ORM i Hibernate

Programowanie obiektowe II

Dr inż. Wojciech Kozioł

Uniwersytet Rzeszowski

ORM

Mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping ORM) – sposób odwzorowania obiektowej architektury systemu informatycznego na bazę danych (lub inny element systemu) o relacyjnym charakterze.

Implementacja takiego odwzorowania stosowana jest m.in. w przypadku, gdy tworzony system oparty jest na podejściu obiektowym, a system bazy danych operuje na relacjach.

(źródło: Wikipedia)

ORM

Korzystanie z ORM wymaga:

- Utworzenia modelu danych w języku obiektowym – klasy i relacje występujące między nimi.
- Utworzenia schematu bazy danych, który będzie odpowiadał utworzonemu wcześniej modelowi obiektowemu.
- Zdefiniowania odwzorowania modelu relacyjnego na bazę danych.
- Utworzenia aplikacji w paradygmacie obiektowym operującej na utworzonych modelach obiektowych (programista nie musi się martwić w jaki sposób jego obiekty będą utrwalone).

Podczas pobierania obiektów z bazy, ich utrwalania, aktualizacji bądź usuwania programista wykorzystuje API stosowanego narzędzia ORM.

JPA

Java Persistence API (skrót JPA) – oficjalny standard mapowania obiektowo-relacyjnego (ORM) firmy Sun Microsystems dla języka programowania Java.

(źródło: Wikipedia)

JPA nie jest konkretnym narzędziem. Stanowi ona pewien standard, specyfikację, interfejs programistyczny określający jak ma działać narzędzie JPA. Istnieją różne implementacje JPA np.: Hibernate, JDO, TopLink itp.

