Wydział Elektroniki
i Technik Informacyjnych
POLITECHNIKA WARSZAWSKA

Bazy Danych 1

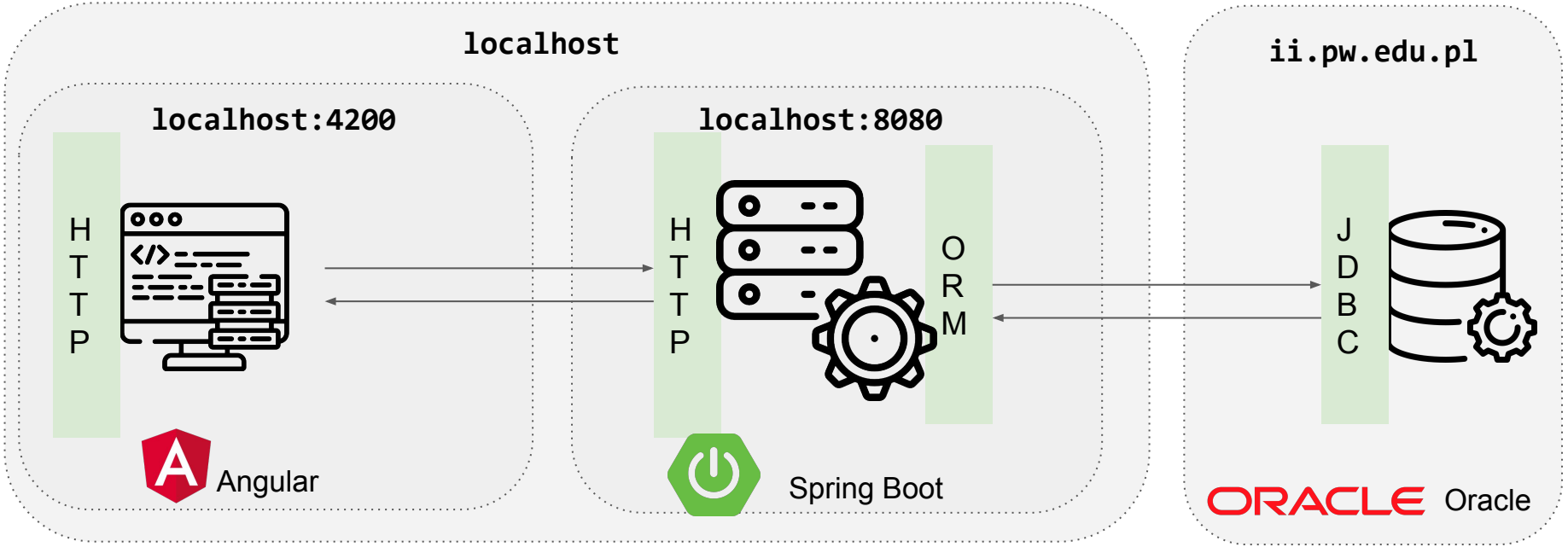
edycja 21L

Laboratorium 10

Przebieg laboratorium

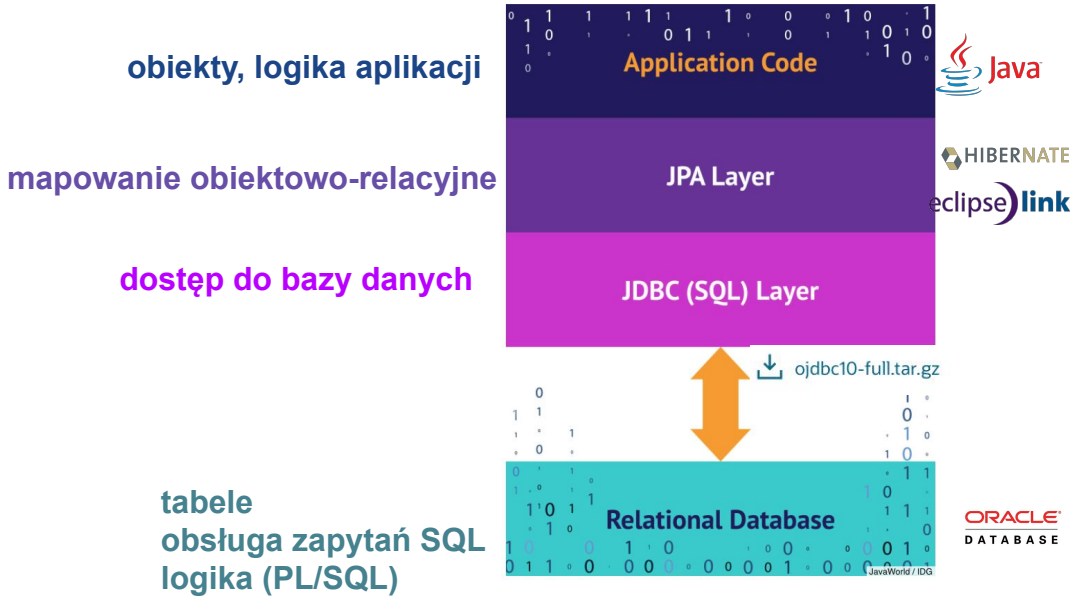
✏️ Integracja aplikacji Java z bazą danych

