Instrukcja 5

Laboratorium 7

Identyfikacja klas reprezentujących logikę biznesową projektowanego oprogramowania, definicja atrybutów i operacji klas oraz związków między klasami - na podstawie analizy scenariuszy przypadków użycia. Opracowanie diagramów klas i pakietów. Zastosowanie projektowych wzorców strukturalnych i wytwórczych

Cel laboratorium:
Tworzenie modelu projektowego programowania (wykład 1) opartego na identyfikacji klas, reprezentujących logikę biznesową projektowanego systemu. Należy dokonać definicji atrybutów klas oraz związków między klasami - na podstawie analizy scenariuszy przypadków użycia (wykład 1,wykład 3, wykład4, wykład 5-część 1, wykład5-część2, wykład6; Dodatek 1 instrukcji).
Uwaga: Należy rozwijać projekt wykonany przy realizacji instrukcji 2-4.

1. Należy wykonać analizę wspólności i zmienności scenariuszy przypadków użycia **(wykład 1, wykład 3, wykład4, Dodatek 1 instrukcji)** i dokonać identyfikacji klas i powiązań miedzy klasami należących do **warstwy biznesowej oprogramowania (wykład6),** umieszczając je na diagramie klas realizowanego projektu UML w środowisku VP CE (**p. 1.3 Dodatku 1**):
   1.1. Klas bazowych
   1.2. Klas pochodnych, powiązanych relacją Generalization (Dziedziczenie)
   1.3. Klas powiązanych relacjami typu Association (Asocjacja), lub/i Dependency (Zależność) lub/i Aggregation (Agregacja słaba, Agregacja silna czyli Kompozycja)
   1.4. Określić kierunek i liczność relacji z p. 1.3. pomiędzy klasami
   1.5. Dokonać identyfikacji wzorców projektowych **(wykład5-część2).**
2. Należy w środowisku **NetBeans** wykonać projekt typu **Java Class Library** i w pakiecie o nazwie dopasowanej do dziedziny realizowanego projektu wykonać definicje klas wg przykładu z **p.1.4 Dodatku 1**.
3. Grupa jednoosobowa powinna wykonać model projektowy 1-2 złożonych przypadków użycia. Grupa dwuosobowa powinna wykonać model projektowy 2-3 złożonych przypadków użycia tzn. opartych na relacjach <<include>> lub/i <<extend>> lub/ i <<use>> – kontynuacja prac wg instrukcji 2-4.

Uwaga:
Notacje stosowane na diagramie klas w Dodatku 1, odpowiadające składni składowych klas w języku Java, różnią się od notacji przedstawionych w instrukcji 1, prezentujących diagramy wykonane w środowisku VP CE. Oczywiście, w realizowanym projekcie w ramach laboratoriów należy zastosować odpowiadające notacje proponowane w środowisku VP CE.

Dodatek 1

Przykład opracowania diagramów klas i pakietów. Zastosowanie projektowych wzorców strukturalnych i wytwórczych (cd. z instrukcji 2,3,4).

1. Przykład wyników analizy - identyfikacja klas na podstawie scenariuszy przypadków użycia

1.1. Wynik analizy wspólności (wykład 1, wykład 3, wykład4):

Wykryto trzy główne klasy typu „Entity” ze względu na odpowiedzialność:

- **TRachune**k (PU: Wstawianie nowego rachunku, Wstawianie nowego zakupu, Obliczanie wartosci rachunku),
- **TZakup** (PU: Wstawianie nowego zakupu, Obliczanie wartosci rachunku),
- **TProdukt1** (PU: Wstawianie nowego produktu, Wstawianie nowego zakupu, Obliczanie wartosci rachunku)

1.2. Wynik analizy zmienności (wykład 1, wykład 3, wykład4):

1.2.1. Wykryto dwa podzbiory typów produktów, które posiadają cenę jednostkową podawaną jako cenę netto, jeśli produkt nie posiada atrybutu podatek lub cenę brutto, jeśli posiada atrybut podatek. Do modelowania tych dwóch typów produktów zastosowano dziedziczenie:

- Zdefiniowano klasę pochodną **TProdukt2** typu „Entity”, która dziedziczy od klasy **TProdukt1**
- Identyfikacji dokonano na podstawie następujących przypadków użycia: PU Wstawianie nowego produktu, PU Obliczanie wartosci rachunku

1.2.2. Wykryto strategię zmniejszania ceny jednostkowej wynikającej z promocji powiązaną z produktem zarówno z podatkiem, jak i bez podatku:

- Zdefiniowano klasę **TPromocja** typu „Entity”
- Zdefiniowano związek typu asocjacja między klasami **TProdukt1** i **TPromocja**, który jest dziedziczony przez pozostałe typy produktu tzn. **TProdukt2**. Ponieważ jednak promocja nie musi dotyczyć każdego produktu, jest w związku asocjacji 0..1 do 0..* z bazowym (głównym) produktem typu **TProdukt1**
- Dzięki temu produkty typu **TProdukt1** i **TProdukt2** powinny podawać uogólnioną cenę detaliczną: bez podatku, z podatkiem oraz w razie potrzeby z uwzględnieniem scenariusza dodawania promocji do ceny detalicznej produktu dla dwóch pierwszych przypadków (stąd cztery typy ceny detalicznej)
- Identyfikacji dokonano na podstawie następujących przypadków użycia: **PU Wstawianie nowego produktu, PU Wstawianie nowego zakupu, PU Obliczanie wartosci rachunku**.

1.2.3. Wykryto następujące związki pomiędzy klasami:

- Silna agregacja między obiektem typu **TRachunek** i obiektami typu **TZakup** (rachunek posiada kolekcję zakupów)
- Słaba agregacja między obiektem typu **TZakup** a obiektem typu **TProdukt1** (zakup składa się z produktu bazowego lub jego następców)
- Identyfikacji dokonano na podstawie następujących przypadków użycia: **PU Wstawianie nowego zakupu, PU Obliczanie wartosci rachunku.**

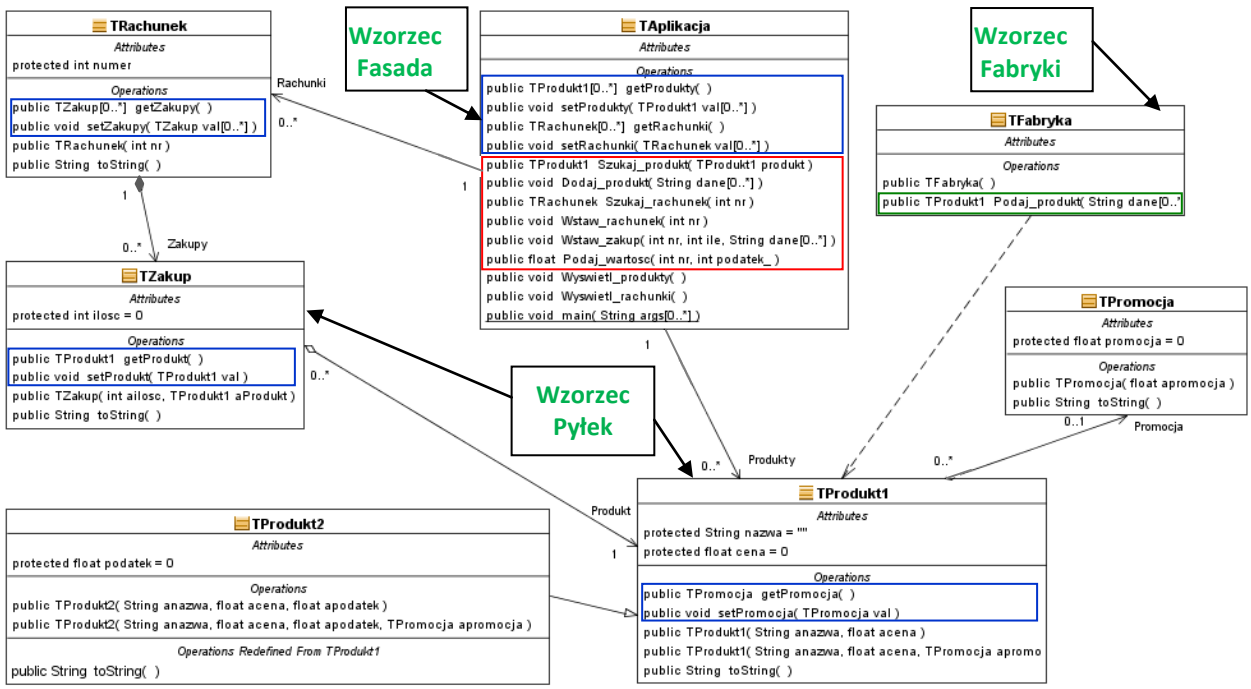
1.2.4. Wykryto następujące wzorce projektowe (wykład5-część2):

- Wzorzec wytwórczy Fabryka Abstrakcyjna jako klasę **TFabryka** typu „Control” do tworzenia różnych typów produktów – czyli obiektów typu **TProdukt1** i **TProdukt2**.
- Wzorzec strukturalny Fasada jako klasę **TAplikacja** typu „Control” do oddzielenia przetwarzania obiektów typu „Entity” od pozostałej części systemu
- Wzorzec strukturalny Pyłek, który obejmuje klasy, których instancje pełnią rolę pyłków: klasę **TProdukt1** i klasę pochodną **TProdukt2** oraz klasę **TZakup**, której

instancje pełnią rolę klientów pyłków. Oznacza to, że wiele różnych obiektów typu **TZakup** może posiadać referencję do tego samego obiektu typu **TProdukt1** lub **TProdukt2**.

1.3. Wynik analizy wspólności i zmienności – wstępny diagram klas reprezentujący warstwę biznesową oprogramowania (wykład6)

Diagram klas jest uzupełniony wstępnie o wzorce projektowe, z zaznaczeniem pełnych wyników analizy wspólności i zmienności oraz lokalizacji głównych operacji wynikających ze scenariuszy przypadków użycia, wprowadzanie do klasy **TAplikacja** implementującej projektowy wzorzec strukturalny typu Fasada.



