Zasady generowania kluczy głównych
Język „Java Persistence”
Podstawowa architektura wielowarstwowych aplikacji w oparciu o wzorce oprogramowania

Autor

Zofia Kruczkiewicz

Wzorce oprogramowania 6

1. Różne mechanizmy generowania kluczy głównych w procesie mapowania modelu obiektowego na relacyjny

Zasady generowania kluczy głównych (1)

@Id oznacza adnotację dla klucza głównego.

Adnotacja **@GeneratedValue (strategy=GenerationType.<...>)** oznacza strategię tworzenia wartości kluczy głównych, zależną od systemu baz danych

(javax.persistence.GeneratedValue, javax.persistence.GenerationType).

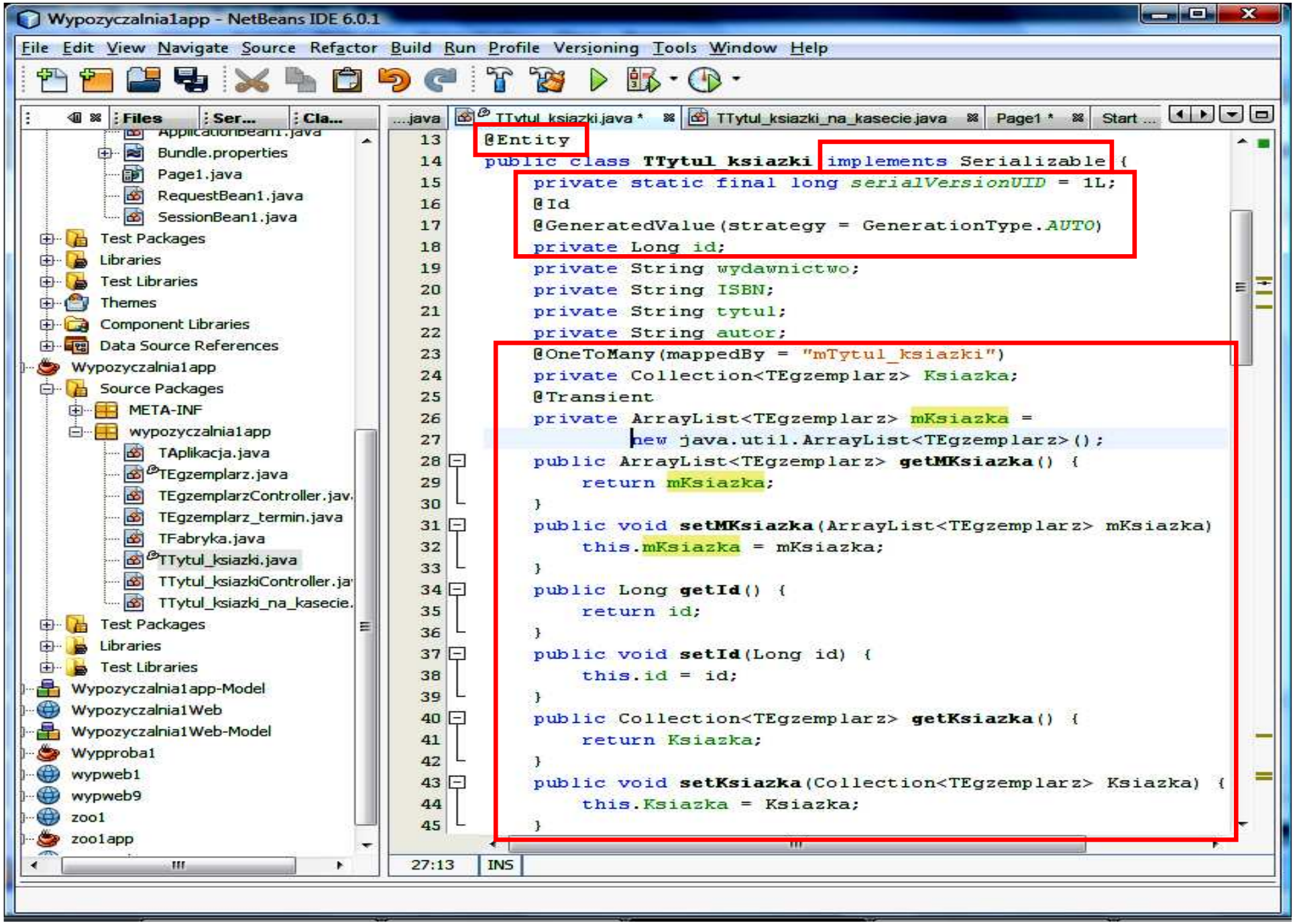
1. **Domyślna strategia** pozwala na przejęcie odpowiedzialności za generowanie kluczy głównych przez TopLink, który posługuje się pomocniczą tabelą przechowującą wartości potrzebne do generowania kluczy – **przykład 1**

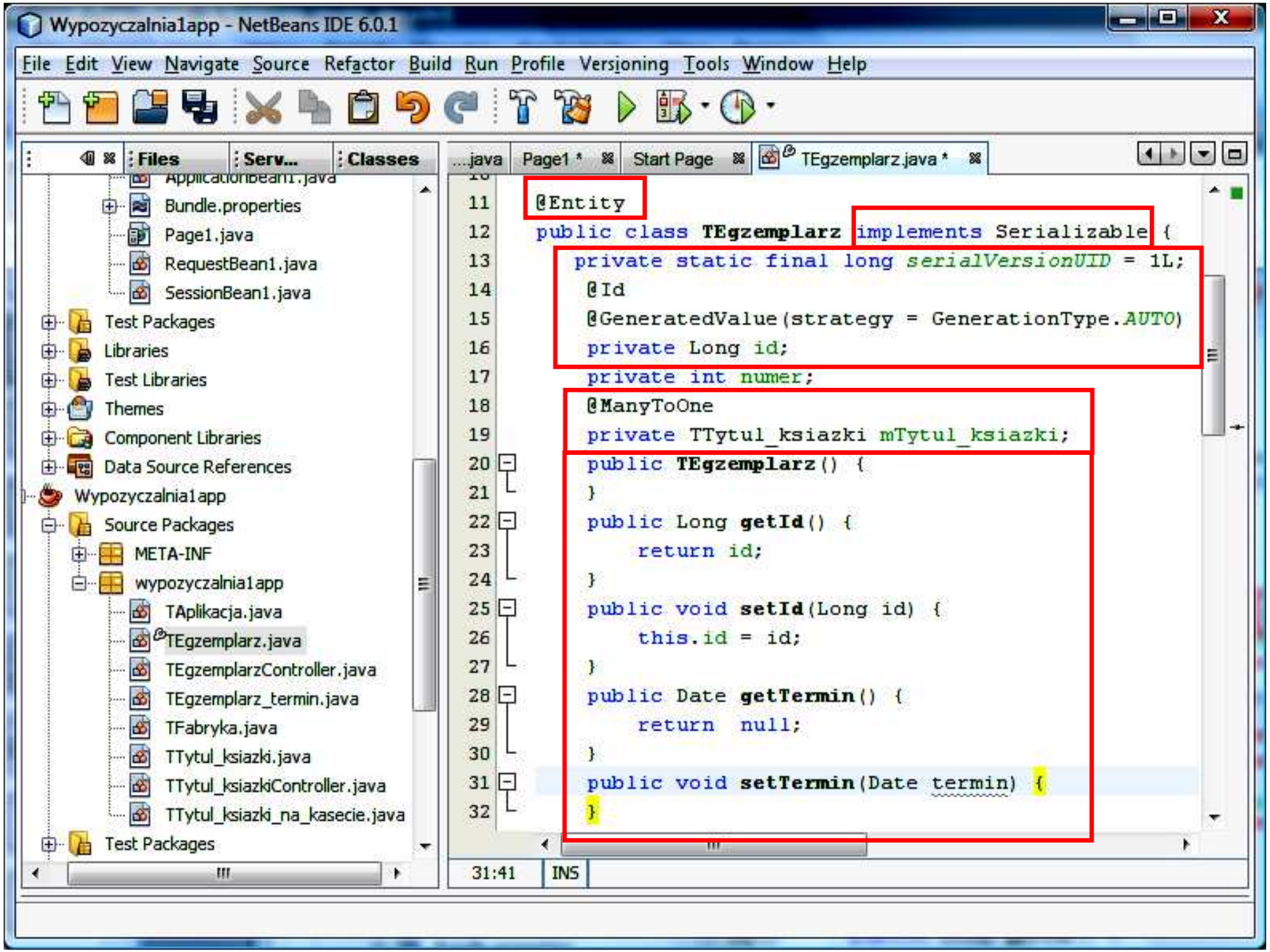
```
@Entity
    public class Inventory implements Serializable {
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        private long id;
```

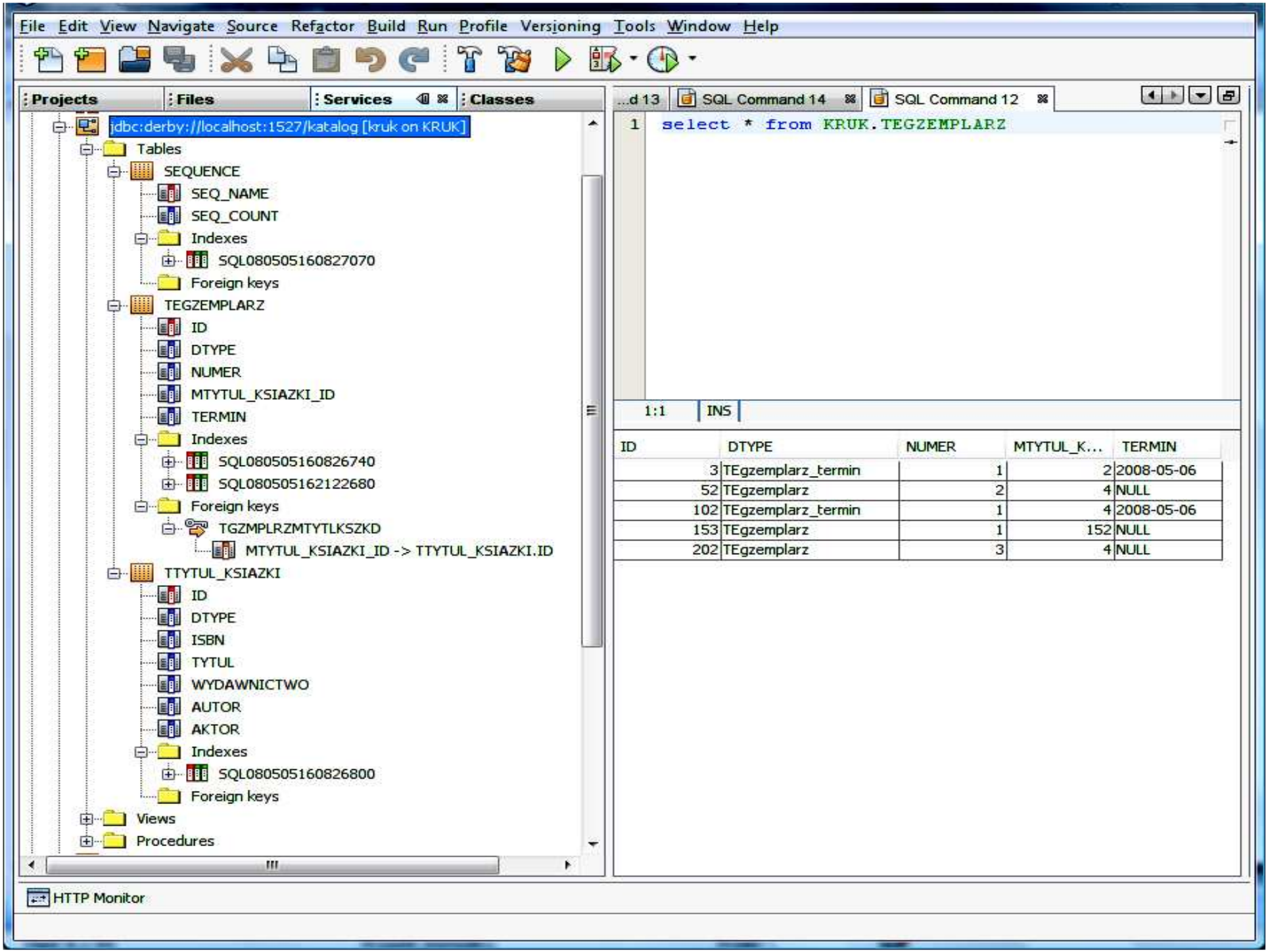
2. **Wykorzystanie mechanizmu identyczności kolumny do generowania wartości klucza głównego** oznacza odpowiedzialność za generowanie wartości kluczy głównych przez bazę danych – **przykład 2**

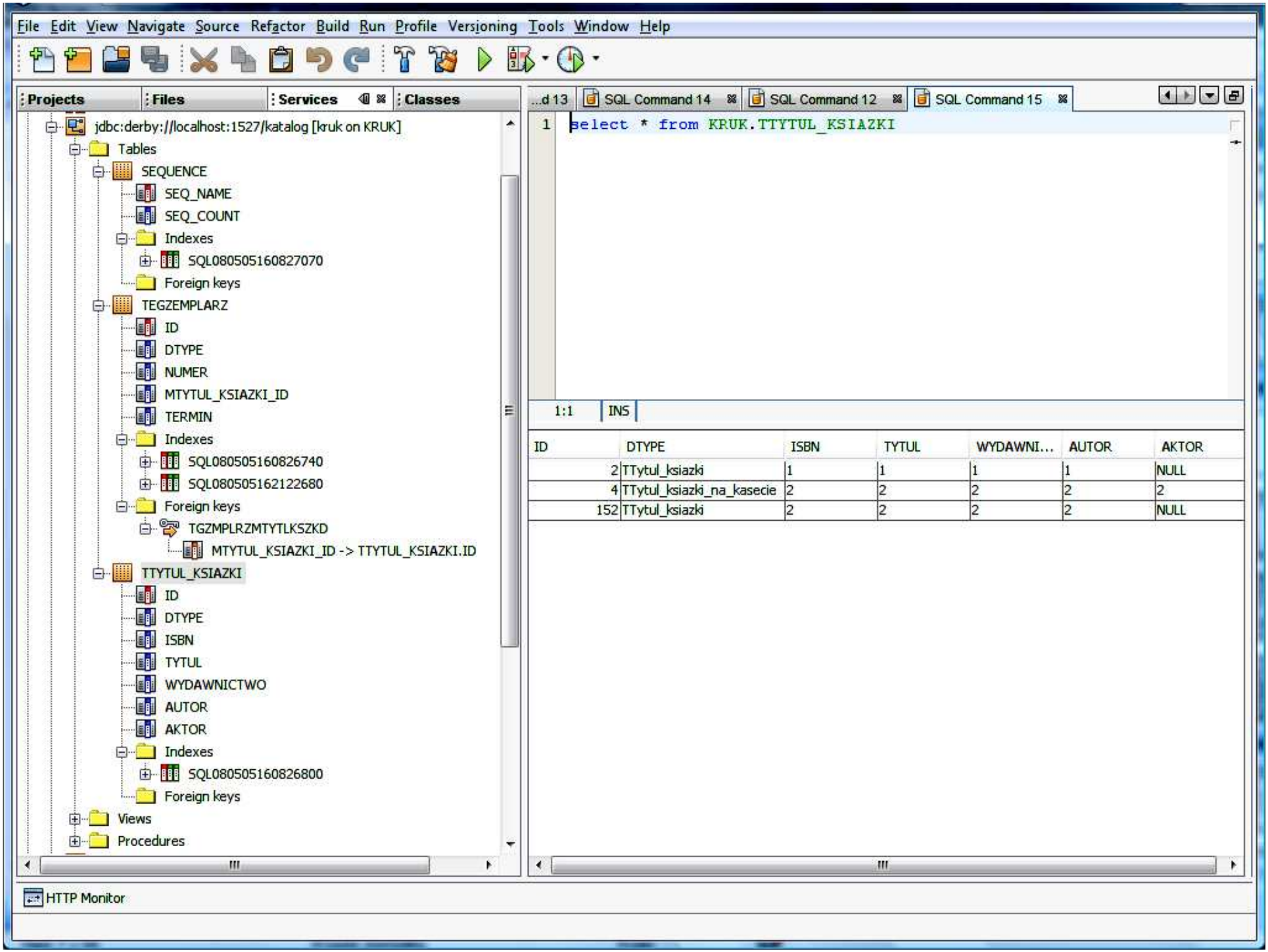
```
@Entity
    public class Inventory implements Serializable {
        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private long id;
```

Przykład 1 - @GeneratedValue(strategy=GenerationType.AUTO)

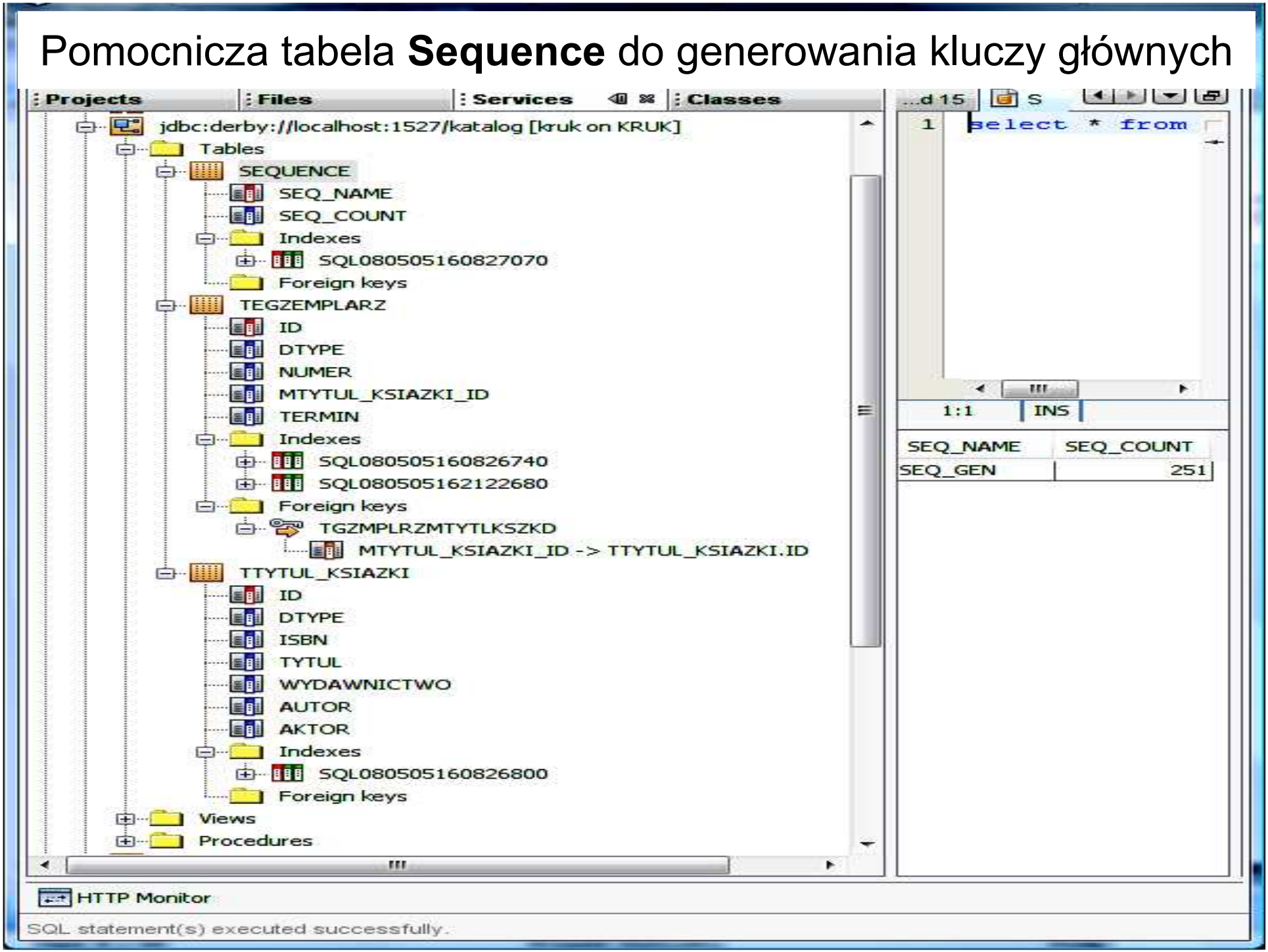



Wypozyczalnia1app - NetBeans IDE 6.0.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Files
Serv...
Classes
Bundle.properties
Page1.java
RequestBean1.java
SessionBean1.java
Test Packages
Libraries
Test Libraries
Themes
Component Libraries
Data Source References
Wypozyczalnia1app
Source Packages
META-INF
wypozyczalnia1app
TAplikacja.java
TEgzemplarz.java
TEgzemplarzController.java
TEgzemplarz_termin.java
TFabryka.java
TTytul_ksiazki.java
TTytul_ksiazkiController.java
TTytul_ksiazki_na_kasecie.java
Test Packages
Page1 *
Start Page
TEgzemplarz.java *
@Entity
public class TEgzemplarz implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
private int numer;
@ManyToOne
private TTytul_ksiazki mTytul_ksiazki;
public TEgzemplarz() {
}
public Long getId() {
return id;
}
public void setId(Long id) {
this.id = id;
}
public Date getTermin() {
return null;
}
public void setTermin(Date termin) {
}
31:41
INS

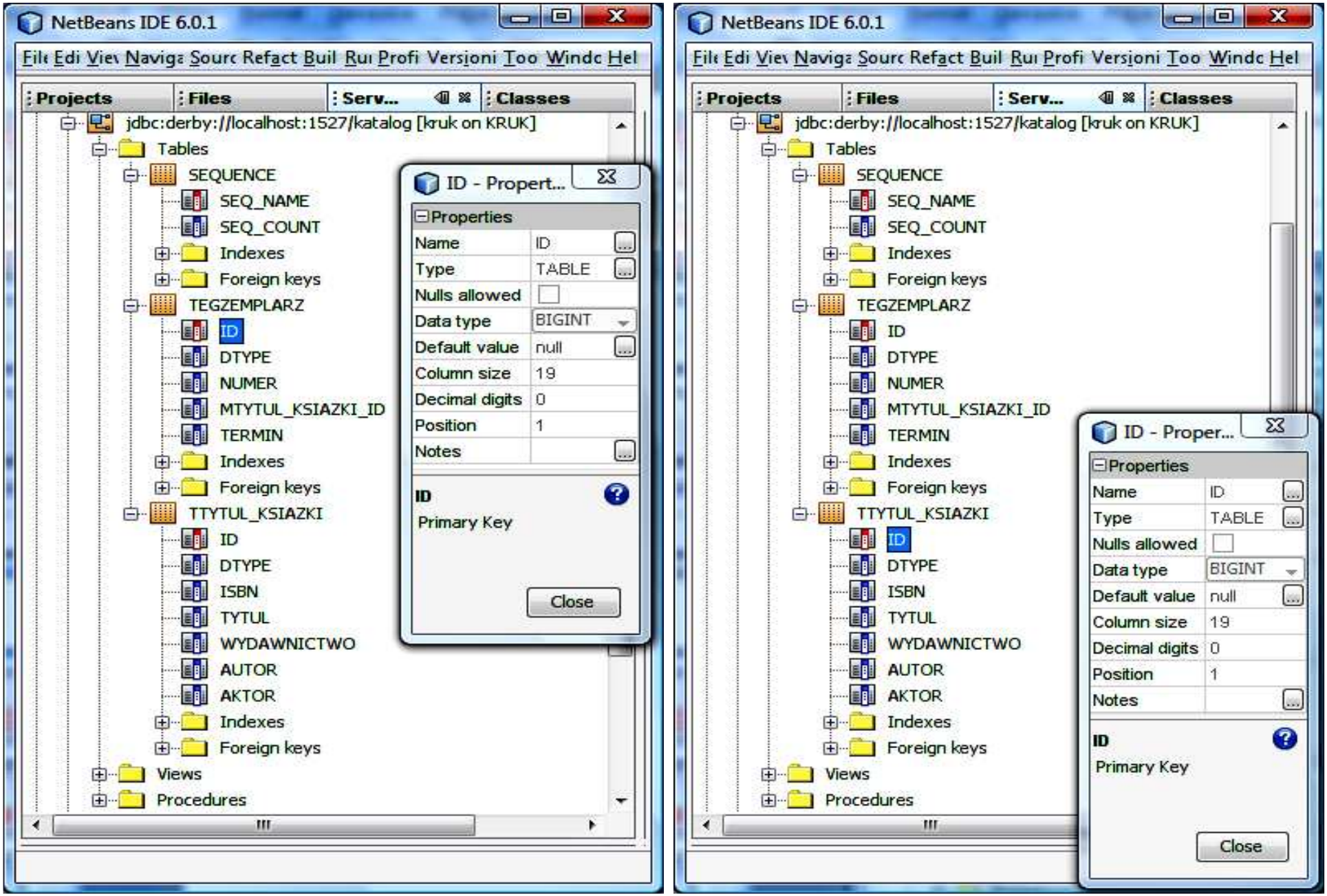

File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
Files
Services
Classes
jdbc:derby://localhost:1527/katalog [kruk on KRUK]
Tables
SEQUENCE
SEQ_NAME
SEQ_COUNT
Indexes
SQL080505160827070
Foreign keys
TEGZEMPLARZ
ID
DTYPE
NUMER
MTYTUL_KSIAZKI_ID
TERMIN
Indexes
SQL080505160826740
SQL080505162122680
Foreign keys
TGZMPLRZMTYTLKSZKD
MTYTUL_KSIAZKI_ID -> TTYTUL_KSIAZKI.ID
TTYTUL_KSIAZKI
ID
DTYPE
ISBN
TYTUL
WYDAWNICTWO
AUTOR
AKTOR
Indexes
SQL080505160826800
Foreign keys
Views
Procedures
...d 13
SQL Command 14
SQL Command 12
1 select * from KRUK.TEGZEMPLARZ
1:1
INS
ID
DTYPE
NUMER
MTYTUL_K...
TERMIN
3 TEgzemplarz_termin 1 2 2008-05-06
52 TEgzemplarz 2 4 NULL
102 TEgzemplarz_termin 1 4 2008-05-06
153 TEgzemplarz 1 152 NULL
202 TEgzemplarz 3 4 NULL
HTTP Monitor


File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
Files
Services
Classes
jdbc:derby://localhost:1527/katalog [kruk on KRUK]
Tables
SEQUENCE
SEQ_NAME
SEQ_COUNT
Indexes
SQL080505160827070
Foreign keys
TEGZEMPLARZ
ID
DTYPE
NUMER
MTYTUL_KSIAZKI_ID
TERMIN
Indexes
SQL080505160826740
SQL080505162122680
Foreign keys
TGZMPLRZMTYTLKSZKD
MTYTUL_KSIAZKI_ID -> TTYTUL_KSIAZKI.ID
TTYTUL_KSIAZKI
ID
DTYPE
ISBN
TYTUL
WYDAWNICTWO
AUTOR
AKTOR
Indexes
SQL080505160826800
Foreign keys
Views
Procedures
...d 13
SQL Command 14
SQL Command 12
SQL Command 15
1 select * from KRUK.TTYTUL_KSIAZKI
1:1
INS
ID
DTYPE
ISBN
TYTUL
WYDAWNI...
AUTOR
AKTOR
2 TTytul_ksiazki 1 1 1 1 NULL
4 TTytul_ksiazki_na_kasecie 2 2 2 2 2
152 TTytul_ksiazki 2 2 2 2 NULL
HTTP Monitor

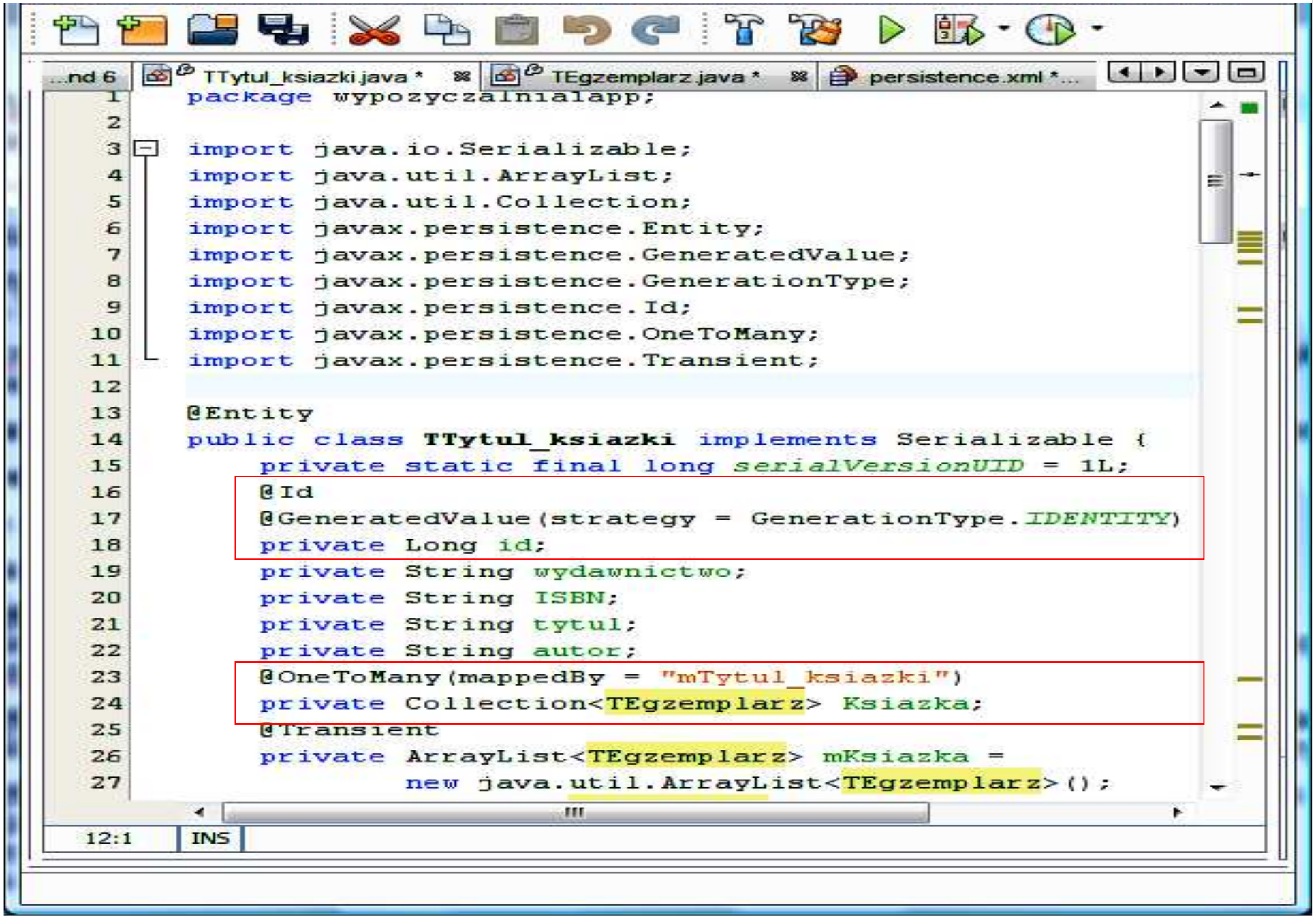
Pomocnicza tabela **Sequence** do generowania kluczy głównych

