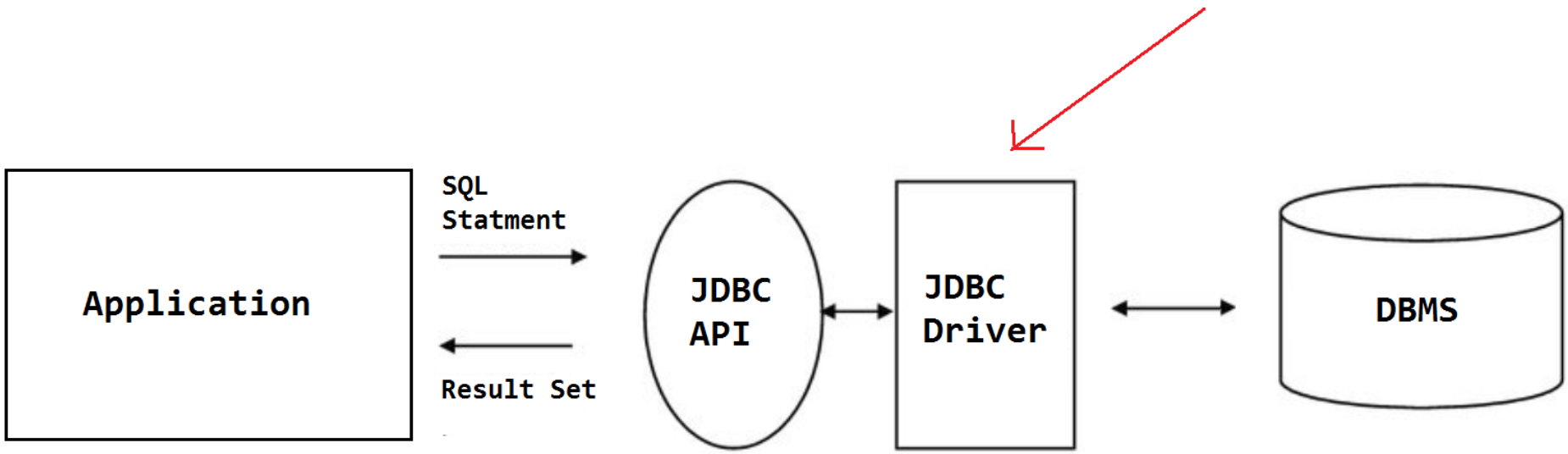
Technologia Programowania 2017/2018
Wykład 8

JDBC + Hibernate

Jakub Lemiesz

JDBC – Java DataBase Connectivity

1. JDBC – interfejs standaryzujący dostęp do relacyjnych baz danych dla aplikacji Java
2. Należy jedynie dostarczyć sterownik do wybranego systemu zgodny z tym interfejsem (*np. sqlserverjdbc.jar, postgresql.jar, derby.jar*)



Stawiamy serwer bazy danych Derby

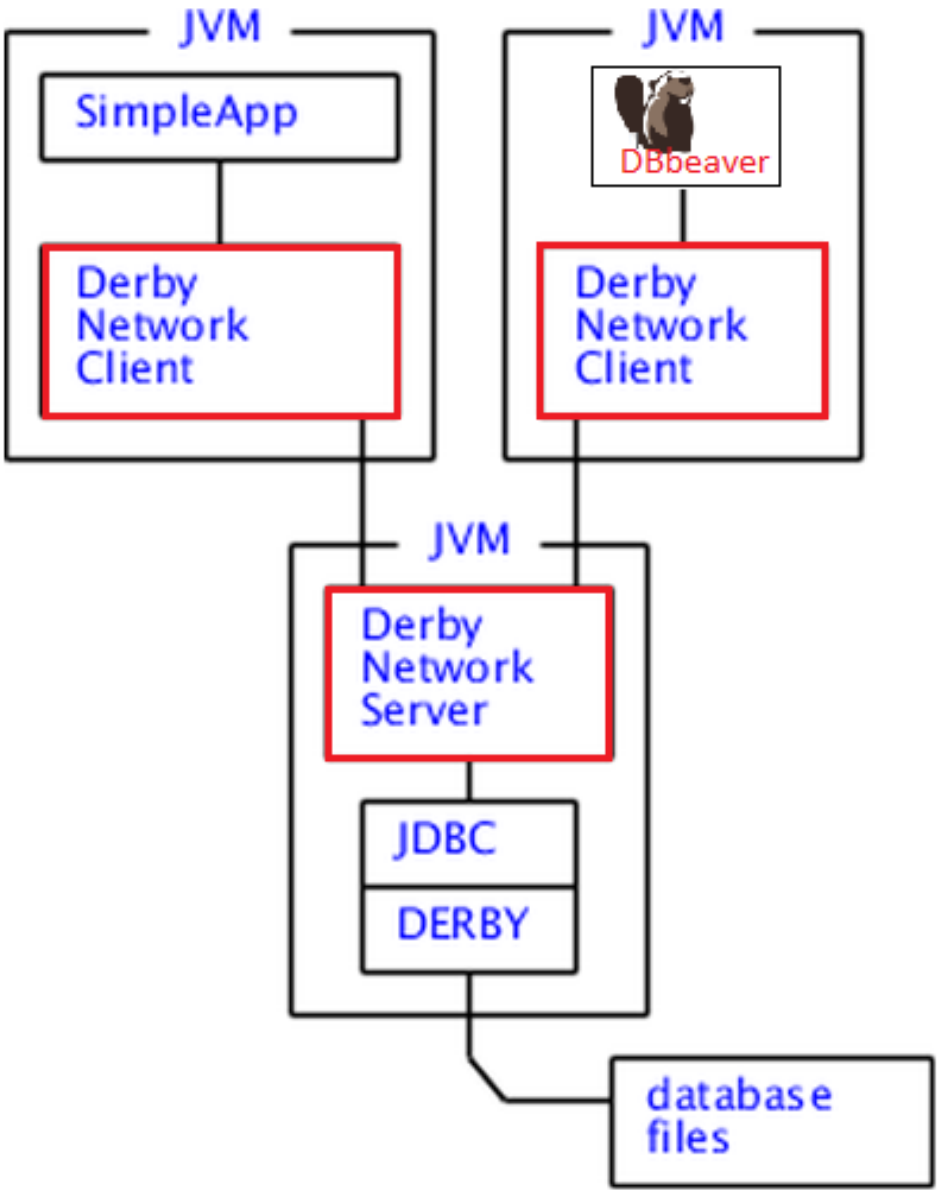
- Omawiamy na przykładzie Derby, ale dla innych systemów bazodanowych (*np. MySql, SQL Server*) jest analogicznie
- Apache Derby – system bazodanowy napisany w Java o małych wymaganiach systemowych (*np. urządzenia mobilne*)
- Może pracować jako niezależny proces (serwer) oraz w trybie wbudowanym (embedded), w ramach tej samej JVM bez odpalania osobnego procesu
- Pobieramy, odpalamy skrypty konfigurujące, a następnie serwer bazy danych na wybranym porcie localhosta:

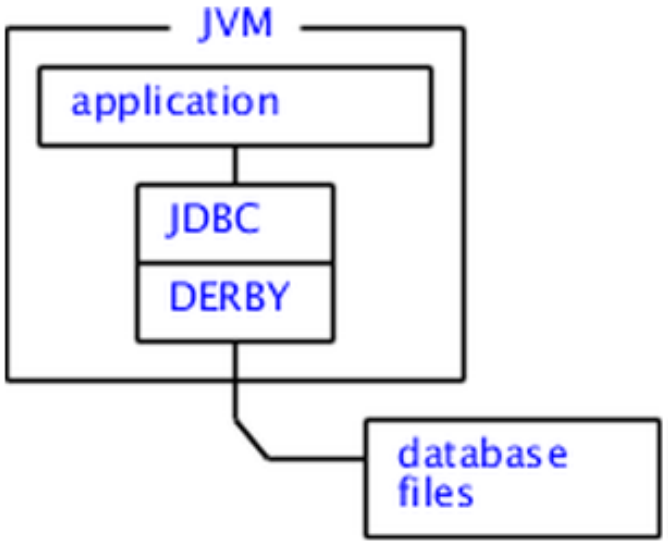
$$E{:}\backslash derby\backslash bin> \ NetworkServerControl.bat$$