Uwaga!

Zakres prac dotyczących diagramu klas:

Definiowanie w sposób iteracyjno - rozwojowy modelu projektowego podczas kolejnych laboratoriów (laboratoria 8-10 wg instrukcji 6-7). Proces ten polega na definiowaniu operacji i atrybutów kolejnej klasy (dziedziczenie, powiązania i agregacje) na diagramie klas, zidentyfikowanych w wyniku modelowania kolejnego przypadku użycia.

1.4. Wykonanie szkieletu kodu źródłowego wynikającego z diagramu klas z p.1.3.

Niebieskim kolorem zaznaczono metody, które służą do implementacji powiązań pomiędzy klasami (Agregacja słaba i Kompozycja).

Czerwonym kolorem zaznaczono metody obsługujące główne przypadki użycia:

- **Dodaj_produkt (PU Wstawianie nowego produktu),**
- **Wstaw_rachunek (PU Wstawianie nowego rachunku),**
- **Wstaw_zakup (PU Wstawianie nowego zakupu),**
- **Podaj_Wartosc (PU Obliczanie wartosci rachunku)**

i pomocnicze:

Szukaj_produkt (PU Szukanie produktu),

Szukaj_rachunek (PU Szukanie rachunku).

Zielonym kolorem zaznacza się metody wynikające z decyzji projektowych związanym z wybranymi wzorcami projektowymi.

Pozostałe metody nie zaznaczono kolorem – są to konstruktory i metody pomocnicze do prezentacji wyników programu działania programu.

```
package rachunek1;
import java.util.ArrayList;
public class TAplikacja
{   private List<TProdukt1> Produkty = new ArrayList<TProdukt1>();
    private List<TRachunek> Rachunki = new ArrayList<TRachunek>();
    public List<TProdukt1> getProdukty ()                    { return null;  }
    public void setProdukty (List<TProdukt1> val)            {   }
    public List<TRachunek> getRachunki ()                    { return null;  }
    public void setRachunki (List<TRachunek> val)            {   }
    public TProdukt1 Szukaj_produkt (TProdukt1 produkt)      { return null;  }
    public void Dodaj_produkt (String[] dane)                {   }
    public TRachunek Szukaj_rachunek (int nr)                { return null;  }
    public void Wstaw_rachunek (int nr)                      {   }
    public void Wstaw_zakup (int nr, int ile, String dane[]) {   }
    public float Podaj_wartosc (int nr, int podatek_)        {  return 0F;  }
    public static void main (String[] args)                  {   }
    public void Wyswietl_produkty ()                         {   }
    public void Wyswietl_rachunki ()                         {   }
}
```

```
package rachunek1;
import java.util.ArrayList;
class TRachunek
{
   protected int numer;
   private List<TZakup> Zakupy = new ArrayList<TZakup>();
   public List<TZakup> getZakupy ()                 { return null;   }
   public void setZakupy (List<TZakup> val)         {   }
   public TRachunek (int nr)                        {   }
   @Override
   public String toString ()                        { return null;   }
}
```

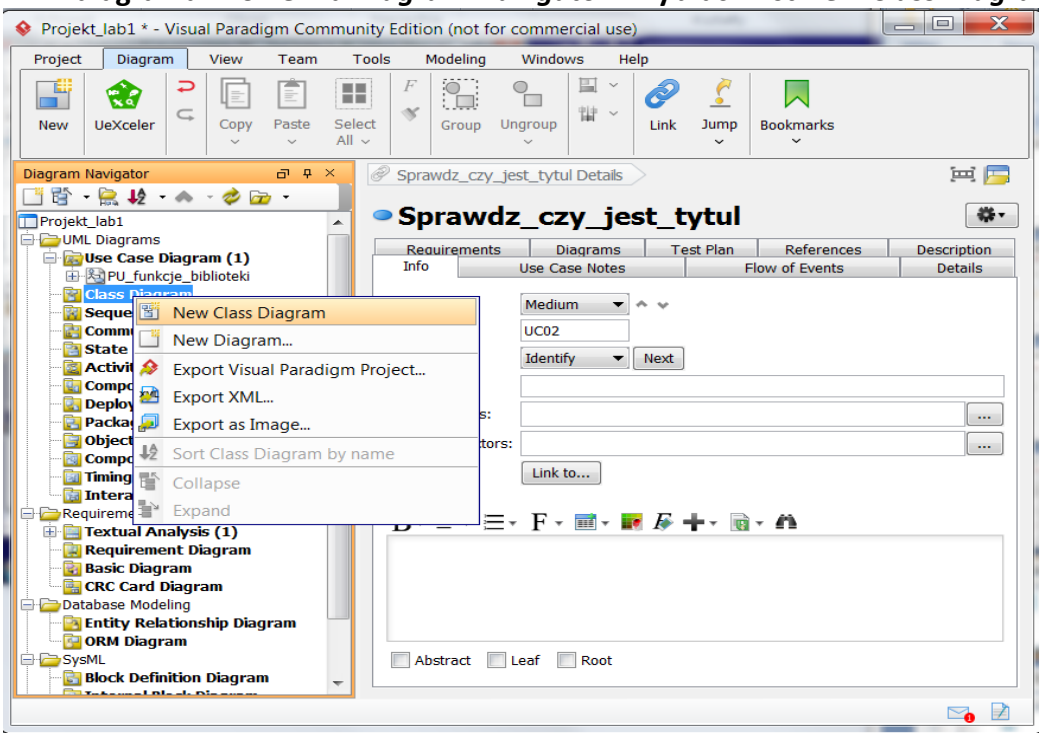
```
package rachunek1;
class TZakup
{
 protected int ilosc ;
 private TProdukt1 Produkt ;
 public TProdukt1 getProdukt ()                   {   return null;   }
 public void setProdukt (TProdukt1 val)           {   }
 public TZakup (int ailosc, TProdukt1 aProdukt)   {   }
  @Override
 public String toString ()                        {  return null;   }
}
```

```
package rachunek1;
class TProdukt1
 {
    protected String nazwa = "";
    protected float cena;
    protected TPromocja Promocja;
    public TPromocja getPromocja ()
        { return null; }
    public void setPromocja (TPromocja val)
        {   }
    public TProdukt1 (String anazwa, float acena)
        {   }
    public TProdukt1 (String anazwa, float acena, TPromocja apromocja)
        {   }
   @Override
    public String toString ()
        { return null; }
}

package rachunek1;
class TProdukt2 extends TProdukt1
{
  protected float podatek;
  public TProdukt2 (String anazwa, float acena, float apodatek)
        {  super(anazwa, acena);  }
  public TProdukt2 (String anazwa, float acena, float apodatek,TPromocja promocja)
        {  super(anazwa, acena, apromocja);  }
 @Overridel
  public String toString ()
        {  return null; }
}

package rachunek1;
class TPromocja
{
    protected float promocja;
    public TPromocja (float apromocja)
        {   }
   @Override
    public String toString ()
        { return null;    }
}

package rachunek1;
public class TFabryka
 {
   public TFabryka ()
        {   }
   public TProdukt1 Podaj_produkt (String[] dane)
       { return null;    }
}
```

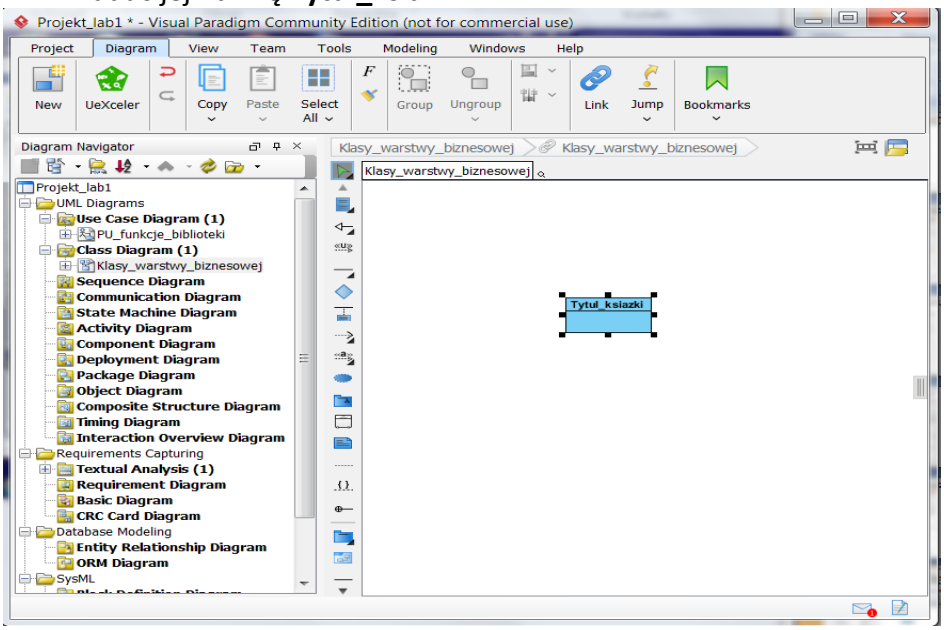
Dodatek 2

1. **Tworzenie diagramów klas i sekwencji użycia w wybranym środowisku np Visual Paradigm**

1.1. Dodanie diagramu klas do projektu – należy kliknąć prawym klawiszem na nazwę diagramu w okienku *Diagram Navigator* i wybrać z list *New Class Diagram*