Przykładowa architektura aplikacji Full Stack



JDBC i JPA

- **Java Database Connectivity** - interfejs dostępu do bazy danych z aplikacji Java (JDBC)
- **Java Persistence API** - interfejs mapowania obiektowo-relacyjnego



Sposoby tworzenia aplikacji bazodanowych

Podstawowe dwa podejścia:

- DB-First
  - Projektowana baza danych, stworzenie schematu. Kod (obiekty) jako pochodna schematu bazy danych.
  - Moze byc wykorzystany zarówno JDBC jak i JPA
- Code-First
  - Tworzony jest kod aplikacji (klasy modelowe). Schemat bazy danych jest pochodną klas modelowych
  - Stworzenie schematu realizowane przez JPA.

JDBC

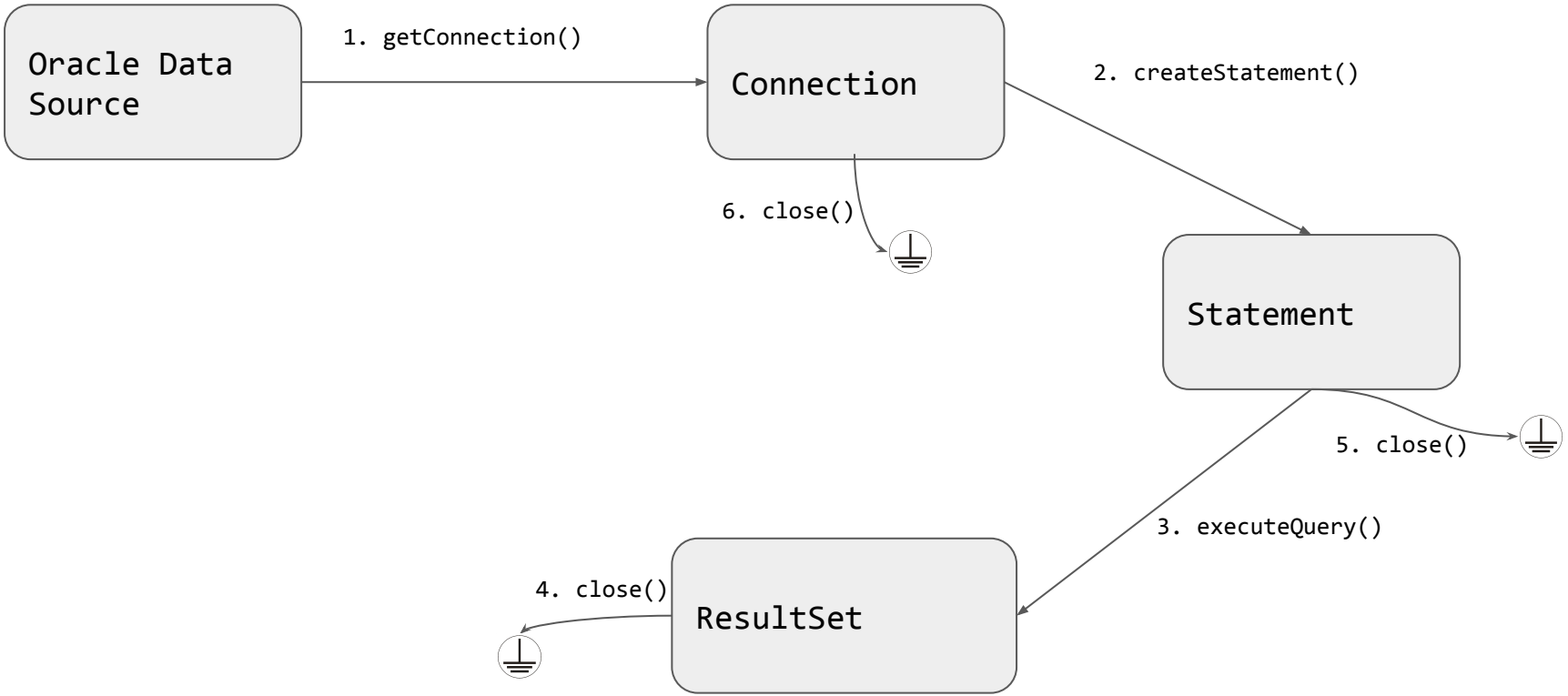
Wydanie poleceń SELECT, przyjęcie wyniku