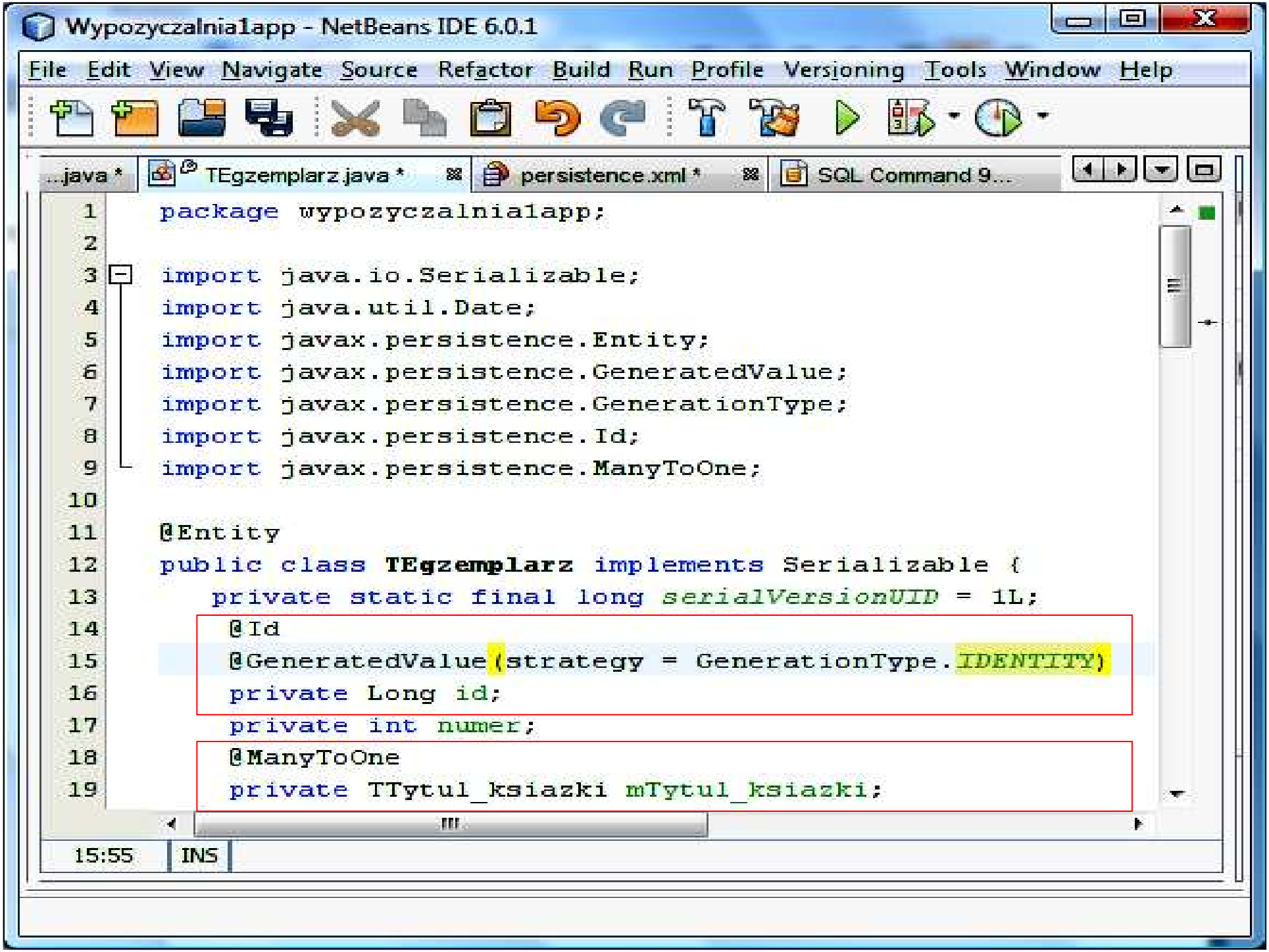


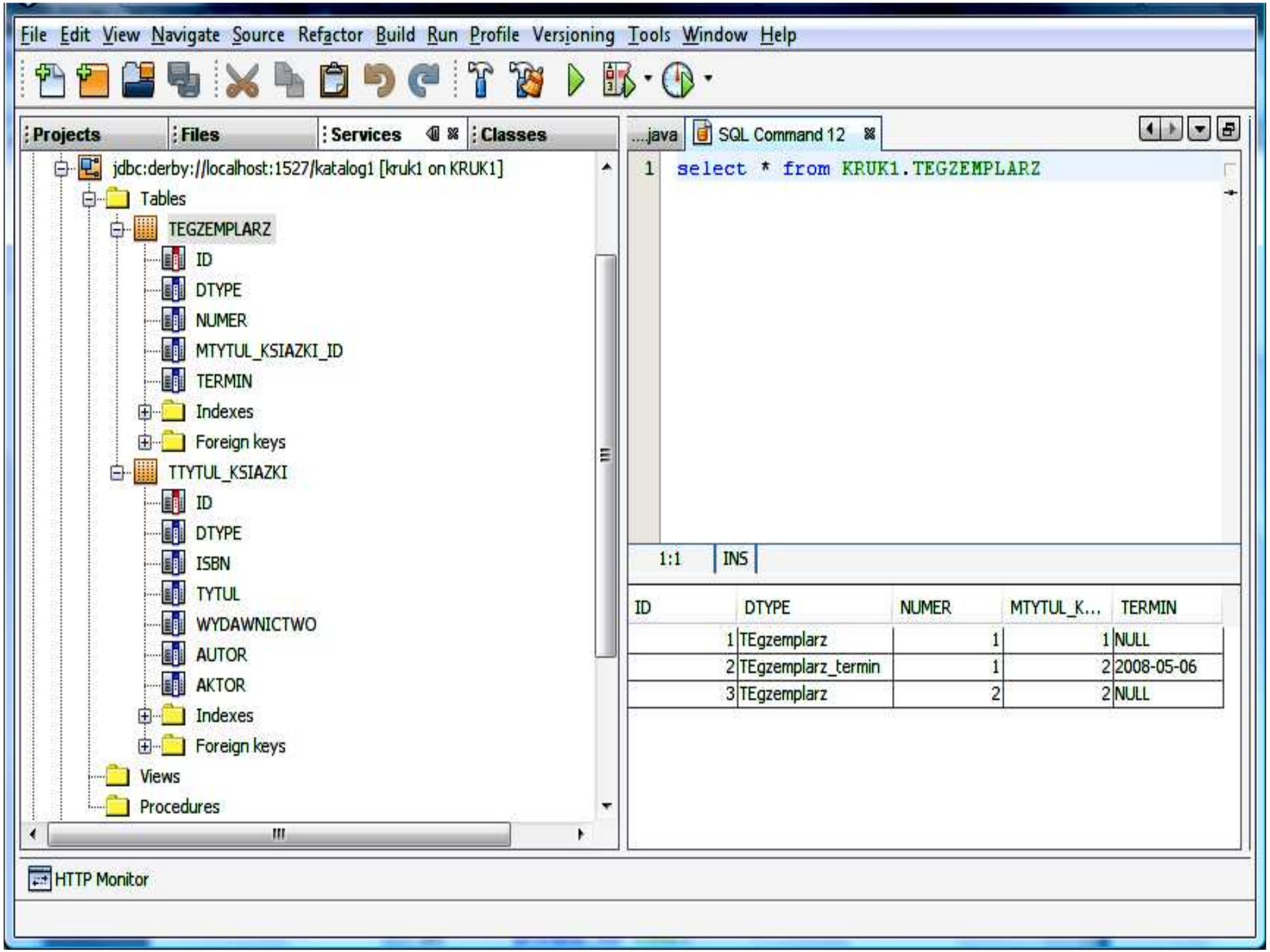
Właściwości kluczy głównych przy zastosowanej strategii AUTO

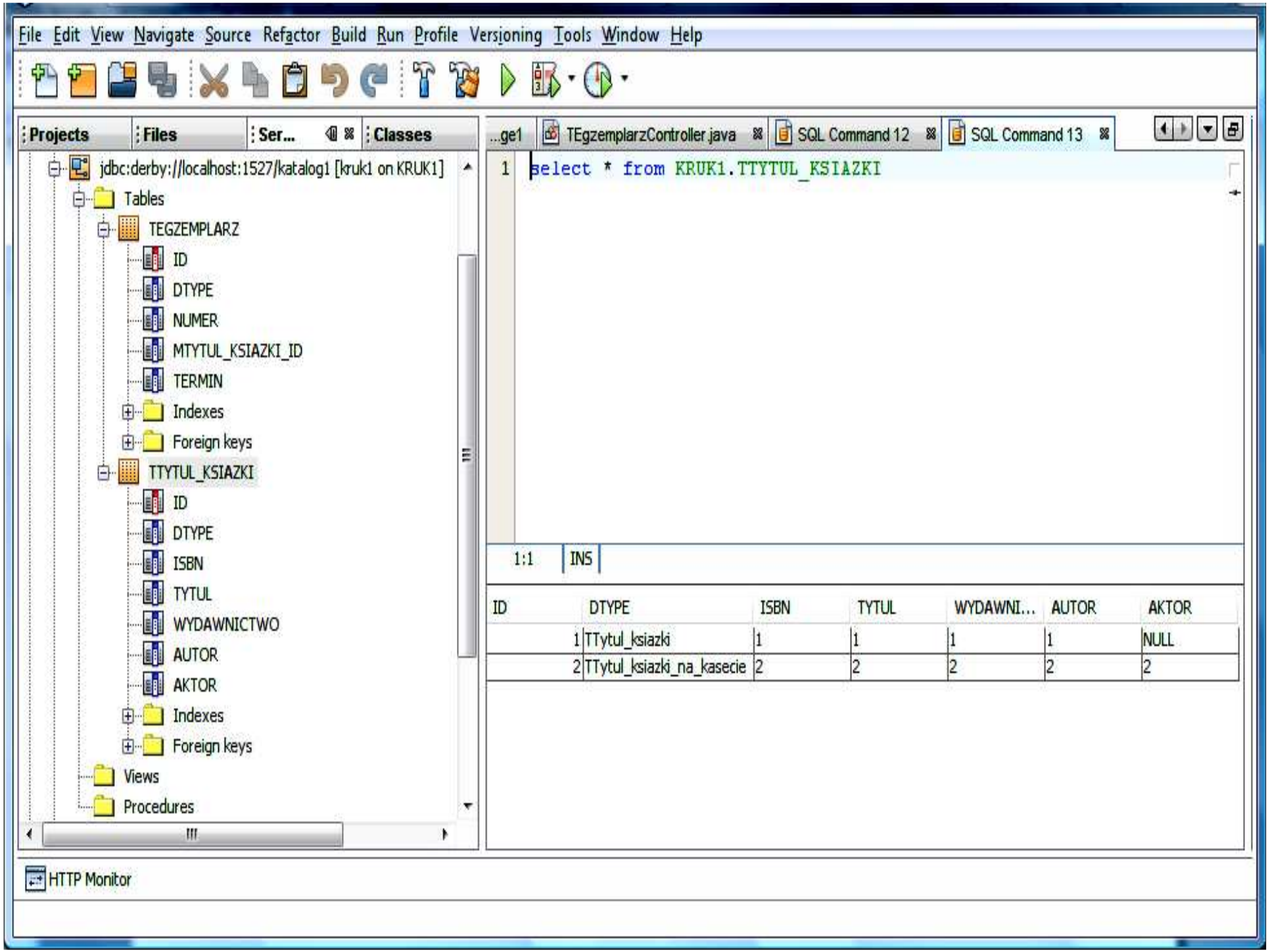


Przykład 2 - @GeneratedValue(strategy=GenerationType.IDENTITY)

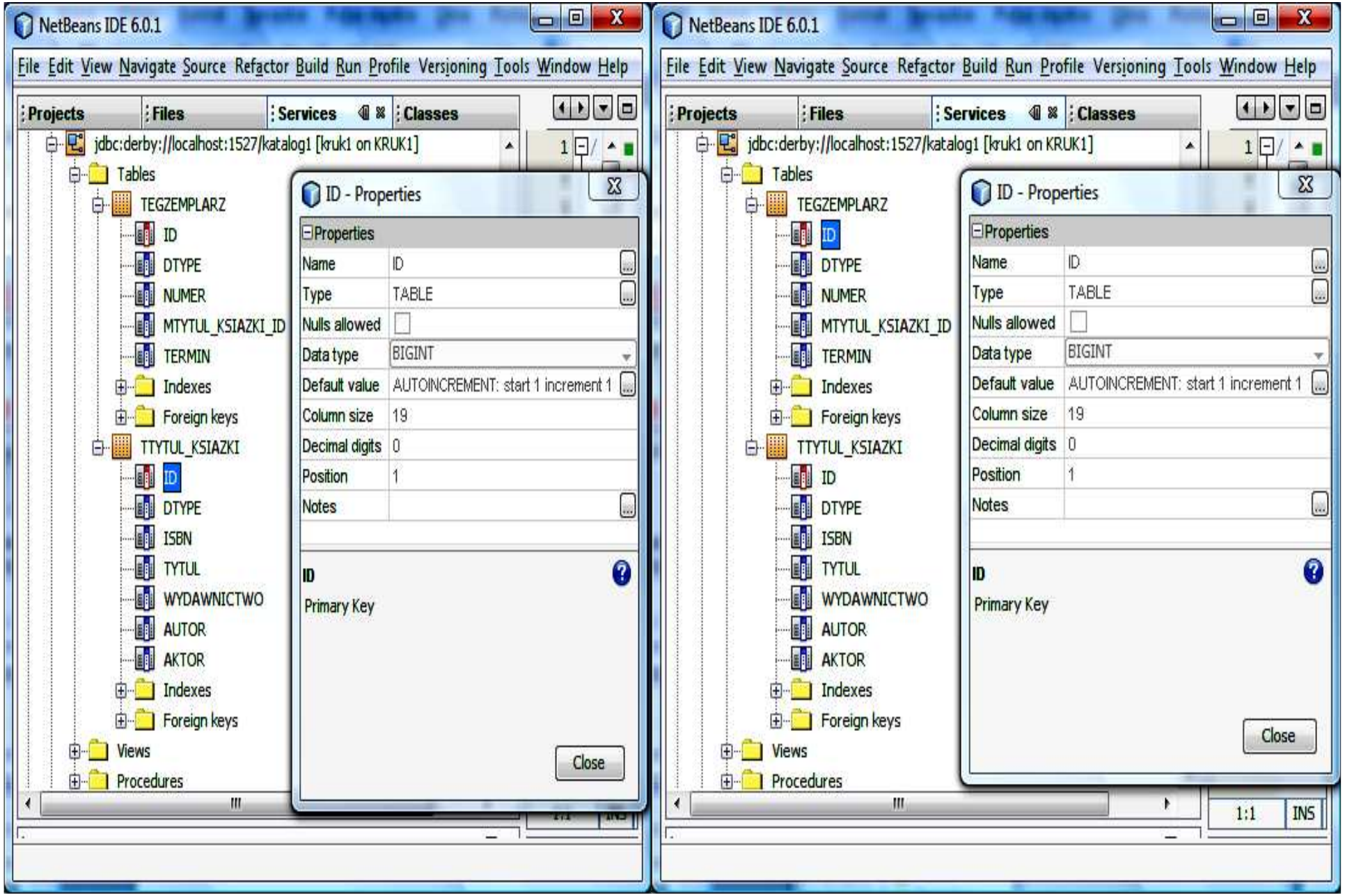



Wypozyczalnia1app - NetBeans IDE 6.0.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
...java *
TEgzemplarz.java *
persistence.xml *
SQL Command 9...
1 package wypozyczalnia1app;
2
3 import java.io.Serializable;
4 import java.util.Date;
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.GenerationType;
8 import javax.persistence.Id;
9 import javax.persistence.ManyToOne;
10
11 @Entity
12 public class TEgzemplarz implements Serializable {
13 private static final long serialVersionUID = 1L;
14 @Id
15 @GeneratedValue(strategy = GenerationType.IDENTITY)
16 private Long id;
17 private int numer;
18 @ManyToOne
19 private TTytul_ksiazki mTytul_ksiazki;
15:55
INS


File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
Files
Services
Classes
jdbc:derby://localhost:1527/katalog1 [kruk1 on KRUK1]
Tables
TEGZEMPLARZ
ID
DTYPE
NUMER
MTYTUL_KSIAZKI_ID
TERMIN
Indexes
Foreign keys
TTYTUL_KSIAZKI
ID
DTYPE
ISBN
TYTUL
WYDAWNICTWO
AUTOR
AKTOR
Indexes
Foreign keys
Views
Procedures
...java
SQL Command 12
1 select * from KRUK1.TEGZEMPLARZ
1:1
INS
ID DTYPE NUMER MTYTUL_K... TERMIN
1 TEgzemplarz 1 1 NULL
2 TEgzemplarz_termin 1 2 2008-05-06
3 TEgzemplarz 2 2 NULL
HTTP Monitor


File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Projects
Files
Ser...
Classes
jdbc:derby://localhost:1527/katalog1 [kruk1 on KRUK1]
Tables
TEGZEMPLARZ
ID
DTYPE
NUMER
MTYTUL_KSIAZKI_ID
TERMIN
Indexes
Foreign keys
TTYTUL_KSIAZKI
ID
DTYPE
ISBN
TYTUL
WYDAWNICTWO
AUTOR
AKTOR
Indexes
Foreign keys
Views
Procedures
...ge1
TEgzemplarzController.java
SQL Command 12
SQL Command 13
1 select * from KRUK1.TTYTUL_KSIAZKI
1:1
INS
ID DTYPE ISBN TYTUL WYDAWNI... AUTOR AKTOR
1 TTytul_ksiazki 1 1 1 1 NULL
2 TTytul_ksiazki_na_kasecie 2 2 2 2 2
HTTP Monitor

Właściwości kluczy głównych przy zastosowanej strategii IDENTITY



Zasady generowania kluczy głównych (2)

3. Specyfikowanie metody generowania za pomocą tabeli (3.1 - za pomocą domyślnej tabeli TopLink lub 3.2 – za pomocą tabeli zdefiniowanej przez programistę = **przykład 3 (wg przykładu z Java EE 5 Tutorial)**)

3.1. @Entity

```
public class Inventory implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.TABLE)
    private long id;
```

3.2. @Entity

```
public class Inventory implements Serializable {
    @Id
    @GeneratedValue(generator="InvTab")
    @TableGenerator(name="InvTab", table="ID_GEN",
      pkColumnName="ID_NAME", valueColumnName="ID_VAL",
      pkColumnValue="INV_GEN")
      private long id;
```

```
np
ID_GEN
ID_NAME                 ID_VAL                  // Nazwy kolumn
INV_GEN                 <last generated value >  // wartości kolejnej krotki
```

4. Wykorzystanie sekwencyjnego mechanizmu nadawania wartości kluczy głównych wspieranych przez bazy danych (4.1 - za pomocą domyślnego mechanizmu lub 4.2 – za pomocą generatora zdefiniowanego przez programistę)

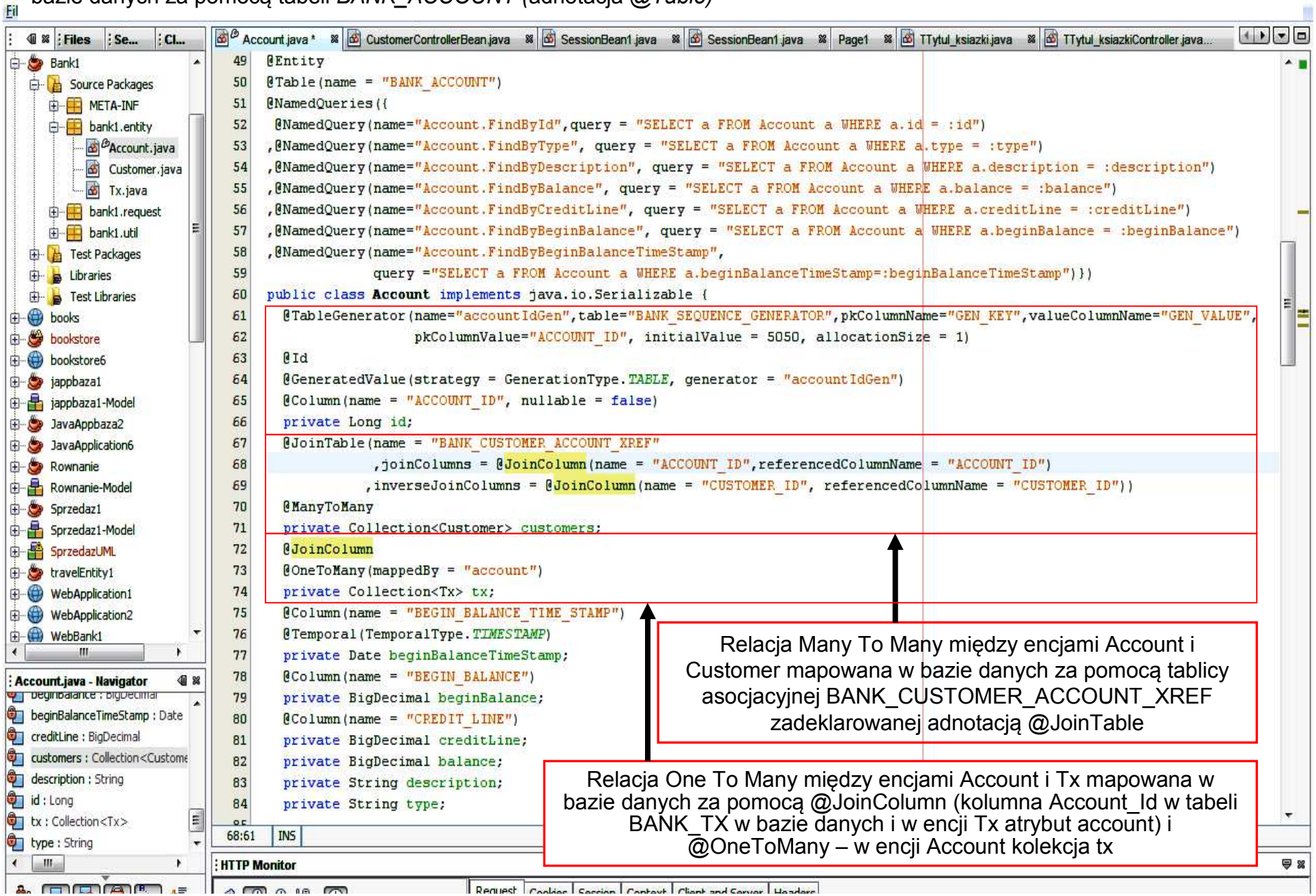
4.1. @Entity

```
public class Inventory implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    private long id;
```

4.2. @Entity

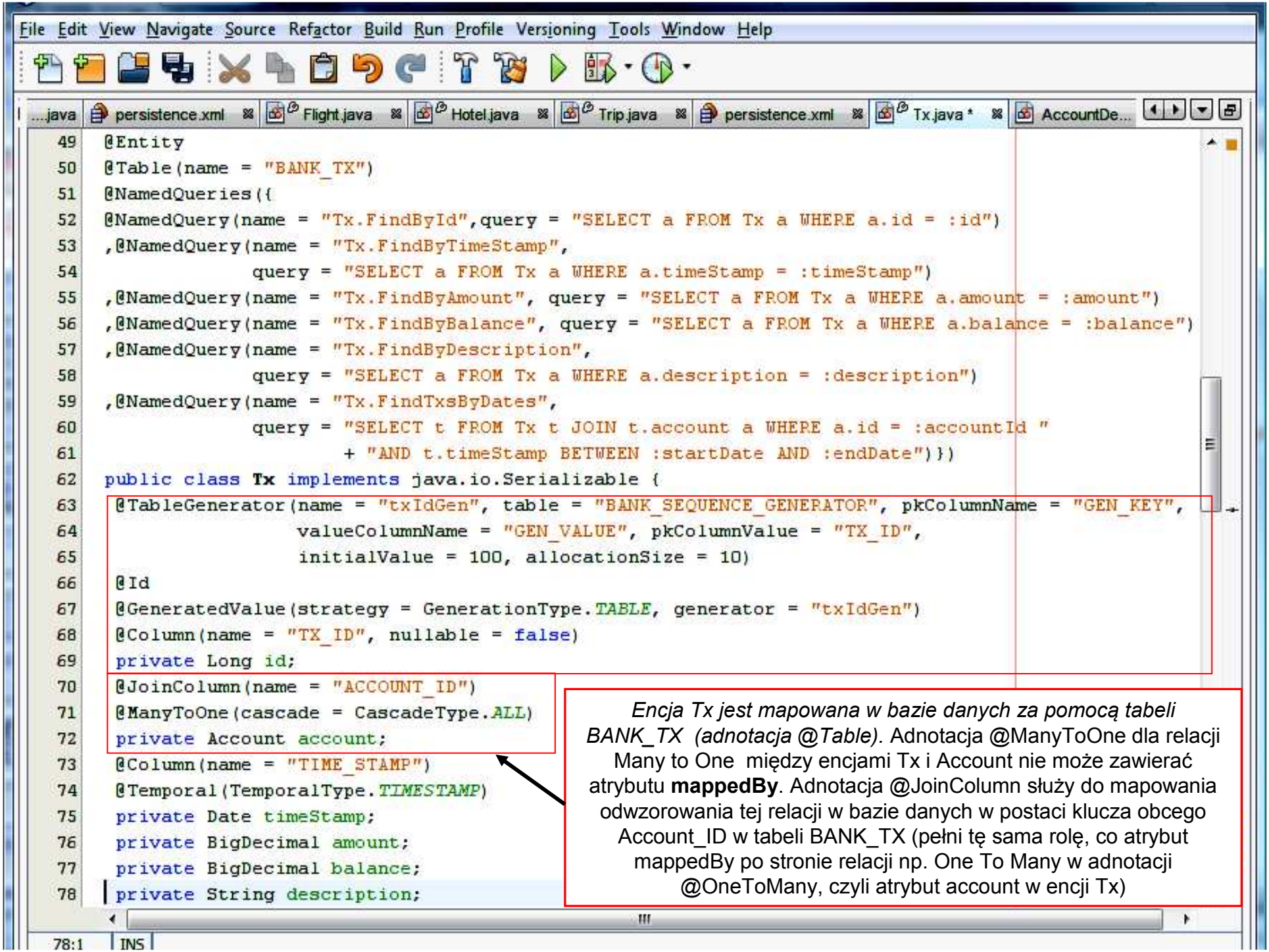
```
public class Inventory implements Serializable {
    @Id
    @GeneratedValue(generator="InvSeq")
    @SequenceGenerator(name="InvSeq",
        sequenceName="INV_SEQ", allocationSize=5)
    private long id;
```

Przykład 3 – **@GeneratedValue(strategy = GenerationType.TABLE, generator = "accountIdGen") - wg Java EE 5 Tutorial**
Adnotacja dla relacji *Many To Many* pomiędzy encjami *Account i Customer (@ManyToMany).* Encja *Account* jest mapowana w bazie danych za pomocą tabeli *BANK_ACCOUNT (*adnotacja *@Table)*