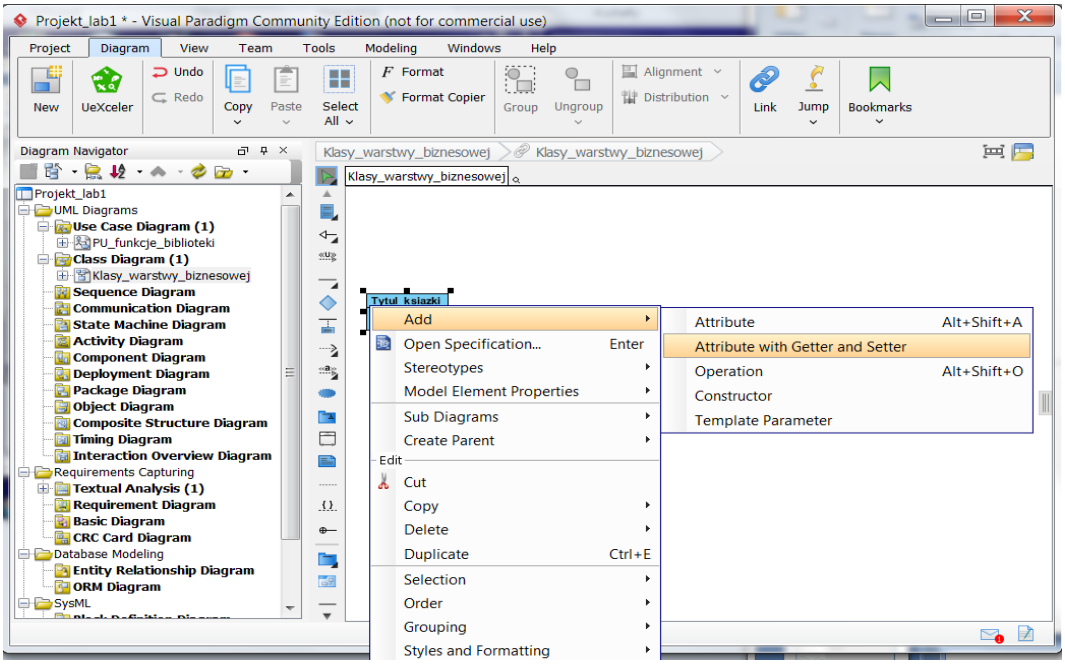


1.2. Po nadaniu nazwy diagramowi klas warstwy biznesowej jako **Klasy_warstwy_biznesowej** należy zdefiniować klasy zidentyfikowane na podstawie scenariuszy przypadków użycia. Pierwsza definiowana klasa zawiera dane tytułu książki – należy przeciągnąć ikonę klasy z palety lewym klawiszem myszy i położyć na diagramie i nadać jej nazwę **Tytul_ksiazki**

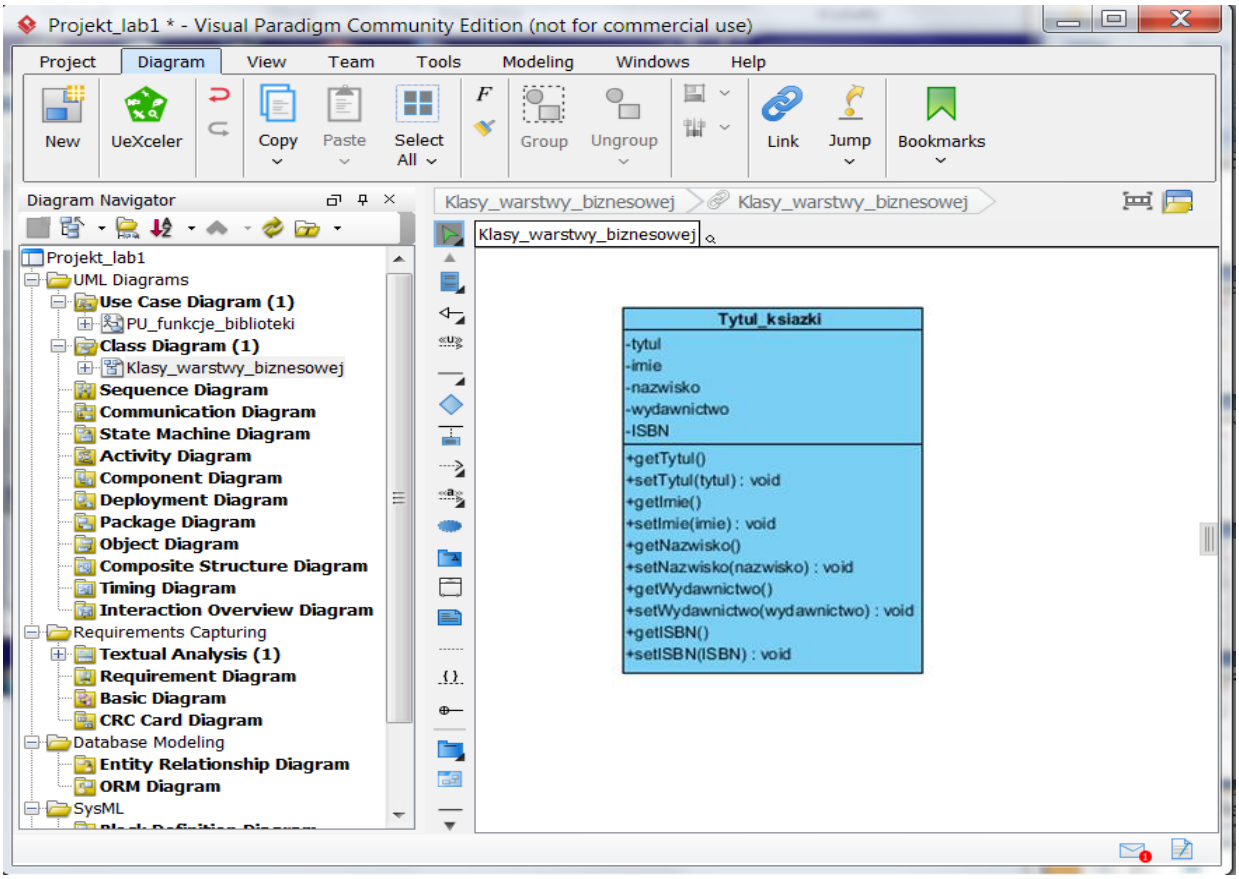


1.3. Zdefiniowanie atrybutów i metod – po kliknięciu prawym klawiszem na klasę należy wybrać z listy pozycje *Attribute* do definiowania nowych atrybutów lub *Operation* do definiowania metod. Zdefiniowanie atrybutów i metod dostępu do atrybutów – po

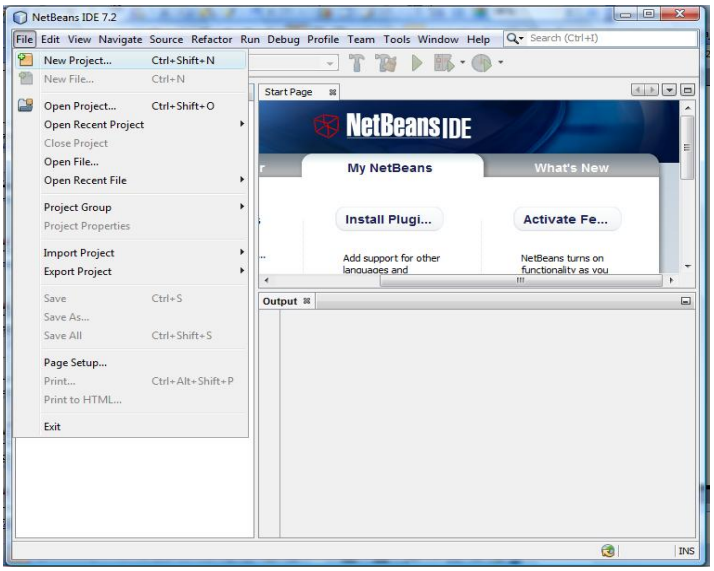
kliknięciu prawym klawiszem na klasę należy wybrać z listy pozycję *Attribute with Getter and Setter*



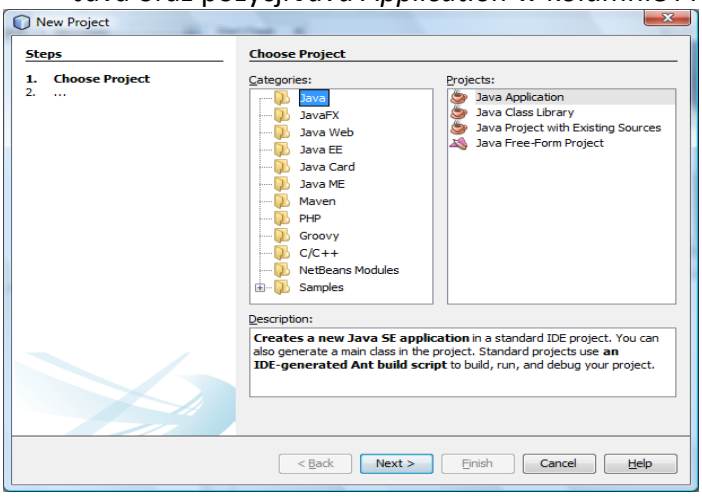
1.4. Dodano prywatne atrybuty i publiczne metody dostępu do atrybutów typu getter i setter klasie **Tytul_ksiazk**i



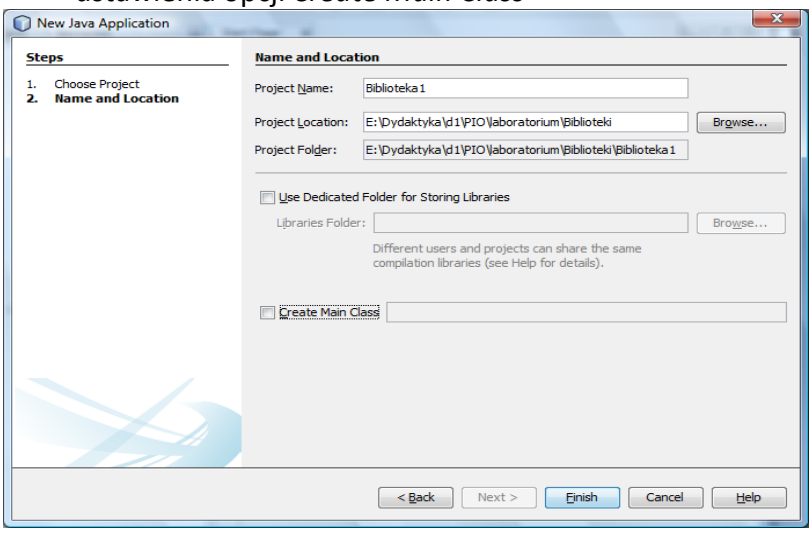
1.5. Wykonanie projektu typu aplikacja Javy w środowisku typu NetBeans – *File/New Project*



