `main` (wykorzystując klasę `ConnectionFactory` i plik '`database.properties`') pobierz obiekt `Connection`
4. Dodaj metodę, która wyświetli listę wszystkich książek zapisanych w bazie, które nie są wypożyczone, wykorzystaj `SQL SELECT`
5. Dodaj metodę, która wstawi do tabelki z książkami nową pozycję (`SQL INSERT`):

{title: „Linux. Komendy i polecenia. Wydanie IV rozszerzone”, publisher: „Helion”, pages: 176, netto_price: 16.18, vat_rate: 5, isbn: „978-83-246-8838-8”, publish_date „2014-06-13”, category: „Systemy Operacyjne, Linux”, author: „Łukasz Sosna”}

5. Dodaj metodę, która wyświetli wszystkie książki z ich kategoriami, wykorzystaj `SQL SELECT` i połącz tabelki za pomocą `JOIN`a
6. Zaktualizuj rekord z książką z zadania 4 i podnieś cenę netto o 20%. `SQL UPDATE`
7. * Dodaj metodę, która wyświetli listę książek, wydanych w przedziale czasu:

```
listBooksByPublishDate(Date fromDate, String toDate)
```

Metoda powinna obsługiwać takie argumenty:

- `listBooksByPublishDate(2020-07-12, null)` – wszystkie książki wydane po *2020-07-12*
- `listBooksByPublishDate(null, 2020-07-12)` – wszystkie książki wydane do *2020-07-12*
- `listBooksByPublishDate(2020-07-01, 2020-07-12)` – wszystkie książki wydane między *2020-07-01* a *2020-07-12*

JDBC — DDL

JDBC – DDL

DDL - (ang . ***D**ata **D**efinition **L**anguage*), służy do tworzenia, modyfikowania i usuwania obiektów struktury bazy danych (tabele, widoki, klucze itp), polecenia: `CREATE`, `DROP`, `ALTER`. Pozostałe rodzaje zapytań: DML, DCL. Z poziomu Javy wywołujemy polecenia DDL za pomocą

Zadanie 5

1. W lokalnej bazie MySQL tworzymy nową bazę/schemę "*library*"
2. W module library-manager uzupełniamy plik '*library-database.properties*' danymi z naszej bazy danych
3. Otwieramy klasę `DatabaseManager` i uzupełniamy metodę `createDb()` kodem, który stworzy strukturę bazy danych. Polecenia DDL znajdują się w pliku *resources/library-db-structure.sql*. Do pobierania obiektów `Connection` używamy `ConnectionFactory` (z plikiem '*library-database.properties*'). Wywołujemy metodę `createDb()` i sprawdzamy czy w lokalnej bazie danych pojawiły się odpowiednie struktury.
4. W klasie `DatabaseManager` uzupełniamy metodę `initializeDb()`. Dodajemy dane do tabelek: '*users*', '*categories*', '*books*' - po kilka wpisów do każdej z tabelek. Wywołujemy i sprawdzamy czy dane się uzupełniły.
5. W klasie `DatabaseManager` uzupełniamy metodę `dropDb()` - usuwamy po kolei wszystkie tabelki. Wywołujemy i sprawdzamy czy operacja się udała.
6. Ponownie uruchamiamy metody `createDb()` i `initializeDb()` - powinniśmy mieć przygotowaną bazę do dalszej pracy


Liquibase

Liquibase

- Narzędzie do tworzenia i zarządzania strukturą bazy danych w projekcie
- Konfiguracja w pliku `liquibase.properties`
- Struktura oparta o `changelog`, zawierający jeden lub więcej `changeset`
- Uruchamiany z linii poleceń lub jako plugin maven'owy
- Tabela `DATABASECHANGELOG` zawiera `changeset'y`
- Tabela `DATABASECHANGELOGLOCK` umożliwia współbieżną pracę na bazie

Zadanie 6

1. Przejdź do modułu *courses-manager*. Stwórz w swojej lokalnej bazie danych nową bazę: ‘sda_courses’. Zmień ustawienia bazy danych w pliku: ‘*liquibase.properties*’ tak żeby wskazywały lokalną bazę. Uruchom polecenie Mavena: *compile* (z panelu *Maven Projects*). Sprawdź czy w lokalnej bazie pojawiły się tabele.
2. Wywołaj kilkukrotnie polecenie `mvn compile`. Sprawdź, czy stan bazy uległ zmianie.
3. Zmodyfikuj plik `db-changelog.xml`. Ustaw typ kolumy `name` w tabeli `courses` na `VARCHAR(60)`. Uruchom polecenie `mvn compile`.
4. W module *library-manager* Stwórz strukturę bazy *library*, korzystając z liquibase.

https://www.liquibase.org/quickstart.html

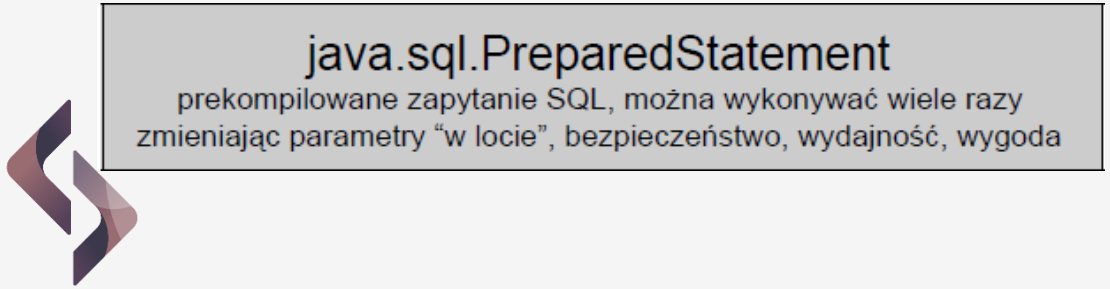
JDBC – DML

JDBC – DML

DML - (ang. ***D**ata **M**anipulation **L**anguage*), język do zarządzania danymi w bazie, polecenia:
`SELECT, INSERT, UPDATE, DELETE`
Z poziomu Javy najlepiej wykonywać polecenia DDL za pomocą obiektu `PreparedStatement`