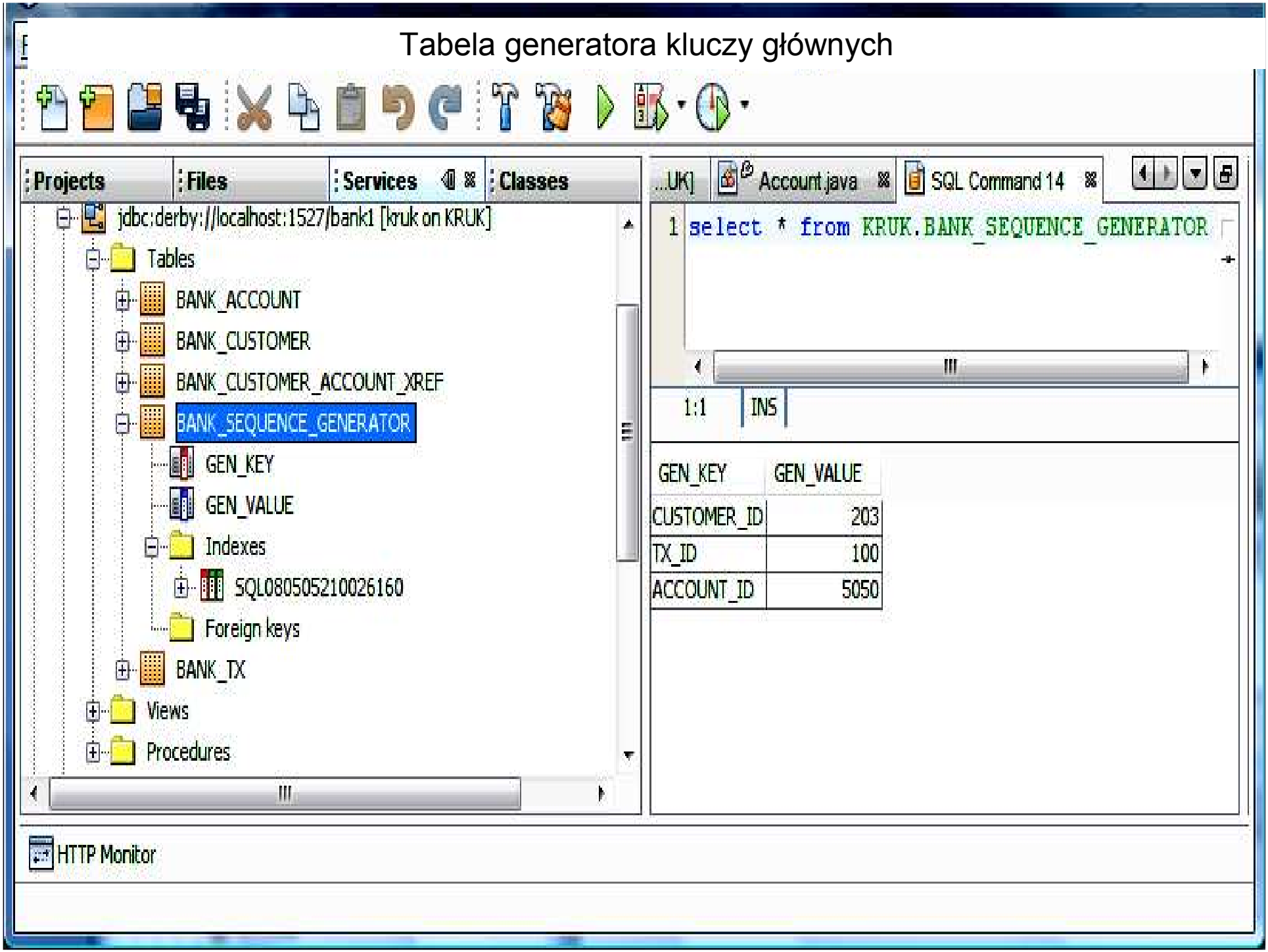
Encja *Customer* jest mapowana w bazie danych za pomocą tabeli *BANK_CUSTOMER (*adnotacja *@Table).* Adnotacja @ManyToMany dla relacji *Many To Many* zawiera jedynie z jednej strony relacji atrybut ***mappedBy*** deklarujący pole relacji ***customers*** w encji ***Account***

```
44  @Entity
45  @Table(name = "BANK_CUSTOMER")
46  @NamedQueries({@NamedQuery(name = "Customer.FindById",query = "SELECT a FROM Customer a WHERE a.id = :id")
47  ,@NamedQuery(name = "Customer.FindByLastName", query = "SELECT a FROM Customer a WHERE a.lastName LIKE :lastName")
48  ,@NamedQuery(name = "Customer.FindByFirstName", query = "SELECT a FROM Customer a WHERE a.firstName = :firstName")
49  ,@NamedQuery(name = "Customer.FindByMiddleInitial", query = "SELECT a FROM Customer a WHERE a.middleInitial = :middleInitial")
50  ,@NamedQuery(name = "Customer.FindByStreet", query = "SELECT a FROM Customer a WHERE a.street = :street")
51  ,@NamedQuery(name = "Customer.FindByCity", query = "SELECT a FROM Customer a WHERE a.city = :city")
52  ,@NamedQuery(name = "Customer.FindByState", query = "SELECT a FROM Customer a WHERE a.state = :state")
53  ,@NamedQuery(name = "Customer.FindByZip", query = "SELECT a FROM Customer a WHERE a.zip = :zip")
54  ,@NamedQuery(name = "Customer.FindByPhone", query = "SELECT a FROM Customer a WHERE a.phone = :phone")
55  ,@NamedQuery(name = "Customer.FindByEmail", query = "SELECT a FROM Customer a WHERE a.email = :email")
56  ,@NamedQuery(name="Customer.FindAllCustomersOfAccount",query="SELECT c FROM Customer c JOIN c.accounts a WHERE a.id = :accountId")})
57  public class Customer implements java.io.Serializable {
58      @ManyToMany(mappedBy = "customers")
59      private Collection<Account> accounts;
60      @TableGenerator(name = "customerIdGen", table = "BANK_SEQUENCE_GENERATOR", pkColumnName = "GEN_KEY",
61                      valueColumnName = "GEN_VALUE", pkColumnValue = "CUSTOMER_ID", initialValue = 203, allocationSize = 1)
62      @Id
63      @GeneratedValue(strategy = GenerationType.TABLE, generator = "customerIdGen")
64      @Column(name = "CUSTOMER_ID", nullable = false)
        private Long id;
66      @Column(name = "FIRST_NAME")
67      private String firstName;
68      @Column(name = "LAST_NAME")
69      private String lastName;
70      @Column(name = "MIDDLE_INITIAL")
71      private String middleInitial;
72      private String city;
73      private String email;
74      private String phone;
        private String state;
76      private String street;
```

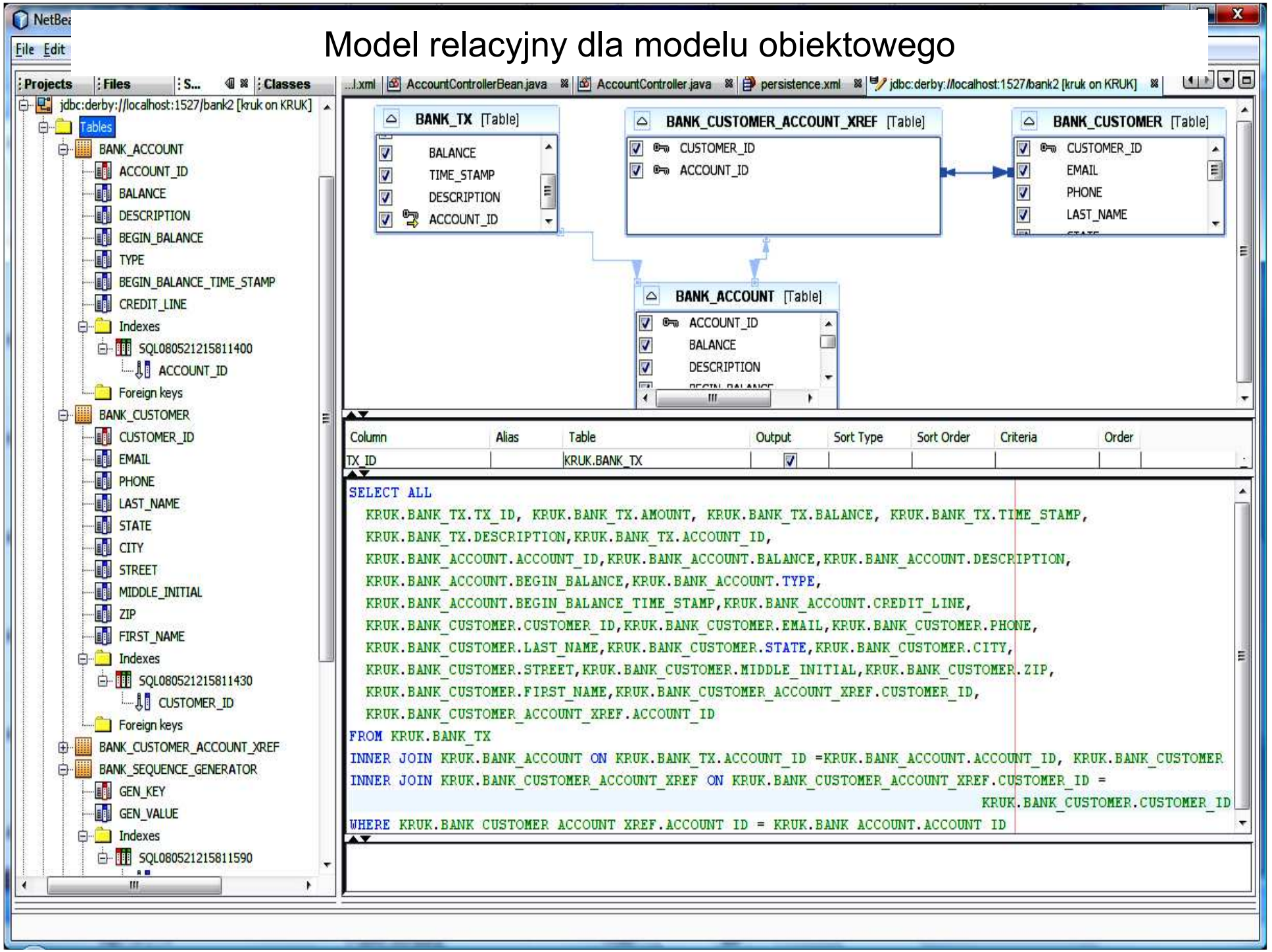


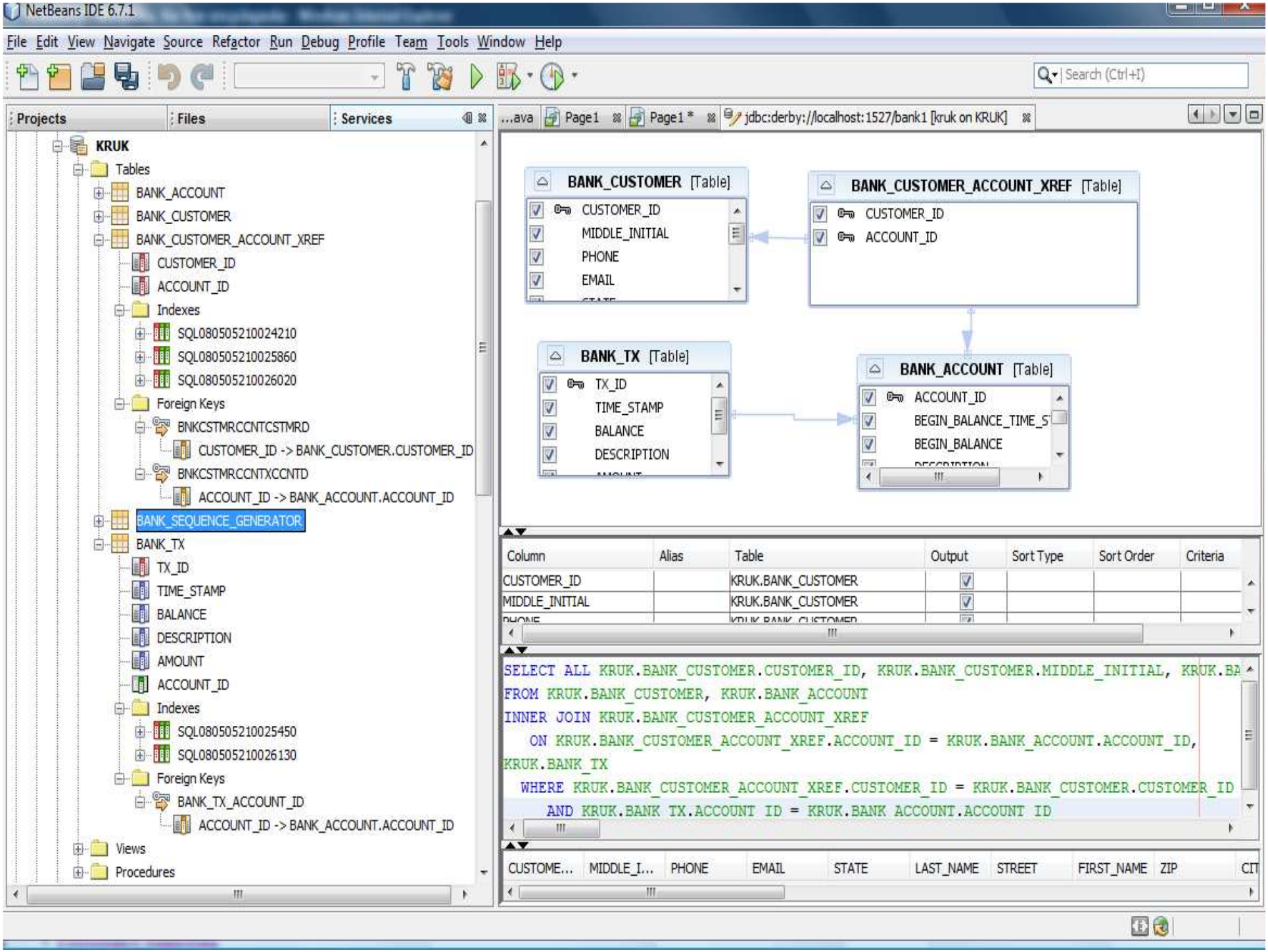
Encja Tx jest mapowana w bazie danych za pomocą tabeli BANK_TX (adnotacja @Table). Adnotacja @ManyToOne dla relacji Many to One między encjami Tx i Account nie może zawierać atrybutu **mappedBy**. Adnotacja @JoinColumn służy do mapowania odwzorowania tej relacji w bazie danych w postaci klucza obcego Account_ID w tabeli BANK_TX (pełni tę sama rolę, co atrybut mappedBy po stronie relacji np. One To Many w adnotacji @OneToMany, czyli atrybut account w encji Tx)

Tabela generatora kluczy głównych



Model relacyjny dla modelu obiektowego




NetBeans IDE 6.7.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
Files
Services
KRUK
Tables
BANK_ACCOUNT
BANK_CUSTOMER
BANK_CUSTOMER_ACCOUNT_XREF
CUSTOMER_ID
ACCOUNT_ID
Indexes
SQL080505210024210
SQL080505210025860
SQL080505210026020
Foreign Keys
BNKCSTMRCCNTCSTMRD
CUSTOMER_ID -> BANK_CUSTOMER.CUSTOMER_ID
BNKCSTMRCCNTXCCNTD
ACCOUNT_ID -> BANK_ACCOUNT.ACCOUNT_ID
BANK_SEQUENCE_GENERATOR
BANK_TX
TX_ID
TIME_STAMP
BALANCE
DESCRIPTION
AMOUNT
ACCOUNT_ID
Indexes
SQL080505210025450
SQL080505210026130
Foreign Keys
BANK_TX_ACCOUNT_ID
ACCOUNT_ID -> BANK_ACCOUNT.ACCOUNT_ID
Views
Procedures
...ava
Page1
Page1 *
jdbc:derby://localhost:1527/bank1 [kruk on KRUK]
BANK_CUSTOMER [Table]
CUSTOMER_ID
MIDDLE_INITIAL
PHONE
EMAIL
BANK_CUSTOMER_ACCOUNT_XREF [Table]
CUSTOMER_ID
ACCOUNT_ID
BANK_TX [Table]
TX_ID
TIME_STAMP
BALANCE
DESCRIPTION
BANK_ACCOUNT [Table]
ACCOUNT_ID
BEGIN_BALANCE
Column
Alias
Table
Output
Sort Type
Sort Order
Criteria
CUSTOMER_ID
KRUK.BANK_CUSTOMER
MIDDLE_INITIAL
KRUK.BANK_CUSTOMER
SELECT ALL KRUK.BANK_CUSTOMER.CUSTOMER_ID, KRUK.BANK_CUSTOMER.MIDDLE_INITIAL, KRUK.BA
FROM KRUK.BANK_CUSTOMER, KRUK.BANK_ACCOUNT
INNER JOIN KRUK.BANK_CUSTOMER_ACCOUNT_XREF
ON KRUK.BANK_CUSTOMER_ACCOUNT_XREF.ACCOUNT_ID = KRUK.BANK_ACCOUNT.ACCOUNT_ID,
KRUK.BANK_TX
WHERE KRUK.BANK_CUSTOMER_ACCOUNT_XREF.CUSTOMER_ID = KRUK.BANK_CUSTOMER.CUSTOMER_ID
AND KRUK.BANK_TX.ACCOUNT_ID = KRUK.BANK_ACCOUNT.ACCOUNT_ID
CUSTOME...
MIDDLE_I...
PHONE
EMAIL
STATE
LAST_NAME
STREET
FIRST_NAME
ZIP

2. Wprowadzenie do języka zapytań „Java Persistence” wg The Java EE 5 Tutorial

Terminologia

- ***Schemat abstrakcyjny (Abstract schema)*: Abstrakcyjny schemat utrwalania (**utrwalane encje, ich stan i powiązania**), na którym operują zapytania. Język zapytań tłumaczy zapytania nad abstrakcyjnym schematem utrwalania na zapytania wykonywane na schemacie bazy danych, do której odwzorowano encje.
- ***Typ abstrakcyjnego schematu utrwalania**:* Wszystkie wyrażenia wyznaczają wynik należący do pewnego typu utrwalanych encji. Typ abstrakcyjnego schematu utrwalania wywodzi się z klas z adnotacjami typu Entity
- ***Nawigacja***: Przechodzenie po związkach między encjami w wyrażeniach języka zapytań – operatorem nawigacji jest kropka.
- ***Ścieżka wyrażenia***: Wyrażenie, które nawiguje do pola stanu encji lub pola relacji.
- ***Pole stanu***: Pole utrwalanej encji
- ***Pole relacji***: Pole utrwalanej encji wykorzystane w relacji w wyrażeniu nawigacji, które jest typu *abstrakcyjnego schematu utrwalania* powiązanej encji

Proste zapytania

Podstawowe zapytanie typu Select

SELECT p FROM Customer p

Wynik: Wszystkie wystąpienia encji Customer.

Opis: Klauzula **FROM** deklaruje zmienną identyfikacji p, z pominięciem słowa kluczowego **AS**.
Jeśli użyto słowo kluczowe **AS**,

FROM AS p

element **Customer** jest abstrakcyjnym schematem encji **Customer**.

Eliminowanie duplikatów wartości

SELECT DISTINCT p FROM Customer p WHERE p.city = ?'Warszawa'

Wynik: Wystąpienia encji **Custromer** (klientów) z wartościami atrybutu **city** równych wartości parametru zapytania ?'Waszawa'.

Opis: Słowo kluczowe **DISTINCT** eliminuje powtórzenia wystąpień wartości **city** w zbiorze klientów.
Klauzula **WHERE** wyszczególnia wystąpienia encji **Customer** za pomocą sprawdzania pola **city** utrwalanej encji **Customer**. Element ?'Warszawa' jest parametrem zapytania jako stała wartość.

Użycie parametrów o podanych nazwach

SELECT DISTINCT p FROM Customer p
WHERE p.city = :city AND p. firstName = :firstname

Wynik: Wystąpienia encji **Customer** (klienci) posiadający wyspecyfikowane atrybuty **city** i **firstname.**

Opis: Atrybuty **city** i **firstname** są utrwalanymi polami encji **Customer**. Klazula **WHERE** porównuje wartości pól z wartościami parametrów zapytania, używając metody **Query.setNamedParameter**. Język zapytań pozwala nazywanie parametrów zapytania wprowadzając symbol (:) poprzedzający nazwę parametru. Pierwszym parametrem wejściowym jest **:city**, drugim jest **:firstname**.

Nawigacja powiązanymi encjami (1)

Proste zapytanie dotyczące encji powiązanych relacją

SELECT DISTINCT p FROM Customer p, IN(p. accounts) t

Wynik: Wszystkie wystąpienia encji **Customer**, które należą do wystąpień encji **Account**.

Opis: Klazula **FROM** deklaruje dwie identyfikujące zmienne: **p**, **t**. Zmienna **p** reprezentuje encję **Customer**, zmienna **t** reprezentuje encję **Account**. Deklaracja zmiennej **t** odpowiada wcześniejszej deklaracji zmiennej **p**. Słowo kluczowe **IN** oznacza, że **accounts** jest kolekcją powiązanych wystąpień encji **Account**. Wyrażenie **p.accounts** nawiguje od encji **Customer** do powiązanej encji **Account**. Operator **kropka** w wyrażeniu **p.accounts** jest operatorem nawigacji.

Można zamiennie użyć polecenie **JOIN** do wyrażenia takiego samego zapytania:

SELECT DISTINCT p FROM Customer p JOIN p.accounts t

lub:

SELECT DISTINCT p FROM Customer p WHERE p.accounts IS NOT EMPTY

Nawigacja do jednowartościowych pól relacji

Użycie klauzuli JOIN:

```
SELECT t FROM Tx t JOIN t.account a
        WHERE a. beginBalanceTimeStamp = '2008-05-23'
          OR a.beginBalanceTimeStamp ='2008-05-28'
```

W przykładzie zapytanie zwraca wszystkie wystąpienia encji **Tx**, które są powiązane z wystąpieniami **a** encji **Account** (**t.account**) i w tych wystąpieniach encji **Account** wartość atrybutu **beginBalanceTimeStamp** jest równa wartościom ’008-05-23’ lub ‘2008-05-28' .

Nawigacja powiązanymi encjami (2)

Użycie parametrów wejściowych w zapytaniach na powiązanych encjach

SELECT DISTINCT p FROM Customer p, IN (p.accounts) AS t WHERE t.type = :type

Wynik : Wystąpienia encji **Customer**, których rachunki posiadają atrybut **type** jest równy wartości parametru **:type**.

Opis : To zapytanie jest podobne do poprzedniego przypadku, różni się wprowadzeniem wejściowego parametru **:type**. Słowo kluczowe **AS** w klauzuli **FROM** jest opcjonalne. W klauzuli **WHERE**, wyrażenie **t.type** (operator kropki) przetwarzające pole utrwalane **type** pełni rolę operatora eliminacji niż operatora nawigacji. Jest ono zastosowane, ponieważ pole relacji **p.accounts** jest kolekcją powiązanych wystąpień encji **Account** z wystąpieniami encji **Customer**, a wyrażenia nie mogą nawigować polami relacji, które są kolekcjami (kolekcje są symbolami końcowymi wyrażeń nawigacji). Dlatego wyrażenie **p.accounts.type** (jest to nieprawidłowe wyrażenie) zastąpiono wyrażeniem **t.type,** udostępniającym utrwalane pole **type.**

Zapytanie na wielokrotnych relacjach

SELECT DISTINCT p FROM Customer p, IN (p.accounts) t JOIN t.tx q
WHERE q.account = :account

Wynik: Wystąpienia ***p*** różnych encji **Customer** posiadających rachunki zawierające się w zbiorze transakcji **tx** i są równe parametrowi **:account**

Opis: Wyrażenia w zapytaniu nawigują trzema relacjami. Wyrażenie **p.accounts** nawiguje relacją **Customer-Account**, i wyrażenie **t.tx** relacją **Account-Tx** oraz **q.account** relacją **Tx-Account**. W innych przykładach, wejściowymi parametrami są obiekty typu String, w przykładzie parametrem jest obiekt typu **account**. Ten typ oznacza **pole relacji** w wyrażeniu porównania klauzuli **WHERE.**

Nawigacja między powiązanymi polami

SELECT DISTINCT p FROM Account p, IN (p.tx) t
WHERE t.account.balance = :balance

Wynik: Wystąpienia encji **Account**, które posiadają wyspecyfikowaną wartość **balance** za pomocą parametru **:balance**.

Opis: Pole **balance** należy do encji **Account**. Aby osiągnąć pole **account**, zapytanie musi nawigować od encji **Account** do **Tx** (**p.tx**) i potem od encji **Tx** do encji **Account**(**t.account**). Ponieważ pole relacji **account** nie jest kolekcją, dlatego prawidłowe jest wyrażenie **t.account.balance.**

Zapytania z różnymi wyrażeniami warunkowymi klauzuli WHERE

- **Wyrażenie LIKE**

```
SELECT p FROM Customer p WHERE p.lastName LIKE 'Mich%'
```

Wynik: Wszystkie wystąpienia encji **Customer**, których atrybuty **lastName** zaczynają się od podłańcucha "Mich."

Opis: Wyrażenia **LIKE** używają maski ze znakami % do wyszukiwania łańcuchów. W tym przypadku wyrażenie **LIKE** wykrywa wartości "Michael" i "Michelle„ za pomocą użytej maski (wzorca).

- **Wyrażenie IS NULL**

```
SELECT t FROM Tx t WHERE t.account IS NULL
```

Wynik: Wszystkie wystąpienia encji **Tx** związane z atrybutem **account** równym **NULL**.

Opis: Wyrażenie **IS NULL** może być użyte do testowania zbioru powiązań między wystąpieniami dwóch encji.

- **Wyrażenie IS EMPTY**

```
SELECT p FROM Customer p WHERE p.accounts IS EMPTY
```

Wynik: Wszystkie wystąpienia encji **Customer**, które nie należą do wystąpień encji **Account.**

Opis: Pola relacji są kolekcjami wystąpień encji **Customer**. Jeśli klient nie posiada rachunków, wtedy kolekcja wystąpień encji Account w polu **accounts** jest pusta, wtedy wyrażenie **IS EMPTY** jest równe **TRUE**.

- **Wyrażenie BETWEEN**

```
SELECT DISTINCT p FROM Account p
        WHERE p.balance BETWEEN :lowerBalance AND :higherBalance
```

Wynik : Wszystkie wystąpienia encji **Account**, których atrybut **balance** zawiera się między podanymi granicami za pomocą parametrów :**lowerBalance** i :**higherBalancy**.

Opis: Wyrażenie **BETWEEN** ma trzy arytmetyczne wyrażenia: pole utrwalania (**p.balance**) i dwa wejściowe parametry (:**lowerBalance** i :**higherBalance**). Następujące wyrażenie jest równoważne wyrażeniu **BETWEEN**:

```
p.balance >= :lowerbalance AND p.balance <= :higherBalance
```

- **Operatory porównania**

```
SELECT DISTINCT p1 FROM Account p1, Account p2
        WHERE p1.balance>p2.balance AND p2.type=:type
```

Wynik: Wszystkie wystąpienia encji **Account**, których atrybut **balance** jest wyższy od wartości atrybutu **balance** tych wystąpień, których atrybut **type** jest równa wartości parametru **type**.

Opis: Klauzula **FROM** deklaruje dwie zmienne identyfikujące **p1** i **p2** tego samego typu **Account**. Obie zmienne są potrzebne, ponieważ klauzula **WHERE** porównuje atrybut **balance** jednego rachunku **p2** z atrybutem **balance** innego rachunku **p1**.

Zapytania typu Update i Delete

Następujące wyrażenia pokazują, jak użyć wyrażenia UPDATE i DELETE w zapytaniach. UPDATE i DELETE operują na wielokrotnych encjach w odniesieniu do warunków użytych w klauzuli WHERE. Klauzula WHERE w zapytaniach UPDATE i DELETE pełni taką samą jak w zapytaniach SELECT.