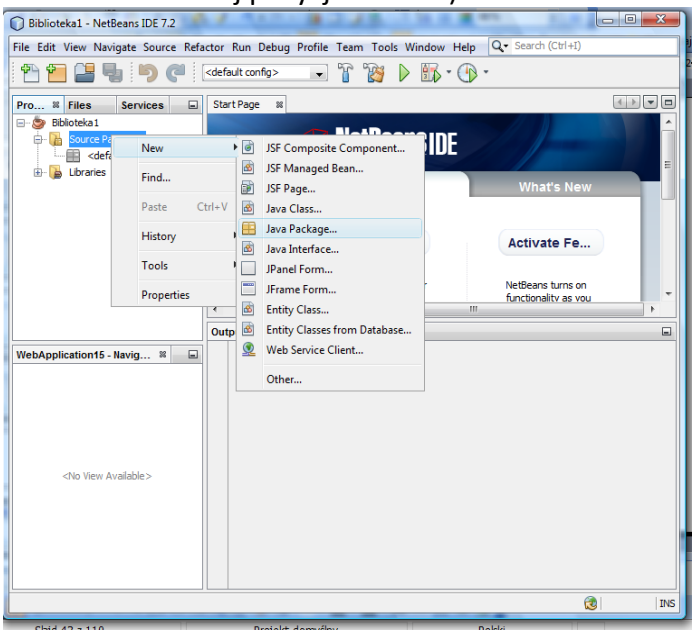
1.6. Wykonanie projektu typu *Java/ Java Application* – wybór w kolumnie *Categories* pozycji Java oraz pozycji *Java Application* w kolumnie *Projects*



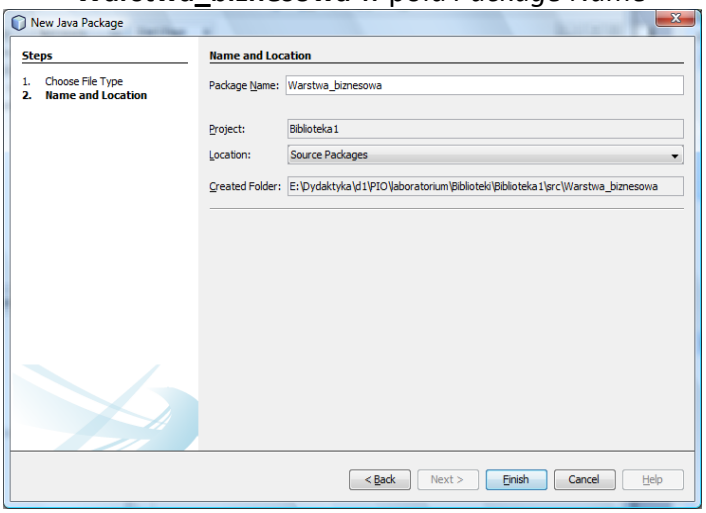
1.7. Wykonanie projektu typu *Java/ Java Application* – nadanie nwazy projektowi w polu *Project Name* oraz lokalizacji za pomocą klawiasza *Browse…* w polu *Project Location* bez ustawienia opcji *Create Main Class*



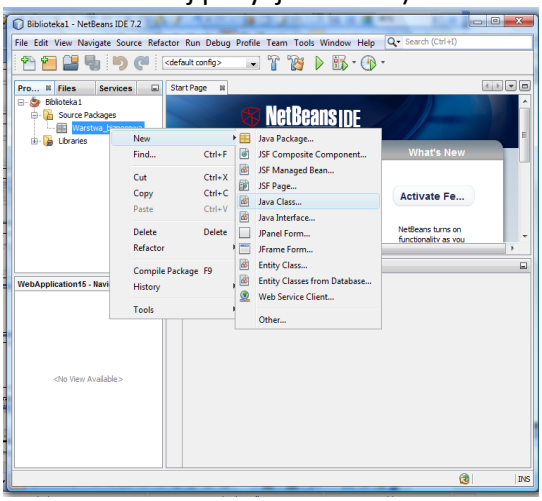
1.8. Wstawienie nowego pakietu do projektu – prawym klawiszem należy kliknąć na pozycję *Source Package* w okienku *Projects i wybrać z listy pozycję Java Package* (lub *Other*, jeśli nie ma takiej pozycji na liście)



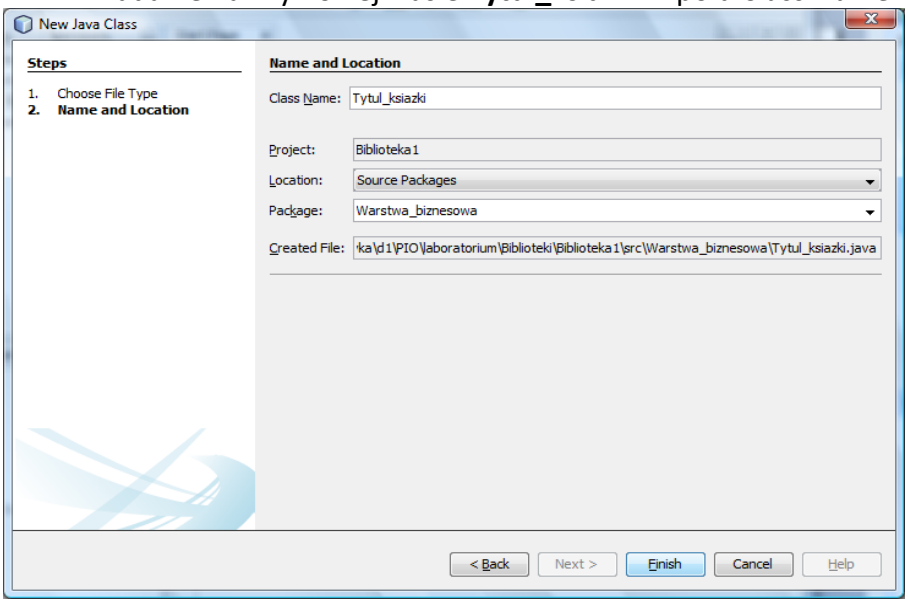
1.9. Wstawienie nowego pakietu do projektu – nadanie nazwy pakietowi **Warstwa_biznesowa** w polu *Package Name*



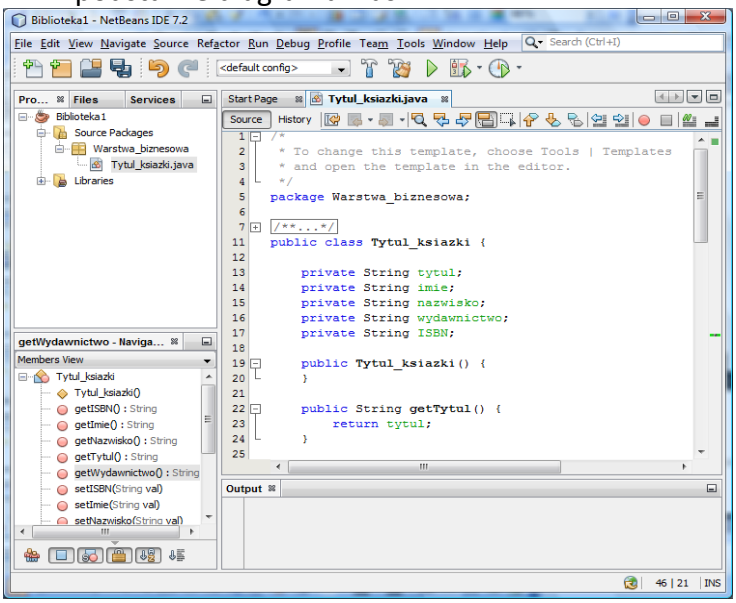
1.10. Wstawienie do pakietu **Warstwa_biznesowa** nowej klasy – należy kliknąć prawym klawiszem myszy na nazwę pakietu i wybrać z listy pozycję *Java Class* (lub *Other*, jeśli nie ma takiej pozycji na liście)



1.11. Nadanie nazwy nowej klasie **Tytul_ksiazki** w polu *Class Name*



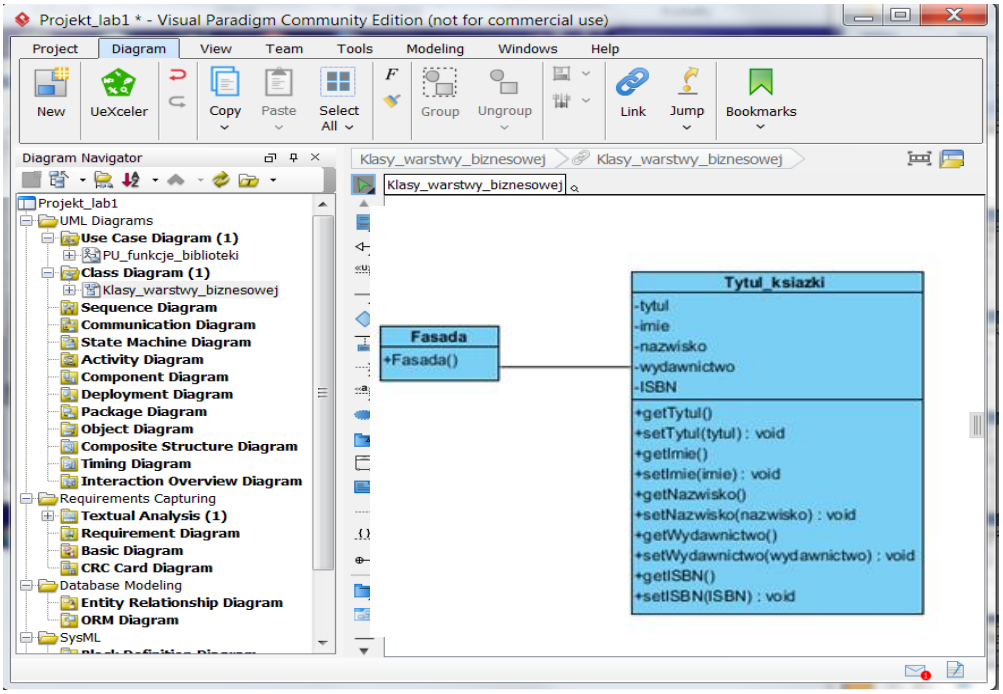
1.12. Zdefiniowanie kodu klasy **Tytul_ksiazki** (kod klasy zawiera następny slajd) na podstawie diagramu klas



