

Programowanie obiektowe

Wykład 13

Marcin Młotkowski

2 czerwca 2022

Plan wykładu

1 Trwałość obiektów

- Połączenie z relacyjnymi bazami danych
- Realizacja trwałości w Javie
- Realizacja trwałości w C#
- Obiektowe bazy danych

2 Obiekty rozproszone

Plan wykładu

1 Trwałość obiektów

- Połączenie z relacyjnymi bazami danych
- Realizacja trwałości w Javie
- Realizacja trwałości w C#
- Obiektowe bazy danych

2 Obiekty rozproszone

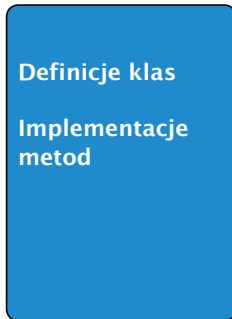
Trwałość (persistence)

Definicja

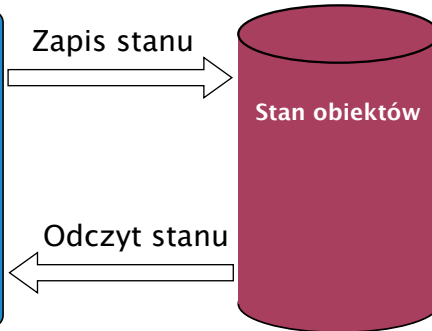
Cecha danej, zmiennej lub obiektu oznaczająca zachowanie jej wartości dłużej niż czas pojedynczego uruchomienia programu.

Przechowywanie stanu w BD

Aplikacja



Baza danych



Scenariusz użycia trwałego obiektu

- Odczyt obiektu (stanu obiektu z bd)
- Modyfikacja stanu
- Wywołanie metod obiektu
- Zapis

Zagadnienia

- Zwykle przechowywany jest stan obiektu a nie obiekt (tj. klasa, nadklasa, metody itp);
- gorliwe czy leniwe aktualizacje (odczyt/zapis) stanu;
- mechanizm aktualizacji zmian (dodawanie nowych pól, zmiana typów pól w nadklasie etc);
- ustalenie, jaka część stanu powinna być przechowywana.

Naturalne podejście

- połączenie aplikacji obiektowej z istniejącym systemem baz danych;
- rozszerzenie środowiska obiektowego o możliwości przetwarzania dużych danych;
- zbudowanie całego środowiska od początku;
- rozszerzenie systemu BD o właściwości obiektowe.

Łączenie środowiska obiektowego z systemem baz danych

Języki obiektowe

JAVA, C#, C++, Python,
JavaScript, Smalltalk ...

Systemy baz danych

MySQL, Oracle, PostgreSQL,
Sybase, Microsoft SQL Server
...

Składowe integracji języka z bazą danych

- Odwzorowanie cech obiektowych w relacyjnych bazach danych (object-relational mapping, ORM)
- Implementacja warstwy pośredniej (zapytania)

Porównanie modelu relacyjnego z obiekowym

Model obiektowy

- Ukrywanie danych i dostęp tylko przez wskazany interfejs
- Relacje między obiektami: asocjacje, dziedziczenie, kompozycja
- wywołania metod
- Tożsamość obiektów

Model relacyjny

- Rekord
- Tabela: kolekcja rekordów tego samego typu
- Związki między tabelami
- Język zapytań

Java Data Object

- specyfikacja;
- obiekty mogą być pamiętane w plikach, relacyjnych i obiektowych bazach danych;
- szczegóły (gorliwość, leniwość, etc) są definiowane w zewnętrznych plikach xml;
- przykładowa realizacja: Apache JDO, DataNucleus

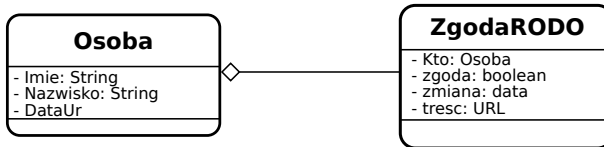
Java Persistence API (JPA)

- specyfikacja;
- głównie do przechowywania stanu obiektów w relacyjnych BD;
- szczegóły są definiowane za pomocą anotacji lub w zewnętrznych plikach;
- JPA Query Language: język zapytań przypominający SQL;
- implementacje: Hibernate, TopLink.