Update

```
UPDATE Customer p SET p.city = ‘Warszawa'
        WHERE p.middleInitial < :middleInitial
```

Opis: To zapytanie ustawia atrybut *city* zbioru wystapień encji **Customer** , jeśli stan początkowy konta rachunku (wartość atrybutu **middleInitial**) była mniejsza niż wyspecyfikowana w parametrze :**middleInitial**

Delete

```
DELETE FROM Customer p
        WHERE p.lastName = :lastName AND p.accounts IS EMPTY
```

Opis: Zapytanie usuwa wszystkich klientów (zbioru wystąpień encji **Customer,** którzy mają atrybut **lastName** ma wartość „**lastName**”), którzy nie posiadają rachunków (atrybut **p.accounts** typu kolekcja jest pusty)

Przykład 4 – adnotacje zapytań, przykład zastosowania

```
@NamedQueries ({
@NamedQuery(name = "Customer.FindById",
      query = "SELECT a FROM Customer a WHERE a.id = :id")
,@NamedQuery(name = "Customer.FindByLastName",
      query = "SELECT a FROM Customer a WHERE a.lastName LIKE :lastName")
,@NamedQuery(name = "Customer.FindByFirstName",
      query = "SELECT a FROM Customer a WHERE a.firstName = :firstName")
,@NamedQuery(name = "Customer.FindByMiddleInitial",
      query = "SELECT a FROM Customer a WHERE a.middleInitial = :middleInitial")
,@NamedQuery(name = "Customer.FindByStreet",
      query = "SELECT a FROM Customer a WHERE a.street = :street")
,@NamedQuery(name = "Customer.FindByCity",
     query = "SELECT a FROM Customer a WHERE a.city = :city")
,@NamedQuery(name = "Customer.FindByState",
     query = "SELECT a FROM Customer a WHERE a.state = :state")
,@NamedQuery(name = "Customer.FindByZip",
     query = "SELECT a FROM Customer a WHERE a.zip = :zip")
,@NamedQuery(name = "Customer.FindByPhone",
     query = "SELECT a FROM Customer a WHERE a.phone = :phone")
,@NamedQuery(name = "Customer.FindByEmail",
     query = "SELECT a FROM Customer a WHERE a.email = :email")
,@NamedQuery(name="Customer.FindAllCustomersOfAccount",
    query="SELECT c FROM Customer c JOIN c.accounts a WHERE a.id = :accountId" )})
```

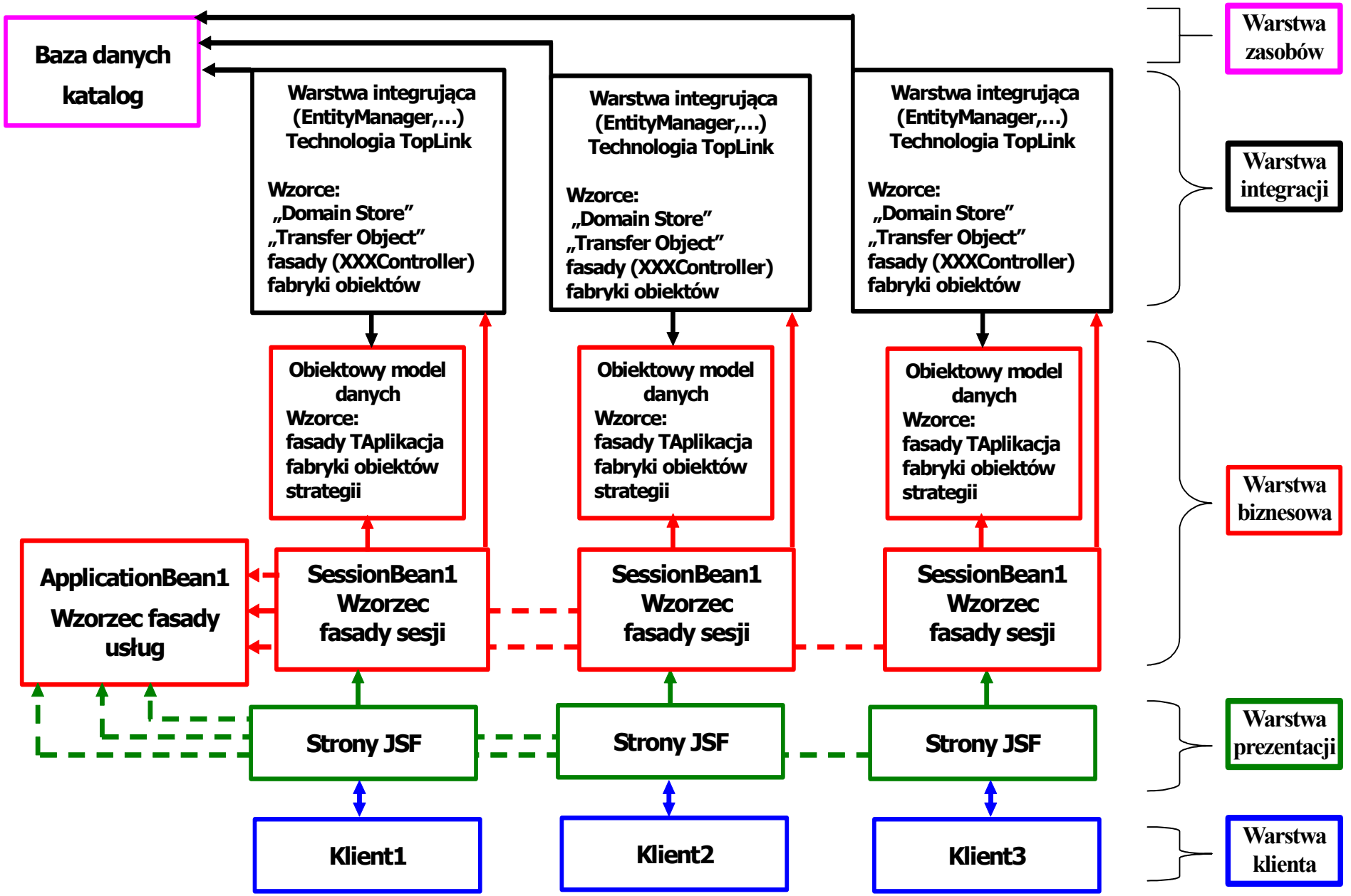
```
@NamedQuery(name="Customer.FindAllCustomersOfAccount",
            query="SELECT c FROM Customer c JOIN c.accounts a
                   WHERE a.id = :accountId")
```

```
customers = em.createNamedQuery("Customer.FindAllCustomersOfAccount")
                .setParameter("accountId", accountId)
                .getResultList();
```

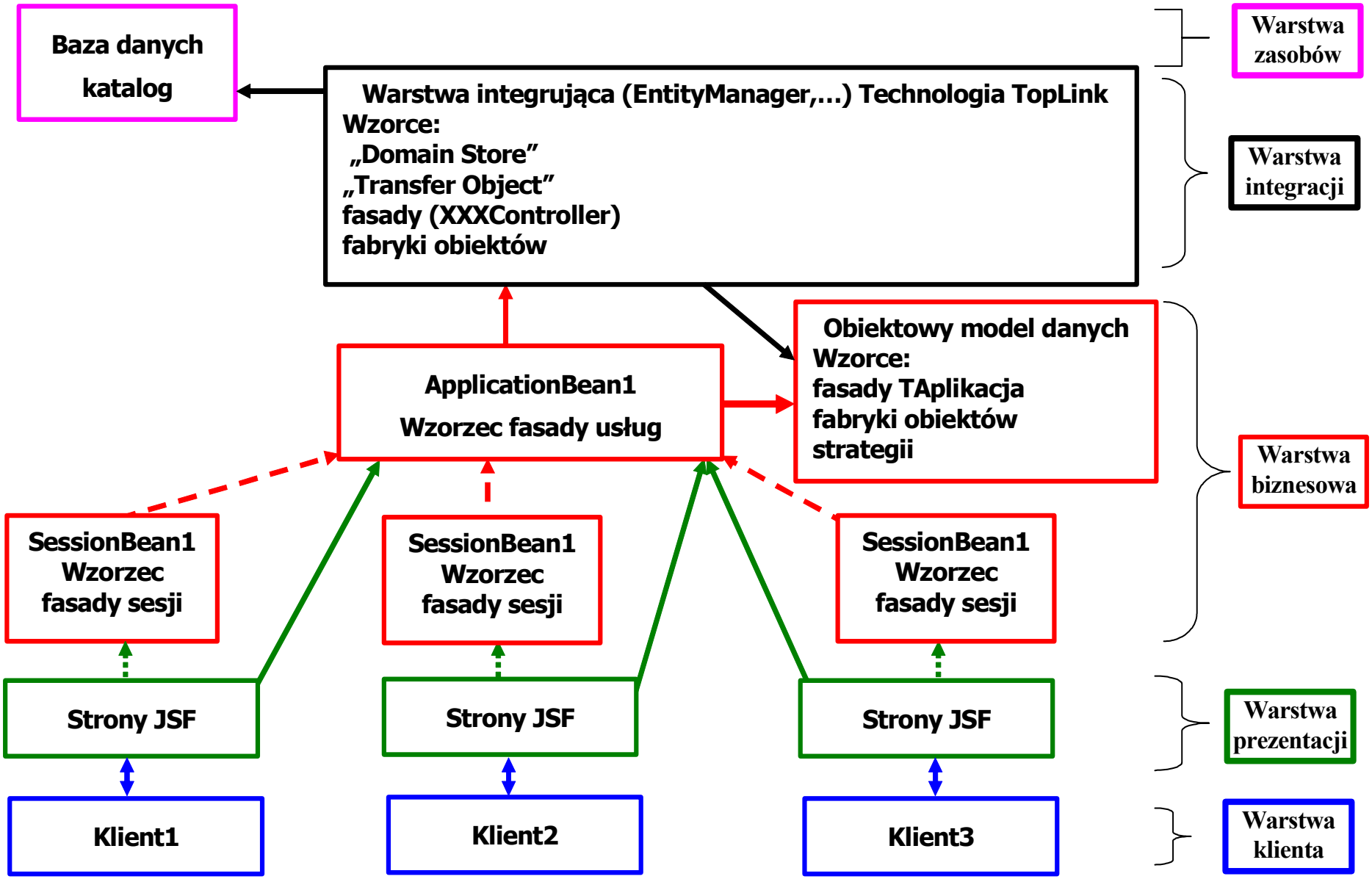
```
public List<Long> getCustomerIds(Long accountId)   throws RuntimeException
 {
   Debug.print("AccountControllerBean getCustomerIds");
   List<Customer> customers = null;
   EntityManager em=getEntityManager();
   if (accountId == null)
        {  throw new RuntimeException("null accountId");  }
   try
    { customers = em.createNamedQuery( "Customer.FindAllCustomersOfAccount")
                        .setParameter("accountId", accountId).getResultList();
      if (customers == null)
                { throw new RuntimeException();    }
     } catch (Exception ex)
                { throw new RuntimeException(ex);  }
    em.close();
    return copyCustomerIdsToList(customers);
   }
```

3. Wzorce warstwy biznesowej: fasada sesji i fasada usług

Architektura aplikacji pięciowarstwowej (przykład z wykładu 5 - punkt 4)- linie przerywane oznaczają powiązania nie wykorzystane w aplikacji



Architektura aplikacji pięciowarstwowej (przykład z wykładu 6 – punkt 4)-
linie przerywane oznaczają powiązania nie wykorzystane w aplikacji



Problem 1 – udostępnianie komponentów i usług biznesowych zdalnym klientom

wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe

Uwagi:

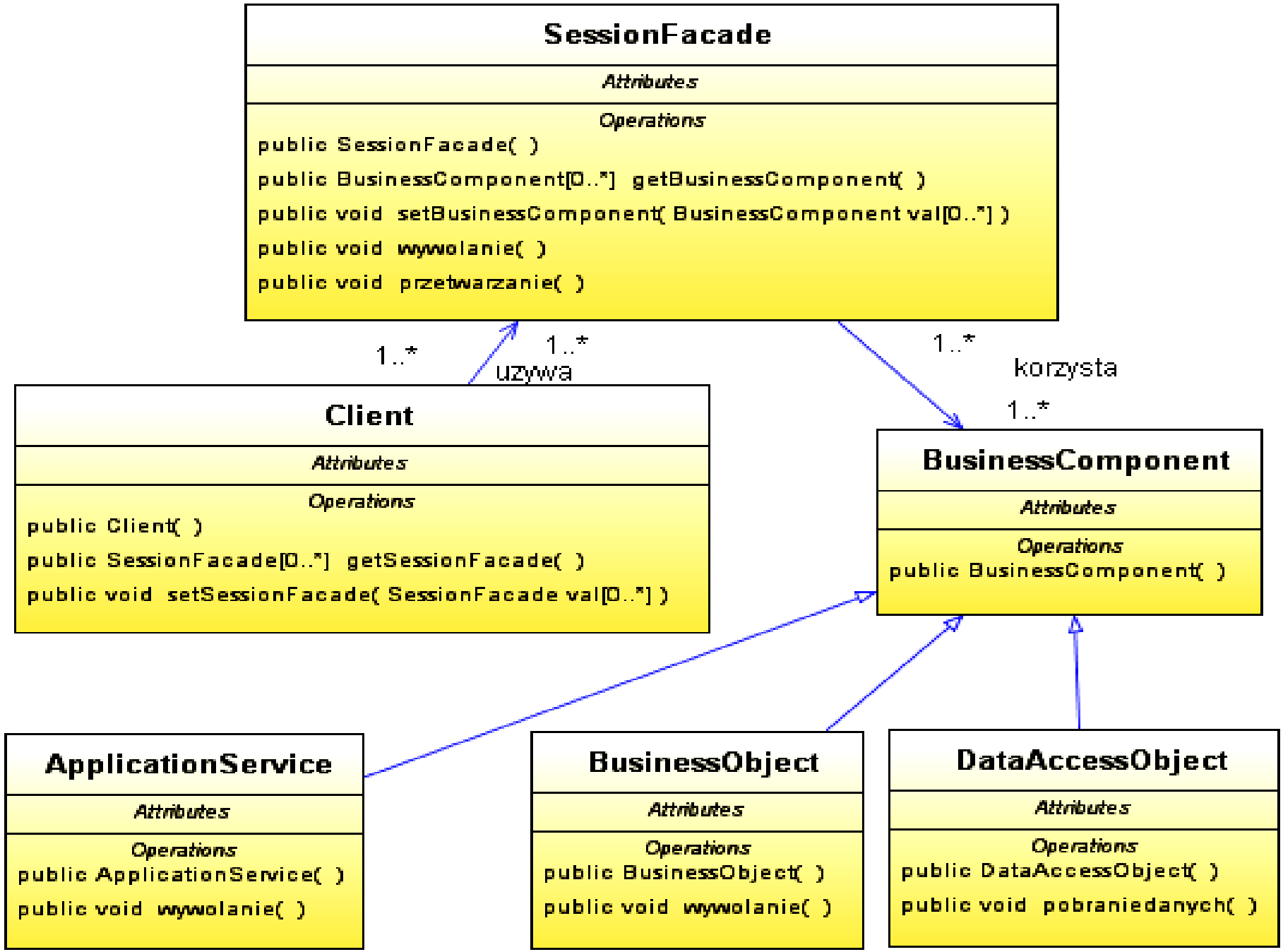
1. Często **warstwa biznesowa** jest realizowana po stronie serwera w postaci obiektów *Business Object*, zwykłych obiektów Javy lub komponentów Entity – wszystkie typu „Entity".
   - Kiedy klienci bezpośrednio komunikują się z komponentami usług biznesowych, zmiana interfejsu komponentu biznesowego ma bezpośredni wpływ na kod klienta.
   - Bezpośredni dostęp wymaga, by klient zawierał złożoną logikę zajmującą się koordynacją i interakcją pomiędzy wieloma komponentami biznesowymi powiązanymi złożonymi relacjami. Wtedy kod klienta zawiera złożone operacje wyszukiwania, zarządzania transakcjami, bezpieczeństwem i sam przeprowadza przetwarzanie biznesowe.
   - Kiedy istnieje wiele typów klientów, bezpośredni dostęp do komponentów biznesowych prowadzi do nierównomiernego korzystania z usług i powielania kodu u klientów. Utrudnia to pielęgnację kodu klienta i zmniejsza elastyczność jego zastosowania.
2. **Rozdrobnienie komponentów na jedną usługę biznesową** prowadzi do wielu komunikacji w sieci z powodu zdalnych wywołań do rozdrobnionych komponentów

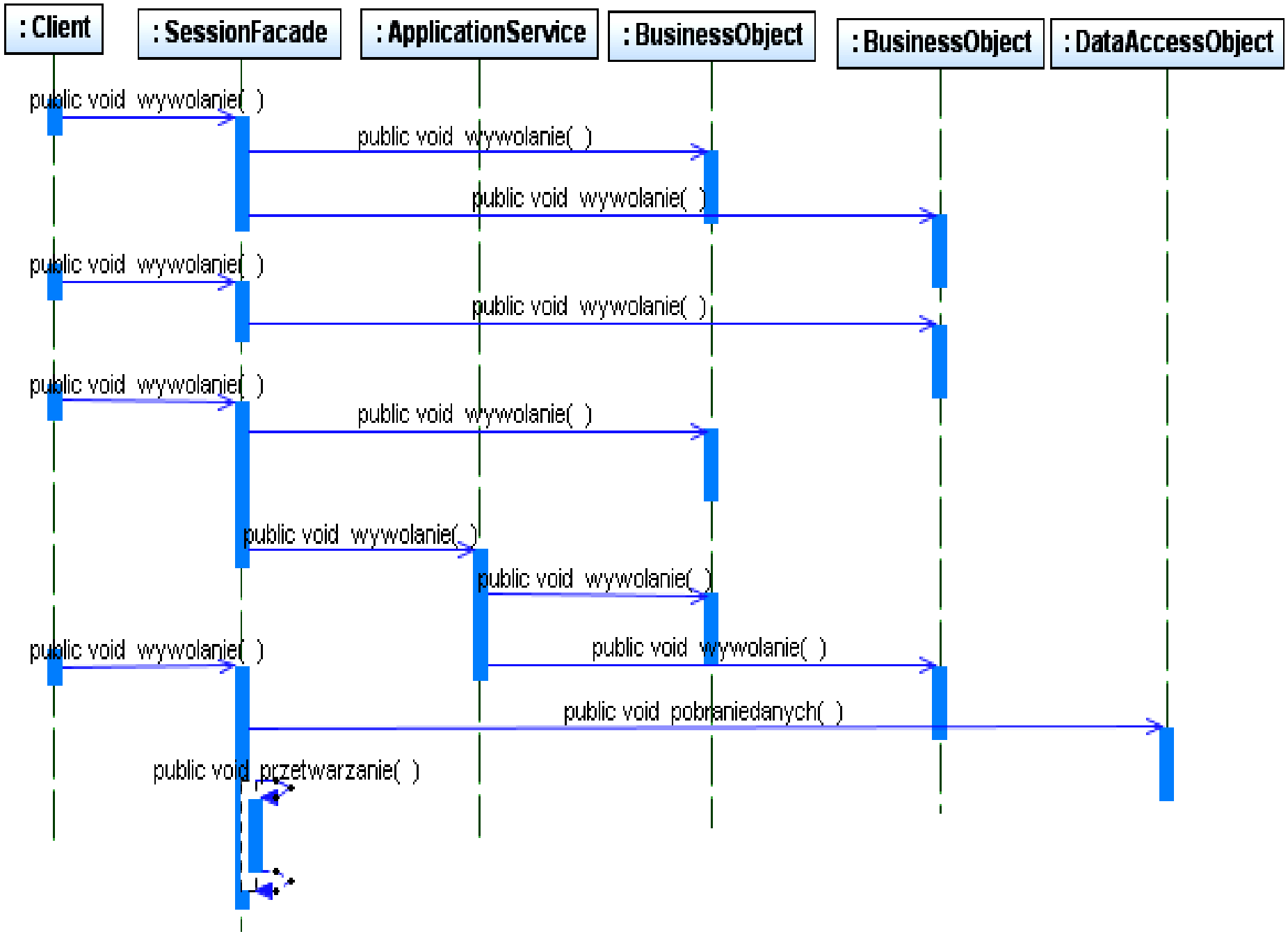
Wymagania:

1. Należy uniknąć bezpośredniego udostępnienia klientom komponentów warstwy biznesowej, aby nie dopuścić do powstania zbyt wielu zależności między klientami a warstwą biznesową
2. Należy zapewnić zdalną warstwę dostępu do obiektów **Business Object** lub innych obiektów warstwy biznesowej
3. Należy zgrupować i udostępnić zdalnym klientom usługi aplikacji (implementować wzorzec **Application Service**) oraz dowolne inne usługi
4. Należy scentralizować i połączyć całą logikę biznesową udostępnianą zdalnym klientom
5. Należy ukryć złożone interakcje i wzajemne zależności między komponentami i usługami biznesowymi w celu ułatwienia zarządzania, centralizacji logiki, zwiększenia elastyczności i ułatwienia wprowadzania zmian.

Wzorzec Session Facade (fasada sesji) – udostępnia klientom ogólne usługi biznesowe realizujące powiązane przypadki użycia, na przykład służące do obsługi rachunku bankowego

- **Client** – klient komponentu **Session Facade**, który chce uzyskać dostęp do usług biznesowych.
- **BusinessComponent** – uczestniczy w realizacji żądań klienta. Może nim być **Business Object** jako model obiektowy danych i zachowanie biznesowe lub jako **ApplicationService** lub jako **DataAccessObject**
- **ApplicationService** – obiekt ten korzysta z obiektów biznesowych i implementuje logikę biznesową. Komponent **Session Facade** może korzystać z wielu takich obiektów
- **DataAccessObject** – pośredniczy w dostępie do bazy danych w prostych aplikacjach, w których nie ma warstwy obiektów biznesowych tworzących model obiektowy danych


SessionFacade
Attributes
Operations
public SessionFacade( )
public BusinessComponent[0..*] getBusinessComponent( )
public void setBusinessComponent(BusinessComponent val[0..*])
public void wywolanie( )
public void przetwarzanie( )
1..*
1..*
uzywa
1..*
korzysta
1..*
Client
Attributes
Operations
public Client( )
public SessionFacade[0..*] getSessionFacade( )
public void setSessionFacade(SessionFacade val[0..*])
BusinessComponent
Attributes
Operations
public BusinessComponent( )
ApplicationService
Attributes
Operations
public ApplicationService( )
public void wywolanie( )
BusinessObject
Attributes
Operations
public BusinessObject( )
public void wywolanie( )
DataAccessObject
Attributes
Operations
public DataAccessObject( )
public void pobraniedanych( )


: Client
: SessionFacade
: ApplicationService
: BusinessObject
: BusinessObject
: DataAccessObject
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void wywolanie( )
public void pobraniedanych( )
public void przetwarzanie( )

Problem 2 – centralizacja logiki biznesowej kilku komponentów i usług biznesowych

wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe

Uwagi:

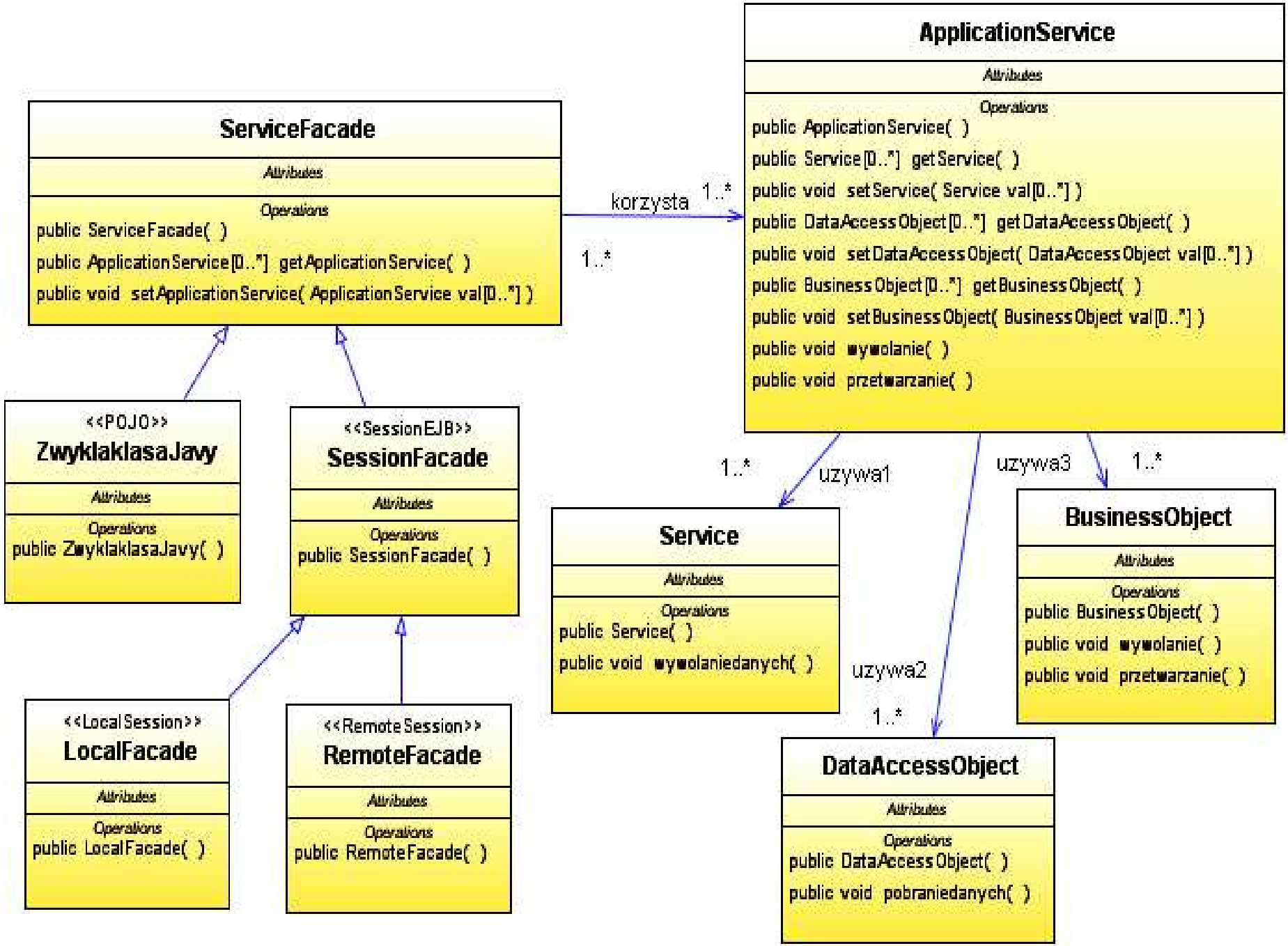
1. Fasady sesji, na przykład **Session Facade** lub fasady zwykłych obiektów zawierają niewiele logiki biznesowej i udostępniają prosty, nierozdrobniony interfejs.
2. Aplikacje implementują przypadki użycia koordynujące współpracę kilku obiektów biznesowych i usług. Nie należy jednak implementować logiki zarządzającej relacjami pomiędzy obiektami biznesowymi w samych obiektach, gdyż zwiększa to zależności pomiędzy nimi i zmniejsza elastyczność. Logiki nie należy umieszczać w fasadzie, gdyż mogłaby ona zostać powielona w wielu komponentach fasady.
3. Aplikacje, które nie stosują komponentów EJB, implementują komponenty warstwy biznesowej jako zwykłe obiekty. Ta logika biznesowa nie powinna być umieszczana ani w fasadzie sesji, ani w kodzie klienta.

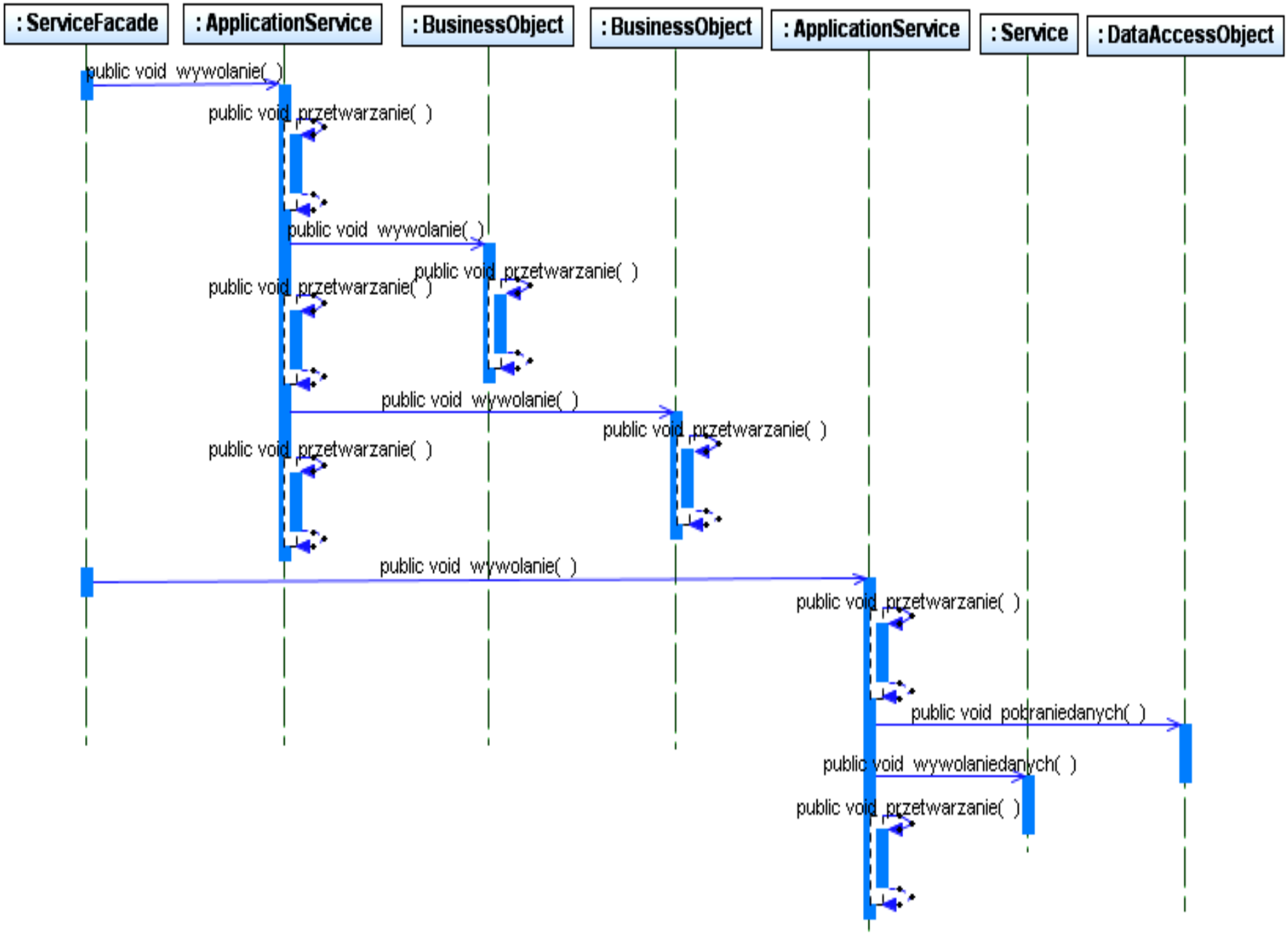
Wymagania:

1. Należy ograniczyć ilość logiki biznesowej w fasadach sesji.
2. Logika biznesowa operuje na kilku obiektach biznesowych lub usługach.
3. Należy utworzyć nierozdrobniony interfejs usług nad istniejącymi komponentami i usługami biznesowymi.
4. Należy umieścić logikę związaną z konkretnymi przypadkami użycia obiektami biznesowymi (głównie dotyczącą wzajemnych zależności pomiędzy obiektami biznesowymi) poza obiektami **Business Object**

Wzorzec **Application Service (fasada usług)** – zapewnia jednolitą warstwę usług i stanowi zaplecze fasad.