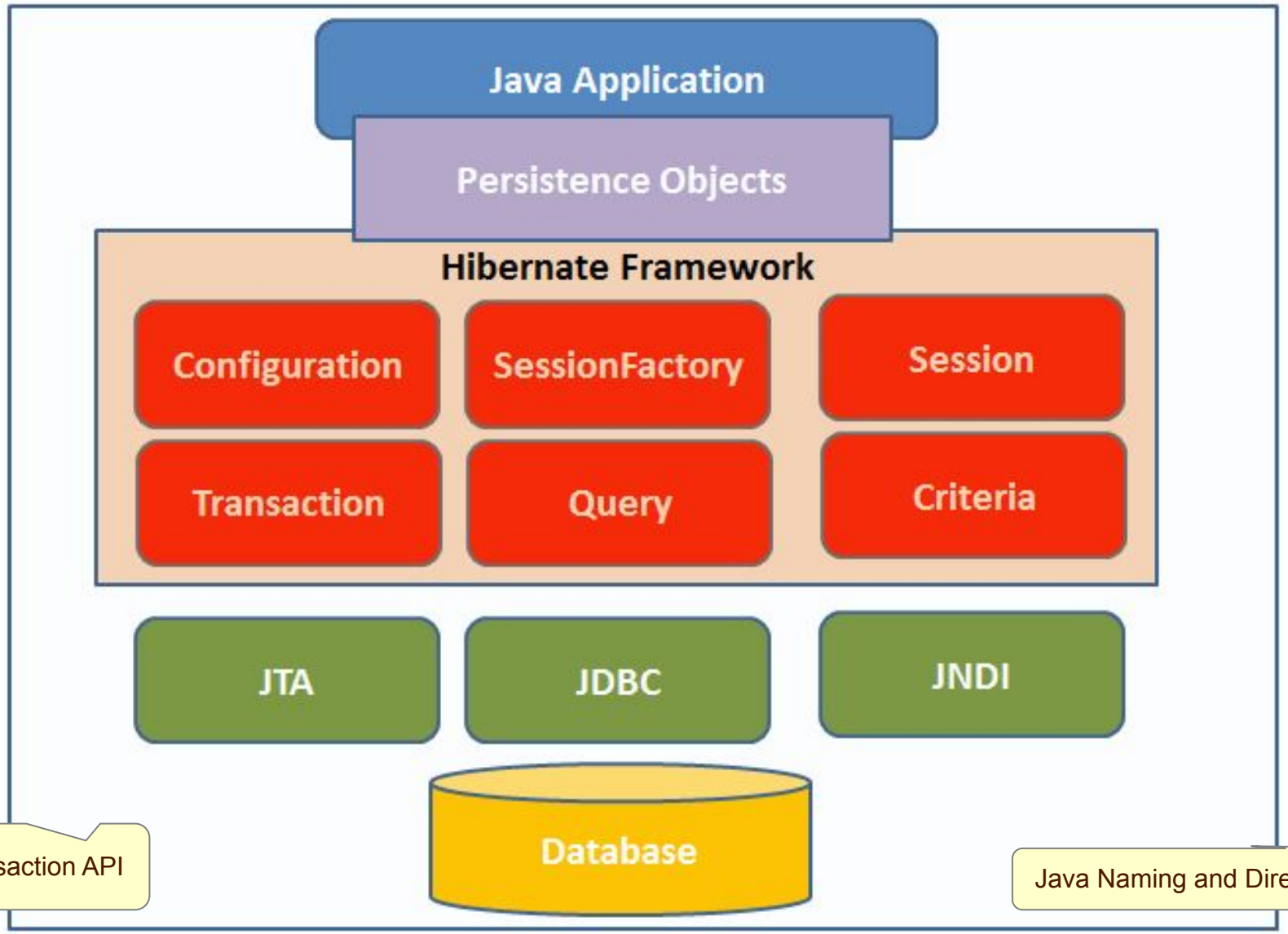
Hibernate ORM

Hibernate:

1) jest obecnie jednym z najpopularniejszych frameworków ORM. Projekt ten jest tworzony przez firmę JBoss Inc. na licencji open source.
2) zapewnia translację danych pomiędzy światem obiektowym a relacyjną bazą danych.
3) umożliwia zwiększenie wydajności operacji wykonywanych na bazie danych poprzez buforowanie oraz minimalizację liczby zapytań.
4) ma na celu zwolnienie programistów z ręcznego kodowania większości funkcjonalności związanych z utrwalaniem danych w bazach danych.

Można go pobrać ze strony http://www.hibernate.org .

Architektura Hibernate



Źródło:http://www.onlinetechvision.com/wpcontent/uploads/2011/09/Hibernate_Architecture.png

Configuration

1) Obiekty klasy Configuration służą do konfiguracji Hibernate. Zdefiniowana konfiguracja określa sposób działania biblioteki w tworzonej aplikacji.
2) Znajdują się tu m.in. właściwości dotyczące łączenia się Hibernate z bazą danych, wybór dialektu dla bazy danych danych i inne.

Configuration (cd..)

1. Konfiguracja Hibernate bez użycia plików konfiguracyjnych XML

```
Configuration configuration = new Configuration().
    setProperty("connection.driver_class","com.mysql.jdbc.Driver").
    setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/dbname").
setProperty("hibernate.connection.username", "root").
    setProperty("hibernate.connection.password", "").
    setProperty("dialect", "org.hibernate.dialect.MySQLDialect").
    setProperty("hibernate.hbm2ddl.auto", "validate").
    addAnnotatedClass(Student.class).
    addAnnotatedClass(Przedmiot.class);
```

2. Można również utworzyć plik konfiguracji definiując ścieżki do zasobów XML przechowujących konfiguracje i mapowanie.

```
Configuration configuration = new Configuration().
     configure("cfg\\hibernateMySQL.cfg.xml").
     addResource("cfg\\plikMapowania.hbm.xml");
```

3. Często mapowanie osadzone jest bezpośrednio w pliku konfiguracyjnym, wtedy podawana jest jedynie ścieżka do tego pliku:

```
Configuration configuration = new Configuration().
    configure("cfg\\hibernateMySQL.cfg.xml");
```

Configuration (cd..)

1. Konfiguracja Hibernate bez użycia plików konfiguracyjnych XML

```
Configuration configuration = new Configuration().
    setProperty("connection.driver_class","com.mysql.jdbc.Driver").
    setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/dbname").
setProperty("hibernate.connection.username", "root").
    setProperty("hibernate.connection.password", "").
    setProperty("dialect", "org.hibernate.dialect.MySQLDialect").
    setProperty("hibernate.hbm2ddl.auto", "update").
    addAnnotatedClass(Student.class).
    addAnnotatedClass(Przedmiot.class);
```

2. Można również utworzyć plik konfiguracji definiując ścieżki do zasobów XML przechowujących konfiguracje i mapowanie.

```
Configuration configuration = new Configuration().
     configure("cfg\\hibernateMySQL.cfg.xml").
     addResource("cfg\\plikMapowania.hbm.xml");
```

3. Często mapowanie osadzone jest bezpośrednio w pliku konfiguracyjnym, wtedy podawana jest jedynie ścieżka do tego pliku:

```
Configuration configuration = new Configuration().
    configure("cfg\\hibernateMySQL.cfg.xml");
```

Mapowanie OR nie musi być w tym przypadku definiowane w pliku XML. Można dodać klasy z adnotacjami przy użyciu metod `addAnnotatedClass` wywołując je po metodzie `configure` tj. w podpunkcie 1.

Configuration (cd..)