$$E{:}\backslash derby\backslash bin> \ set*.bat$$

$$E{:}\backslash derby\backslash bin> \ startNetworkServer.bat \ \ -p \ \ 3000$$

Tryb serwera a tryb wbudowany



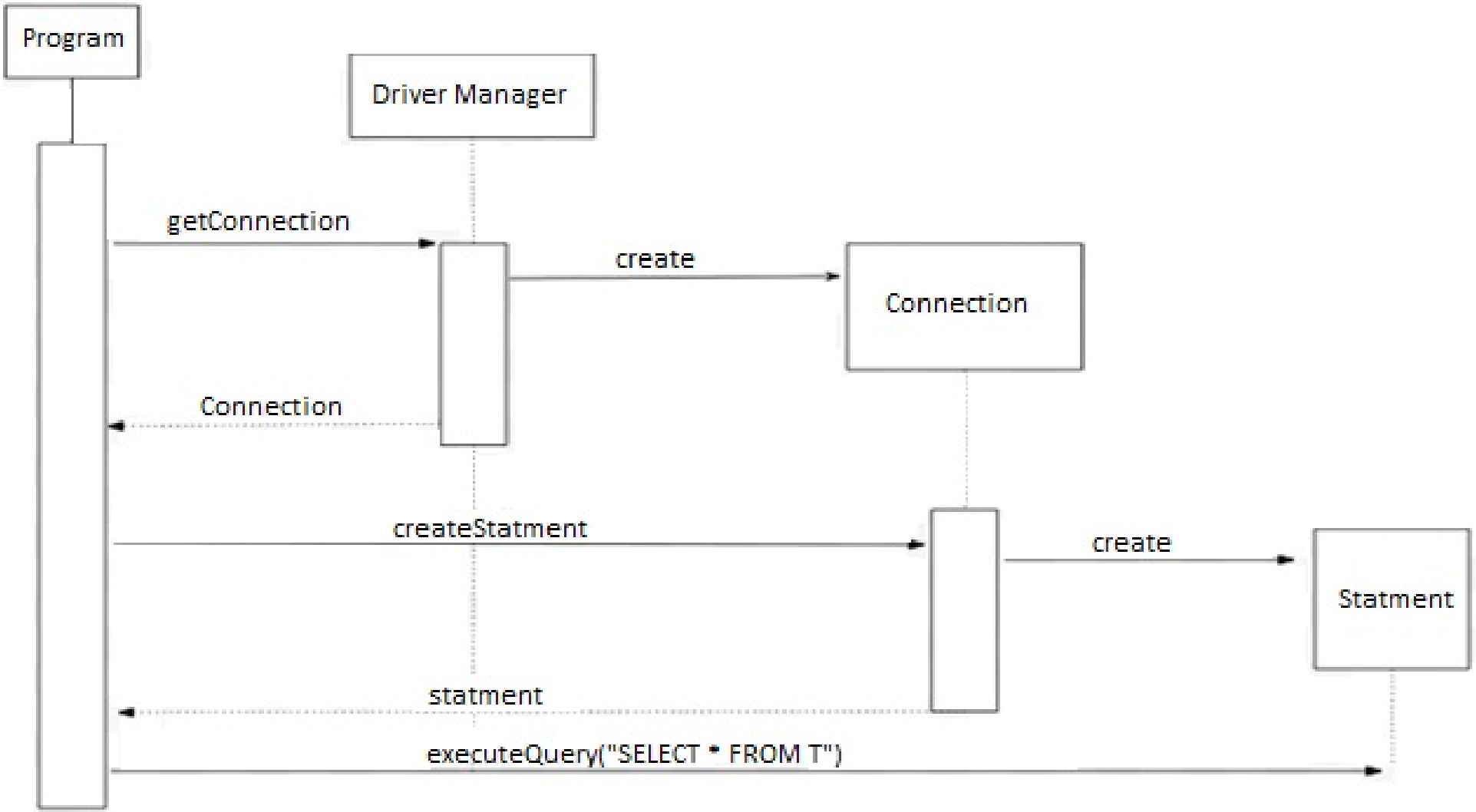


JDBC – najważniejsze komponenty

- import java.sql.DriverManager; (*class*)
- import java.sql.Connection; (*interface*)
- import java.sql.Statement; (*interface*)
- import java.sql.PreparedStatement; (*interface*)
- import java.sql.ResultSet; (*interface*)
- ...

Aby skorzystać z klas java.sql.*, należy pobrać odpowiedni sterownik JDBC i umieścić go w katalogu widocznym dla maszyny wirtualnej (*set classpath, Maven dependencies,...*)

JDBC – jaki to wzorzec?



JDBC – podstawowe metody

```
//DriverManager zwróci pasujący sterownik
//na podstawie adresu, GoF:Abstract Factory
String addr = "jdbc:derby://localhost:3000/kubaDB";

//cały kod opiera się na interfejsach:
//Connection, Statement, ResultSet
Connection conn = DriverManager.getConnection(addr);
Statement s = conn.createStatement();
s.execute("create table ...");
s.executeUpdate("insert ...");
conn.commit(); //GoF:Command

//zbiór wierszy spełniających zapytanie
ResultSet rs = s.executeQuery("select * from ...");
while(rs.next())
        System.out.println(rs.getInt(1));
```