Przykład

Rejestr wyrażonych zgód w ramach dyrektywy RODO. Rejestr dla każdej osoby przechowuje rejestr udzielonych bądź odwołanych zgód.

Model danych w UML



Implementacja klasy Osoba

```
import javax.persistence.*;

public class Osoba
{
    private Integer id;
    String Imie;
    String Nazwisko;
    GregorianCalendar DataUr;
}
```


Implementacja klasy Osoba

```
import javax.persistence.*;

@Entity
@Table
public class Osoba implements Serializable
{
    @id
    @GeneratedValue
    private Integer id;
    String Imie;
    String Nazwisko;
    GregorianCalendar DataUr;
}
```

Implementacja klasy ZgodaRODO

```
@Entity
@Table
public class ZgodaRODO implements Serializable {
    @ManyToOne
    Osoba kto;
    boolean Zgoda = false;
    GregorianCalendar zmiana;
    URL Tresc;
}
```

Plik persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence
  version="2.0">

  <persistence-unit name="przyklad" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <class>Osoba</class>
    <class>ZgodaRODO</class>

    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:magazyn" />
      <property name="javax.persistence.jdbc.user" value="ja" />
      <property name="javax.persistence.jdbc.password" value="" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="show_sql" value="true"/>
      <property name="hibernate.temp.use_jdbc_metadata_defaults" value="false"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.use_sql_comments" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

Podstawowe operacje

Create

Read

Update

Delete

Szkielet programu

```
EntityManagerFactory sessionFactory =  
    Persistence.createEntityManagerFactory("ii.progobiekt.j  
EntityManager entityManager =  
    sessionFactory.createEntityManager();  
entityManager.getTransaction().begin();
```

```
entityManager.getTransaction().commit();  
entityManager.close();
```

Create

```
entityManager.persist( new Osoba() );
```

Create

```
entityManager.persist( new Osoba() );
```

```
INSERT INTO OSOBA (Nazwisko, Imie, DataUr)  
VALUES (" ", " ", NULL)
```

Read

```
List<Osoba> result =  
    entityManager.  
        createQuery("from Osoba", Osoba.class ).  
            getResultList();  
for (Osoba osoba : result )  
{  
    System.out.println(osoba.ToString());  
}
```


Read

```
List<Osoba> result =  
    entityManager.  
        createQuery("from Osoba", Osoba.class ).  
            getResultList();  
for (Osoba osoba : result )  
{  
    System.out.println(osoba.ToString());  
}
```

```
SELECT * FROM OSOBA
```

Update

- Jeśli obiekt został pobrany z BD i zmodyfikowany wewnątrz transakcji, wtedy jest automatycznie zapisany po wywołaniu metody `.commit()`;
- można zaznaczyć, że obiekt jest *brudny*, gdy manager nie zauważy zmiany (np. zmiany w tablicy).

Delete

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaDelete<User> deleteUser =
    cb.createCriteriaDelete(Osoba.class);
Root<User> cUser = deleteUser.from(Osoba.class);
List<Predicate> predicates = new ArrayList<Predicate>();
predicates.add(cb.equal(cUser.get("id"), id_to_delete));
deleteUser.where(predicates.toArray(new Predicate[] ));
Query query = entityManager.createQuery(deleteUser);
query.executeUpdate();
```

Delete

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaDelete<User> deleteUser =
    cb.createCriteriaDelete(Osoba.class);
Root<User> cUser = deleteUser.from(Osoba.class);
List<Predicate> predicates = new ArrayList<Predicate>();
predicates.add(cb.equal(cUser.get("id"), id_to_delete));
deleteUser.where(predicates.toArray(new Predicate[] ));
Query query = entityManager.createQuery(deleteUser);
query.executeUpdate();
```

```
DELETE FROM OSOBA WHERE OSOBA.id=1203
```

Utworzenie/aktualizacja schematu bazy danych

Każda implementacja ma swoje narzędzia do automatyzacji.