Plik konfiguracyjny w formie `hibernateMySQL.cfg.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>
    <!-- Database connection settings -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.url">jdbc:mysql://localhost/dbname</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.username">root</property>
    <property name="connection.password"/>
    <property name="hbm2ddl.auto">validate</property>

    <mapping package="hibernatestudentdb" resource="cfg/student.hbm.xml"/>
  </session-factory>

</hibernate-configuration>
```

W tym przypadku mapowanie zdefiniowane jest w osobnym pliku student.hbm.xml. Można również zdefiniować mapowanie bezpośrednio w pliku konfiguracyjnym, jednak zazwyczaj stosuje się do tego pliki zewnętrzne.

Gdy mapowanie definiowane jest w pliku XML nie ma potrzeby stosowania adnotacji w mapowanych klasach.

Configuration (cd..)

Plik mapowania `student.hbm.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="hibernatestudentdb.Student" table="student">
        <id name="studentId" column="idStudent">
            <generator class="increment"/>
        </id>
        <property name="imie" column="fistName" />
        <property name="nazwisko" column="secondName" />
        <property name="nrIndeksu" column="indexNumber" />
   </class>
</hibernate-mapping>
```

Configuration (cd..)

Istnieje również możliwość podania nazw mapowanych klas zawierających adnotacje bezpośrednio w pliku konfiguracyjnym. Są to klasy: `Student, Adres, Przedmiot,` które znajdują się w tym przypadku w pakiecie `hibernatestudentdb.`

W takim przypadku nie ma potrzeby definiowania mapowania w plikach XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>
    <!-- Database connection settings -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.url">jdbc:mysql://localhost/dbname</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.username">root</property>
    <property name="connection.password"/>
    <property name="hbm2ddl.auto">validate</property>

    <mapping class="hibernatestudentdb.Student" />
    <mapping class="hibernatestudentdb.Adres" />
    <mapping class="hibernatestudentdb.Przedmiot" />

  </session-factory>

</hibernate-configuration>
```

SessionFactory

1) Twórcy Hibernate zaprojektowali interfejs SessionFactory z myślą o współdzieleniu aplikacji.
2) Gdy w ramach jednej aplikacji istnieje potrzeba łączenia się z kilkoma bazami danych, należy wtedy dla każdej z nich utworzyć oddzielny obiekt SessionFactory.
3) Obiekt SessionFactory produkuje obiekty Session, poprzez które następuje komunikacji z bazą danych.
4) Obiekt SessionFactory służy do buforowania poleceń SQL wygenerowanych przez Hibernate. Przechowuje on również meta-dane odwzorowania wykorzystywane przez Hibernate podczas działania aplikacji.
5) Obiekt SessionFactory służy również do buforowania danych odczytanych z pojedynczej jednostki zadaniowej w celu udostępnienia tych danych innej jednostce zadaniowej.

Session

1) Session jest głównym interfejsem w każdej aplikacji wykorzystującej Hibernate.
2) Obiekty klasy Session są lekkie i generują niewielki koszt ich tworzenia i usuwania. Jest to ważne z punktu widzenia wydajności. Hibernate prawdopodobnie tworzy nową sesję przy każdym żądaniu.
3) Sesje reprezentują pojedynczą jednostkę pracy.
4) Stanowią one bufor (kolekcję załadowanych obiektów, które są powiązane jedną jednostką zadaniową).
5) Hibernate jest w stanie zmiany zachodzące w obiektach w ramach danej jednostki zadaniowej (sesji).

Definiowanie obiektów do mapowania

Obiekty używane do mapowania obiektowo-relacyjnego to zwykłe obiekty Java. Z uwagi jednak na to, że Hibernate najlepiej współpracuje z klasami spełniającymi reguły POJO (ang. Plain Old Java Object), podczas tworzenia klas zaleca się stosowanie następujących reguł:

- klasa musi implementować pusty interfejs Serializable,
- klasa musi posiadać bezparametrowy konstruktor,
- zaleca się aby klasa zawierała specjalny identyfikator serialVersionUID,
- klasa oraz jej pola i metody nie mogą być oznaczone słowem kluczowym final.

Klasa student

```
public class Student implements Serializable{

    private static final long serialVersionUID = -300025L;

    private int studentId;
    private String imie;
    private String nazwisko;
    private long nrIndeksu;

    public Student(String imie, String nazwisko, long nrIndeksu)       {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
    }

    public Student() { }


    public int getStudentId() { return studentId; }

    public void setStudentId(int studentId) { this.studentId = studentId; }

    public String getImie() { return imie; }

    public void setImie(String imie) { this.imie = imie; }

    public String getNazwisko() { return nazwisko; }

    public void setNazwisko(String nazwisko) { this.nazwisko = nazwisko; }

    public long getNrIndeksu() { return nrIndeksu; }

    public void setNrIndeksu(long nrIndeksu) { this.nrIndeksu = nrIndeksu; }
}
```

Pole serialVersionUID nie podlega seralizacji

Klasa wymaga zdefiniowania konstruktora bezparametrowego
z uwagi na, to że istnieje w niej inna postać konstruktora z parametrami

Plik konfiguracyjny Hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>
    <!-- Database connection settings -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.url">jdbc:mysql://localhost/BazaStudentow</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.username">root</property>
    <property name="connection.password"/>
    <property name="hbm2ddl.auto">update</property>

    <mapping package="hibernatestudentdb" resource="cfg/student.hbm.xml"/>
  </session-factory>

</hibernate-configuration>
```