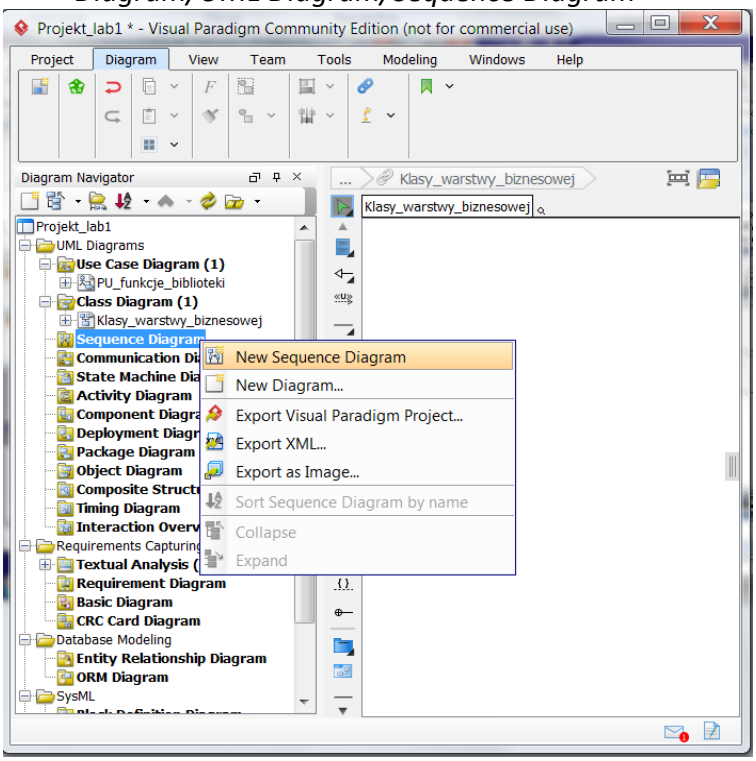
1.13. Kod klasy Tytul_ksiazki

```
public class Tytul_ksiazki {
   private String tytul;
   private String imie;
   private String nazwisko;
   private String wydawnictwo;
   private String ISBN;
   public String getTytul()                 {  return tytul;  }
   public void setTytul(String val)         {  this.tytul = val; }
   public String getImie()                  { return imie;   }
   public void setImie(String val)          {  this.imie = val;   }
   public String getNazwisko()              {  return nazwisko;   }
   public void setNazwisko(String val)      { this.nazwisko = val;  }
   public String getWydawnictwo()           { return wydawnictwo;   }
   public void setWydawnictwo(String val)   { this.wydawnictwo = val;  }
   public String getISBN()                  {  return ISBN;   }
   public void setISBN(String val)          {  this.ISBN = val; }
}
```

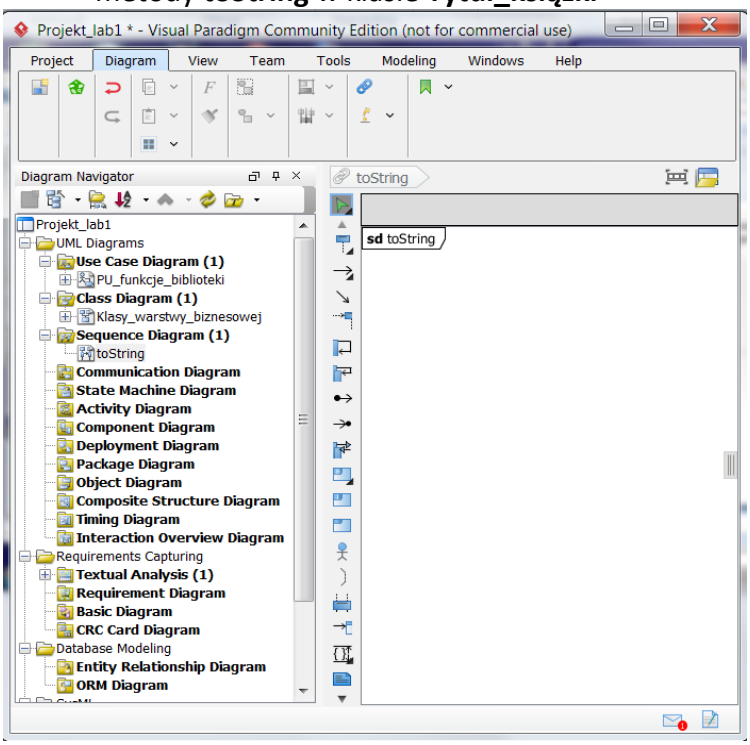
1.14. Wstawienie nowej klasy **Fasada** (podobnie jak klasę Tytul_ksiazki) powiązanej za pomocą relacji *Association* 1..0 z klasą typu **Tytul_ksiazki** – relację należy wybrać z palety z lewej strony lewym klawiszem myszy oraz położyć ją na klasie **Fasada** i przeciągnąć na klasę **Tytul_ksiazki.** Klasa ta reprezentuje wzorzec projektowy Fasada - będzie zastosowana do obsługi wywołań przypadków użycia przez warstwę interfejsu graficznego użytkownika.



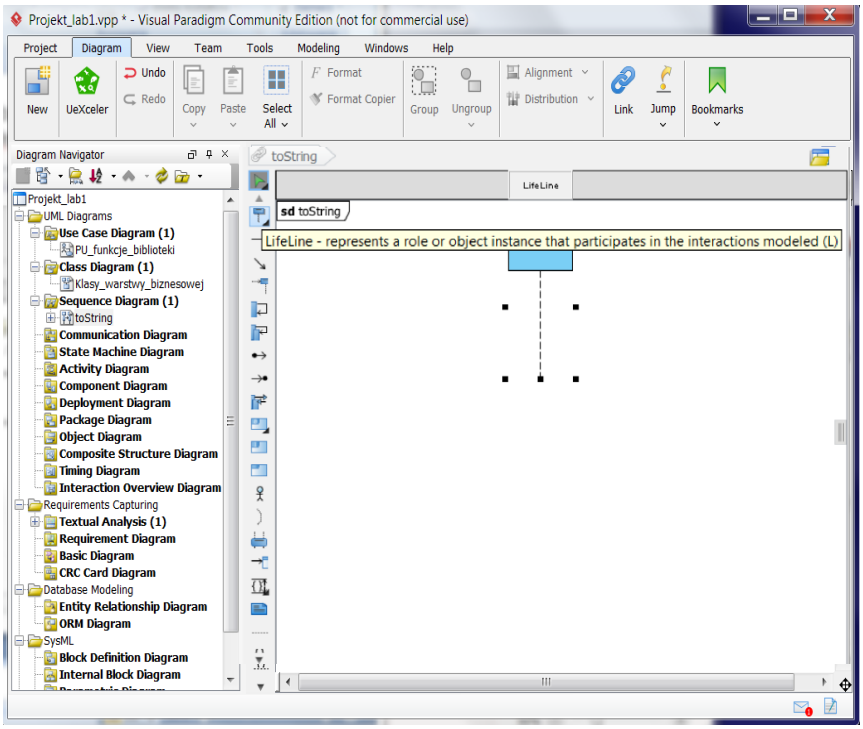
1.15. Wstawianie diagramu sekwencji – należy kliknąć prawym klawiszem myszy na nazwę modelu w okienku Model Explorer i wybrać z listy opcję *Diagram/UML Diagram/Sequence Diagram*



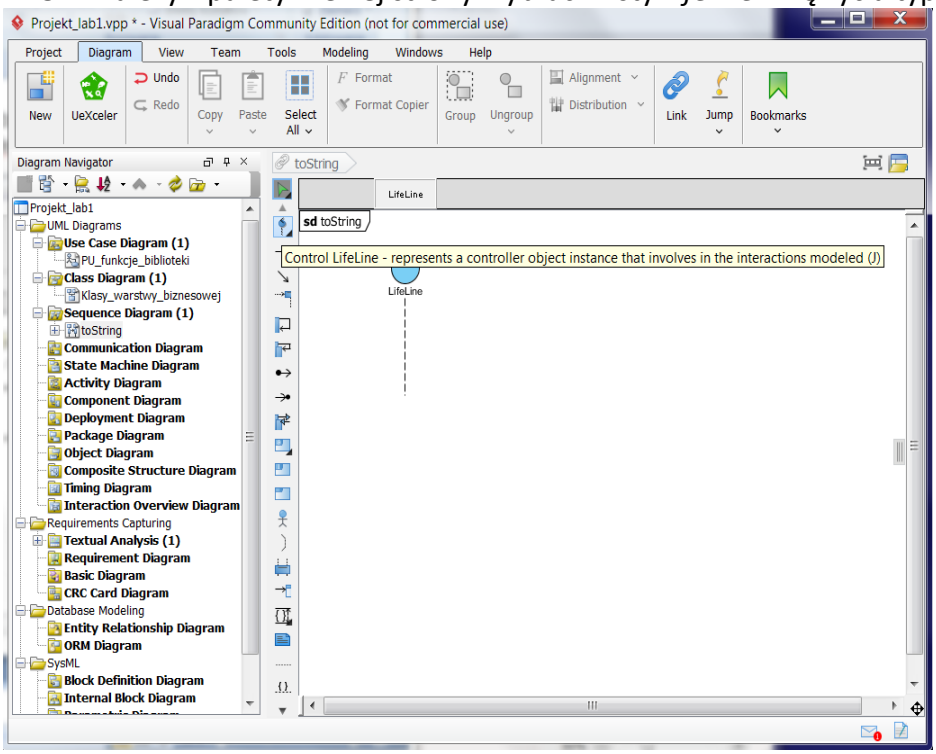
1.16. Należy nadać nazwę t**oString** diagramowi sekwencji – diagram będzie zawierał definicję metody **toString** w klasie **Tytul_książki**