Wydanie poleceń INSERT/UPDATE/DELETE oraz DDL

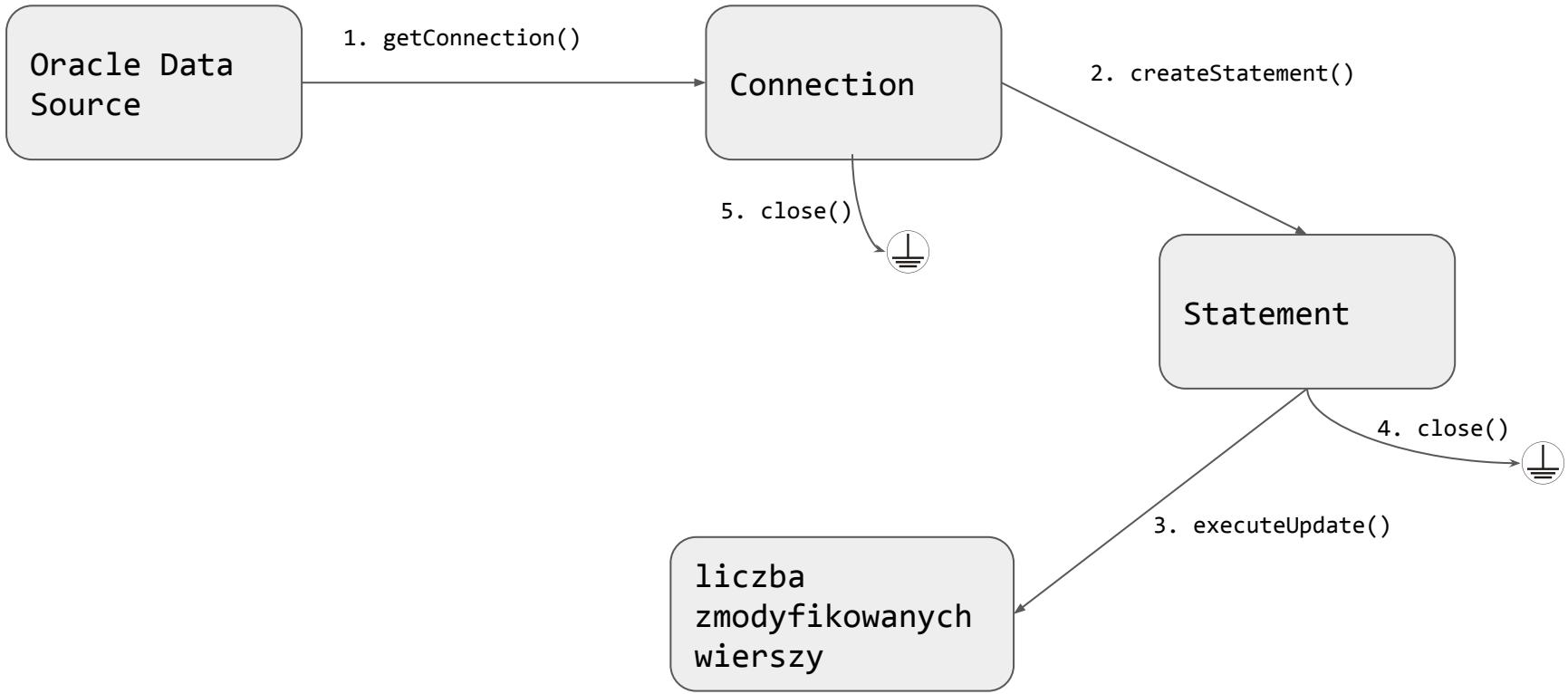
Wykonanie funkcji i procedur składowanych PL/SQL, przyjęcie wyników