1. Konfiguracja Hibernate definiowana jest w elemencie głównym `<hibernate-configuration>`.
2. Parametry konfiguracyjne dla konkretnej bazy danych zdefiniowane są w `<session-factory>`.
3. Element `<session-factory>` zawiera zestaw właściwości `<property>`. W dalszej części wykładu omówiono najważniejsze z nich.

Plik konfiguracyjny Hibernate

Właściwość „hibernate.dialect”

```
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
```

Parametr ten określa dialekt dla bazy, z którą ma nastąpić połączenie. Zdefiniowanie odpowiedniej wartości dla tego parametru umożliwia Hibernate generowanie kodu SQL zoptymalizowanego dla konkretnej relacyjnej bazy danych. W tym przypadku dla omawianego kodu dialekt ustawiono dla serwera MySql.

W większości przypadków Hibernate będzie w stanie wybrać poprawną implementację **org.hibernate.dialect.Dialect** na podstawie metadanych JDBC zwróconych przez sterownik JDBC.

Dialekty dla popularnych systemów bazodanowych pokazano w tabeli na następnym slajdzie.

Plik konfiguracyjny Hibernate

Dialekty Hibernate dla różnych systemów zarządzania bazami danych:

RDBMS	Dialect
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL5	org.hibernate.dialect.MySQL5Dialect
MySQL5 with InnoDB	org.hibernate.dialect.MySQL5InnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
Sybase	org.hibernate.dialect.SybaseASE15Dialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server 2000	org.hibernate.dialect.SQLServerDialect
Microsoft SQL Server 2005	org.hibernate.dialect.SQLServer2005Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
H2 Database	org.hibernate.dialect.H2Dialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

Plik konfiguracyjny Hibernate

Właściwości „connection.driver_class” i "connection.url"

```
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost/BazaStudentow</property>
```

Właściwości te pozwalają ustalić sterownik JDBC oraz łańcuch połączeniowy wskazujący adres serwera bazodanowego oraz nazwą bazy danych. Hibernate używa ich do łączenia się z konkretną bazą danych. W omawianym przypadku jest to sterownik oraz łańcuch połączeniowy właściwy dla serwera MySQL.

Właściwości „connection.username” i "connection.password"

```
<property name="connection.username">nazwa-użytkownika</property>
<property name="connection.password">hasło</property>
```

Podczas łączenia się z bazą danych najczęściej wymagane jest podanie nazwy użytkownika przez, którego następuje połączenie z bazą danych oraz hasła.

Plik konfiguracyjny Hibernate

Właściwość „hbm2ddl.auto”

```
<property name="hbm2ddl.auto">update</property>
```

Parametr ten umożliwia testowanie i tworzenie struktury bazy danych podczas uruchamiania aplikacji. Gdy aplikacja jest już przetestowana zalecane jest ustawienie go na wartość **validate** lub usunięcie linii definiującej tę właściwość, przez co wartość domyślnie zostanie ustawiona na **validate**.

Wartości, które można ustawić dla właściwości hbm2ddl.auto:

create - Hibernate usuwa tabelę z bazy danych, a następnie tworzy nową i wykonuje operacje na nowo utworzonej tabeli.

validate - jeśli ta opcja jest ustawiona Hibernate sprawdza czy tabela i kolumny istnieją w bazie danych. Jeśli nie to wyrzuca wyjątek. Taka wartość jest domyślna dla właściwości `hbm2ddl.auto`.

update - jeśli ta opcja jest ustawiona Hibernate sprawdza tabelę i jej kolumny. Jeśli tabela nie istnieje tworzy ją, jeśli kolumny nie istnieją również je tworzy. Wartość `update` nie usuwa żadnych istniejących tabel, zatem dane nie są w tym przypadku tracone.

create-drop – Hibernate usuwa tabelę z bazy danych, a następnie tworzy nową i wykonuje operacje na nowo utworzonej tabel, a na końcu przy zakończeniu sesji usuwa tabelę. Wartość `create-drop` jest wykorzystywana przy testach kodu Hibernate np. testach jednostkowych.