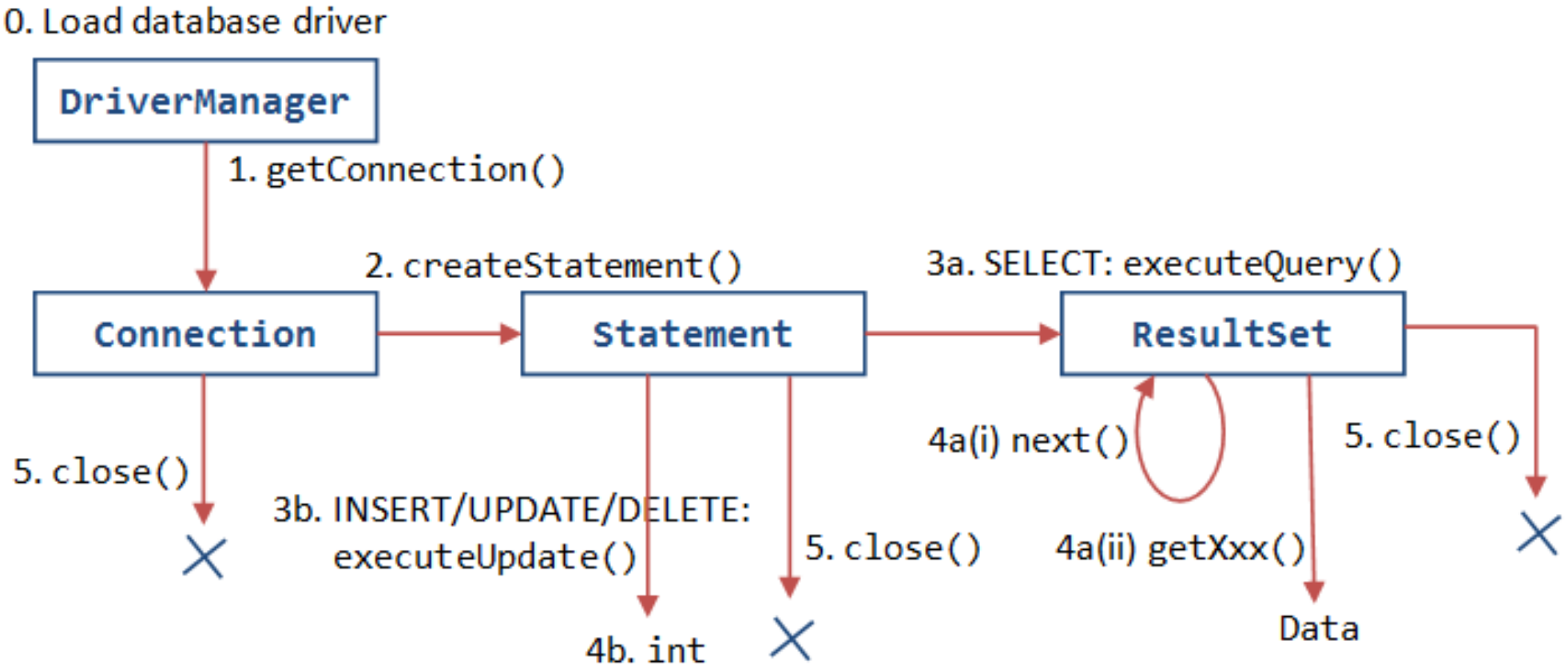
JDBC – podstawowe metody

1. s.executeQuery(...)
   zwraca zbiór wierszy reprezentowany przez ResultSet.
   ResultSet ma kursor wskazujący aktualnie przetwarzany rekord, przesuwany przez metodę `next()`.

2. s.executeUpdate(...)
   używa się do operacji INSERT, DELETE i UPDATE.
   Zwraca liczbę wierszy, na które zapytanie miało wpływ.

3. s.execute(...)
   zwraca true gdy pierwszy obiekt zwrócony przez zapytanie jest typu ResultSet. Gdy zapytanie zwraca kilka ResultSet przetwarzanie w pętli `s.getResultSet()`.
   Używa się przeważnie do operacji CREATE, DROP.

Przebieg komunikacji w JDBC



https://www3.ntu.edu.sg/home/ehchua/programming/java/images/JDBC_Cycle.png

PreparedStatement

```
// efficiently  add many rows by using
// predefined statement with parameters;
// the statement is compiled by DBMS only once

String sql = "insert into jdbc_student values (?, ?)"
PreparedStatement ps = conn.prepareStatement(sql);

ps.setInt(1, 193456);
ps.setString(2, "Jan Kowalski");
ps.executeUpdate();

ps.setInt(1, 191111);
ps.setString(2, "Anna Nowak");
ps.executeUpdate();
```

Przykład użycia JDBC

1. Uruchamiamy aplikację _JDBC_TEST (w trybie debuggowania)
2. Dodatkowo do podglądu zmian na serwerze bazy dany uruchamiamy program DBeaver
3. DBeaver – umożliwia interaktywny dostęp do dowolnej bazy (*używa JDBC, potrzebuje derby.jar*)

Co to jest mapowanie obiektowo-relacyjne?

1. Znaczna cześć tworzonych obecnie aplikacji jest zorientowana obiektowo
2. W biznesie dane typowo zapisuje się w relacyjnych bazach danych (*szybkość, poprawność*)
3. Mapowanie obiektowo-relacyjne to "autmatyczne" konwertowanie danych relacyjnych na obiekty (*i obiektów na dane relacyjne*)
4. Czy "czyste" JDBC wystarczy?

Problemy z JDBC, czyli po co nam ORM?

Chcę stworzyć dla programu obiektowego schemat bazy danych. Mogę ręcznie napisać kod SQL, tzn.

1. stworzyć tabele i ręcznie ustalać typy odwzorowujące pola klas,
2. ręcznie zdefiniować powiązania 1:1, 1:m, m:n (*klucze, klucze obce, tabele z powiązaniami*)
3. stworzyć kod umożliwiający automatyczne monitorowanie i przenoszenie zmian do bazy

Problemy z JDBC, czyli po co nam ORM?

4. ręcznie tworzyć zapytania typu "3 x join":

   ORM:

   ```
   "SELECT * FROM Bill b WHERE b.Client.Addr.City=..."
   ```

5. ręcznie inicjalizować obiekty i pola wyciągając wiersz po wierszu, pole po polu z *ResultSet*