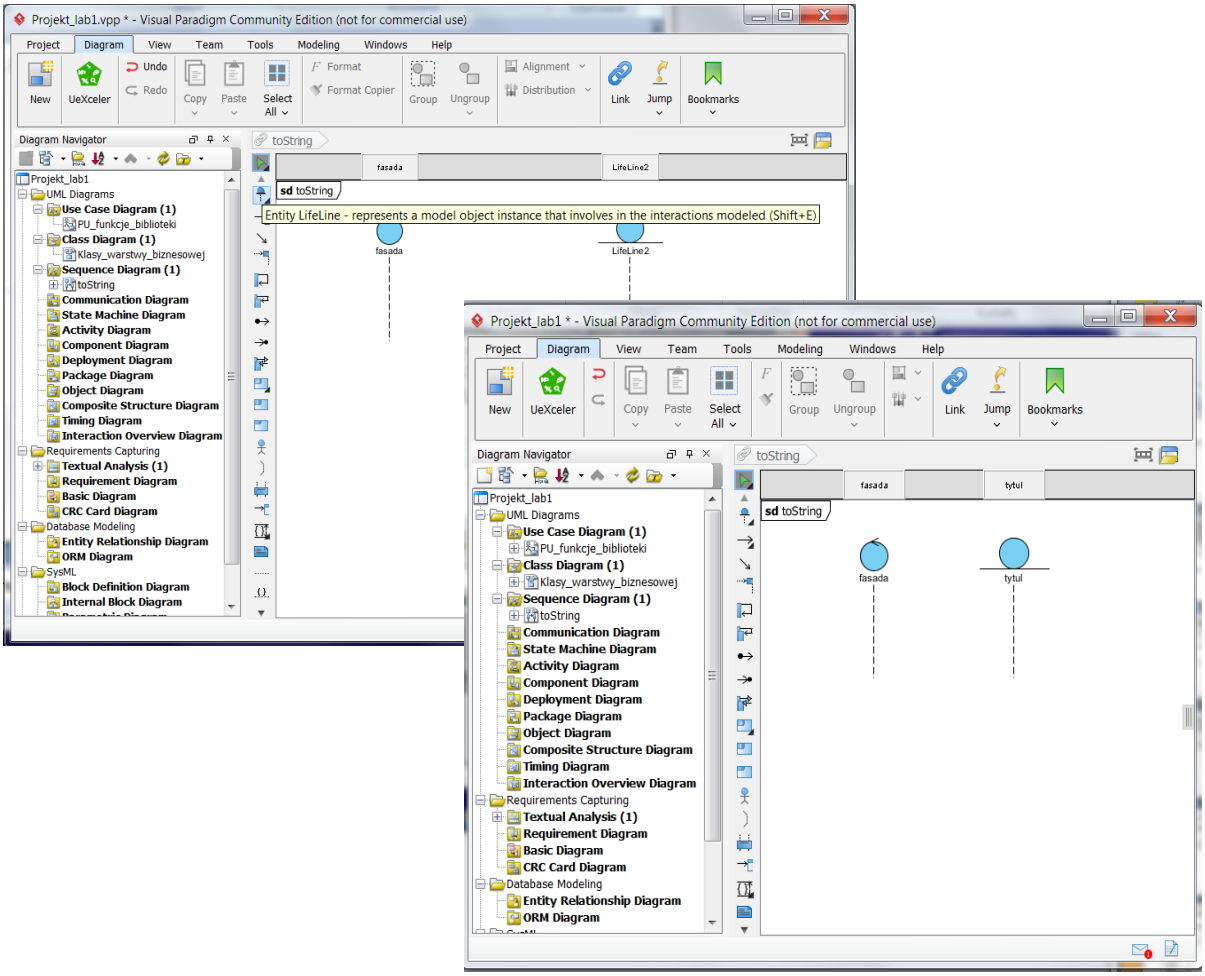
1.17. Można z palety z lewej strony wybrać z listy *Lifeline* linię życia typu *Lifeline* i używać do modelowania wszystkich linii życia. W instrukcji zastosowano jednak zróżnicowane typy linii życia, wynikające z typów klas obiektów: Entity, Boundry, Control



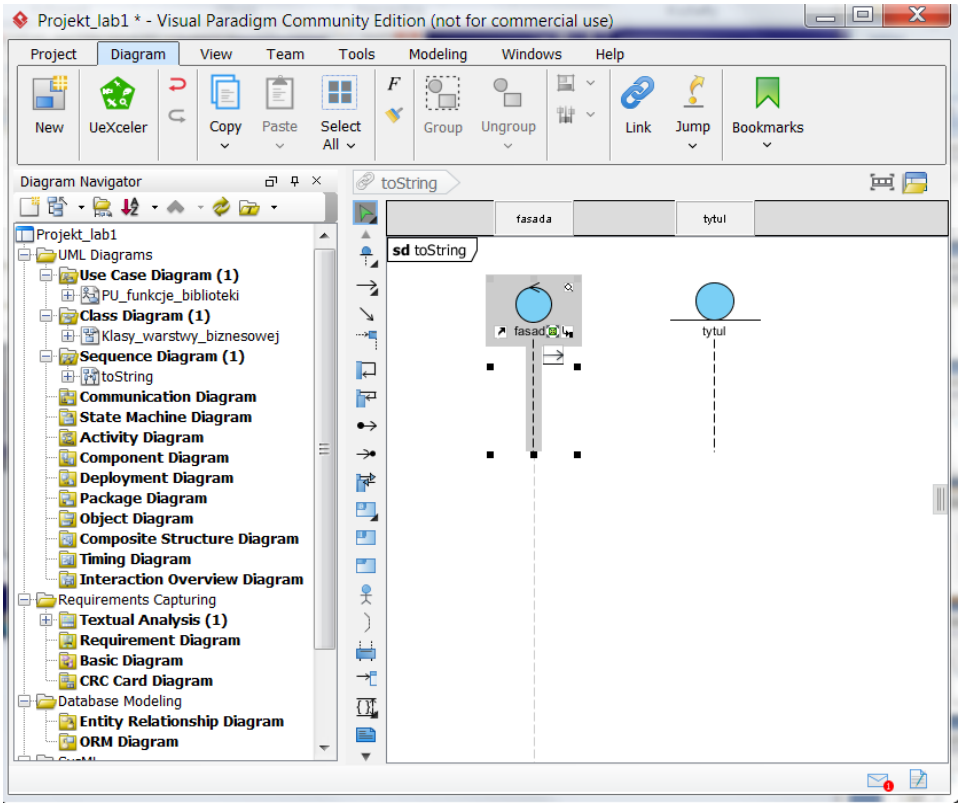
1.18. Należy z palety z lewej strony wybrać z listy *Lifeline* linię życia typu *Control Lifeline*



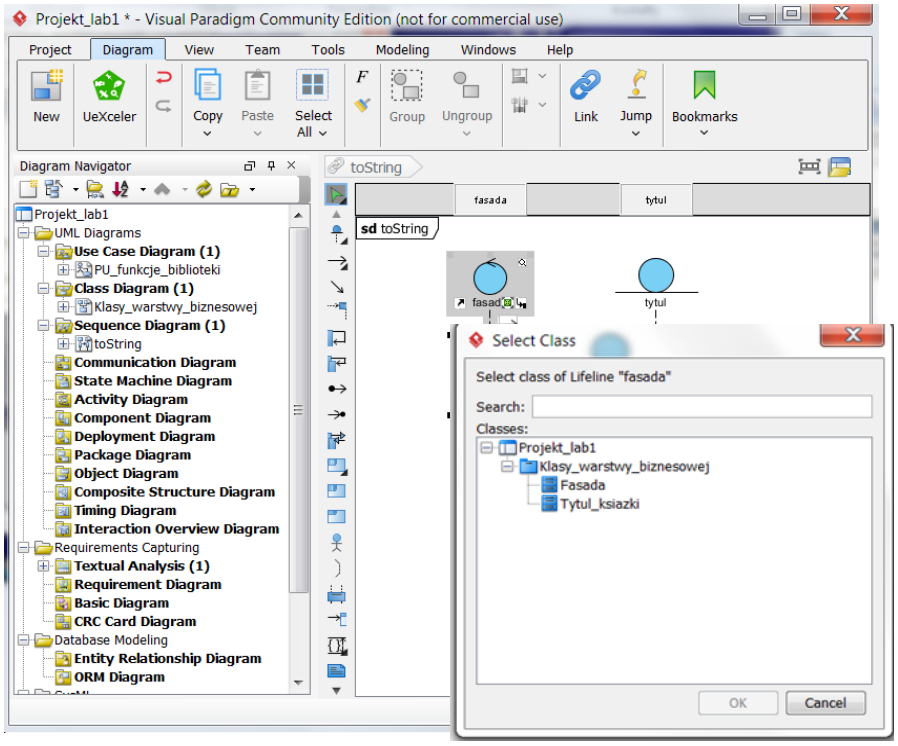
1.19. Należy z palety z lewej strony wybrać z listy Lifeline linię życia typu *Entity Lifeline* i nadać nazwę **tytul**