Plik mapowania relacyjno-obiektowego

Plik `student.hbm.xml` określający mapowanie obiektowo-relacyjne

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="hibernatestudentdb.Student" table="student">
        <id name="studentId" column="idStudent">
            <generator class="increment"/>
        </id>
        <property name="imie" column="fistName" />
        <property name="nazwisko" column="secondName" />
        <property name="nrIndeksu" column="indexNumber" />
   </class>
</hibernate-mapping>
```

Powyższy kod mapuje obiekty klasy student na tabelę student w bazie danych, co zdefiniowane jest w elemencie `<class>`. W elemencie `<id>` zdefiniowano klucz główny tabeli dla bazy danych, którego nazwę ustawiono na `id_student`. Klucz ten będzie odwzorowany polem `studentId` klasy `Student` według parametru `name`. W elemencie tym zagnieżdżono element `<generator>` określający autoinkrementację klucza. Pozostałe kolumny tabeli definiowane są elementami `<property>` i są odwzorowywane podobnie jak klucz główny tj. pole `imie` klasy Student zapisane będzie w kolumnie `firstname` w tabeli `student` itd.

Klasa wykonawcza

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateStudentDB {

  public static void main(String[] args) {

   Student student1 = new Student("Adrian", "Nowak", 8934890);
   Student student2 = new Student("Jan", "Morawski", 8824391);

   Configuration configuration =
     new Configuration().configure("cfg\\hibernateMySQL.cfg.xml");

   ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().
     applySettings(configuration.getProperties()).build();

   SessionFactory factory = configuration.buildSessionFactory(serviceRegistry);

   Session session = factory.openSession();

   Transaction transaction = session.beginTransaction();

   session.save(student1);
   session.save(student2);

   transaction.commit();

   session.close();
   factory.close();

   StandardServiceRegistryBuilder.destroy(serviceRegistry);
  }
}
```

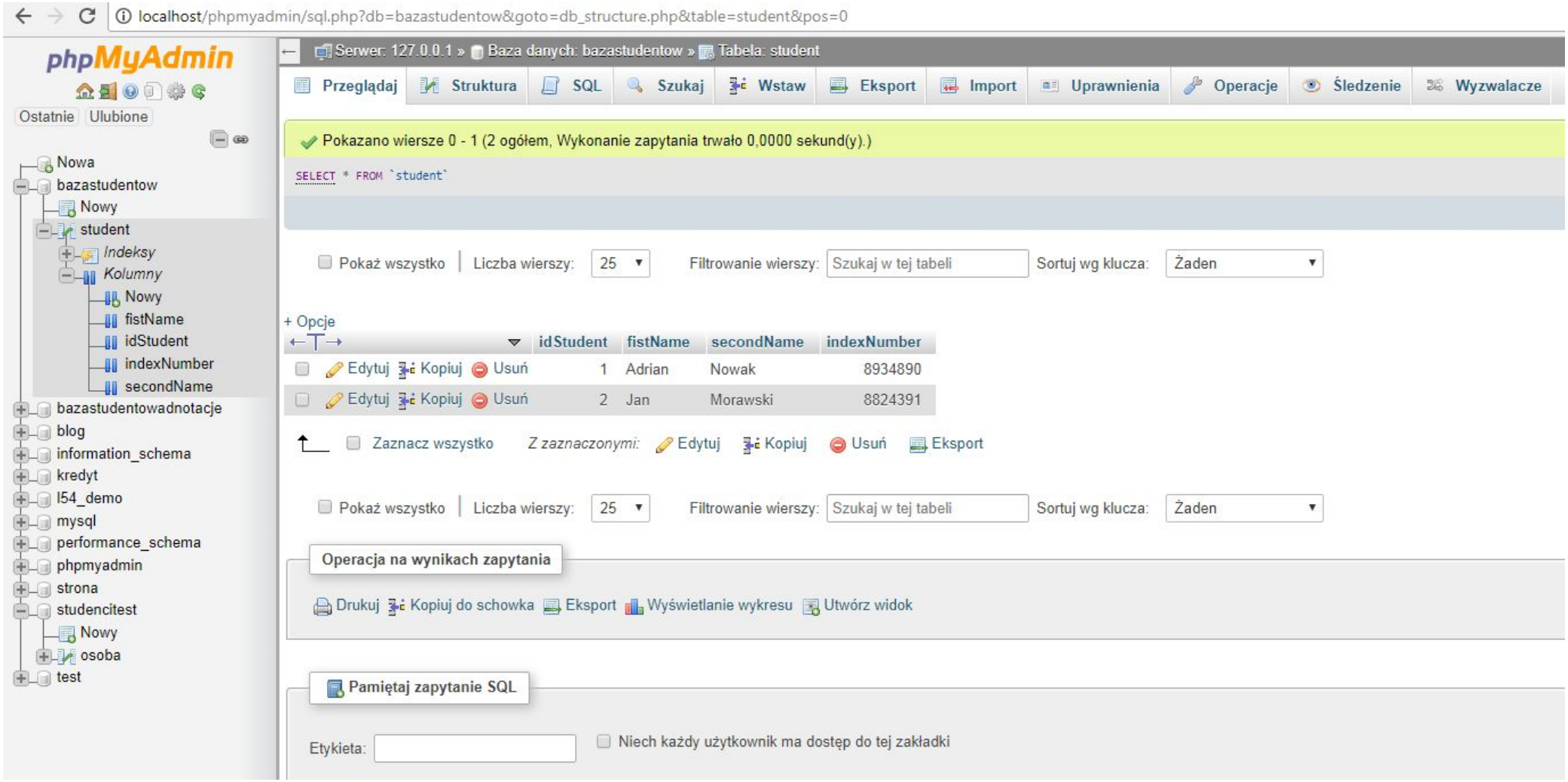
Utworzenie obiektu konfiguracji odczytanej z pliku XML, tutaj też wczytywane jest mapowanie