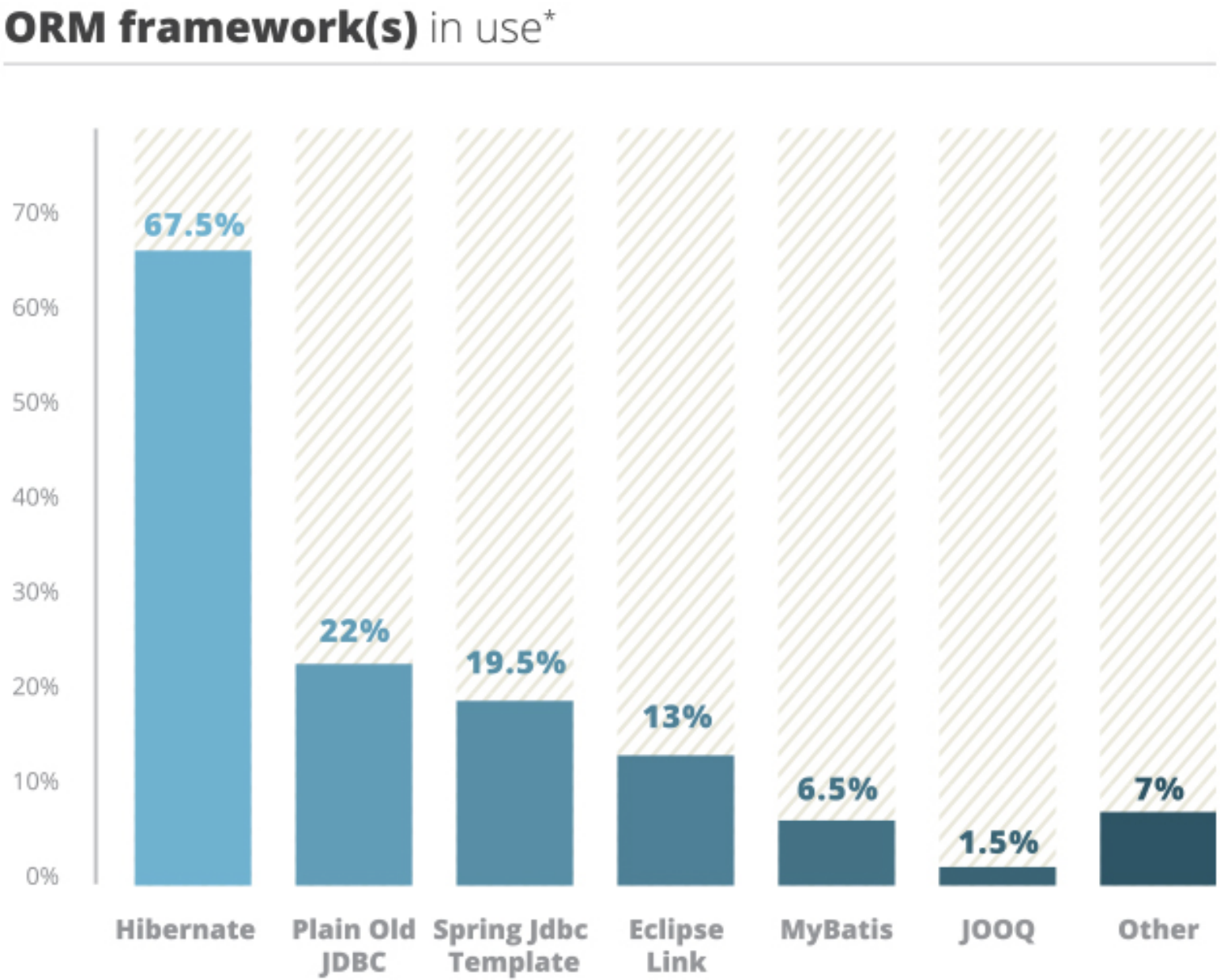
ORM pobiera powiązane dane automatycznie:

```
Query query = session.createQuery("FROM Student WHERE...");
List<Student> studentList = query.list();
studentList.get(0).getLectures().get(0).getName();
```

Hibernate

- Systemy ORM
  - Hibernate (Red Hat, Inc.)
  - NHibernate (dla .NET)
  - EclipseLink
  - TopLink (Oracle)
  - ...
- Java Persistence API (JPA) – standard ORM dla Java. Duży wpływ na jego kształt miał Hibernate.

Popularnosc systemów ORM - źródło



Uruchamianie aplikacji z Hibernate

- Instalacja Hibernate: łatwo przez Maven'a i pom.xml
- Konfiguracja pliku hibernate.cfg.xml