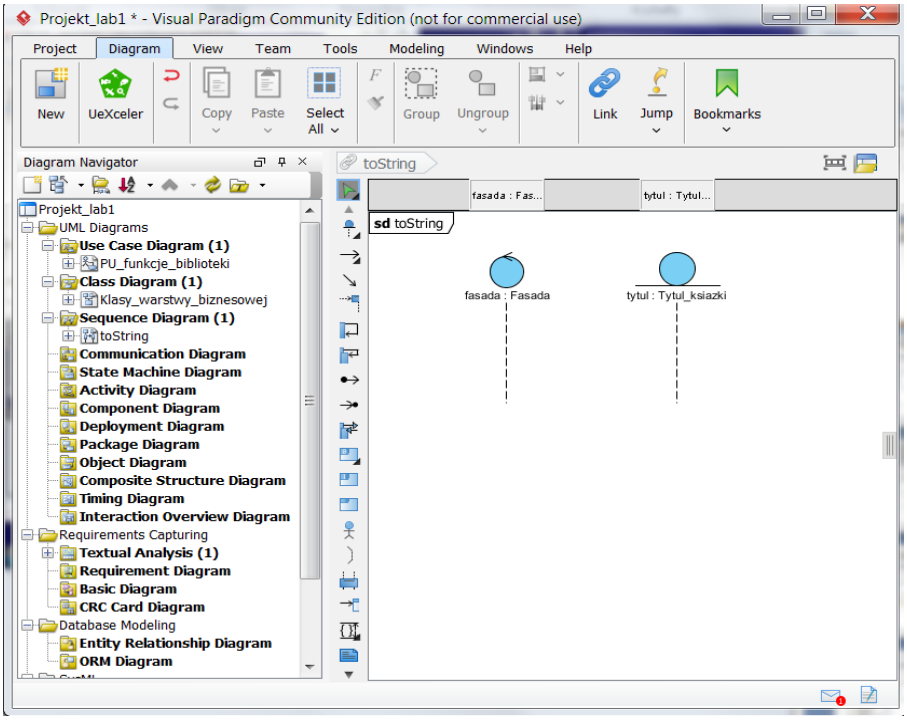
1.20. Należy linie życia obiektów powiązać z klasami z diagramu klas – po wybraniu linii życia **fasada** należy kliknąć prawym klawiszem myszy i wybrać z listy opcję *Select Class/Select Class*



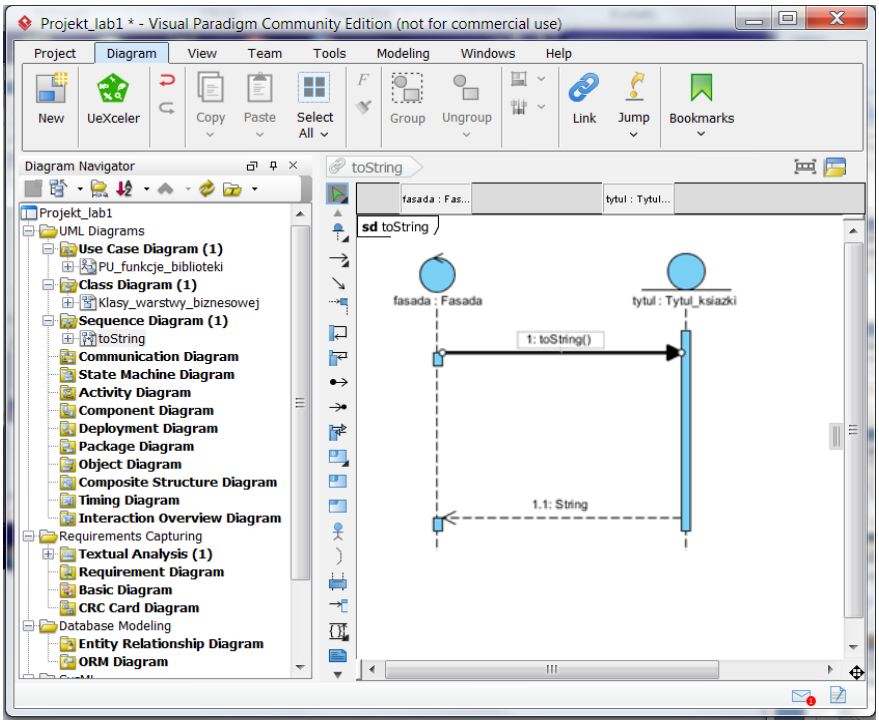
1.21. W formularzu *Select Class* należy w polu *Search* wpisać fragment nazwy klasy **Fasada**. W ukazanym okienku wybrać właściwą klasę i nacisnąć klawisz *OK.* Podobnie należy powiązać linię życia **tytul.**



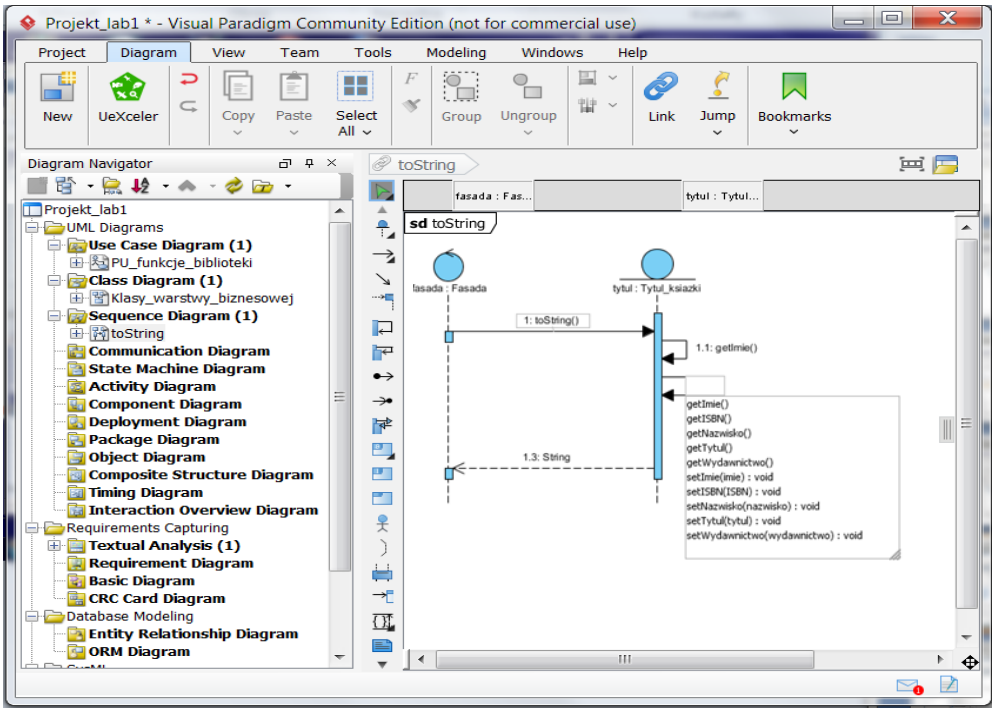
1.22. Należy wybrać z listy *Message* typ metody *Call Message* i przeciągnąć ją kładąc na linii życia **fasada** i przeciągając położyć na linii życia **tytul**. Podobnie należy zrobić z wiadomością typu *Return Message*.



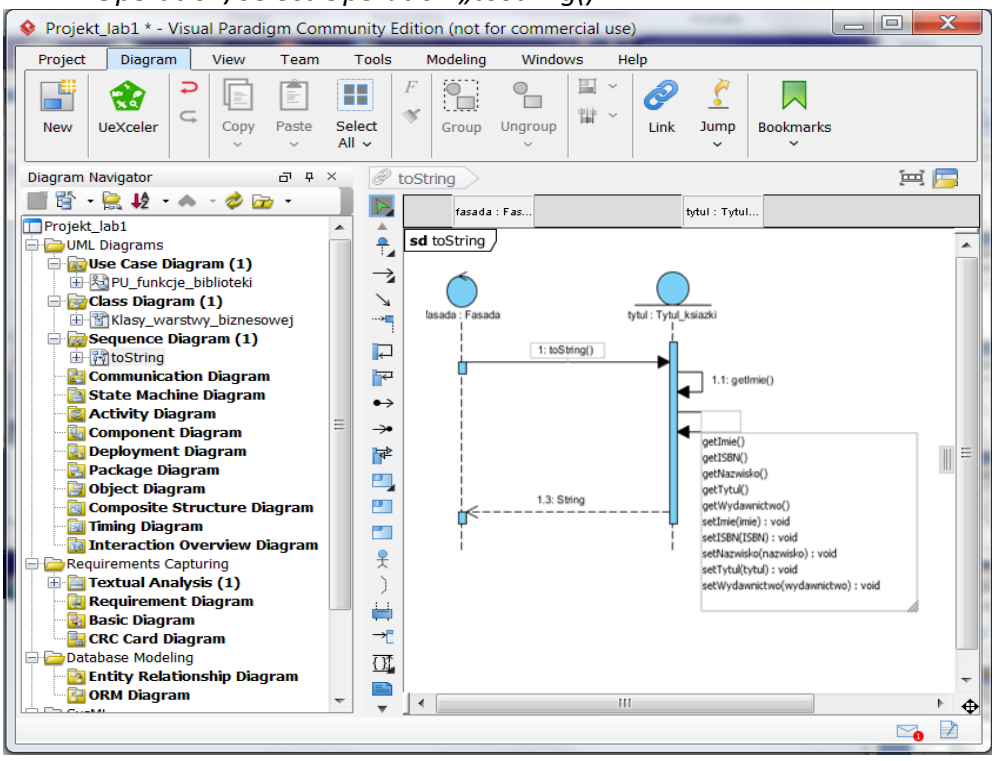
1.23. Należy wybrać z listy *Message* typ metody *Call Message* i przeciągnąć ją kładąc na linii życia **fasada** i przeciągając położyć na linii życia **tytul**. Podobnie należy zrobić z wiadomością typu *Return Message,* która powinna wychodzić z linii życia **tytul** i wchodzić do linii życia **fasada**