Rozpoczęcie transakcji

Utrwalenie obiektów klasy student w bazie danych

Zatwierdzenie transakcji (zakończenie)

Wynik działania aplikacji



Mapowanie OR z użyciem adnotacji

Zamiast definiować plik XML zawierający mapowanie obiektowo-relacyjne, można to mapowanie zdefiniować bezpośrednio w kodzie klasy Student przy użyciu adnotacji.

Należy wtedy usunąć również element <mapping> z pliku konfiguracyjnego tj. tę linię

```
<mapping package="hibernatestudentdb" resource="cfg/student.hbm.xml"/>
```

Mapowanie OR z użyciem adnotacji

```
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="student")
public class Student implements Serializable{
    private static final long serialVersionUID = -300025L;

    @Column(name="id_studenta", unique=true)
    @Id
    @GeneratedValue
    private int studentId;

    @Column(name="imie_studenta")
    private String imie;

    @Column(name="nazwisko_studenta")
    private String nazwisko;

    @Column(name="nr_indeksu")
    private long nrIndeksu;

    public Student(String imie, String nazwisko, long nrIndeksu) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
    }

    public Student() {}
       // dodaj getery i setery oraz przesłoń metodę toString() tak aby wyświetlała dane obiektu
}
```

Kod wykonawczy testujący (adnotacje)

```
public static void main(String[] args) {

  Student student = new Student("Adrian", "Nowak", 8934890);

  Configuration configuration =
    new Configuration().configure("cfg\\hibernateMySQL.cfg.xml");

  configuration.addAnnotatedClass(Student.class);


  ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
    .applySettings(configuration.getProperties()).build();

  SessionFactory factory = configuration.buildSessionFactory(serviceRegistry);

  Session session = factory.openSession();

  Transaction transaction = session.beginTransaction();
    session.save(student);
  transaction.commit();

  session.close();
  factory.close();
}
```

Należy dodać do obiektu konfiguracji adnotowaną klasę

Jednokierunkowa relacja jeden do wiele

Jednokierunkowa relacja jeden do wiele

```
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="student")
public class Student implements Serializable{

    private static final long serialVersionUID = -300025L;

    @Column(name="id_studenta", unique=true)
    @Id
    @GeneratedValue
    private int studentId;

    @Column(name="imie_studenta")
    private String imie;

    @Column(name="nazwisko_studenta")
    private String nazwisko;

    @Column(name="nr_indeksu")
    private long nrIndeksu;

    @OneToMany
    @JoinColumn(name="id_studenta")
    private List<Telefon> telefony;

    public Student(String imie, String nazwisko, long nrIndeksu) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
    }

     public Student(){
     ...
     // dodaj getery i setery oraz przesłoń metodę toString() tak aby wyświetlała dane obiektu
}
```

Jednokierunkowa relacja jeden do wiele

```java
package hihbernatestudentdbadnotationonetomany;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Telefon implements Serializable {
    private static final long serialVersionUID = -300030L;

    @Id
    @GeneratedValue
    private int id_telefonu;
    private String typ;
    private long number;

    public int getId_telefonu() {
        return id_telefonu;
    }

    public void setId_telefonu(int id_telefonu) {
        this.id_telefonu = id_telefonu;
    }

    public String getTyp() {
        return typ;
    }

    public void setTyp(String typ) {
        this.typ = typ;
    }

    public long getNumber() {
        return number;
    }

    public void setNumber(long number) {
        this.number = number;
    }
}
```

Jednokierunkowa relacja jeden do wiele

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
    <property
name="connection.url">jdbc:mysql://localhost/BazaStudentowAdnotacje</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.username">root</property>
    <property name="connection.password"/>
    <property name="hbm2ddl.auto">update</property>

  </session-factory>
</hibernate-configuration>
```

Jednokierunkowa relacja jeden do wiele