- odpowiedni wpis w pliku `persistence.xml` nakazujący utworzenie/aktualizację BD;
- odpowiedni kod w naszym programie;
- narzędzia zewnętrzne.

C#: ADO.NET

- ADO - ActiveX Data Objects
- ADO.NET – środowisko dostępu do danych w BD

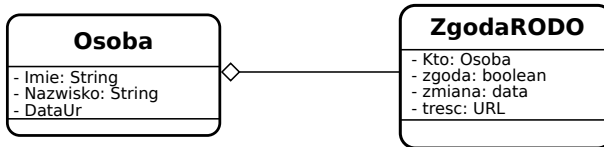
Entity Framework

- Część środowiska ADO.NET
- Powstanie: 2008 rok
- Projekt danych w postaci conceptualnego modelu danych (Entity Data Model)
- Automatyczne odwzorowanie na RBD i model obiektowy
- Zapisywanie modelu danych w pliku XML

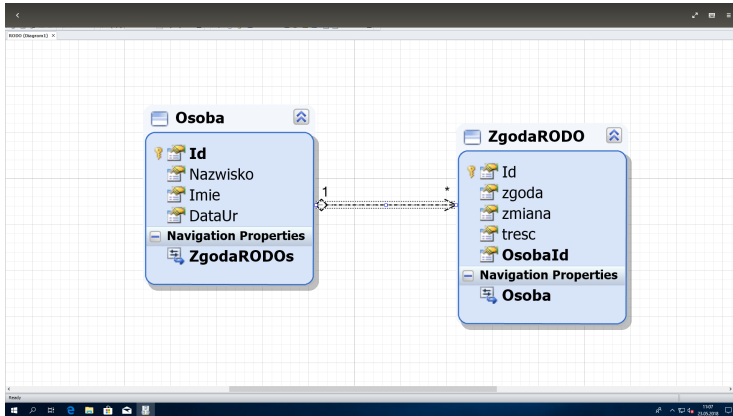
LINQ: Language Integrated Query

- Projekt Microsoftu
- Dostęp na poziomie języka programowania do danych w BD za pomocą składni SQL-podobnej

Model danych w UML (ponownie)



Entity Developer



RODO.Designer.cs

- plik wygenerowany Entity Developerem;
- 553 wiersze.

Przykład LINQ

```
Zgody db = new Zgody(connectionString);  
var q = from o in db.Osoba, z in db.ZgodaRODOs  
        where o.Id == z.OsobaId and z.zgoda and o.Nazwisko =  
        select new {z.zmiana, c.tresc};  
foreach (var t in q) { ... }
```

RUBY

ActiveRecords

- każdej klasie będącej podklasą klasy Base odpowiada tabela w bd;
- narzędzia do tzw. migracji;
- obiektowy dostęp do danych

```
Osoba.find(:all,  
           :conditions => "nazwisko = 'Młotkowski'")
```

Rozszerzenia RBD na przykładzie Oracle

- PL/SQL – rozszerzenie SQL o procedury
- Składnia zapożyczona z Ady
- Możliwość definiowania własnych programów wykonywanych na serwerze
- Możliwość deklarowania własnych typów danych (klas)
- Rozszerzenie SQL o odwołania do obiektów

Serwer STONE

- Przechowuje obiekty
- Obiekty nie są kopiowane do aplikacji klienta, tylko zostają w serwerze
- Metody są wykonywane na serwerze

Rozwiązanie w Smalltalku

Przechowywanie stanu aplikacji w jednym pliku (obrazie).