- **Client –** fasada typu **Session Facade**, obiekty zwykłych klas Javy, inny obiekt typu **Application Service**
- **Application Service –** pełni główną rolę, hermetyzuje logikę biznesową operującą na kilku obiektach biznesowych lub opartą na konkretnym przypadku użycia. Może on wywoływać metodę biznesową obiektu typu **BusinessObject** lub innego obiektu typu **ApplicationService** lub obiektu dostępu do danych **DataAccessObject**
- **BusinessObject –** kilka obiektów tego typu do realizacji żądania **ApplicationService**
- **Service –** komponent udostępniający dowolny rodzaj usługi
- **DataAccessObject** – obiekty dostępu do danych


ServiceFacade
Attributes
Operations
public ServiceFacade( )
public ApplicationService[0..*] getApplicationService( )
public void setApplicationService(ApplicationService val[0..*])
korzysta
1..*
1..*
ApplicationService
Attributes
Operations
public ApplicationService( )
public Service[0..*] getService( )
public void setService(Service val[0..*])
public DataAccessObject[0..*] getDataAccessObject( )
public void setDataAccessObject(DataAccessObject val[0..*])
public BusinessObject[0..*] getBusinessObject( )
public void setBusinessObject(BusinessObject val[0..*])
public void wywolanie( )
public void przetwarzanie( )
<<POJO>>
ZwyklaklasaJavy
Attributes
Operations
public ZwyklaklasaJavy( )
<<SessionEJB>>
SessionFacade
Attributes
Operations
public SessionFacade( )
1..*
uzywa1
Service
Attributes
Operations
public Service( )
public void wywolaniedanych( )
uzywa3
1..*
BusinessObject
Attributes
Operations
public BusinessObject( )
public void wywolanie( )
public void przetwarzanie( )
uzywa2
1..*
DataAccessObject
Attributes
Operations
public DataAccessObject( )
public void pobraniedanych( )
<<Local Session>>
LocalFacade
Attributes
Operations
public LocalFacade( )
<<Remote Session>>
RemoteFacade
Attributes
Operations
public RemoteFacade( )

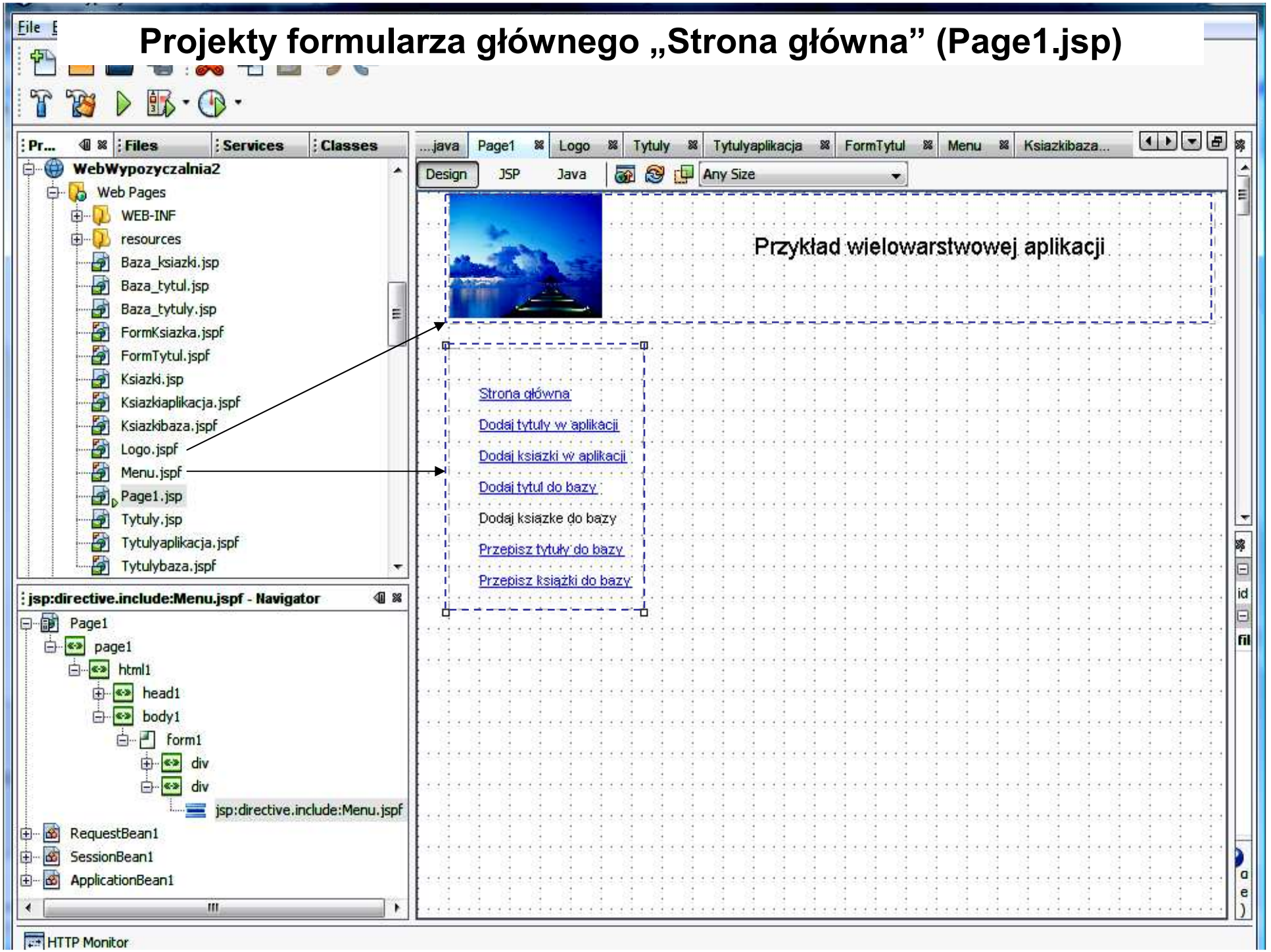

: ServiceFacade
: ApplicationService
: BusinessObject
: BusinessObject
: ApplicationService
: Service
: DataAccessObject
public void wywolanie( )
public void przetwarzanie( )
public void wywolanie( )
public void przetwarzanie( )
public void przetwarzanie( )
public void wywolanie( )
public void przetwarzanie( )
public void przetwarzanie( )
public void wywolanie( )
public void przetwarzanie( )
public void pobraniedanych( )
public void wywolaniedanych( )
public void przetwarzanie( )

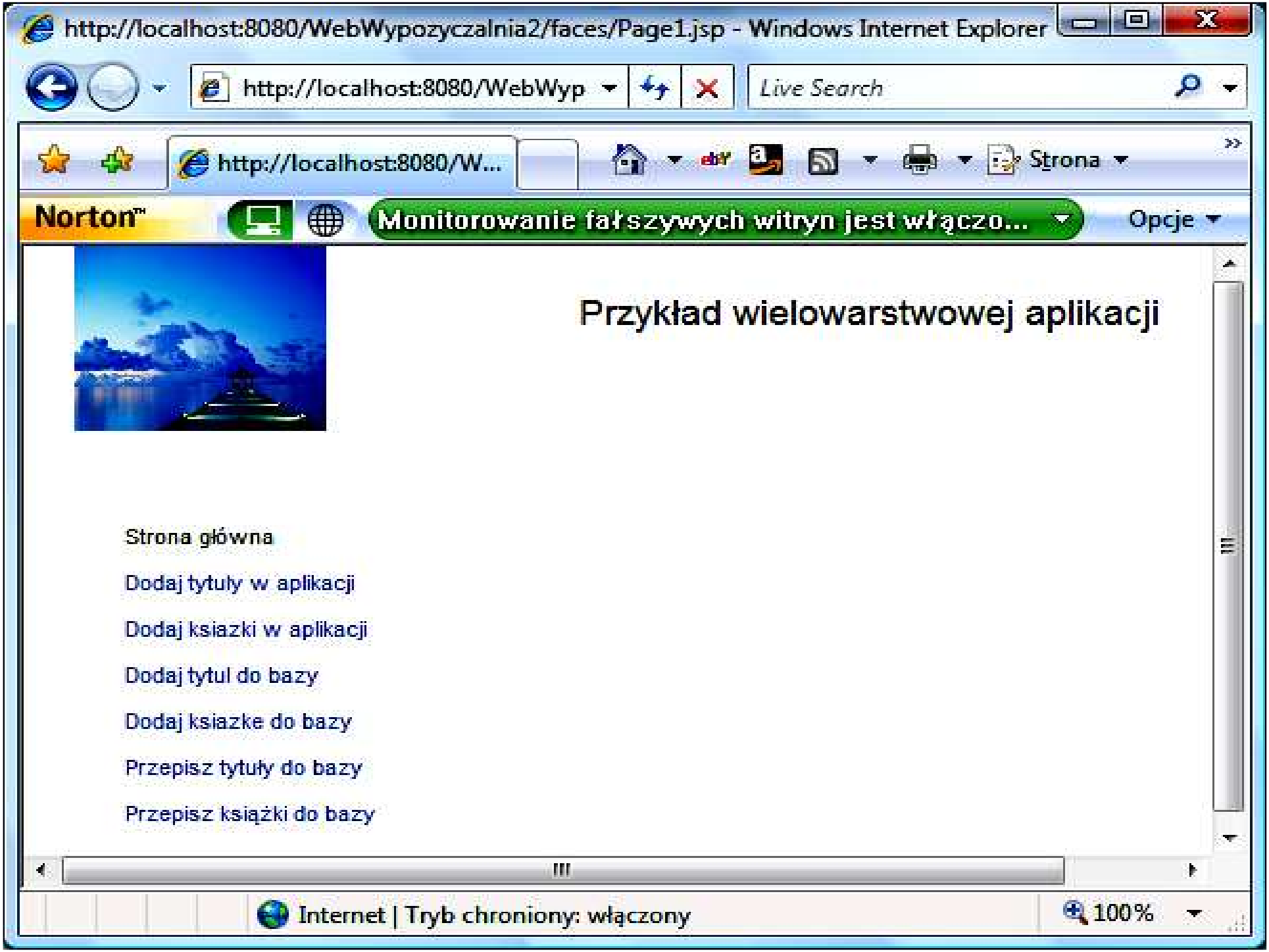
4. Architektura typu wielu klientów ze wspólną warstwą biznesową istniejącą podczas sesji i wspólną warstwą integrująca z bazą danych,

Przykład projektu składającego się z wielu formularzy opartych na fragmentach stron internetowych typu JSPF

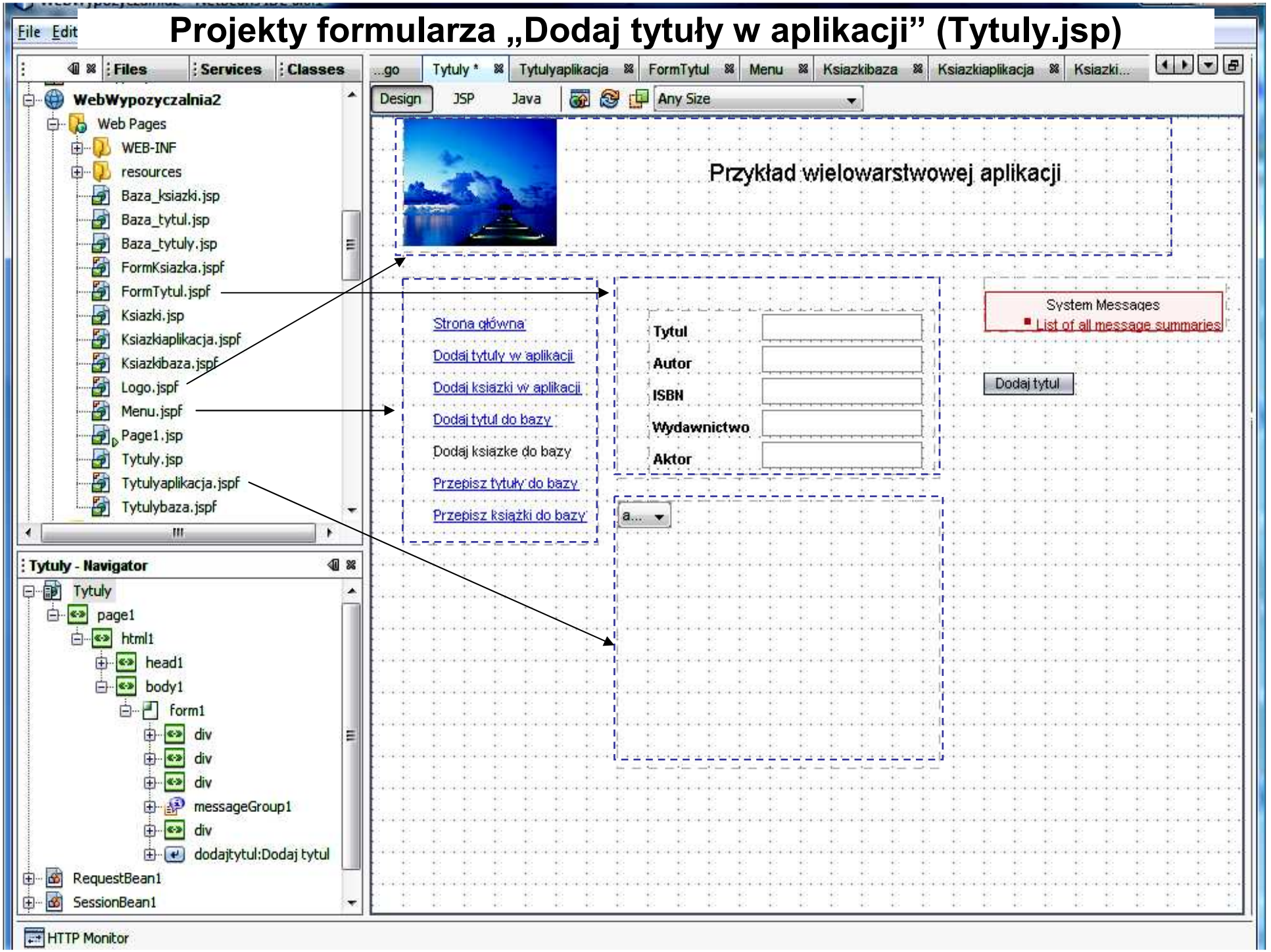
(wg wykładu 5)

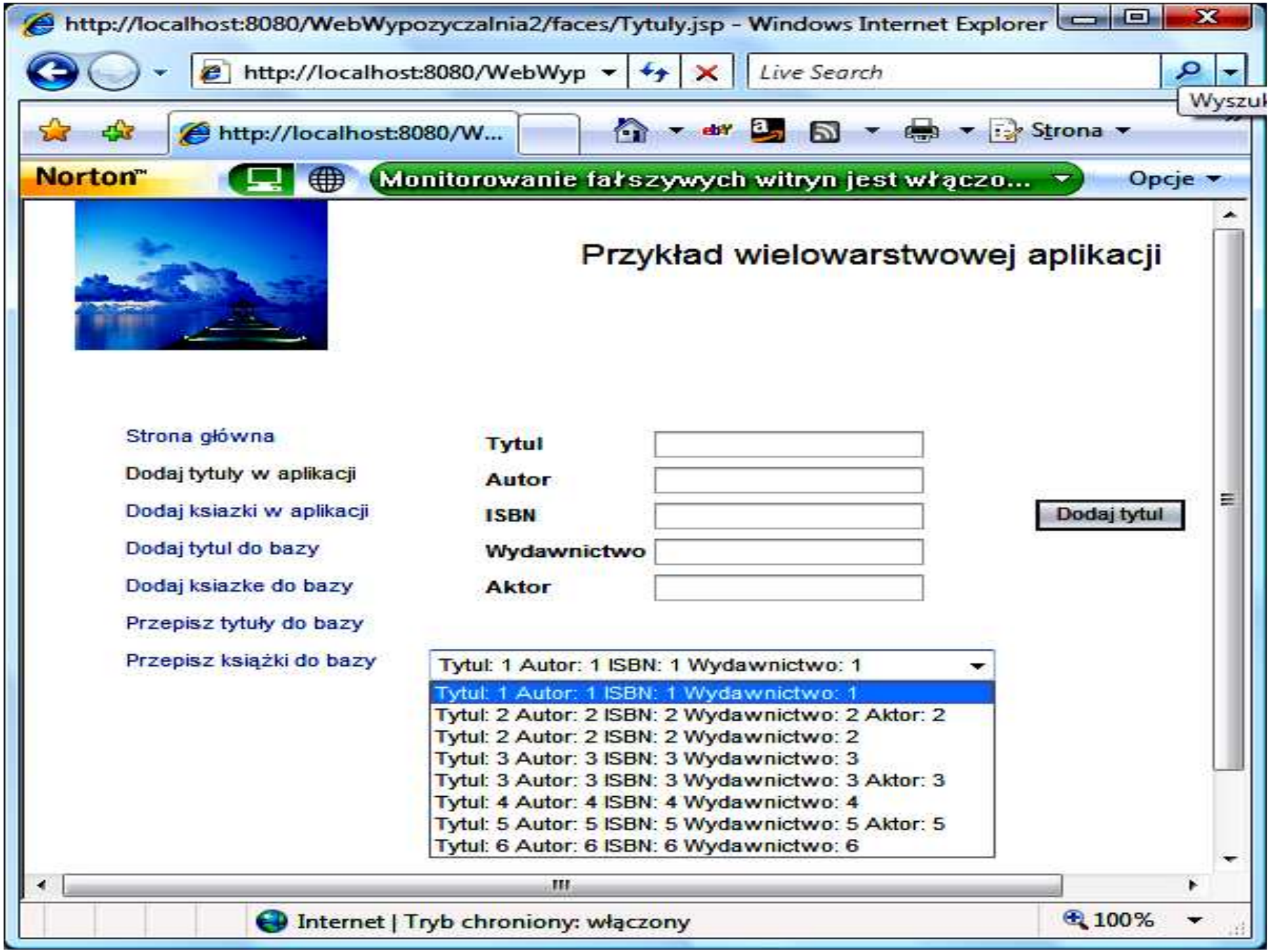
Projekty formularza głównego „Strona główna” (Page1.jsp)



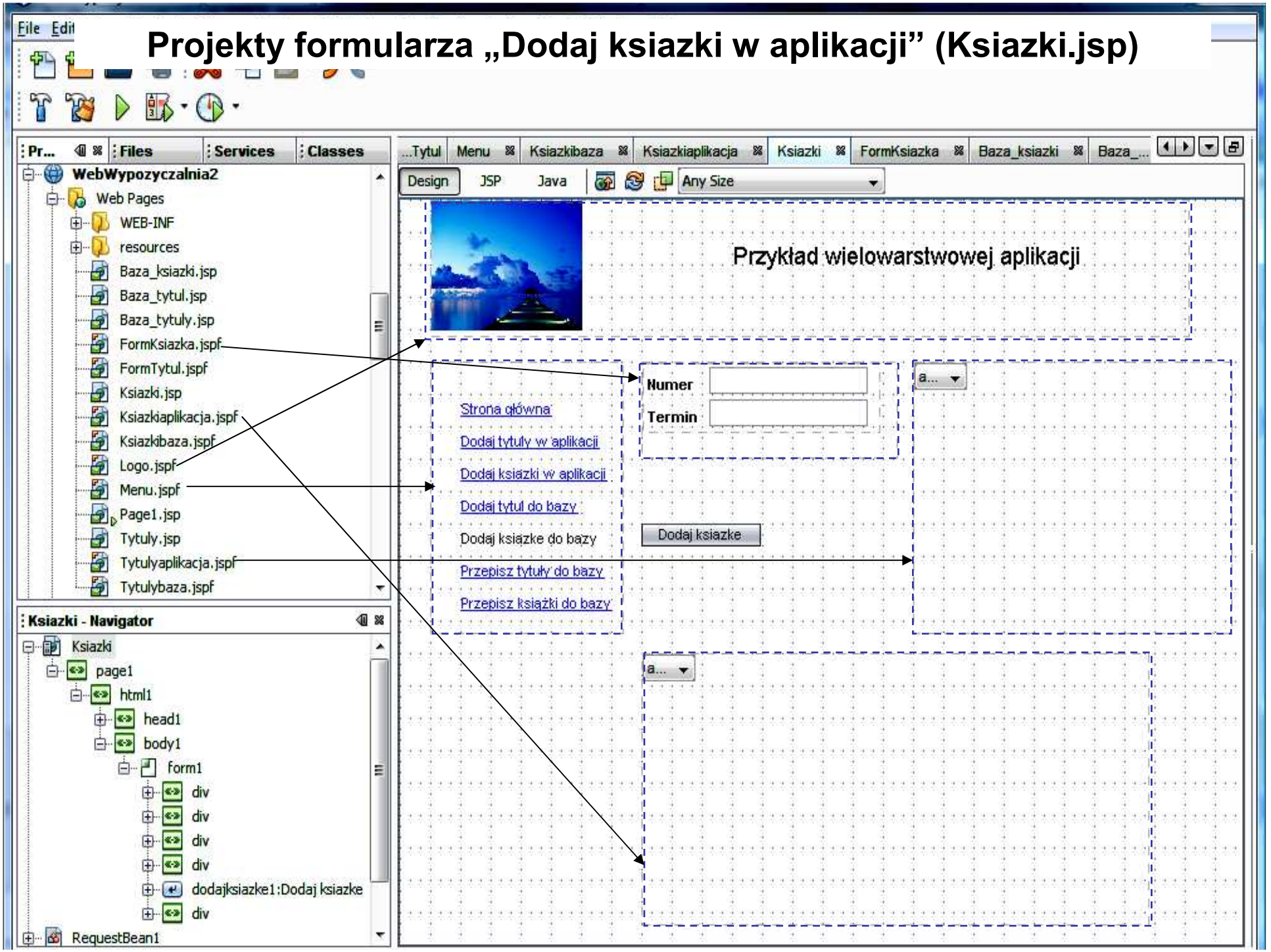

http://localhost:8080/WebWypozyczalnia2/faces/Page1.jsp - Windows Internet Explorer
http://localhost:8080/WebWyp
Live Search
http://localhost:8080/W...
Strona
Norton™
Monitorowanie fałszywych witryn jest włączo...
Opcje
Przykład wielowarstwowej aplikacji
Strona główna
Dodaj tytuły w aplikacji
Dodaj ksiazki w aplikacji
Dodaj tytul do bazy
Dodaj ksiazke do bazy
Przepisz tytuły do bazy
Przepisz książki do bazy
Internet | Tryb chroniony: włączony
100%

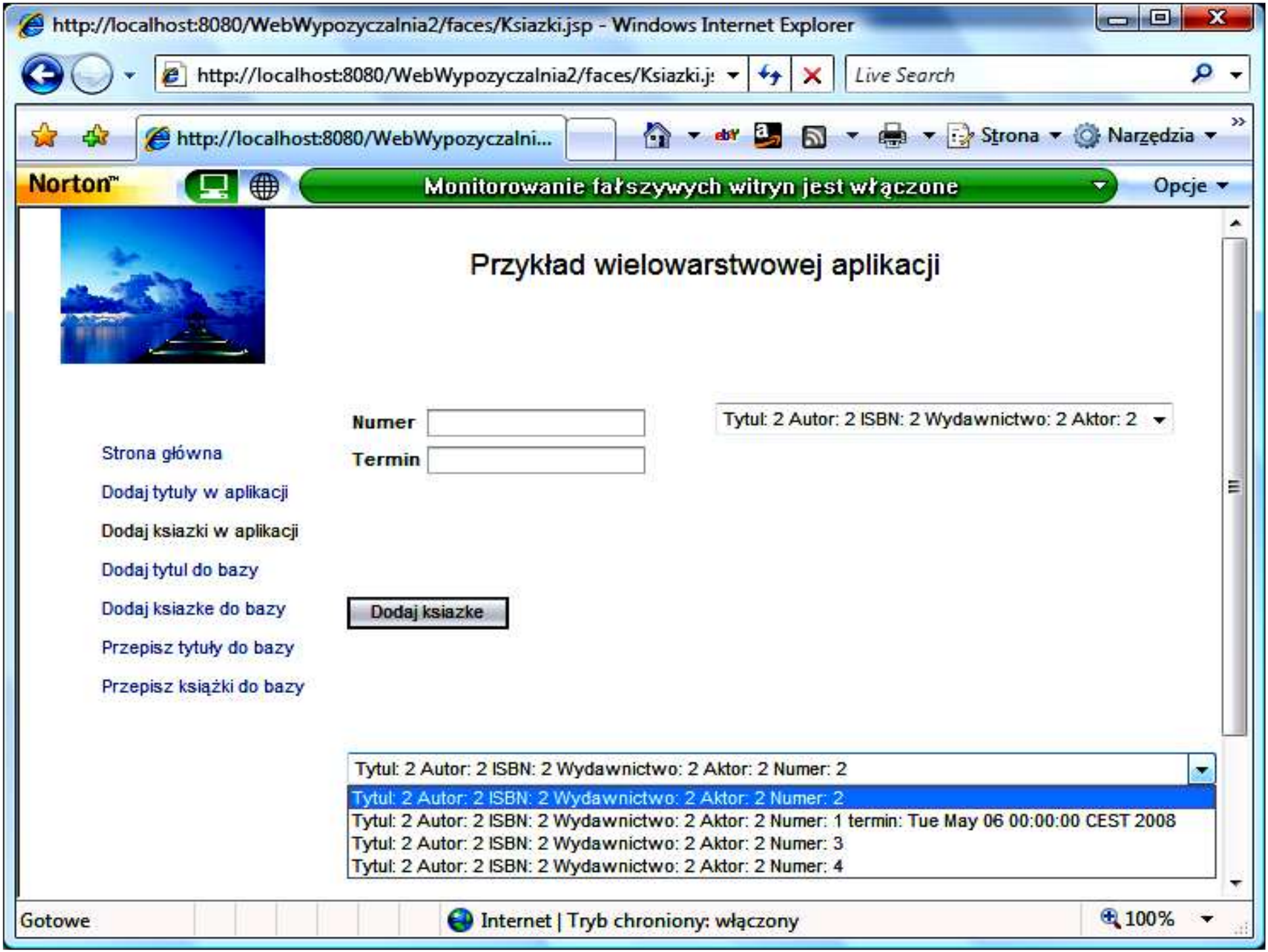
Projekty formularza „Dodaj tytuły w aplikacji” (Tytuly.jsp)



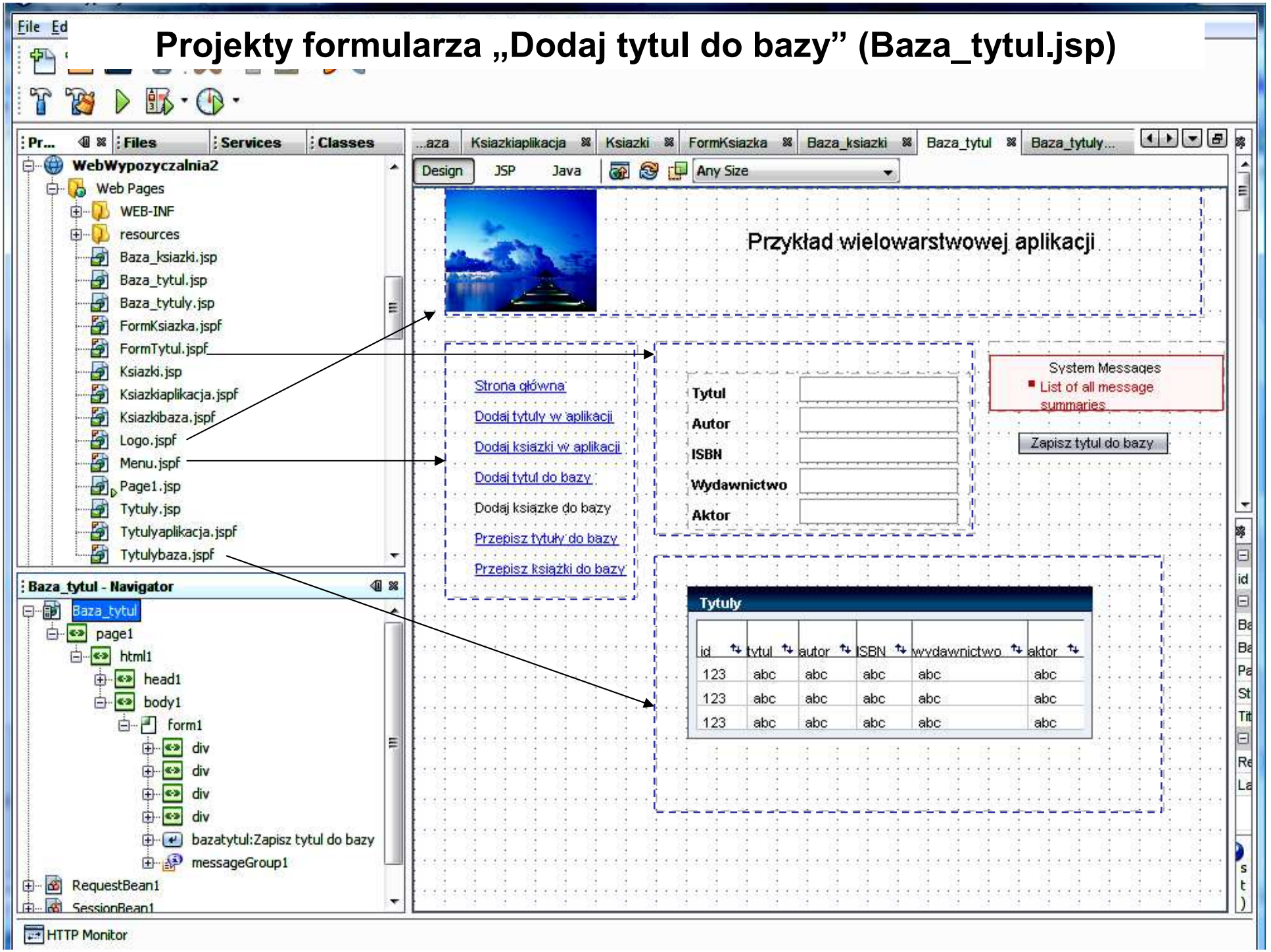

http://localhost:8080/WebWypozyczalnia2/faces/Tytuly.jsp - Windows Internet Explorer
http://localhost:8080/WebWyp
Live Search
http://localhost:8080/W...
Strona
Norton™
Monitorowanie fałszywych witryn jest włączo...
Opcje
Przykład wielowarstwowej aplikacji
Strona główna
Dodaj tytuły w aplikacji
Dodaj ksiazki w aplikacji
Dodaj tytul do bazy
Dodaj ksiazke do bazy
Przepisz tytuły do bazy
Przepisz książki do bazy
Tytul
Autor
ISBN
Wydawnictwo
Aktor
Dodaj tytul
Tytul: 1 Autor: 1 ISBN: 1 Wydawnictwo: 1
Tytul: 1 Autor: 1 ISBN: 1 Wydawnictwo: 1
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2
Tytul: 3 Autor: 3 ISBN: 3 Wydawnictwo: 3
Tytul: 3 Autor: 3 ISBN: 3 Wydawnictwo: 3 Aktor: 3
Tytul: 4 Autor: 4 ISBN: 4 Wydawnictwo: 4
Tytul: 5 Autor: 5 ISBN: 5 Wydawnictwo: 5 Aktor: 5
Tytul: 6 Autor: 6 ISBN: 6 Wydawnictwo: 6
Internet | Tryb chroniony: włączony
100%

Projekty formularza „Dodaj ksiazki w aplikacji” (Ksiazki.jsp)




http://localhost:8080/WebWypozyczalnia2/faces/Ksiazki.jsp - Windows Internet Explorer
http://localhost:8080/WebWypozyczalnia2/faces/Ksiazki.j:
Live Search
http://localhost:8080/WebWypozyczalni...
Strona
Narzędzia
Norton
Monitorowanie fałszywych witryn jest włączone
Opcje
Przykład wielowarstwowej aplikacji
Numer
Termin
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
Strona główna
Dodaj tytuły w aplikacji
Dodaj ksiazki w aplikacji
Dodaj tytul do bazy
Dodaj ksiazke do bazy
Dodaj ksiazke
Przepisz tytuły do bazy
Przepisz książki do bazy
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2 Numer: 2
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2 Numer: 2
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2 Numer: 1 termin: Tue May 06 00:00:00 CEST 2008
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2 Numer: 3
Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2 Numer: 4
Gotowe
Internet | Tryb chroniony: włączony
100%

Projekty formularza „Dodaj tytul do bazy” (Baza_tytul.jsp)



http://localhost:8080/WebWypozyczalnia2/faces/Baza_tytul.jsp - Windows Internet Explorer

http://localhost:8080/WebWypozyczaln

Live Search

http://localhost:8080/W...