Polecenia przygotowane

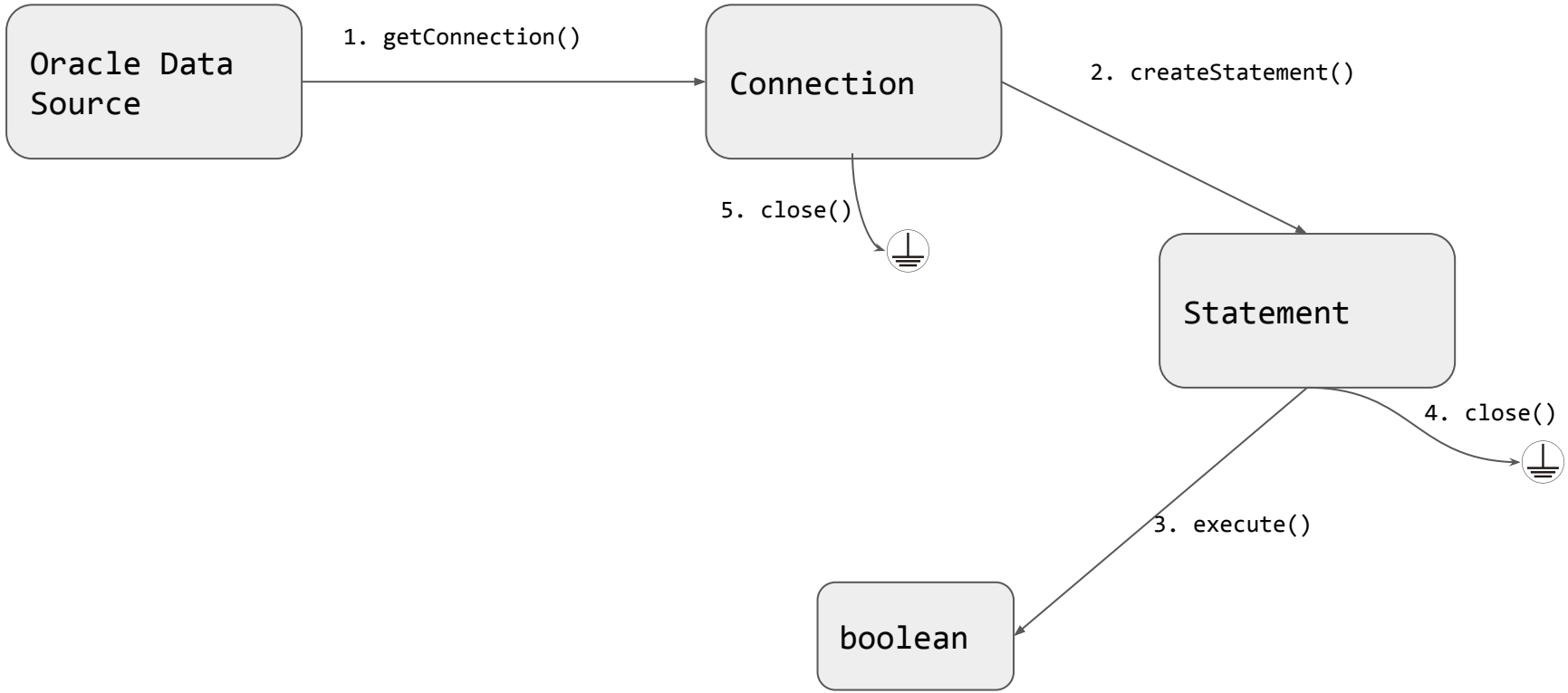
JDBC - wydanie poleceń SELECT



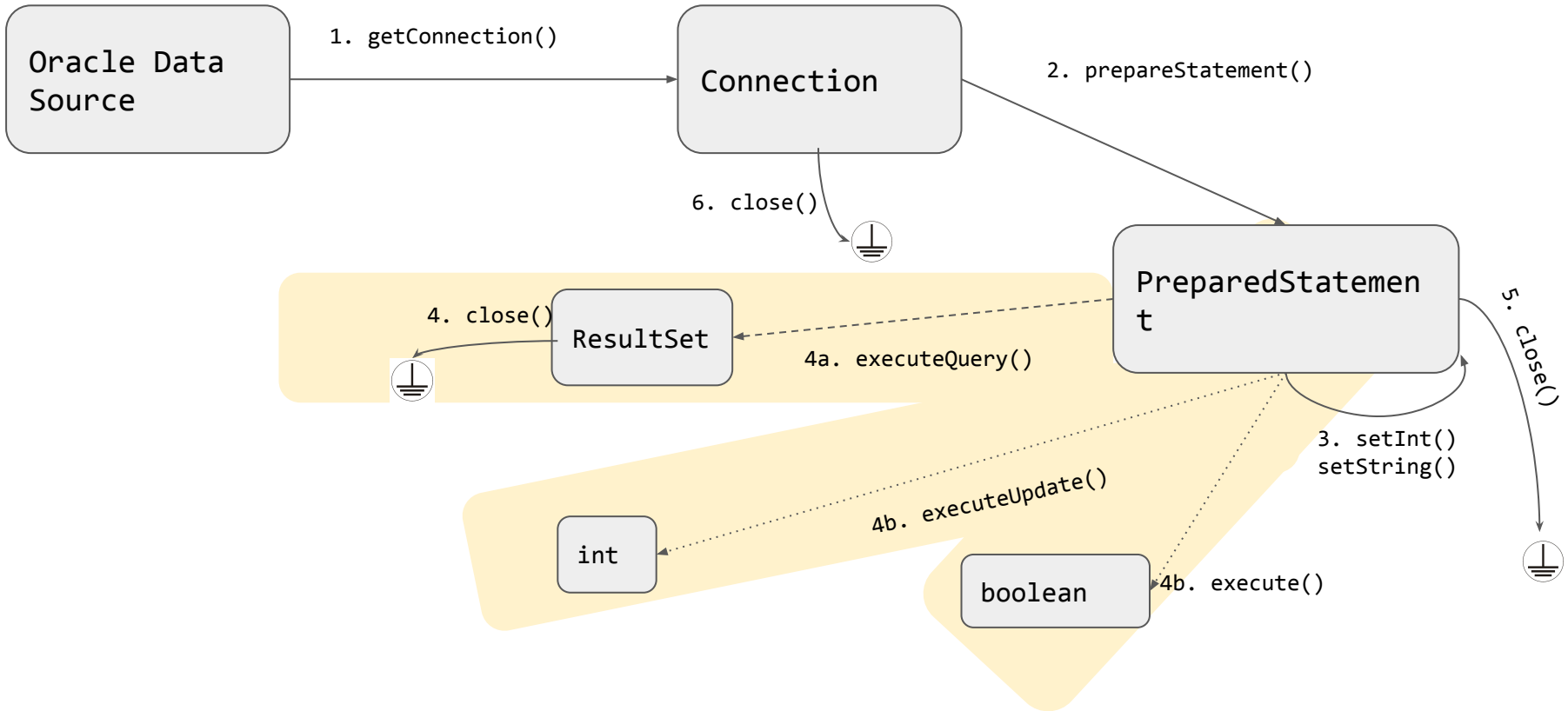
JDBC - wydanie poleceń DML



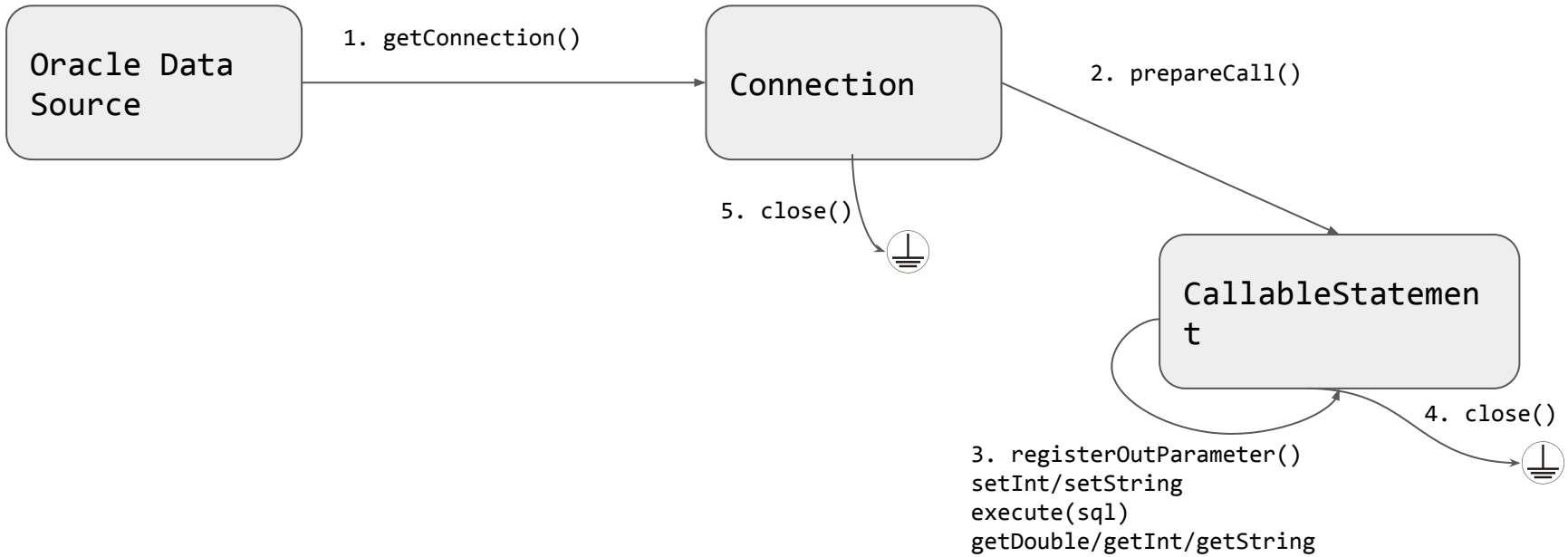
JDBC - wydanie poleceń DDL



JDBC - wydanie poleceń przygotowane



JDBC - wywołanie funkcji/procedury PL/SQL



JDBC - obsługa wyjątków (try-with-resources)

```
13  * Utworzenie tabeli 'dzialy' (JDBC).
14  */
15 import java.sql.*;
16
17 public class OBD_jdbc_3a {
18
19     public static void main(String[] args) {
20
21         String url = "jdbc:oracle:thin:@ora3.elka.pw.edu.pl:1521:ora3inf";
22         String uzytkownik = [redacted]
23         String haslo      = [redacted]