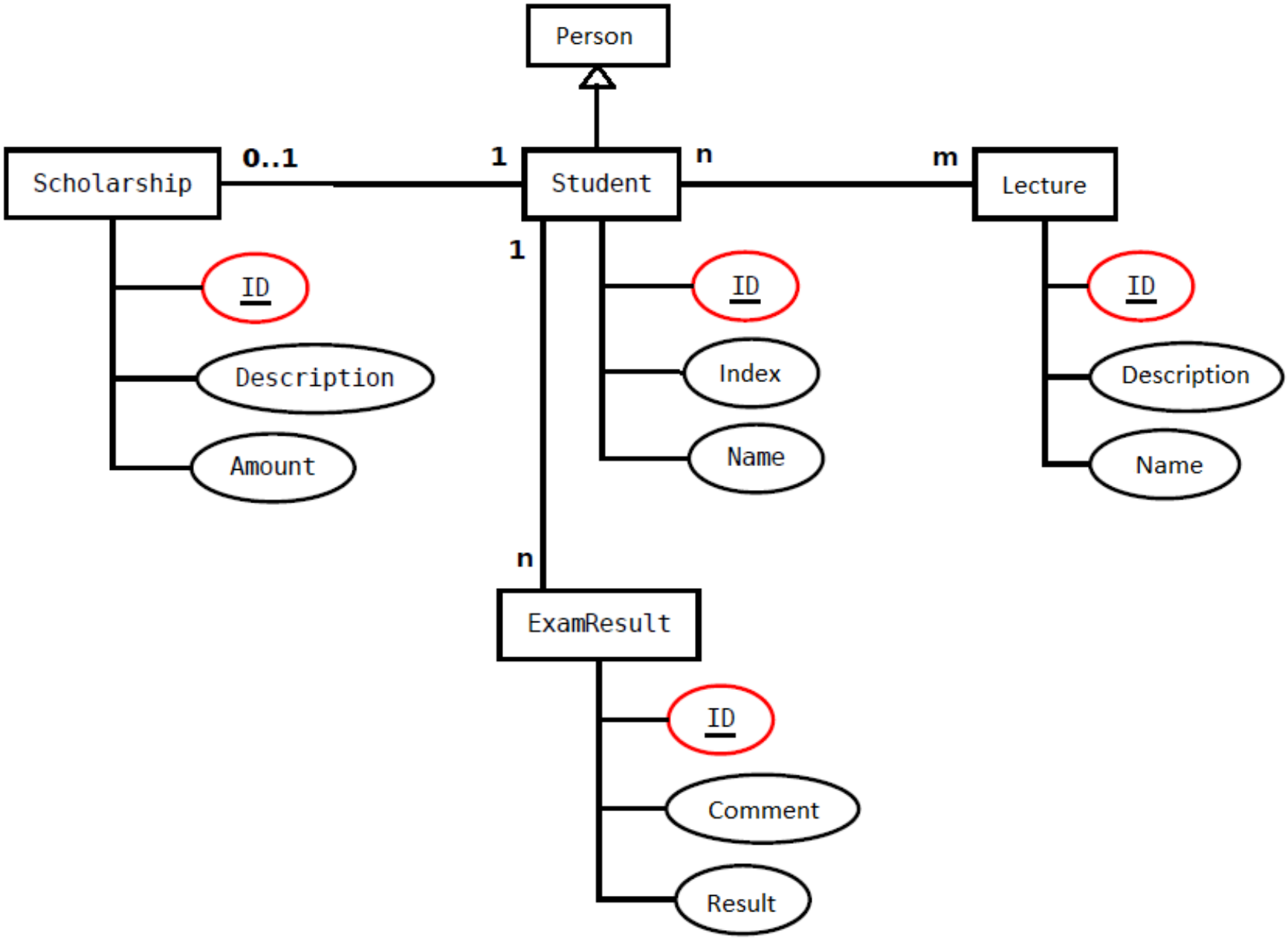
hibernate.cfg.xml

```
<property name="connection.driver_class">
        org.apache.derby.jdbc.ClientDriver
</property>

<property name="connection.url">
        jdbc:derby://localhost:3000/kubaDB
</property>

<mapping class="Student" />
<mapping class="ExamResult" />
<mapping class="Scholarship" />
<mapping class="Lecture" />
```

Co robi nasza aplikacja? (*projekt _Hibernate_TEST*)



Używanie Hibernate ≡ dopisujemy adnotacje

@Entity	Specifies that the class is an entity
@Id	Specifies the primary key of an entity
@Transient	Specifies that the field is not persistent
@MappedSuperclass	Designates a class whose mapping information is applied to the entities that inherit from it. A mapped superclass has no separate table defined for it.
@GeneratedValue	The `GeneratedValue` annotation may be applied to a primary key property or field of an entity or mapped superclass in conjunction with the `Id` annotation.

@Column i walidacja pól

```
@Column(name = "SEX")
@Pattern(regexp = "F|M")
private String sex;
```

Używanie Hibernate ≡ dopisujemy adnotacje

@OneToOne	Defines a single-valued association to another entity that has one-to-one multiplicity.
@FetchType	LAZY = do not load referenced entity, until it is accessed for the first time (domyślny) EAGER = load referenced entity immediately
@CascadeType	Defines the set of cascadable operations that are propagated to the associated entity. ALL is equivalent to cascade={PERSIST, MERGE, REMOVE, REFRESH, DETACH}
mappedBy	References the field that "owns" the relationship in the referenced entity. Required unless the relationship is unidirectional.

@OneToMany, @ManyToOne, @ManyToMany

```
@MappedSuperclass
public class PersistenceObject {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="index")
    private int id;
    ...

@MappedSuperclass
public class Person extends PersistenceObject {
     Column(name="NAME")
     private String name;

     @Column(name = "AGE")
     @Min(0)
     @Max(99)
     private int age;
```

```
@Entity
@Table(name="Student")
public class Student extends Person {

    @Transient
    private String poleNieDoBazy;

    @OneToOne(fetch = FetchType.LAZY,
              mappedBy="grantedTo")
    private Scholarship scholarship;

    @OneToMany(cascade = CascadeType.ALL,
               mappedBy="student")
    private List<ExamResult> examResults;

    @ManyToMany(mappedBy = "students")
    private List<Lecture> lectures;
```

Po koleji, co robimy?

1. Tworzymy plik konfiguracyjny w XML, piszemy adnotacje (*mamy połączenie i schemat bazy*)
2. SessionFactory w oparciu o plik konfiguracyjny hibernate.cfg.xml będzie tworzyć sesje
3. Pobieramy sesję z SessionFactory,
   otwieramy w niej nową transakcję,
   wykonujemy zmiany na obiektach,
   wykonujemy commit(),
   otwieramy nową transakcję...

Po co jest sesja?