Strona

Norton™ Monitorowanie fałszywych witryn jest włączone Opcje

Przykład wielowarstwowej aplikacji

- Strona główna
- Dodaj tytuły w aplikacji
- Dodaj ksiazki w aplikacji
- Dodaj tytul do bazy
- Dodaj ksiazke do bazy
- Przepisz tytuły do bazy
- Przepisz książki do bazy

Tytul

Autor

ISBN

Wydawnictwo

Aktor

Zapisz tytul do bazy

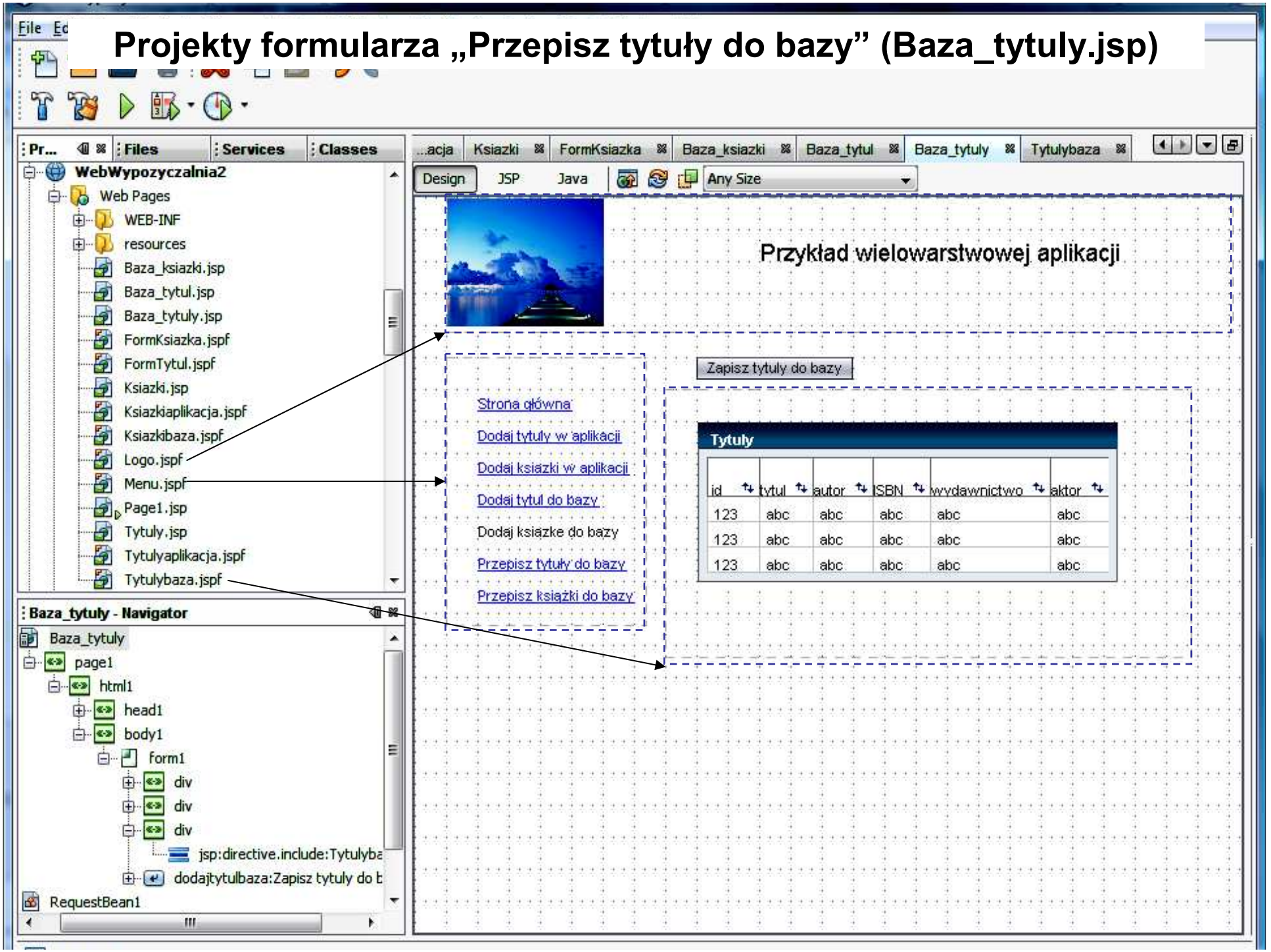
Tytuły

id	tytul	autor	ISBN	wydawnictwo	aktor
2	1	1	1	1	
4	2	2	2	2	2
152	2	2	2	2	
252	3	3	3	3	
352	3	3	3	3	3
353	4	4	4	4	
452	5	5	5	5	5
453	6	6	6	6	

Internet | Tryb chroniony: włączony

100%

Projekty formularza „Przepisz tytuły do bazy” (Baza_tytuly.jsp)



http://localhost:8080/WebWypozyczalnia2/faces/Baza_tytuly.jsp - Windows Internet Exp...

http://localhost:8080/WebWypo

Live Search

http://localhost:8080/W...

Strona

Norton™ Monitorowanie fałszywych witryn jest włączone Opcje

Przykład wielowarstwowej aplikacji

Zapisz tytuły do bazy

Strona główna

Dodaj tytuły w aplikacji

Dodaj ksiazki w aplikacji

Dodaj tytul do bazy

Dodaj ksiazke do bazy

Przepisz tytuły do bazy

Przepisz książki do bazy

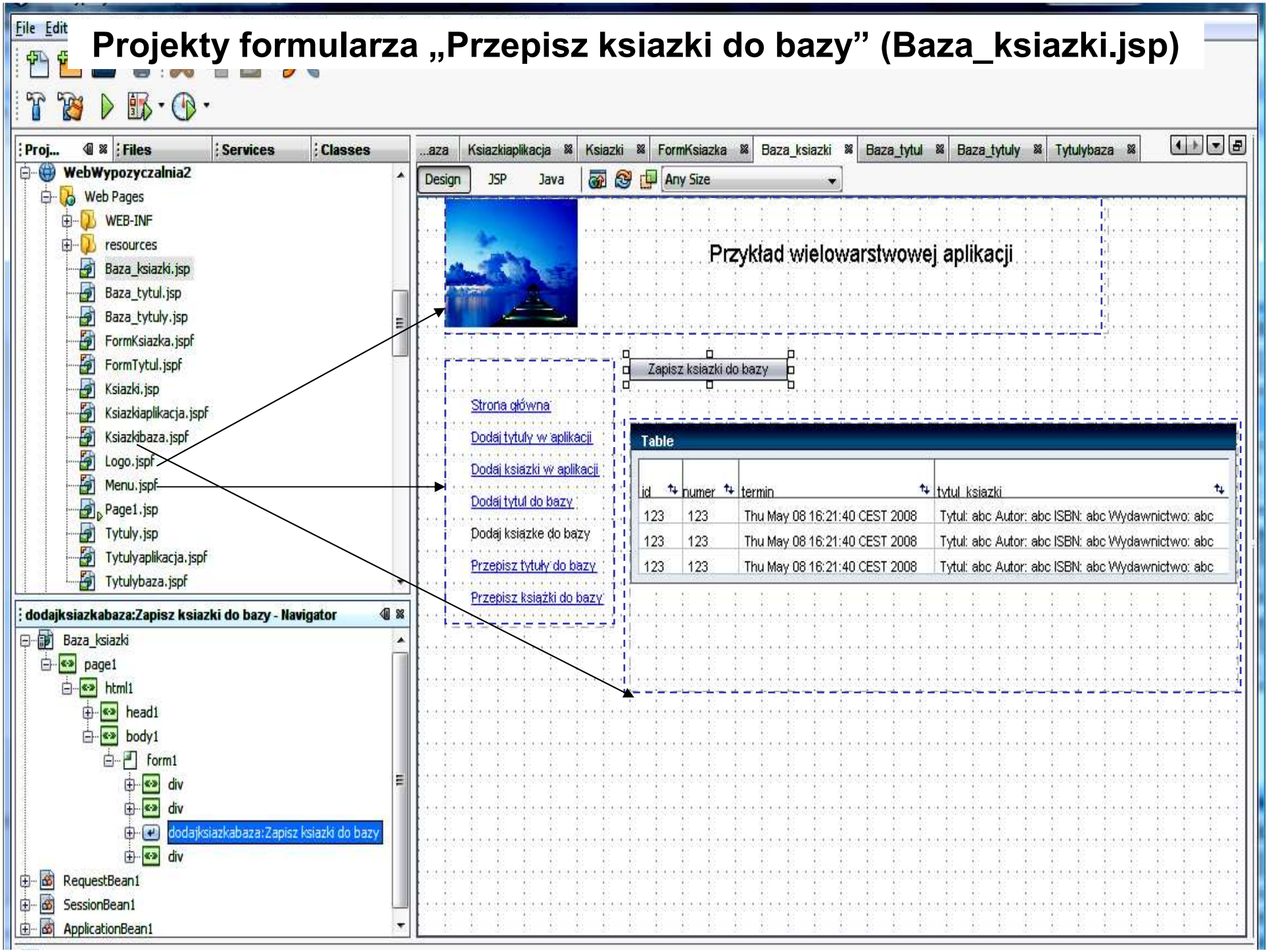
Tytuły

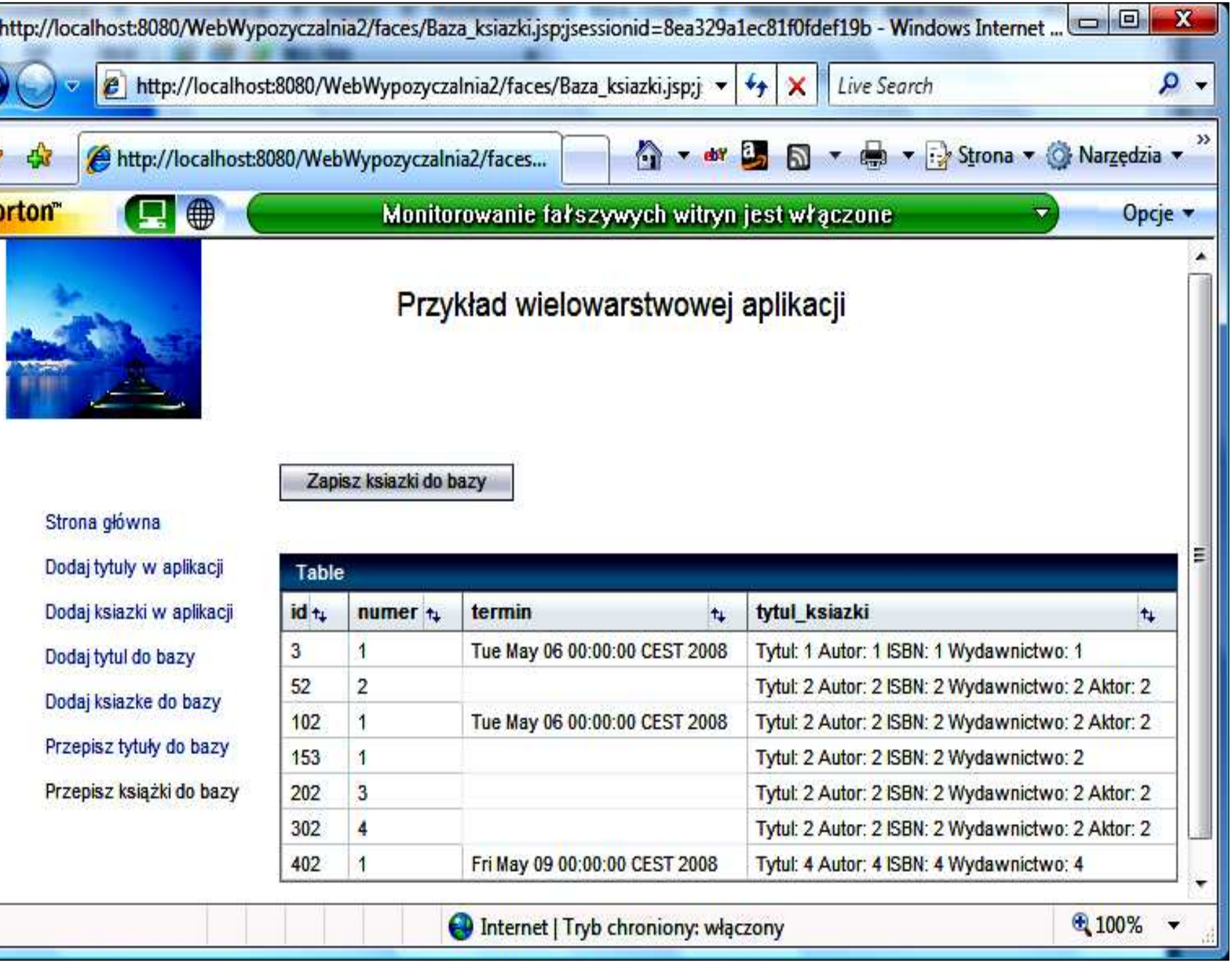
id	tytul	autor	ISBN	wydawnictwo	aktor
2	1	1	1	1	
4	2	2	2	2	2
152	2	2	2	2	
252	3	3	3	3	
352	3	3	3	3	3
353	4	4	4	4	
452	5	5	5	5	5
453	6	6	6	6	

Internet | Tryb chroniony: włączony

100%

Projekty formularza „Przepisz ksiazki do bazy” (Baza_ksiazki.jsp)



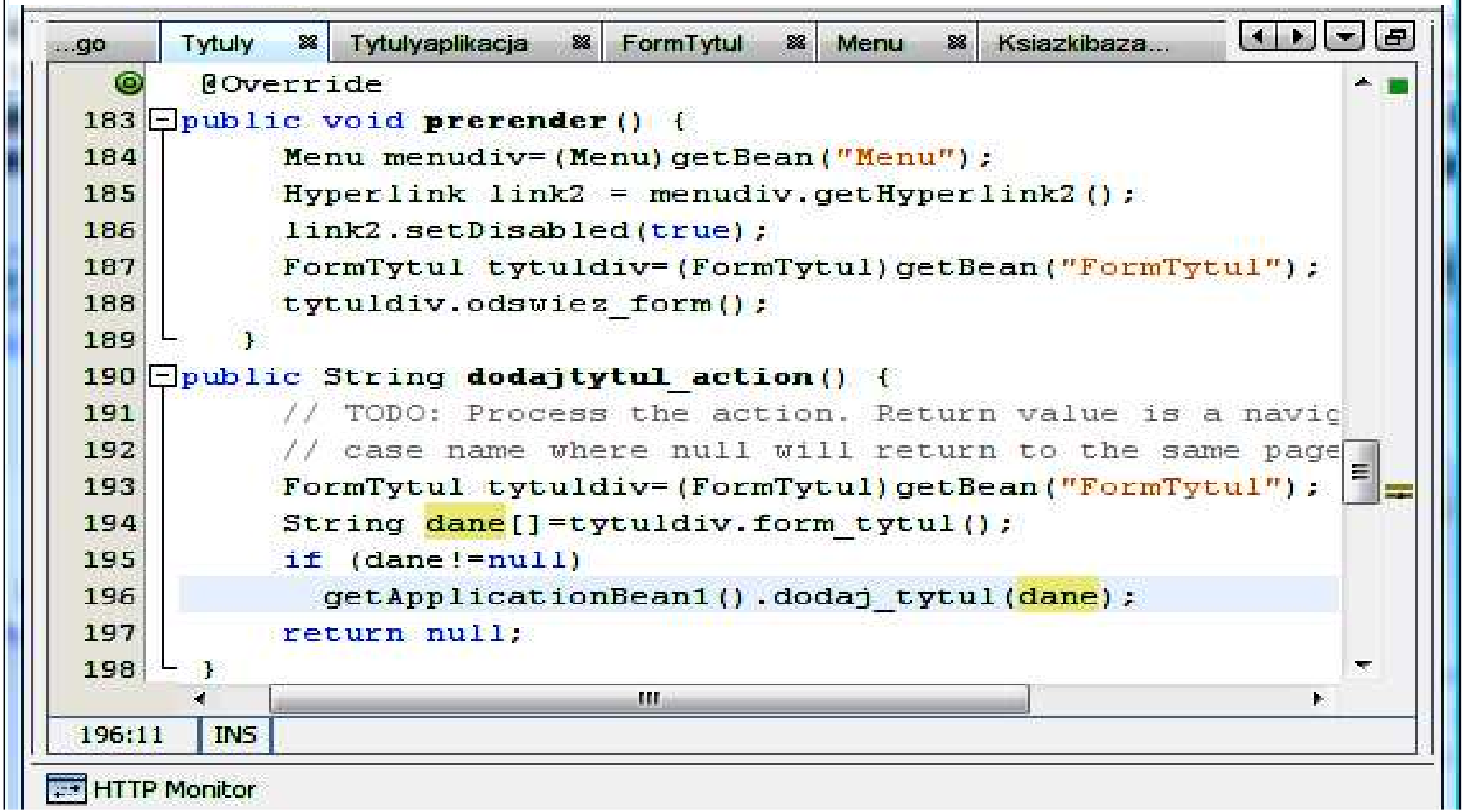

http://localhost:8080/WebWypozyczalnia2/faces/Baza_ksiazki.jsp;jsessionid=8ea329a1ec81f0fdef19b - Windows Internet ...
http://localhost:8080/WebWypozyczalnia2/faces/Baza_ksiazki.jsp;j
Live Search
http://localhost:8080/WebWypozyczalnia2/faces...
Strona
Narzędzia
Norton™
Monitorowanie fałszywych witryn jest włączone
Opcje
Przykład wielowarstwowej aplikacji
Zapisz ksiazki do bazy
Strona główna
Dodaj tytuły w aplikacji
Dodaj ksiazki w aplikacji
Dodaj tytul do bazy
Dodaj ksiazke do bazy
Przepisz tytuły do bazy
Przepisz książki do bazy
Table
id
numer
termin
tytul_ksiazki
3 1 Tue May 06 00:00:00 CEST 2008 Tytul: 1 Autor: 1 ISBN: 1 Wydawnictwo: 1
52 2 Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
102 1 Tue May 06 00:00:00 CEST 2008 Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
153 1 Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2
202 3 Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
302 4 Tytul: 2 Autor: 2 ISBN: 2 Wydawnictwo: 2 Aktor: 2
402 1 Fri May 09 00:00:00 CEST 2008 Tytul: 4 Autor: 4 ISBN: 4 Wydawnictwo: 4
Internet | Tryb chroniony: włączony
100%

5. Oprogramowanie systemu typu wielu klientów ze wspólną warstwą biznesową istniejącą podczas sesji i wspólną warstwą integrująca z bazą danych,

Przykład projektu składającego się z wielu formularzy opartych na fragmentach stron internetowych typu JSPF

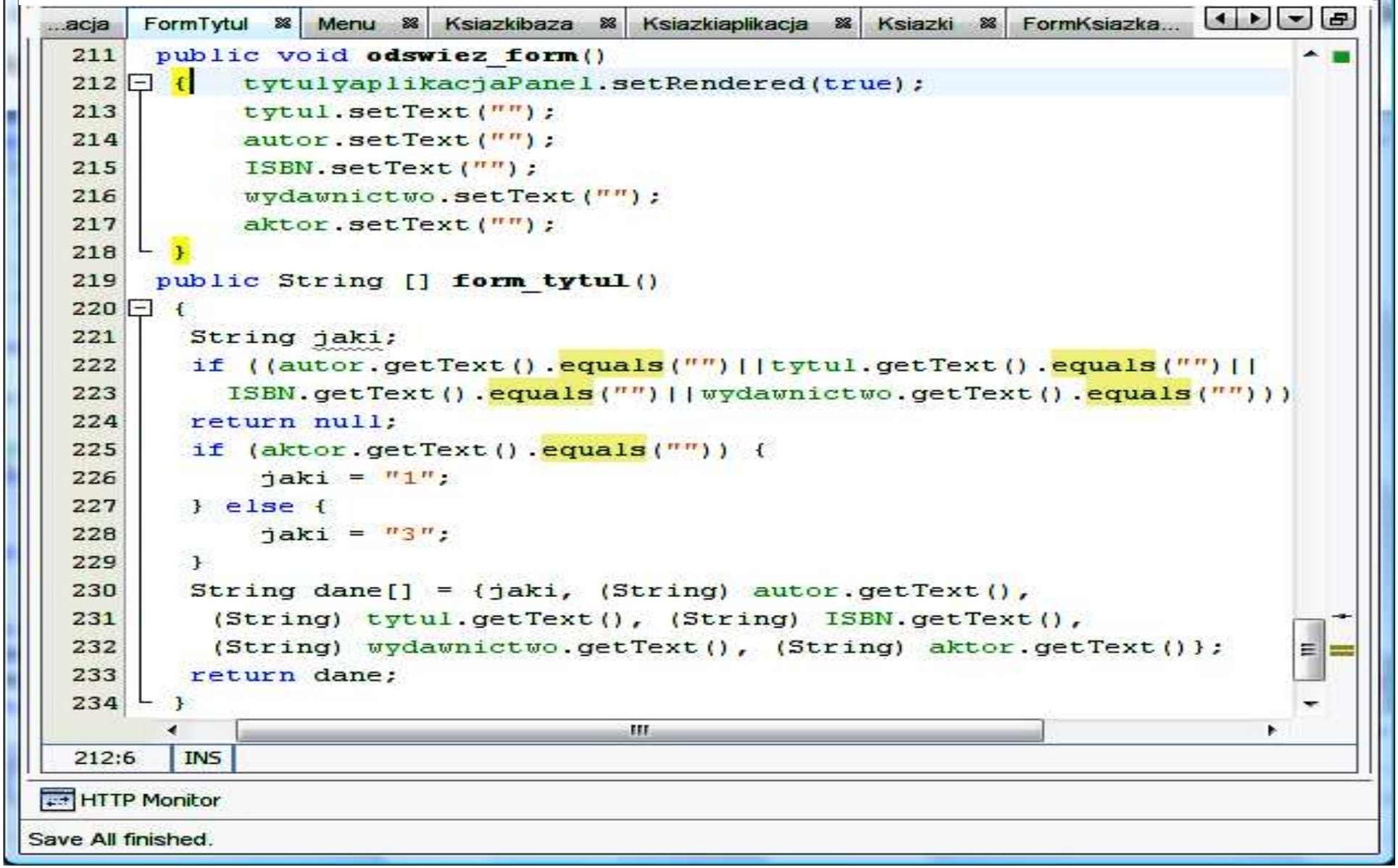
5.1. Oprogramowanie dotyczące formularza Tytuly.jsp

Definicje metod w klasie Tytuly dla strony typu JSP – do wstawiania nowego tytulu do warstwy biznesowej (obsługa zdarzenia dodajtytul_action) oraz generowania widoku w fazie Response przetwarzania strony metodą prerender (wygaszanie linku do bieżącej strony w formularzu Menu typu JSPF i czyszczenie pól formularza FormTytul typu JSPF jego metodą odswiez_form)

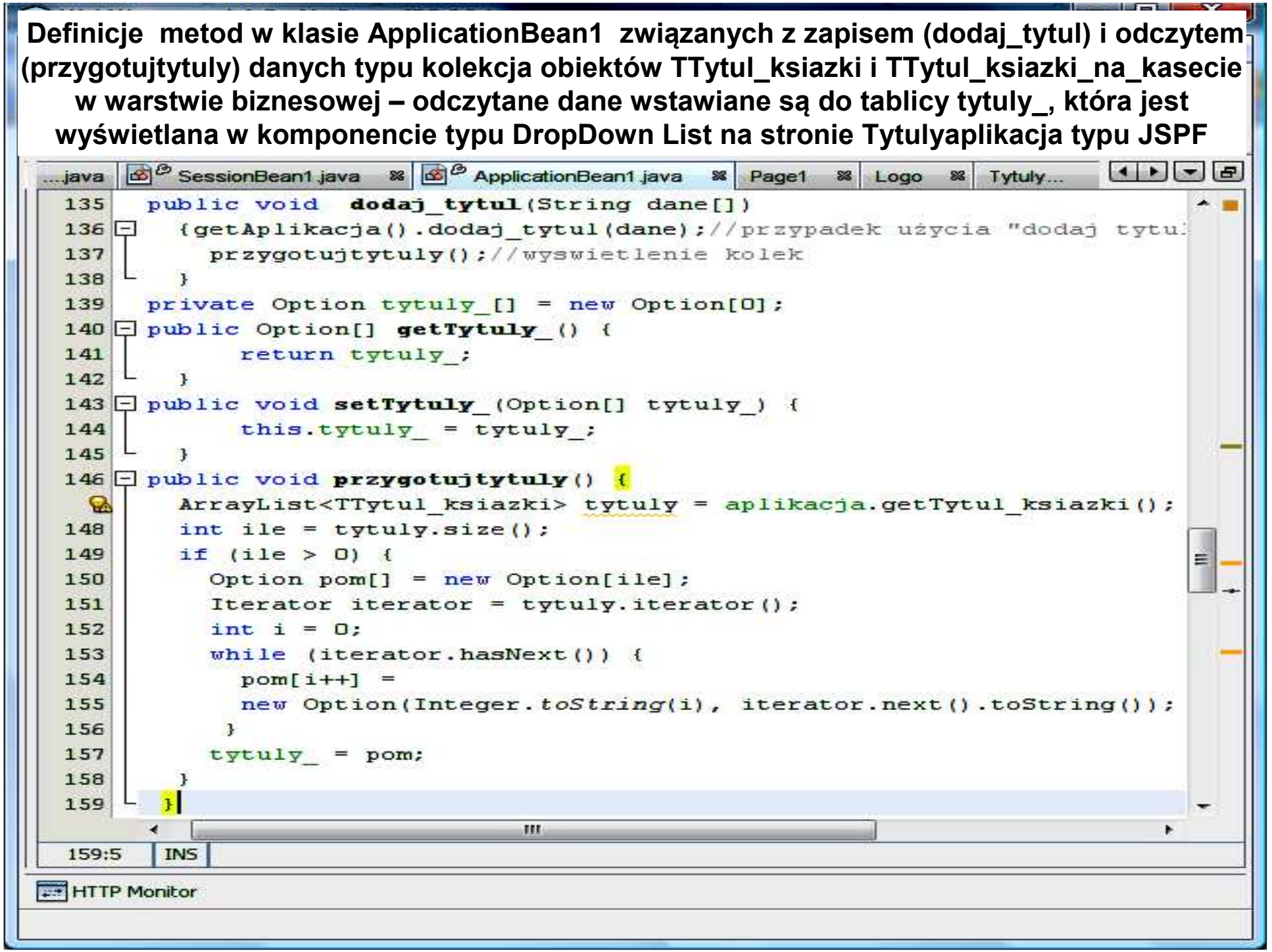


Definicje metod w klasie FormTytul typu BackingBean dla strony typu JSPF –

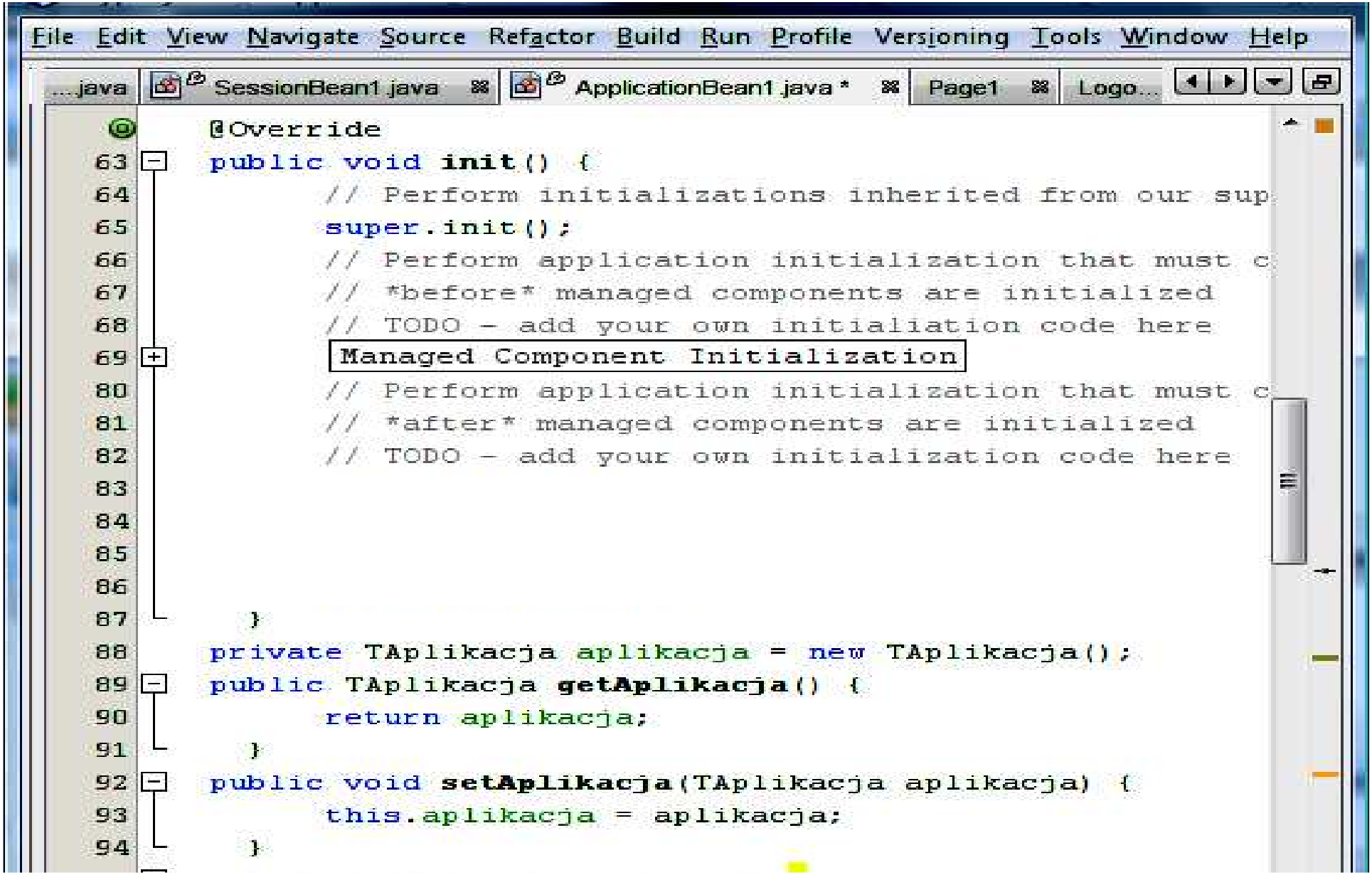
- **do pobierana danych o nowym tytule (form_tytul): dane dla wstawianych tytułów książek**
- **do czyszczenia pól formularza (odswiez_form)**