```
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HihbernateStudentDBAdnotationOneToMany {

    public static void main(String[] args) {
        Student student1 = new Student("Michał", "Adamiak", 8934890);
        Student student2 = new Student("Stanis³aw", "Bia³y", 8824391);

        Telefon telefon1 = new Telefon();
        telefon1.setTyp("mobilny");
        telefon1.setNumber(12121212);

        Telefon telefon2 = new Telefon();
        telefon2.setTyp("stacjonarny");
        telefon2.setNumber(13131313);

        List<Telefon> listaTelefonow1 = new ArrayList<>();
        listaTelefonow1.add(telefon1);
        listaTelefonow1.add(telefon2);

        student1.setTelefony(listaTelefonow1);

        Configuration configuration = new Configuration()
                .configure("cfg\\hibernateMySQL.cfg.xml");
        configuration.addAnnotatedClass(Student.class);
        configuration.addAnnotatedClass(Telefon.class);

        ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
      .applySettings(configuration.getProperties()).build();

        SessionFactory factory = configuration.buildSessionFactory(serviceRegistry);

        Session session = factory.openSession();

        Transaction transaction = session.beginTransaction();
         session.save(student1);
         session.save(student2);
         session.save(telefon1);
         session.save(telefon2);
        transaction.commit();

        System.out.println("Transaction Completed !");
```

Dwukierunkowa relacja jeden do wiele

Dwukierunkowa relacja jeden do wiele

```
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="student")
public class Student implements Serializable{
    private static final long serialVersionUID = -300025L;

    @Column(name="id_studenta", unique=true)
    @Id
    @GeneratedValue
    private int studentId;

    @Column(name="imie_studenta")
    private String imie;

    @Column(name="nazwisko_studenta")
    private String nazwisko;

    @Column(name="nr_indeksu")
    private long nrIndeksu;

    @OneToMany(mappedBy="student")
    private List<Telefon> telefony;

    public Student(String imie, String nazwisko, long nrIndeksu) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
    }

    public Student(){}
        ...
        // dodaj getery i setery oraz przesłoń metodę toString() tak aby wyświetlała dane obiektu
}
```

Dwukierunkowa relacja jeden do wiele

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class Telefon {
    private static final long serialVersionUID = -300030L;

    @Id
    @GeneratedValue
    private int id_telefonu;
    private String typ;
    private long number;

    @ManyToOne
    @JoinColumn(name="id_studenta")
    private Student student;

    public int getId_telefonu() { return id_telefonu; }
    public void setId_telefonu(int id_telefonu) { this.id_telefonu = id_telefonu; }
    public String getTyp() { return typ; }
    public void setTyp(String typ) { this.typ = typ; }
    public long getNumber() { return number; }
    public void setNumber(long number) { this.number = number; }
    public Student getStudent() { return student; }
    public void setStudent(Student student) { this.student = student; }

    // dodaj metodę toString()
}
```

Dwukierunkowa relacja jeden do wiele

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HihbernateStudentDBAdnotationOneToMany {
    public static void main(String[] args) {
        Student student1 = new Student("Adrian", "Nowak", 8934890);
        Student student2 = new Student("Jan", "Morawski", 8824391);

        Telefon telefon1 = new Telefon();
        telefon1.setTyp("mobilny");
        telefon1.setNumber(808201021);
        telefon1.setStudent(student1);

        Telefon telefon2 = new Telefon();
        telefon2.setTyp("stacjonarny");
        telefon2.setNumber(178312812);
        telefon2.setStudent(student1);

        Configuration configuration = new Configuration().configure("cfg\\hibernateMySQL.cfg.xml");
        configuration.addAnnotatedClass(Student.class);
        configuration.addAnnotatedClass(Telefon.class);

        ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(
            configuration.getProperties()).build();

        SessionFactory factory = configuration.buildSessionFactory(serviceRegistry);

        Session session = factory.openSession();

        Transaction transaction = session.beginTransaction();
         session.save(student1);
         session.save(student2);
         session.save(telefon1);
         session.save(telefon2);
        transaction.commit();
        System.out.println("Transaction Completed !");

        session.close(); factory.close();
    }
}
```

Relacja wiele do wiele

Relacja n:m – kod dla klasy Student

```
@Entity
@Table(name="student")
public class Student implements Serializable{

    private static final long serialVersionUID = -300025L;

    @Column(name="id_studenta", unique=true)
    @Id
    @GeneratedValue
    private int studentId;

    @Column(name="imie_studenta")
    private String imie;

    @Column(name="nazwisko_studenta")
    private String nazwisko;

    @Column(name="nr_indeksu")
    private long nrIndeksu;