24         String sql1 = "CREATE TABLE dzialy (nr_dzialu integer not null, nazwa_dzialu varchar2(30), siedziba varchar2
25
26         try (Connection polaczenie = DriverManager.getConnection(url, uzytkownik, haslo)) {
27             System.out.println("AutoCommit: " + polaczenie.getAutoCommit());
28             try (Statement polecenie = polaczenie.createStatement()) {
29                 System.out.println("execute: " + polecenie.execute(sql1));
30             } catch (Exception e) {
31                 System.out.println("catch-1");
32                 e.printStackTrace();
33                 return;
34             }
35             finally {
36                 System.out.println("finally-1");
37             }
38         } catch (Exception e) {
39             System.out.println("catch-2");
40             e.printStackTrace();
41             return;
42         }
43         finally {
44             System.out.println("finally-2");
45         }
46         System.out.println("Sukces.");
47     }
48 }
```

Przykładowe aplikacja

https://gitlab-stud.elka.pw.edu.pl/aszmurlo/simple-emp-app - dla JDBC

https://gitlab-stud.elka.pw.edu.pl/aszmurlo/empservice - dla JPA

https://gitlab-stud.elka.pw.edu.pl/aszmurlo/empapp - dla JPA

Przydatne:

- IDE (Eclipse / IntelliJ IDEA)

Niezbędne:

- Sterownik bazy danych (ojdbc.jar)

Praca domowa

- Uruchom aplikację EmpApp.
- Zmodyfikuj parametry połączenia z bazą danych.
- Przetestuj działanie metod.
- Dokonaj poprawy kodu źródłowego w taki sposób, aby skorzystać z bloków try-with-resources.