1. Sesja od momentu stworzenia do zamknięcia utrzymuje połączenie z bazą
2. Sesja widzi wybrane* obiekty w programie i przenosi aktualne informacje o nich do bazy

*W kontekście Hibernate każdy obiekt może być w jednym z 3 stanów: Transient, Persistent, Detached. Sesja widzi tylko te obiekty, które są Persistent (*ma tzw. 'persistence context'*)

Obiekty Transient

- Sesja Hibernate nic nie wie o obiektach w tym stanie, GC je usuwa gdy nie ma referencji i ślad po nich ginie
- Nowo tworzone obiekty, obiekty usunięty ze bazy

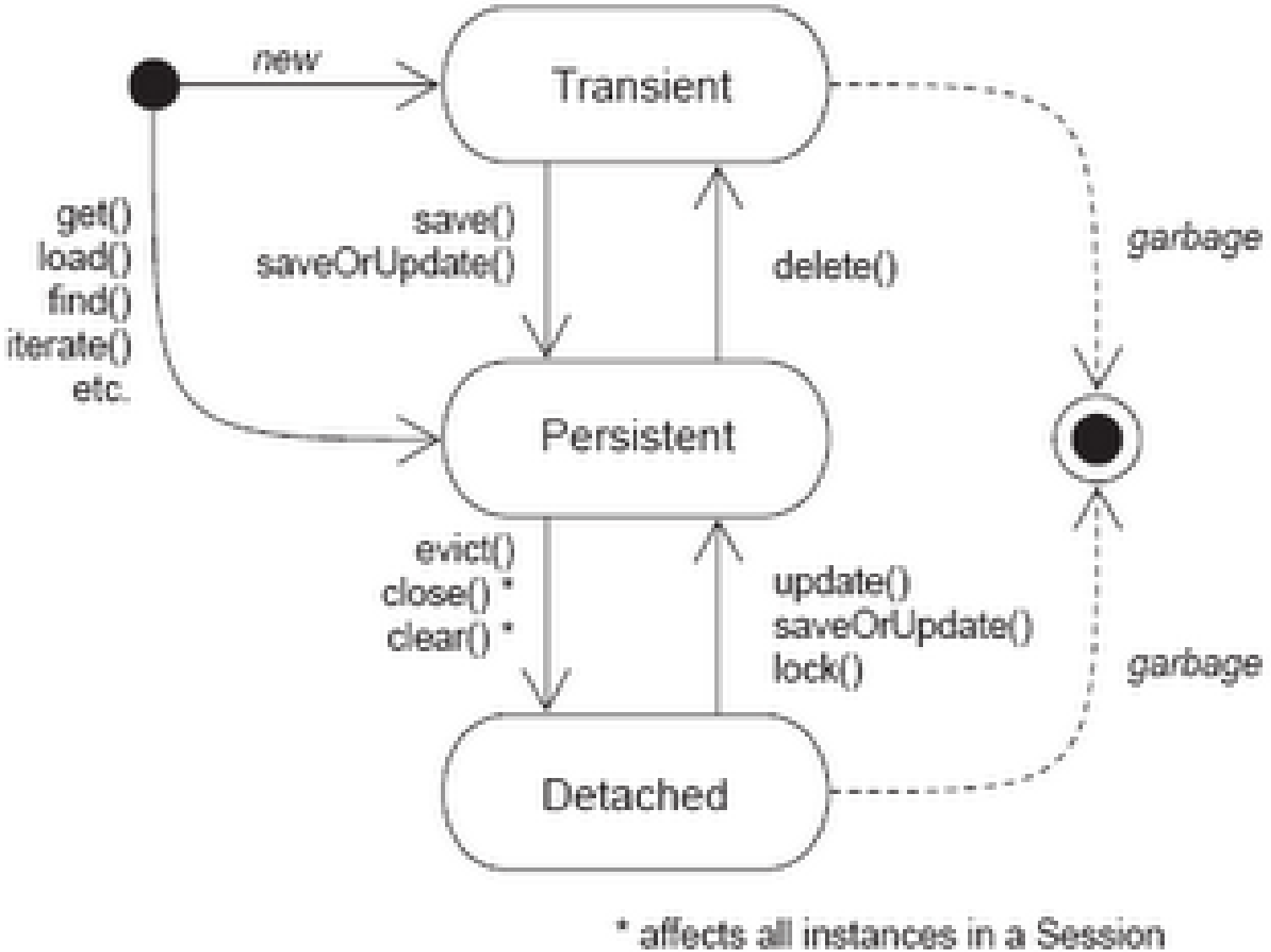
Obiekty Persistent

- Sesja Hibernate 'widzi' obiekty w stanie persistent, wszystkie zmiany w nich są zapisywane do bazy
- Mogą to być obiekty transient na których wykonano operację *save* lub obiekty wczytane z bazy

Obiekty Detached

- Np. obiekty persistent po zamknięciu sesji (*mogą być przywrócony do stanu persistent w innej sesji*)
- Jest z nimi związany rekord w bazie, ale nie są zarządzane przez Hibernate (*zmiany w nich nie są odnotowywane w bazie, mogą zostać usunięte przez GC*)

Cykl życia obiektów w sesji Hibernate – źródło



Hibernate - sesja

```
Session session = getSessionFactory().openSession();

session.beginTransaction();
Student student1 = new Student();//obiekt transient
student1.setName("Zdzisław Hibernate");
session.save(student1);//obiekt persistent

Student student2 = new Student();
student2.setName("Anna Zdolna");
session.save(student2);
session.getTransaction().commit();

session.beginTransaction();
...
session.close();
```