Wady rozwiązania

- Kłopoty z przetwarzaniem dużych porcji danych
- Kłopoty z szybkim wyszukiwaniem danych
- Problem wielodostępu do danych

Plan wykładu

1 Trwałość obiektów

- Połączenie z relacyjnymi bazami danych
- Realizacja trwałości w Javie
- Realizacja trwałości w C#
- Obiektowe bazy danych

2 Obiekty rozproszone

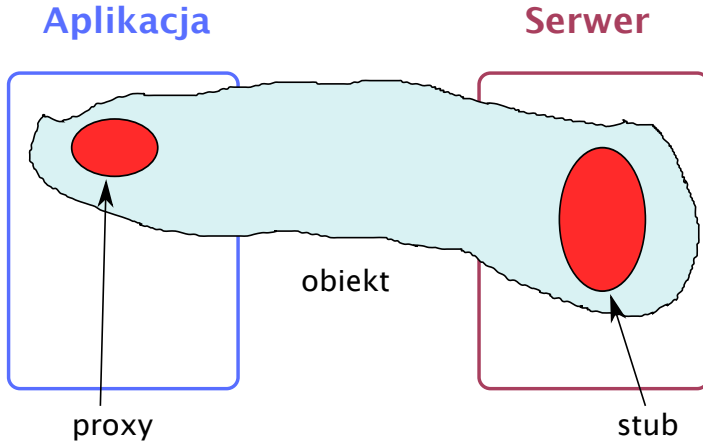
Obiekty trwałe: inna koncepcja

- Obiekt (jego stan) oraz implementacja metod jest pamiętana na serwerze
- Wykonanie metody powoduje odwołanie się do zdalnego obiektu, wykonanie metody na serwerze i zwrócenie klientowi wyniku
- Serwer dba o spójność danych, wielodostęp, transakcyjność etc

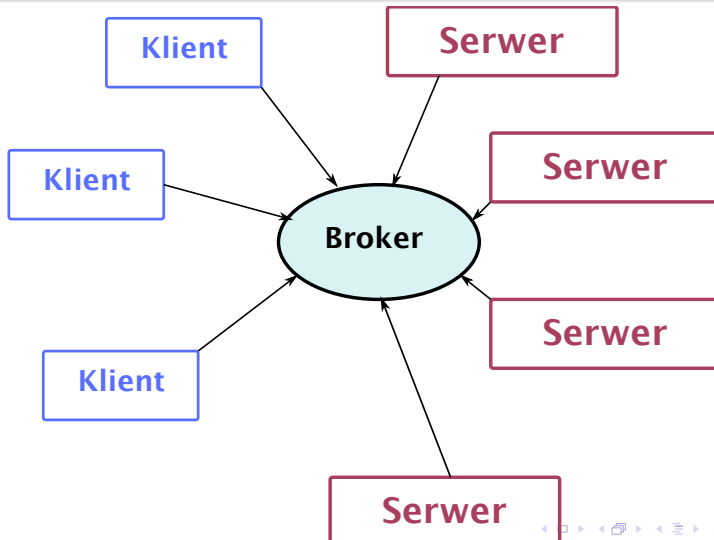
Postulaty dotyczące takiego środowiska

- Jednolity dostęp do obiektów lokalnych i odległych
- Łatwość tworzenia odwołań do odległych obiektów
- Niezależność standardów od platformy

Architektura



Broker



Środowiska obiektów rozproszonych

- Java RMI
- CORBA
- SOAP
- DCOM

Java RMI

- RMI – Remote Method Invocation
- Współpraca tylko między aplikacjami napisanymi w Javie
- java.rmi.*
- Całe środowisko jest częścią SDK

CORBA

- Zbiór standardów opracowanych przez OMG (Object Management Group)
- Niezależność od platformy/sprzętu/języka
- Implementacje: VisiBroker, ORBit
- Bardzo obszerna specyfikacja

SOAP

- Simple Object Access Protocol
- Oparty na XML
- Standard W3C
- Implementacje: .NET Remoting, Apache SOAP
- .NET Remoting: również realizacja binarna

DCOM

- Distributed Component Object Model
- Implementacja tylko na systemy Microsoftu
- Uznany za przestarzały na rzecz .NET Remoting