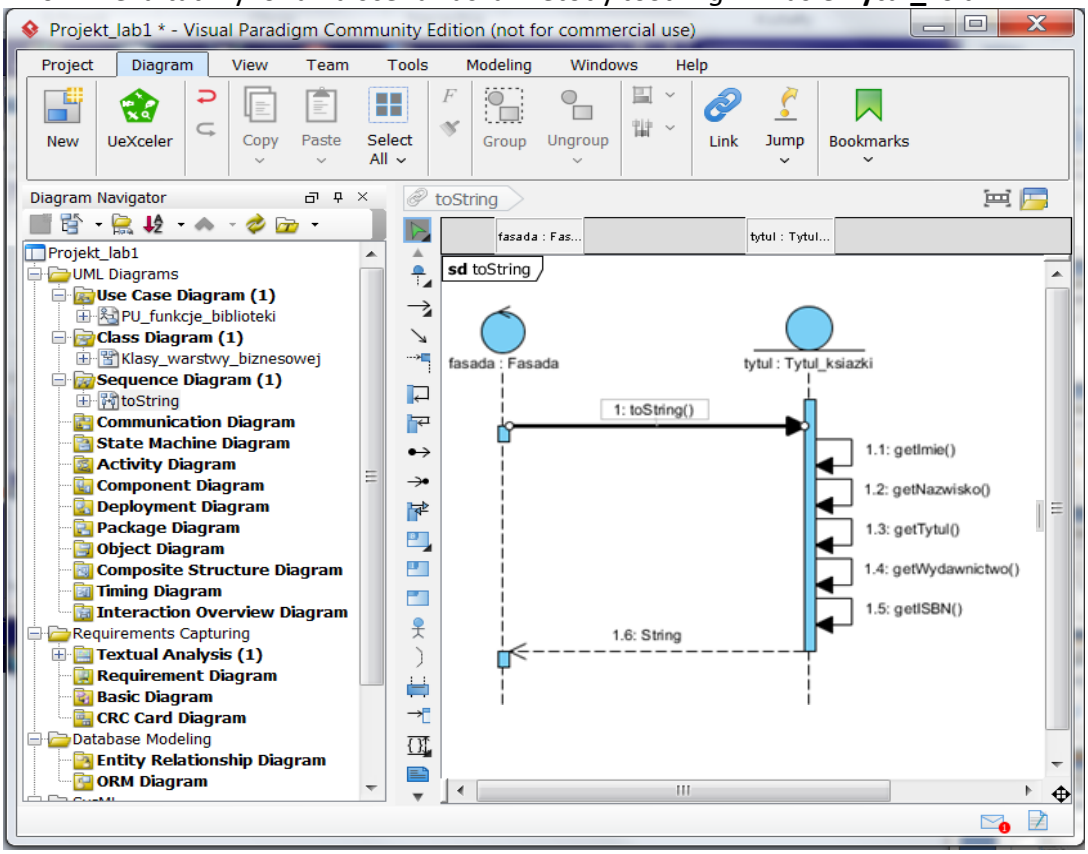
1.24. Należy zdefiniować ciało metody **toString** w klasie **Tytul_ksiazki** za pomocą wiadomości *Self Message*, przeciąganych z palety z lewej strony. Podczas wstawiania wiadomości pokazuje się lista metod klasy **Tytul_ksiazki** zdefiniowanych podczas tworzenia diagramu klas. Należy dokonać wyboru właściwej metody typu get z listy metod klasy **Tytul_ksiazki**.



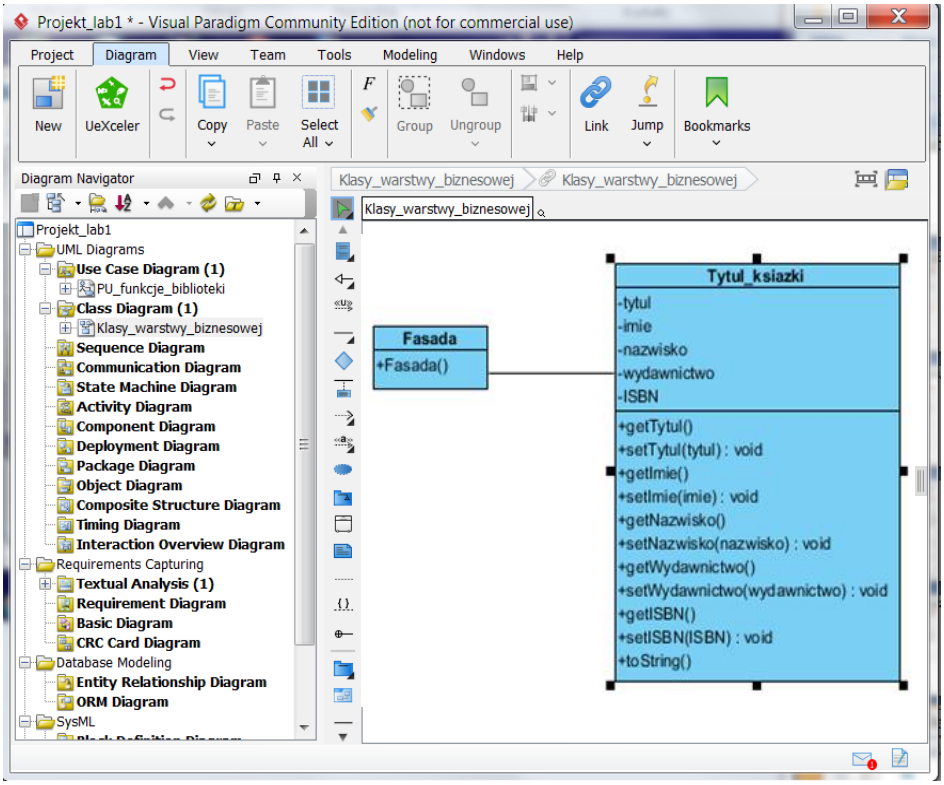
1.25. Włączenie zdefiniowanej metody do klasy **Tytul_ksiazki** – po wybraniu wiadomości o nazwie **toString** klikając prawym klawiszem myszy należy wybrać z list pozycje *Select Operation/Select Operation „toString()”*



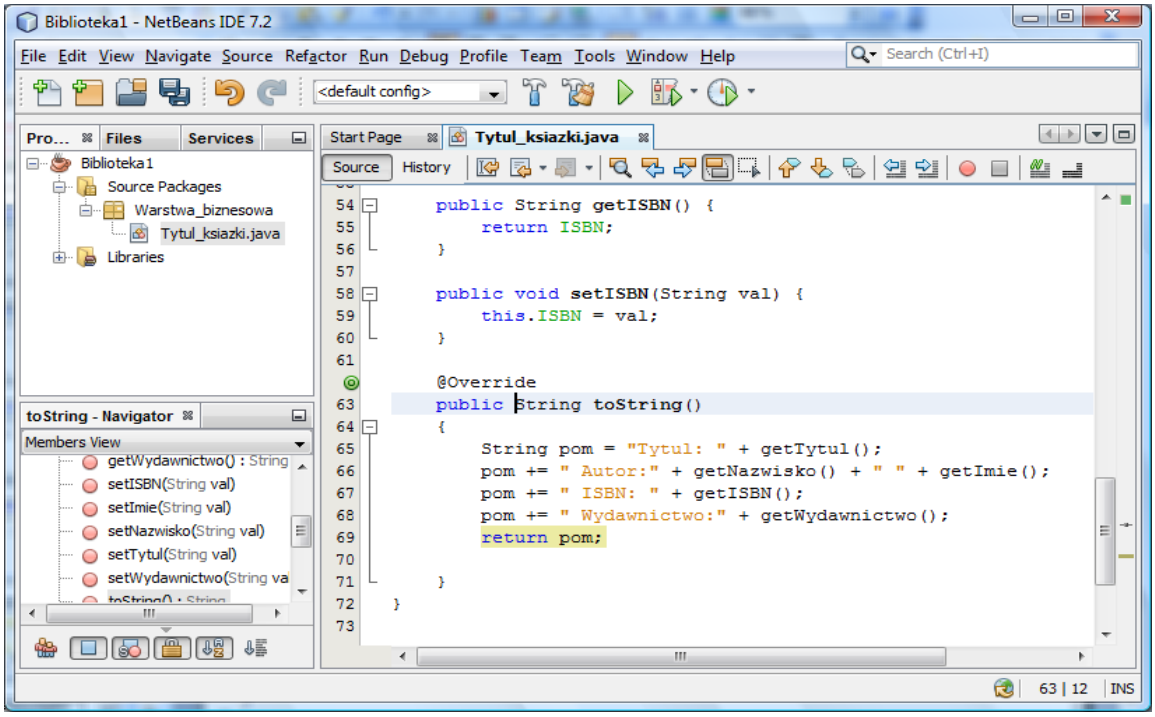
1.26. Rezultat wykonania scenariusza metody *toString* w klasie **Tytul_ksiazki**



1.27. Metoda **toString** zdefiniowana na diagramie sekwencji pojawiła się w klasie **Tytul_ksiazki**



1.28. Definicja kodu metody **toString** w klasie **Tytul_ksiazki** zdefiniowana na podstawie diagramu sekwencji (p. 11.12)



2. Dodatkowa pomoc ze strony Visual Paradigm:

3.1. Pomoc:**Drawing class diagrams.**
(http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2576/7190_drawingclass.html)

3.2. Pomoc:**Drawing sequence diagrams.**
(http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2577/7025_drawingseque.html)