SQL Injections - wstrzyknięcie w normalne zapytanie SQL złośliwego kodu, który ma na celu zniszczyć dane albo uzyskać dane których użytkownik nie powinien widzieć. `PreparedStatement` zapobiega tego typu atakom.
https://www.w3schools.com/sql/sql_injection.asp
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet

JDBC API



JDBC – DML

Proste zapytanie i pobranie wyników:

```
int id = 11;
String name = “Adam”;
java.sql.Date date = new java.sql.Date(2018, 6, 10);

PreparedStatement statement = connection.prepareStatement(
        "INSERT INTO table1(id, name, date) VALUES(?, ?, ?)"
        ));
        //parameterIndex zaczyna się od 1!
        statement.setInt(1, id);
        statement.setInt(2, courseId);
        statement.setDate(3, date);
        statement.executeUpdate();
}
```

Zadanie 6

1. W module *courses-manager* stwórz plik '*database.properties*' z właściwościami do połączenia z bazą '*sda_courses*' . W klasie `CoursesManager` zaimplementuj kod metod tak, żeby zapytania DML wykorzystywały obiekt `PreparedStatement`. Dodaj do odpowiednich metod parametry które będą używane w zapytaniach SQL, np. `insertStudent(String name, int courseId, String description)`. Stwórz klasę `CourseManagerTest`, w której przetestujesz działanie metod.
2. Przejdź do modułu: '*library-manager*'. Stwórz metodę `DatabaseManager.initializeDb2()` i przenieś tam kod z metody `initializeDb()` wykorzystując obiekt `PreparedStatement`. Sprawdź czy działa usuwając i dodając strukturę bazy danych '*library*'.
3. Przejdź do modułu: '*jdbc-starter*'. Potestuj różne ataki SQLInjection poprzez klasę `SqlInjectionSample`. Zmień metody w klasie `SqlInjectionSample` wykorzystując `PreparedStatement` i sprawdź czy to zapobiega atakom SQLInjections.

JDBC — CRUD

JDBC – CRUD

CRUD - (ang. ***C**reate, **R**ead, **U**pdate and **D**elete)* - cztery podstawowe funkcje aplikacji działających z bazami danych. Odpowiedniki SQL: Create - `INSERT`, Read - `SELECT`, Update - `UPDATE`, Delete - `DELETE`

- **DAO** - (ang . ***D**ata **A**ccess **O**bject)* - komponent/klasa dostarczający jednolity interfejs do komunikacji między aplikacją a źródłem danych (np.: bazą danych)

Zadanie 7

1. Uzupełnij klasę `JdbcUsersDao`, dodaj kod do pustych metod, użyj `PreparedStatement` do wykonywania zapytań. W metodzie `list()` napisz kod zwracający wszystkich użytkowników.
2. Uzupełnij klasę `JdbcBooksDao`, dodaj kod do pustych metod, użyj `PreparedStatement` do wykonywania zapytań. W metodzie `list()` napisz kod zwracający wszystkie książki.
3. Uzupełnij klasę `JdbcOrdersDao`, dodaj kod do pustych metod, użyj `PreparedStatement` do wykonywania zapytań.

Zadanie 7 cd.

4. * Do interfejsu `IBooksDao` i klasy `JdbcBooksDao` dodaj metodę która wyszuka książki po podanej frazie. Fraza może zawierać fragment bądź całość tytułu lub autora książki. Czyli jak wyszukamy po frazach:
    a. ‘Adam’, ‘Mickiewicz’, ‘Adam Mickiewicz’, ‘Adam M’ - powinny znaleźć się wszystkie książki gdzie autor to: ‘Adam Mickiewicz’
    b. ‘ogniem’ - powinny znaleźć się wszystkie książki które w tytule mają frazę ‘ogniem’, np. ‘Ogniem i mieczem’, ‘Lód i ogień’ itp
5. * Zmień metodę `JdbcUsersDao.list()` tak żeby poza podstawowymi danymi użytkownika pobierała dane o ilości książek wypożyczonych przez danego czytelnika lub zero gdy użytkownik nie ma książek wypożyczonych. Dodaj tą informację do interfejsu administratora przy opcji nr 1 : ‘Lista wszystkich użytkowników’
6. * Do tabelki z książkami dodaj kolumnę ‘`limit`’, która określa ile maksymalnie książek może być wypożyczonych (czyli ile egzemplarzy jest na stanie biblioteki). Użyj tej kolumny przy metodach z klasy `JdbcOrdersDao` które dodają wypożyczenia i zwroty - tak żeby nie można było wypożyczyć książki której `limit = 0`. Dodatkowo obsłuż aktualizację limitu przy wypożyczeniu i zdaniu książki. Użyj transakcji do zapewnienia spójności danych:

Linki

- Martin Fowler - ORM Hate