    @ManyToMany
    private List<Przedmiot> przedmioty;

    public Student(String imie, String nazwisko, long nrIndeksu) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
    }

    public Student(){}

    // dodaj getery i setery oraz metodę toString()
}
```

Relacja n:m – kod dla klasy Student

```
@Entity
@Table(name="przedmiot")
public class Przedmiot implements Serializable{
    private static final long serialVersionUID = -300040L;

    @Id
    @GeneratedValue
    private int przedmiotId;
    private String nazwaPrzedmiotu;

    @ManyToMany(mappedBy="przedmioty")
    private List<Student> studenci;

    public Przedmiot(String nazwaPrzedmiotu) {
        this.nazwaPrzedmiotu = nazwaPrzedmiotu;
    }

    public Przedmiot() {}

    // dodaj getery i setery oraz metodę toString()
}
```

```
private List<Przedmiot> przedmioty;
```

Zapytania SQL

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

  //Zapytanie pobiera wszystkie encje klasy Student z bazy danych
  ArrayList<Student> lista = (ArrayList<Student>)
    session.createSQLQuery("select * from student").addEntity(Student.class).list();

  for (Student student : lista)
    System.out.println(student.toString());

session.close();
```

student – nazwa tabeli w bazie danych
id_sudenta – nazwa kolumny w bazie danych - klucz główny w tabeli student

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

  //Zapytanie pobiera z bazy danych encję klasy Student o identyfikatorze 1
  Student student = (Student)
  session.createSQLQuery("select * from student where id_studenta=1").
          addEntity(Student.class).uniqueResult();

  System.out.println(student.toString());

session.close();
```

Zapytania HQL

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

  //Zapytanie pobiera wszystkie encje klasy Student z bazy danych
  Query zapytanie = session.createQuery("from "+Student.class.getSimpleName());
  ArrayList<Student> lista = (ArrayList<Student>) zapytanie.list();
  for (Student k : lista) System.out.println(k.toString());

session.close();
```

`Student.class.getSimpleName()` → "Student"

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

//Zapytanie pobiera z bazy danych encję klasy Student o identyfikatorze 1
Query zapytanie = session.createQuery("from Student where studentId=1");
Student student = (Student) zapytanie.uniqueResult();
System.out.println(student.toString());

session.close();
```

Student – nazwa encji (klasy) Student
sudentId – nazwa pola klasy Student przechowującego identyfikator zmapowany z kluczem główny w tabeli student w bazie danych.

Criteria

```
Session session = factory.openSession();
        //Zapytanie pobiera wszystkie encje klasy Student z bazy danych
        //Brak ustawionych ograniczeń
     Criteria c = session.createCriteria(Student.class);

     List<Student> students = c.list();
     for (Student k : students) System.out.println(k.toString());
session.close();
```

```
Session session = factory.openSession();
     //Zapytanie pobiera z bazy danych encję klasy Student o identyfikatorze 1
     Criteria criteria = session.createCriteria(Student.class);
     criteria.add(Restrictions.eq("studentId", 1));
     Student student = (Student) criteria.uniqueResult();

     System.out.println(student.toString());
session.close();
```

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

        Criteria criteria = session.createCriteria(Student.class);
        criteria.add(Restrictions.or(
              Restrictions.like("imie", "A%"),
              Restrictions.like("nazwisko", "M%")
             )).addOrder(Order.asc("nazwisko"));

        List<Student> students = criteria.list();
        for (Student k : students) System.out.println(k.toString());
session.close();
```

Criteria w wersji Hibernate 5.2

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

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Student> criteria =
builder.createQuery(Student.class);
Root<Student> root = criteria.from(Student.class);
criteria.select(root);
criteria.where(builder.or(
builder.equal(root.get("imie"),"Jan"),
    builder.equal(root.get("imie"),"Adrian")
    ));

List<Student> students =
session.createQuery(criteria).getResultList();
for (Student k : students) System.out.println(k.toString());

session.close();
```

Zapytania nazwane

```
import java.io.Serializable;
import java.util.List;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;

@NamedQueries({
    @NamedQuery(
    name = "findByName",
    query = "from Przedmiot s where s.nazwaPrzedmiotu = :name"
    )
})

@Entity
public class Przedmiot implements Serializable{
    private static final long serialVersionUID = -300040L;

    @Id
    @GeneratedValue
    private int przedmiotId;
    private String nazwaPrzedmiotu;

    @ManyToMany(mappedBy="przedmioty")
    private List<Student> studenci;

    public Przedmiot(String nazwaPrzedmiotu) {
        this.nazwaPrzedmiotu = nazwaPrzedmiotu;
    }
    public Przedmiot() {}

// dodaj getery i setery oraz toString()
} // end class
```

Zapytania nazwane

```
Query query = session.getNamedQuery("findByName")
        .setString("name", "Programowanie obiektowe");
```