Hibernate - sesje

```
Session session1 = sessionFactory.openSession();
Student student = session1.get(Student.class, 1);
session1.close();

// ‘detached’ - modification is ignored by Hibernate

student.setName("Zenek");

//re-attach the object, returning  to‘persistent’

Session session2 = sessionFactory.openSession();
session2.update(student);
session2.getTransaction().commit();
```

commit() vs. flush()

1. Zmiany w 'persistence context' są automatycznie przenoszone do bazy metodą flush() wywoływaną:
   - ręcznie
   - przed wykonaniem zapytania do bazy
   - przy commitowaniu transakcji
   - przy zamykaniu sesji
2. Wykonanie 'commit()' jedynie zatwierdza zmiany

commit() vs. flush() – przykład

```
Transaction tran = session.beginTransaction();
for ( int i=0; i<1000; i++ )
{
    session.save( new Student(i,...) );
    if ( i % 20 == 0 ) {
        session.flush();
        session.clear();
    }
}
tran.commit();
session.close();
```

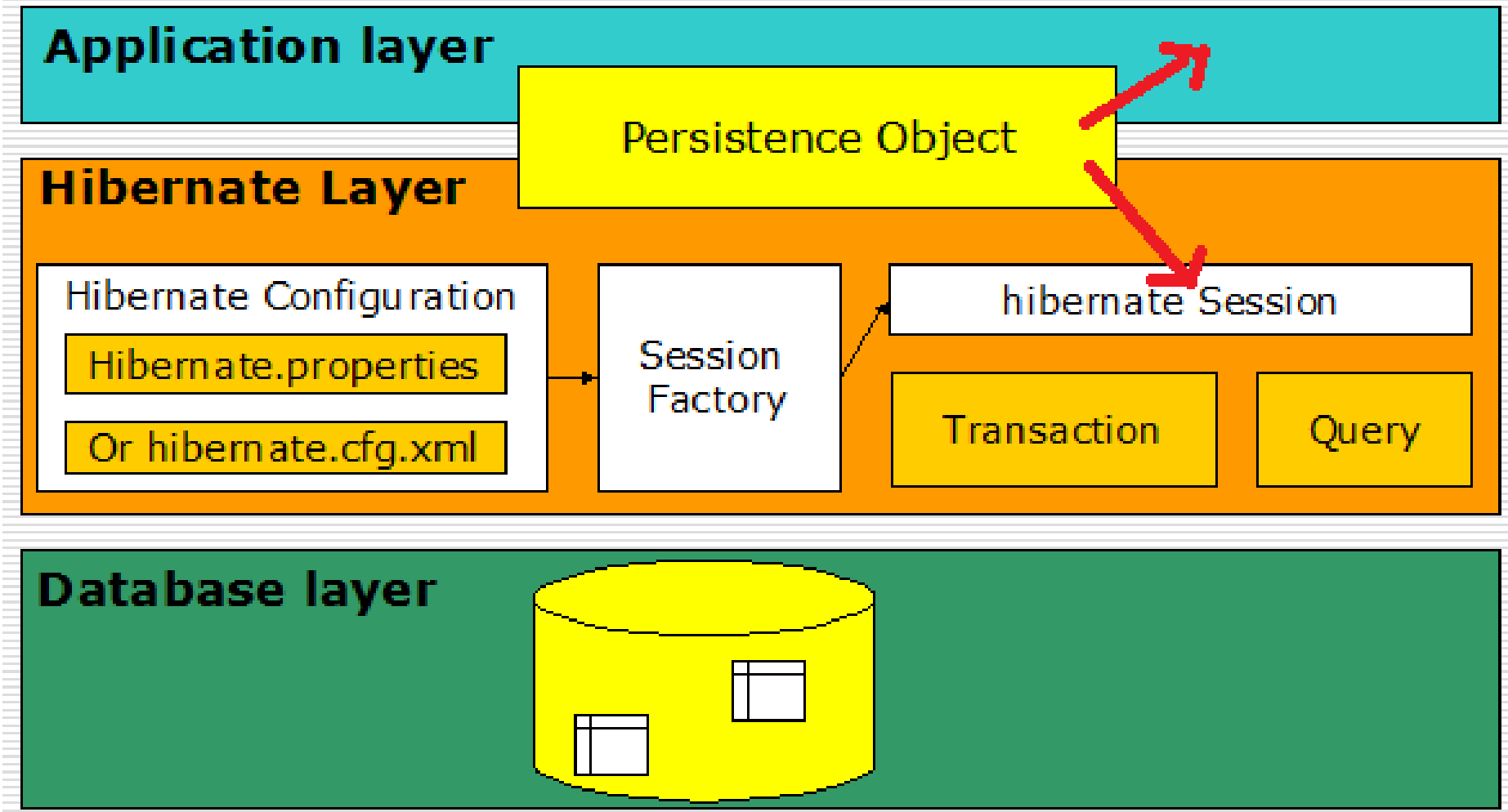
Zapytania do bazy z użyciem HQL

```
String hql = "FROM Student S WHERE S.id > 0
                                ORDER BY S.name DESC";

Query query = session.createQuery(hql);
List<Student> studentList = query.list();

println(
"Student " +
studentList.get(0).getName() +
"jest zapisany na: " +
studentList.get(0).getLectures().get(0).lectureName()
);
```

Architektura Hibernate



Dla zainteresowanych tematem polecam