Definicje metod w klasie ApplicationBean1 związanych z zapisem (dodaj_tytul) i odczytem (przygotujtytuly) danych typu kolekcja obiektów TTytul_ksiazki i TTytul_ksiazki_na_kasecie w warstwie biznesowej – odczytane dane wstawiane są do tablicy tytuly_, która jest wyświetlana w komponencie typu DropDown List na stronie Tytulyaplikacja typu JSPF

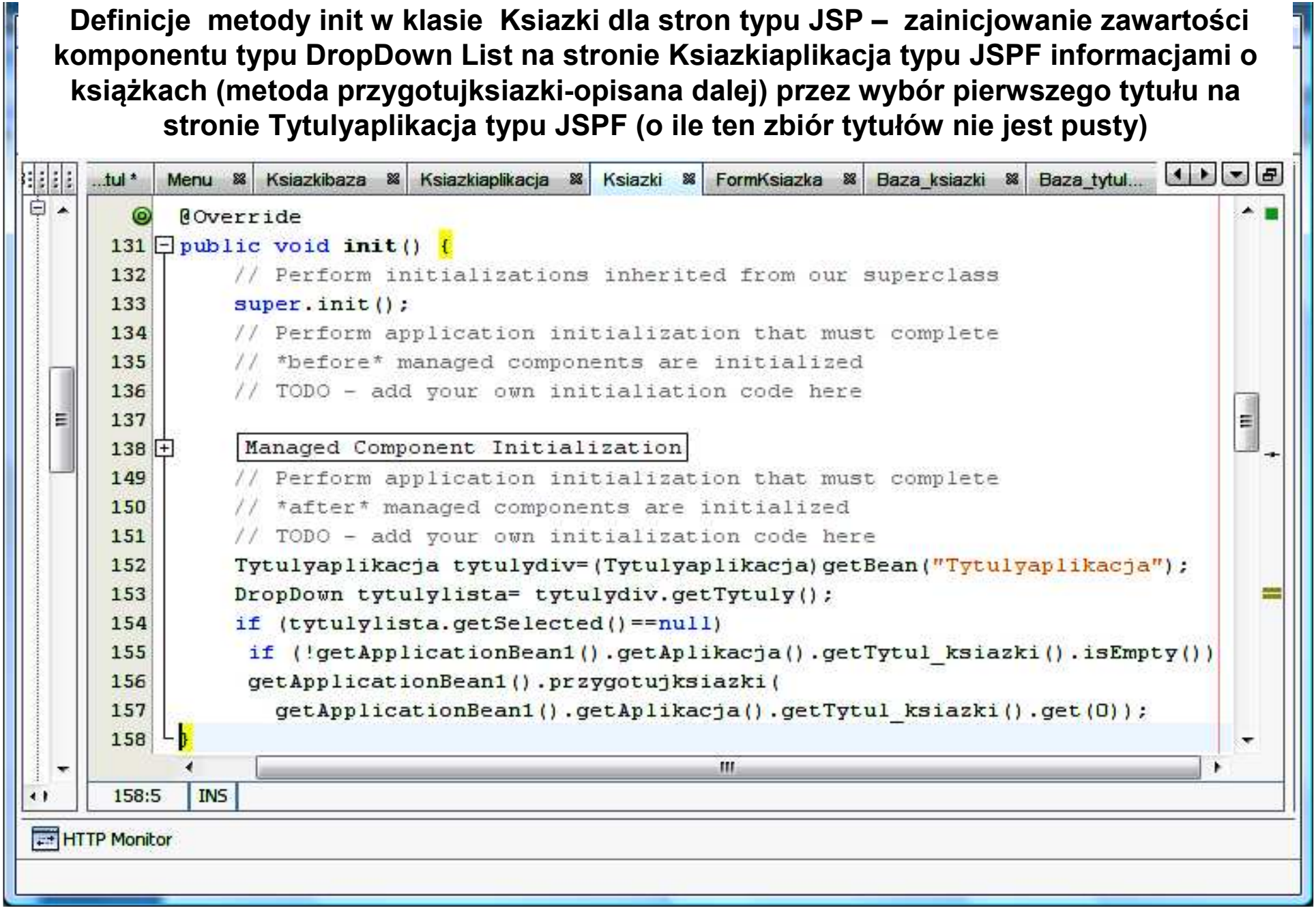


Utworzenie warstwy biznesowej oraz obiektu typu TAplikacja, który jest fasadą warstwy biznesowej w postaci zwykłego obiektu Javy

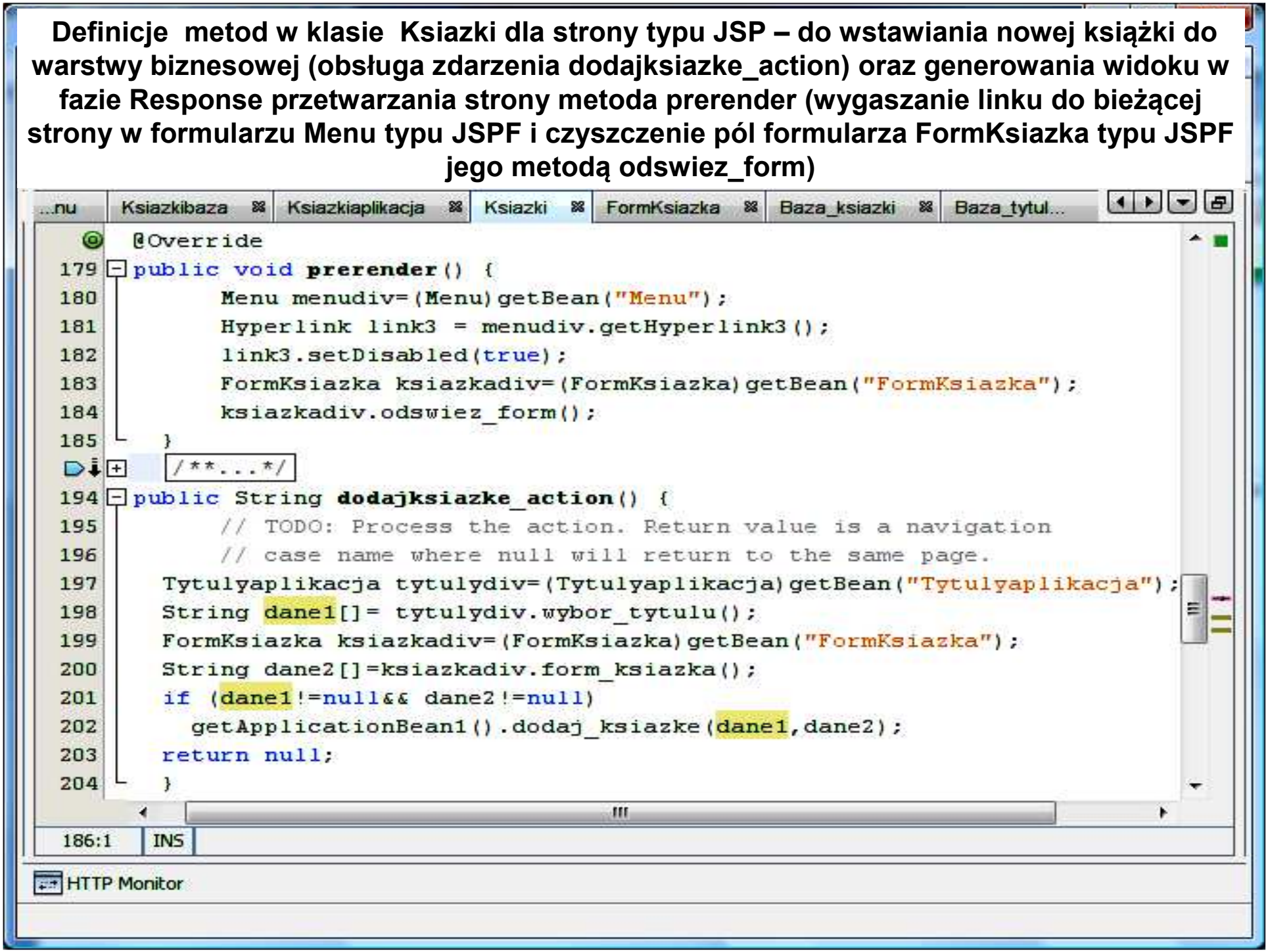


5.2. Oprogramowanie dotyczące formularza Ksiazki.jsp

Definicje metody init w klasie Ksiazki dla stron typu JSP – zainicjowanie zawartości komponentu typu DropDown List na stronie Ksiazkiaplikacja typu JSPF informacjami o książkach (metoda przygotujksiazki-opisana dalej) przez wybór pierwszego tytułu na stronie Tytulyaplikacja typu JSPF (o ile ten zbiór tytułów nie jest pusty)



Definicje metod w klasie Ksiazki dla strony typu JSP – do wstawiania nowej książki do warstwy biznesowej (obsługa zdarzenia dodajksiazke_action) oraz generowania widoku w fazie Response przetwarzania strony metoda prerender (wygaszanie linku do bieżącej strony w formularzu Menu typu JSPF i czyszczenie pól formularza FormKsiazka typu JSPF jego metodą odswiez_form)



Definicje metod w klasie FormKsiazka typu BackingBean dla stron typu JSPF – do pobierana danych o nowej książce (form_ksiazka): dane1 dla wyszukania tytułów wstawianych książek (metoda wybor_tytulu z ze strony Tytulyaplikacja typu JSPF) i dane2 z danymi wstawianej książki; oraz do czyszczenia pól formularza (odswiez_form)

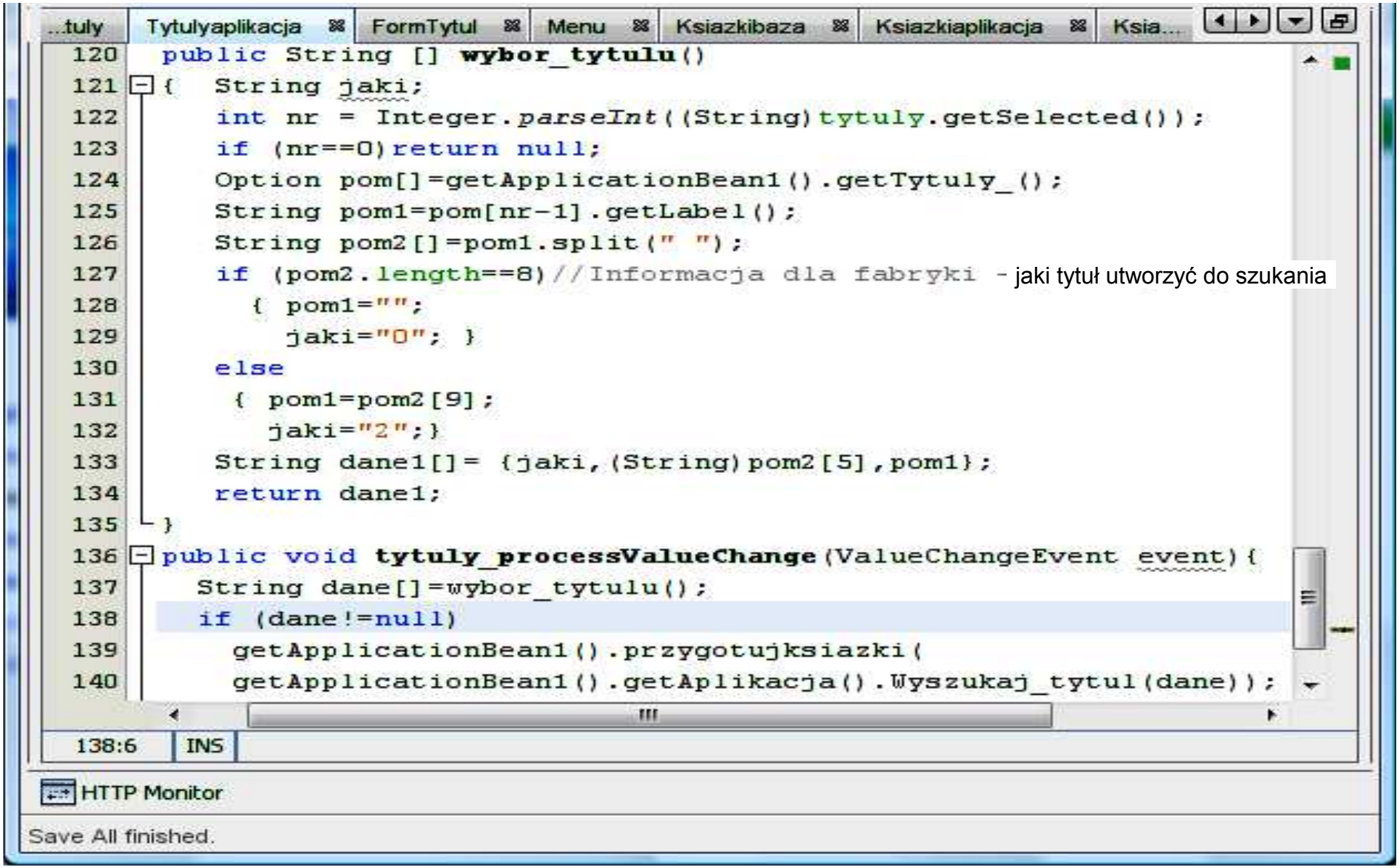
...ibaza | Ksiazkiaplikacja | Ksiazki | FormKsiazka | Baza_ksiazki | Baza_tytul | Baza_tytuly...

```
158   public void odswiez_form()
159   {
160       ksiazkiaplikacjaPanel.setRendered(true);
161       numer.setText("");
162       termin.setText("");
163   }
164   public String[] form_ksiazka() {
165       // TODO: Process the action. Return value is a navigation
166       // case name where null will return to the same page.
167      String jaka;
168      if (numer.getText().equals(""))
169          return null;
170       if (termin.getText().equals(""))
171        { jaka = "0"; }
172       else
173        { jaka = "1"; }//Informacja dla fabryki - jaki egzemplarz utworzyć c
174      String dane[]= {jaka,(String)numer.getText(),(String)termin.getText()};
175      return dane;
176   }
```

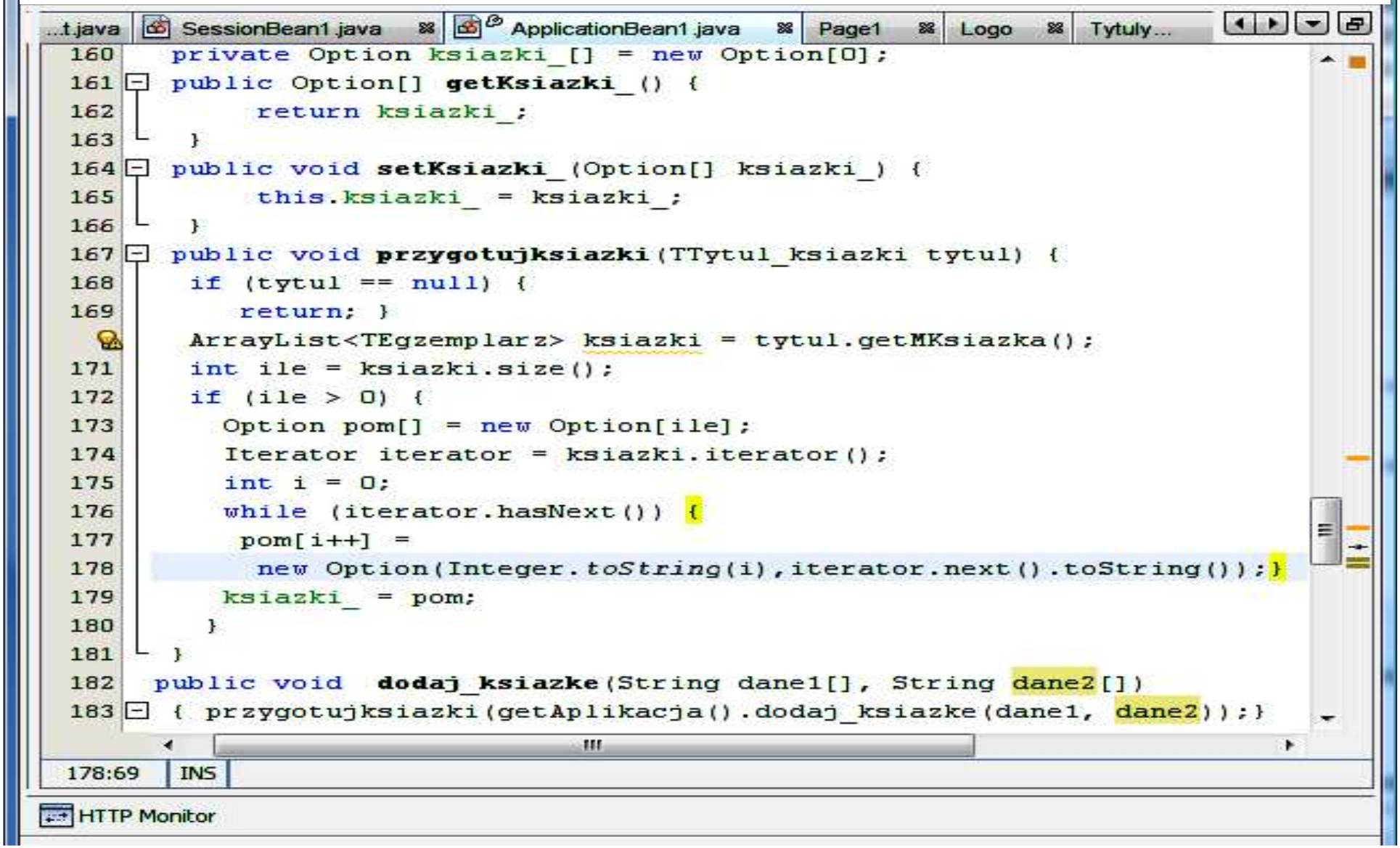
168:39 | INS

HTTP Monitor

Definicje metod w klasie Tytulyaplikacja typu BackingBean dla stron typu JSPF – do pobierana danych o tytule dane1 do wyszukiwania (wybor_tytulu): ISBN lub ISBN i nazwisko aktora; oraz do zmiany zawartości komponentu typu DropDown List na stronie Ksiazkiaplikacja typu JSPF (metodą przygotujksiazki) za pomocą obsługi zdarzenia wyboru pozycji z listy metodą tytuly_processValueChange

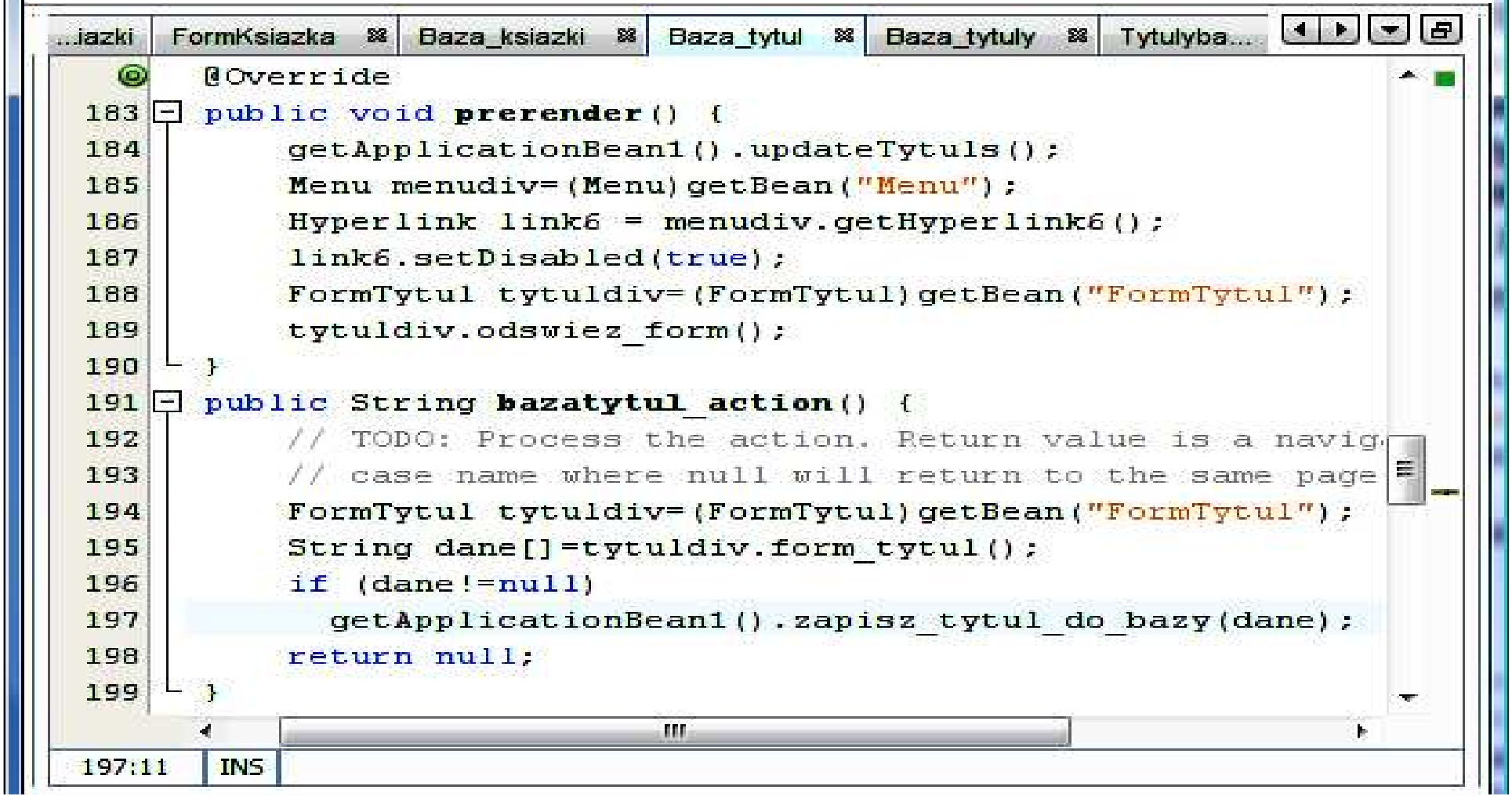


Definicje metod w klasie ApplicationBean1 związanych z zapisem (dodaj_ksiazke) i odczytem (przygotujksiazki) danych typu kolekcja obiektów TEgzemplarz i TEgzemplarz_termin w warstwie biznesowej – odczytane dane wstawiane są do tablicy ksiazki_, która jest wyświetlana w komponencie typu DropDown List na stronie Ksiazkiaplikacja typu JSPF

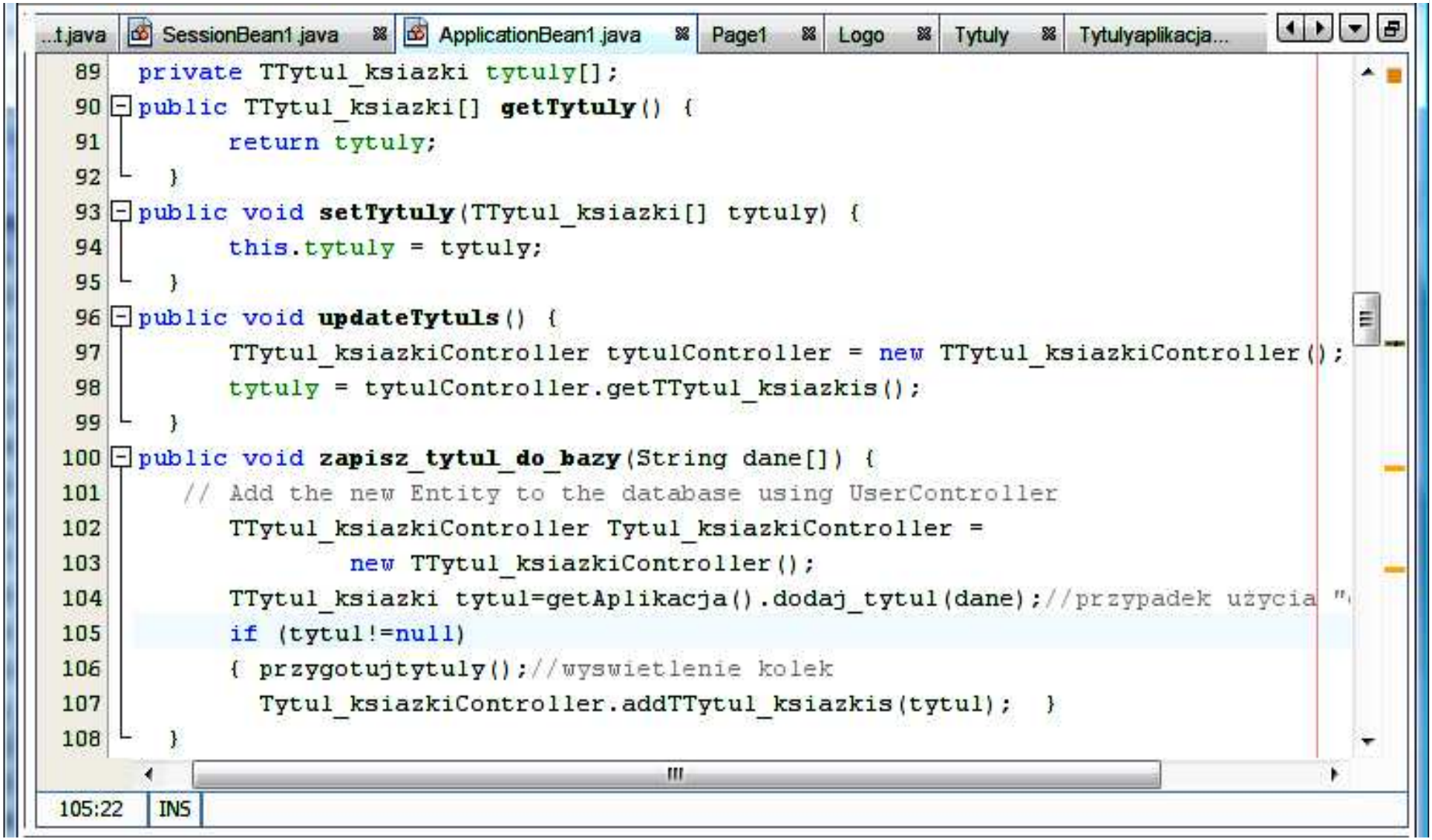


5.3. Oprogramowanie dotyczące formularza „ Dodaj tytul do bazy”

Definicje metod w klasie Baza_tytul dla strony typu JSP – do zapisu tytułów z warstwy biznesowej (obsługa zdarzenia bazatytul_action) oraz generowania widoku w fazie Response przetwarzania strony metoda prerender (wygaszanie linku do bieżącej strony w formularzu Menu typu JSPF, aktualizacja tablicy tytuly metodą updateTytuls w klasie ApplicationBean1, wyświetlanej w komponencie Table strony Tytulybaza typu JSPF, czyszczenie pól formularza FormTytul typu JSPF jego metodą odswiez_form)

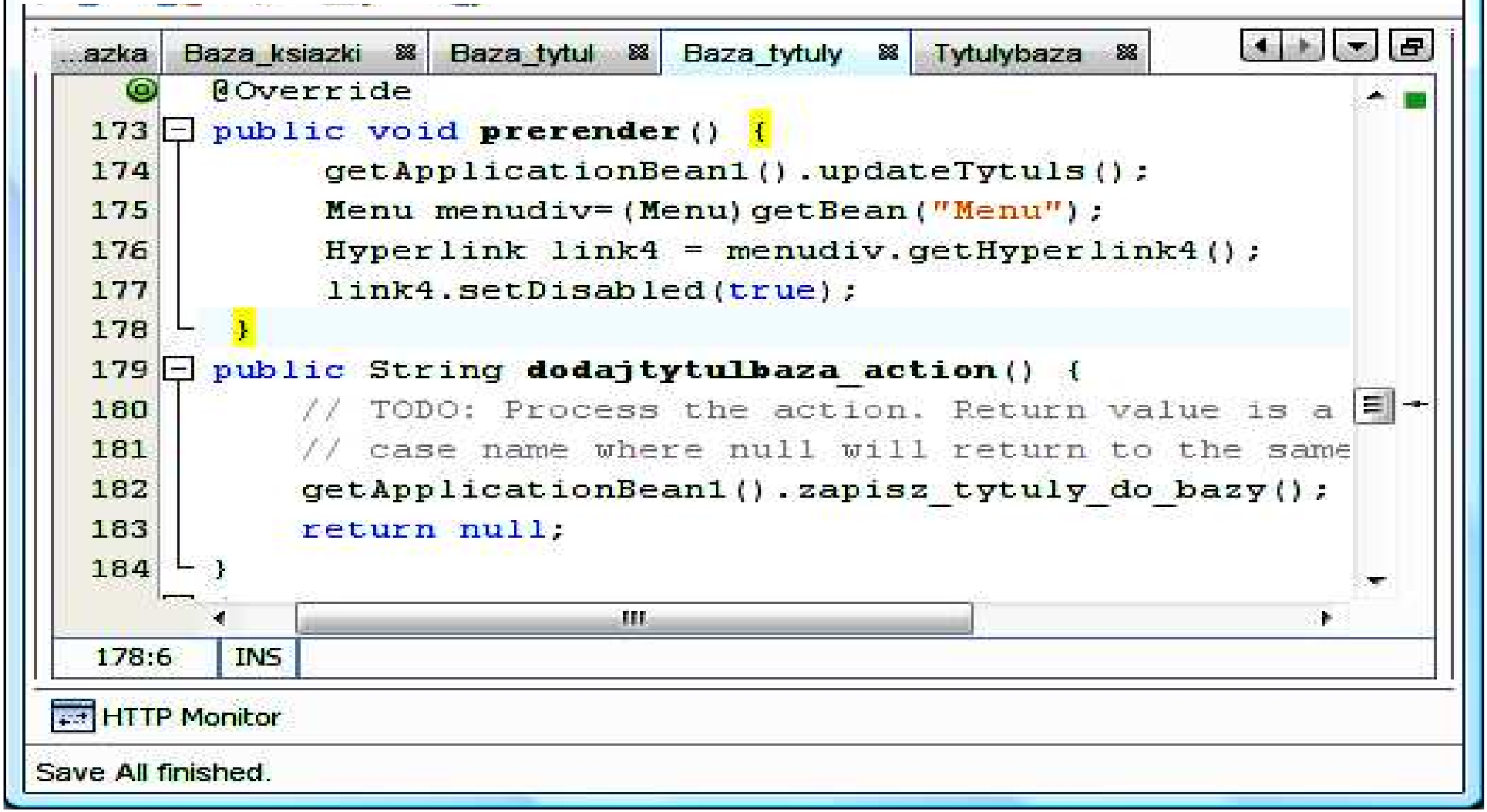


Definicje metod w klasie ApplicationBean1: odczytującą dane typu kolekcja obiektów TTytul_ksiazki i TTytul_ksiazki_na_kasecie z bazy danych (updateTytuls) oraz metoda zapisz_tytul_do_bazy zapisująca do bazy danych (addTTytul_ksiazkis) i warstwy biznesowej (dodaj_tytul) pojedynczych danych typu TTytul_ksiazki i TTytul_ksiazki_na_kasecie oraz odświeżająca zawartość komponentu typu DropDownList na stronie Tytulyaplikacja typu JSPF (przygotujtytuly)

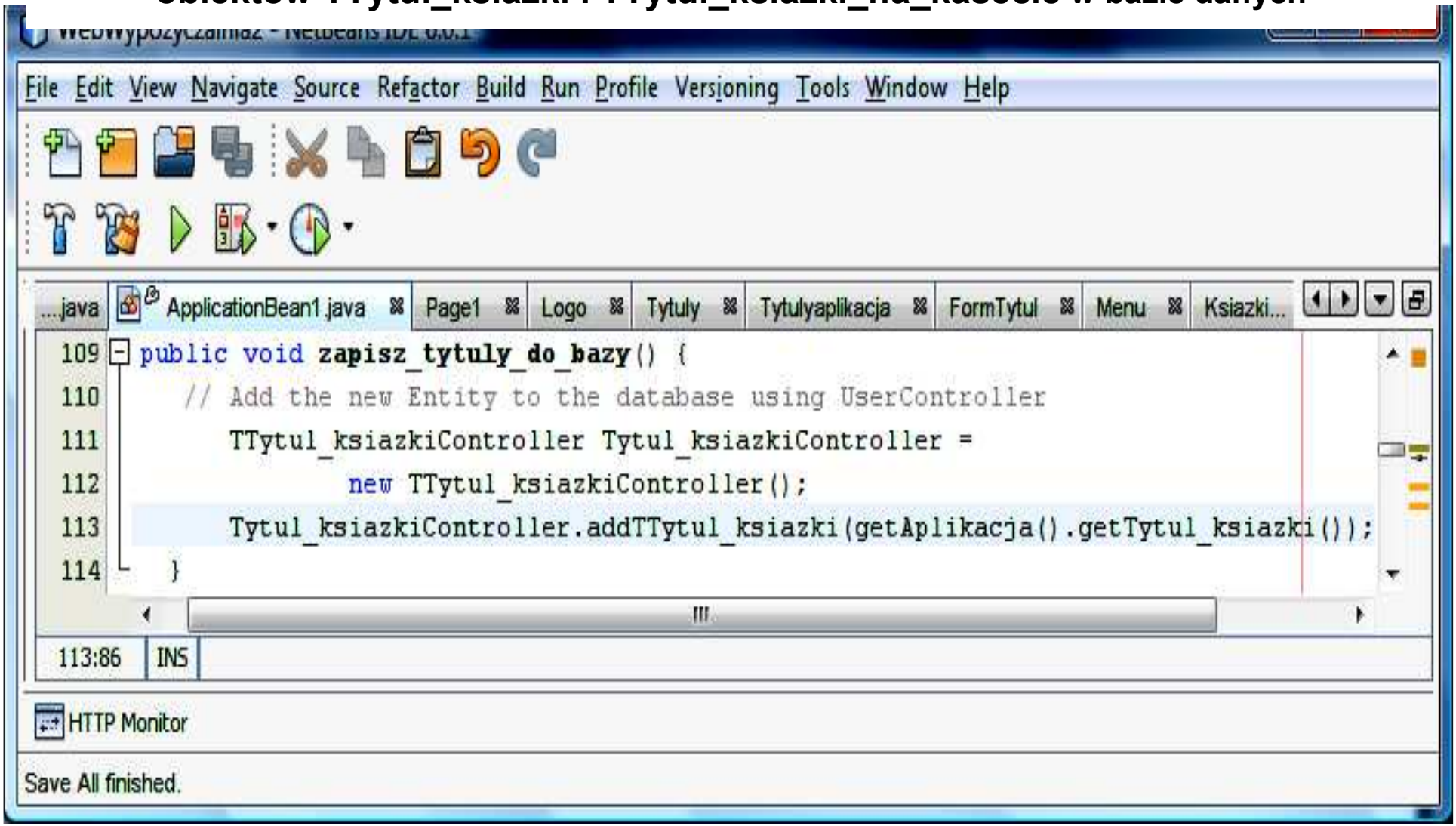


5.4. Oprogramowanie dotyczące formularza Baza_tytuly.jsp

Definicje metod w klasie Baza_tytuly dla strony typu JSP – do zapisu tytułów z warstwy biznesowej (obsługa zdarzenia dodajtytulbaza_action) oraz generowania widoku w fazie Response przetwarzania strony metoda prerender (wygaszanie linku do bieżącej strony w fdormularzu Menu typu JSPF i aktualizacja tablicy tytuly metodą updateTytuls w klasie ApplicationBean1 wyświetlanej w komponencie Table strony Tytulybaza typu JSPF)

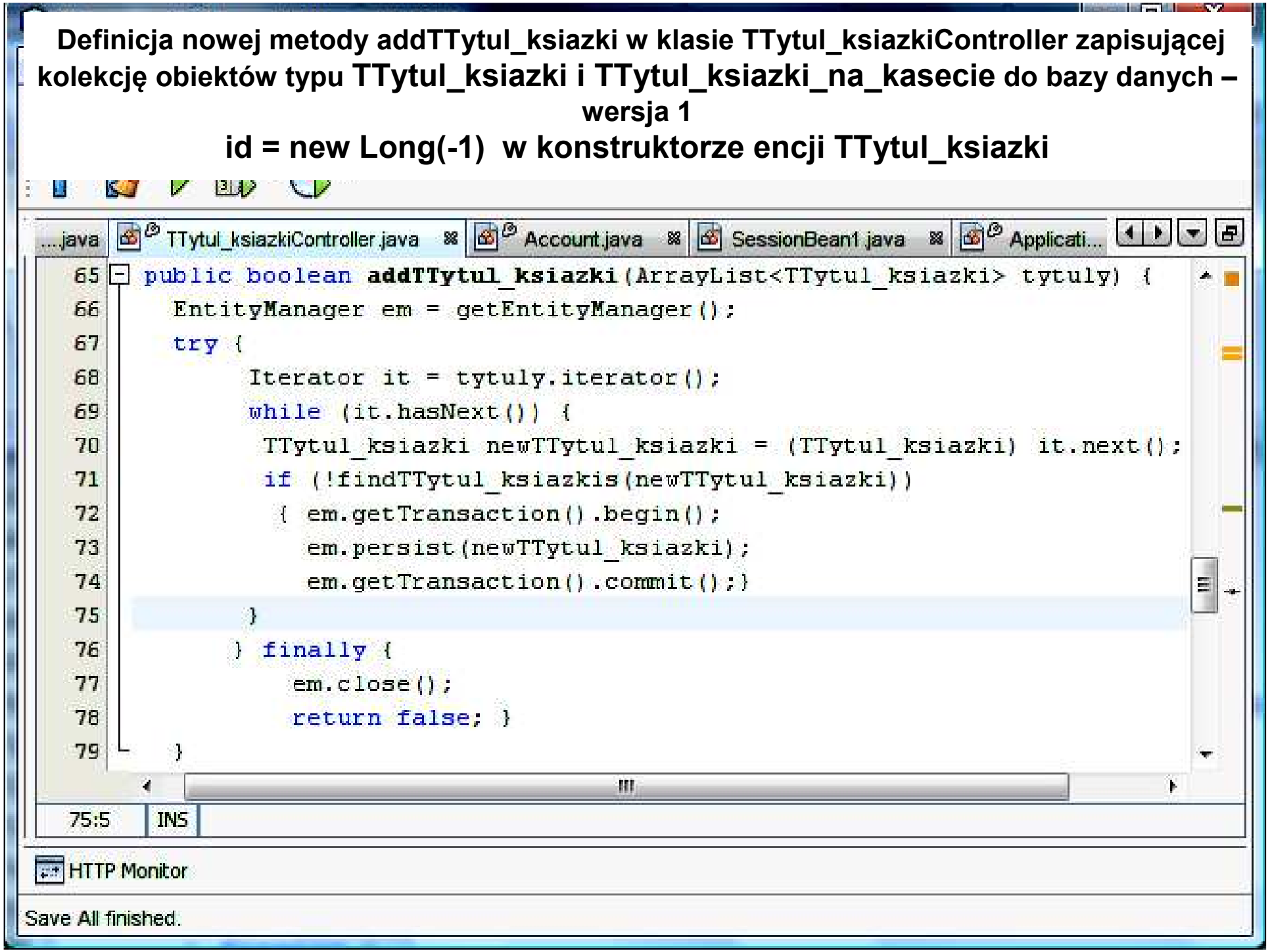


Definicje metody w klasie ApplicationBean1 związanej z zapisem danych typu kolekcja obiektów TTytul_ksiazki i TTytul_ksiazki_na_kasecie w bazie danych

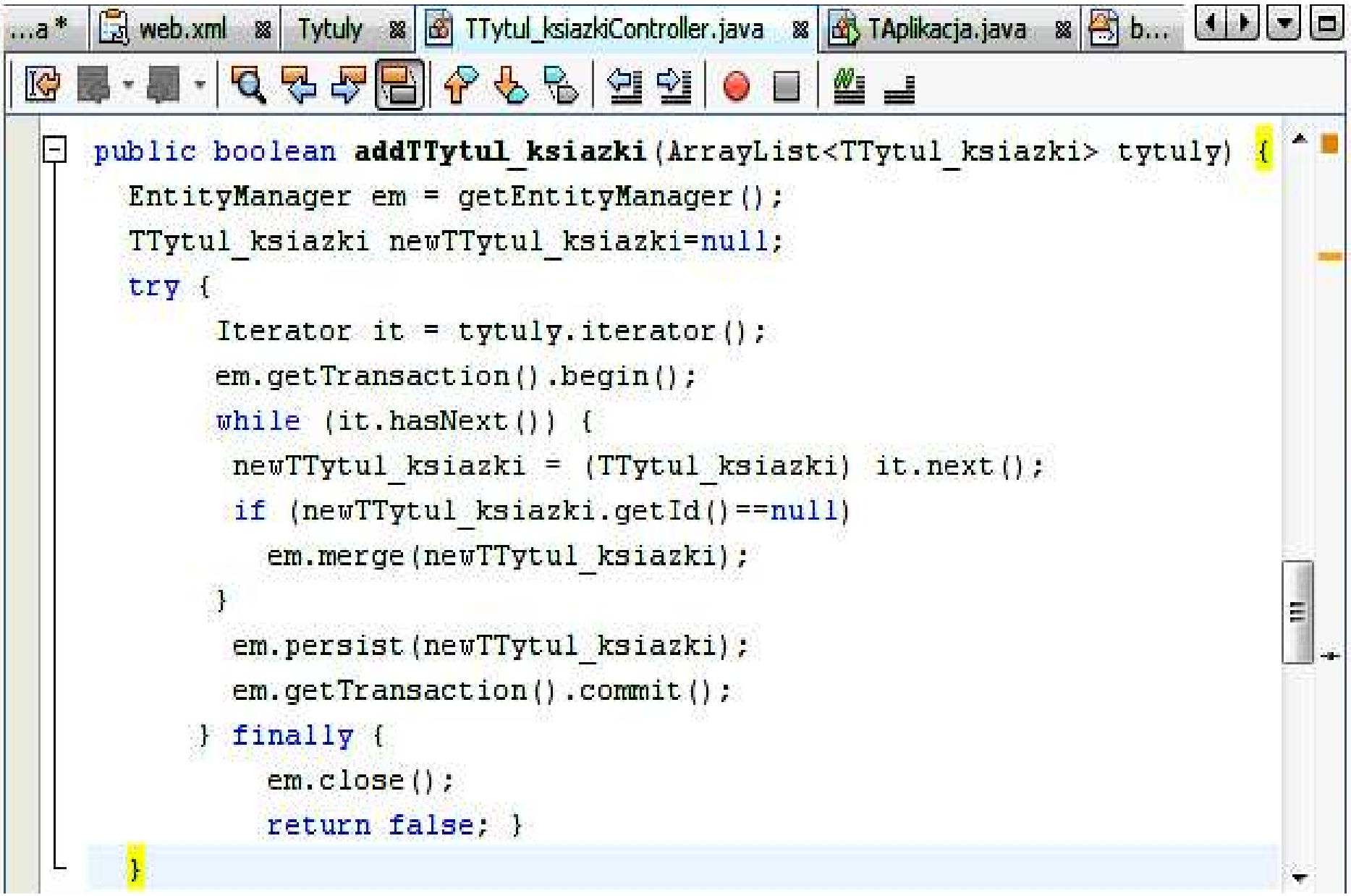


Definicja nowej metody addTTytul_ksiazki w klasie TTytul_ksiazkiController zapisującej kolekcję obiektów typu TTytul_ksiazki i TTytul_ksiazki_na_kasecie do bazy danych – wersja 1

id = new Long(-1) w konstruktorze encji TTytul_ksiazki

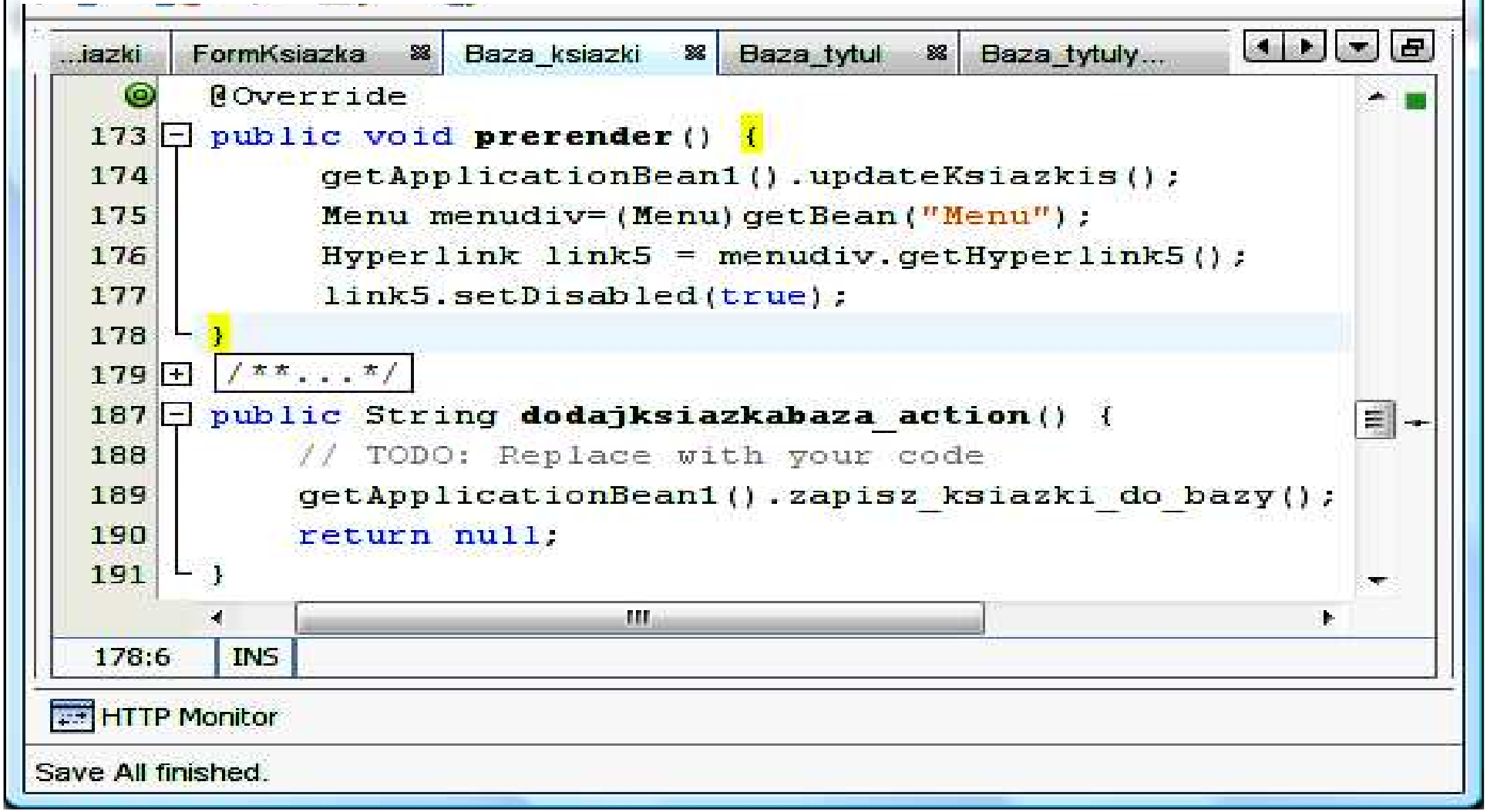


Definicja nowej metody addTTytul_ksiazki w klasie TTytul_ksiazkiController zapisującej kolekcję obiektów typu TTytul_ksiazki i TTytul_ksiazki_na_kasecie do bazy danych - wersja 2 (id = null w konstruktorze encji TTytul_ksiazki)

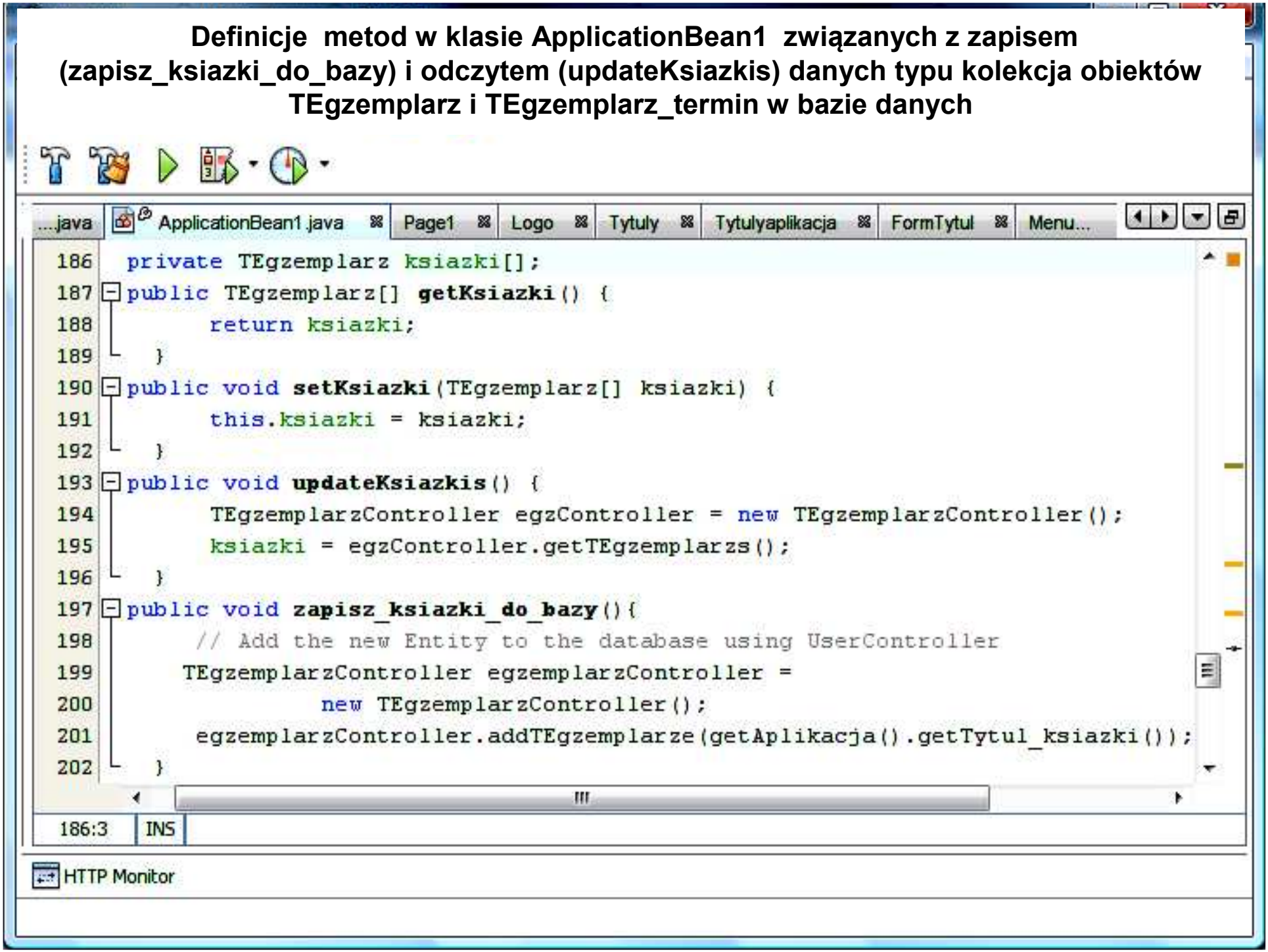


5.5. Oprogramowanie dotyczące formularza Baza_ksiazki.jsp

Definicje metod w klasie Baza_ksiazki dla strony typu JSP – do zapisu książek z warstwy biznesowej (obsługa zdarzenia dodajksiazkabaza_action) oraz generowania widoku w fazie Response przetwarzania strony metoda prerender (wygaszanie linku do bieżącej strony w fdormularzu Menu typu JSPF i aktualizacja tablicy ksiazki metodą updateKsiazkis w klasie ApplicationBean1 wyświetlanej w komponencie Table strony Ksiazkibaza typu JSPF)

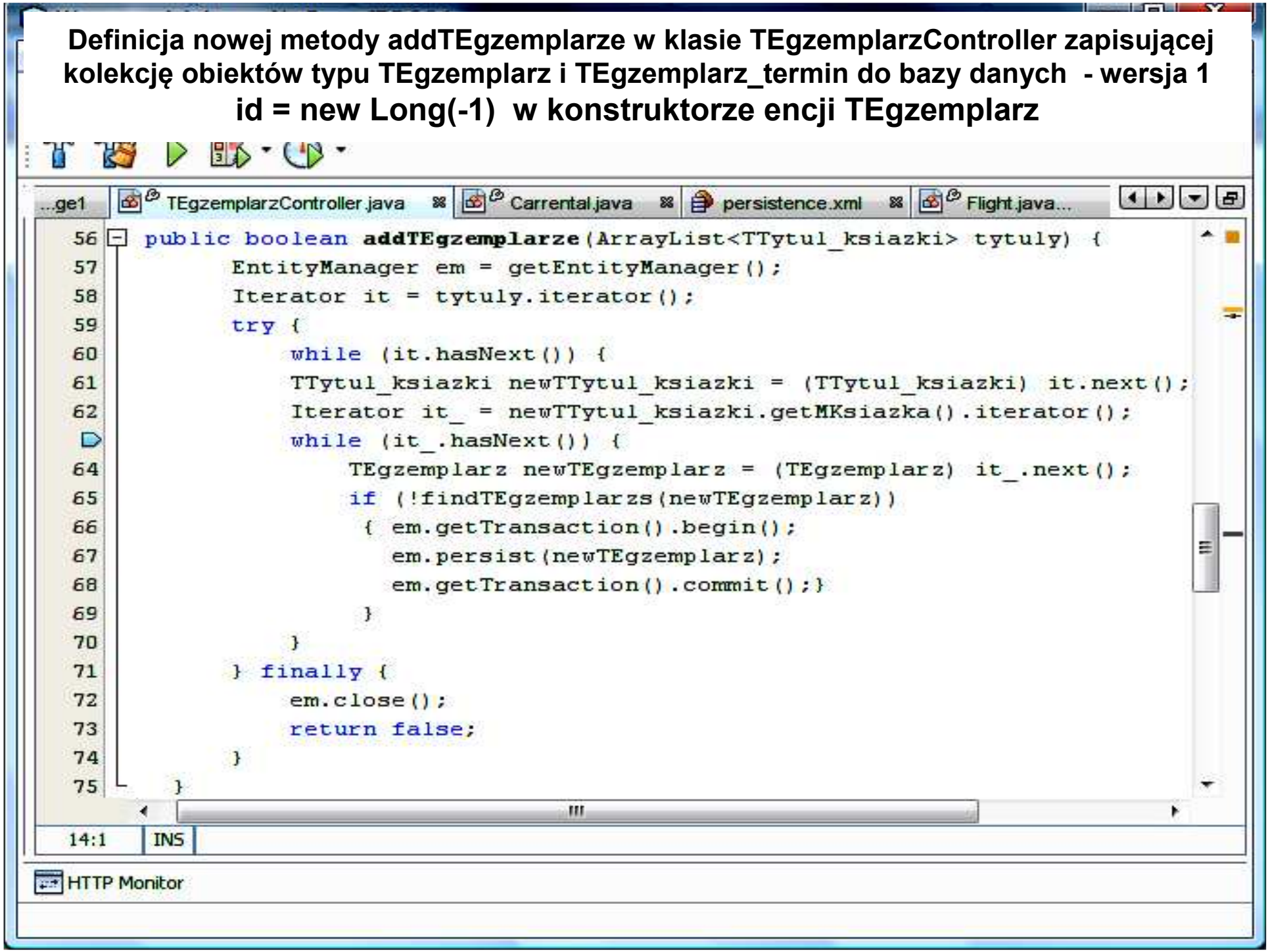


Definicje metod w klasie ApplicationBean1 związanych z zapisem (zapisz_ksiazki_do_bazy) i odczytem (updateKsiazkis) danych typu kolekcja obiektów TEgzemplarz i TEgzemplarz_termin w bazie danych



Definicja nowej metody addTEgzemplarze w klasie TEgzemplarzController zapisującej kolekcję obiektów typu TEgzemplarz i TEgzemplarz_termin do bazy danych - wersja 1

id = new Long(-1) w konstruktorze encji TEgzemplarz



Definicja nowej metody addTEgzemplarze w klasie TEgzemplarzController zapisującej kolekcję obiektów typu TEgzemplarz i TEgzemplarz_termin do bazy danych – wersja 2

id = null w konstruktorze encji TEgzemplarz

...ava | TEgzemplarzController.java * | TEgzemplarz.java | TTytul_ksiazki.java | TTyt...

```
public boolean addTEgzemplarze(ArrayList<TTytul_ksiazki> tytuly) {
        EntityManager em = getEntityManager();
        TEgzemplarz newTEgzemplarz = null;
        Iterator it = tytuly.iterator();
        em.getTransaction().begin();
        try {
            while (it.hasNext()) {
            TTytul_ksiazki newTTytul_ksiazki = (TTytul_ksiazki) it.next();
            if(newTTytul_ksiazki.getId()==null)continue;
            Iterator it_ = newTTytul_ksiazki.getMKsiazka().iterator();
            while (it_.hasNext()) {
                newTEgzemplarz = (TEgzemplarz) it_.next();
                 if (newTEgzemplarz.getId()==null)
                   em.merge(newTEgzemplarz); }
            }
             em.persist(newTEgzemplarz);
             em.getTransaction().commit();
        } finally {
            em.close();
            return false; }
    }
```

5.6. Zsynchronizowanie zawartości bazy danych i warstwy biznesowej przy starcie aplikacji – zastosowanie metod update.. w obiekcie ApplicationBean1

```
@Override
public void init() { // Perform initializations inherited from
        super.init();
        // Perform application initialization that must complete
        // *before* managed components are initialized
        // TODO - add your own initialiation code here
        Managed Component Initialization
        // *after* managed components are initialized
        // TODO - add your own initialization code here
        updateTytuls();
        updateKsiazkis();
        updateAplikacja();
        przygotujtytuly();
    }
private TAplikacja aplikacja = new TAplikacja();
public TAplikacja getAplikacja() {
        return aplikacja;
}
public void setAplikacja(TAplikacja aplikacja) {
        this.aplikacja = aplikacja;
}
public void updateAplikacja() {
 for (int i = 0; i < tytuly.length; i++)
        aplikacja.getTytul_ksiazki().add(tytuly[i]);
 Iterator it = aplikacja.getTytul_ksiazki().iterator();
 while (it.hasNext()) {
   TTytul_ksiazki tytul = (TTytul_ksiazki) it.next();
   for (int j = 0; j < ksiazki.length; j++) {
     TTytul_ksiazki tytul1 = ksiazki[j].getMTytul_ksiazki();
     if(tytul1!=null)
      if( tytul1.equals(tytul))
        { tytul.getMKsiazka().add(ksiazki[j]); }
     } } }
```

5.7. Przystosowanie do pracy z wieloma wątkami warstwy biznesowej – metody typu synchronized