Podstawy modelowania w języku UML

dr hab. Bożena Woźna-Szcześniak, prof. UJD

Uniwersytet Humanistyczno-Przyrodniczy im. Jana Długosza w Częstochowie

Wykład 1

Cel

Język UML to język modelowania systemów informatycznych i ma w inżynierii oprogramowania bardzo duże znaczenie.
W trakcie wykładu zostaną przedstawione następujące zagadnienia:

- Definicja i historia UML
- Widoki modelu 4+1 Kruchtena
- Diagramy UML - obecny standard to 2.5.1; `https://www.omg.org/spec/UML/2.5.1/`

Ćwiczenia bedą realizowane za pomocą narzędzia:

- Visual Paradigm Community Edition
- Osiąglany z `http://www.visual-paradigm.com/`

Przed przystąpieniem do ćwiczeń należy zapozanać się z wykładem.

Literatura - strony, fora, blogi itp. I

- G. Booch, J. Rumbaugh, I. Jacobson I: *The Unified Modeling Language User Guide (2nd Edition)*. Addison-Wesley Professional, 2005.
- Eric J. Naiburg, Robert A. Maksimchuk: *UML dla zwykłych śmiertelników*. PWN, 2007.
- Strona domowa UML: `http://www.uml.org/`
- Specyfikacja języka: `http://www.omg.org/spec/UML/`
- UML 2.5.1: `https://www.omg.org/spec/UML/2.5.1/`
- Tutoriale: `http://www.sparxsystems.com/uml-tutorial.html`
- Zasoby AGH: `http://zasoby.open.agh.edu.pl/~10sdczerner/page/uml.html`.

Modelowanie - zagadnienia związane

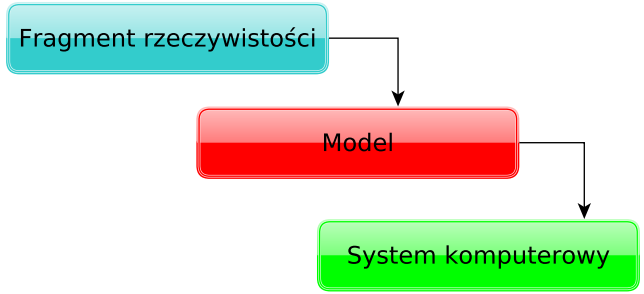
- Definicja modelu i modelowania
- Znaczenie modelu

Model, modelowanie - definicja

- *Model* przedstawia pewien fragment rzeczywistości w uproszczony, ale formalny i uporządkowany sposób.
- *Model* poprzez system założeń (dotyczących wyglądu i zachowania), pojęć (związanych z daną dziedziną i wymaganiami) oraz zależności między nimi pozwala lepiej zrozumieć złożoną rzeczywistość.
- *Modelowanie* to proces prowadzący do zdefiniowania/skonstruowania modelu.

Znaczenie modelu

- Umożliwia odzwierciedlenie/upraszczenie rzeczywistości.
- Umożliwia przejrzystą prezentację projektu.
- Pozwala zapanować nad złożonością projektu
- Umożliwia wychwycenie problemów projektowych, które mogłyby wypłynąć podczas kodowania, znacznie utrudniając pracę, bądź też powodując konieczność przeprojektowania zakodowanej już aplikacji.
- Ułatwia komunikację pomiędzy klientem i realizatorem (twórcą).
- Podnosi jakość oprogramowania: niezawodność, adaptacyjność.



Definicja UML

- UML znaczy UNIFIED MODELING LANGUAGE, czyli zunifikowany język modelowania.
- UML to znormalizowany graficzny język modelowania, służący do opisu projektu systemu informatycznego.
- UML może być stosowany do wizualizacji, specyfikowania, tworzenia, analizy i dokumentowania procesu budowy (obiektowego) systemu informatycznego.

Definicja UML według OMG

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

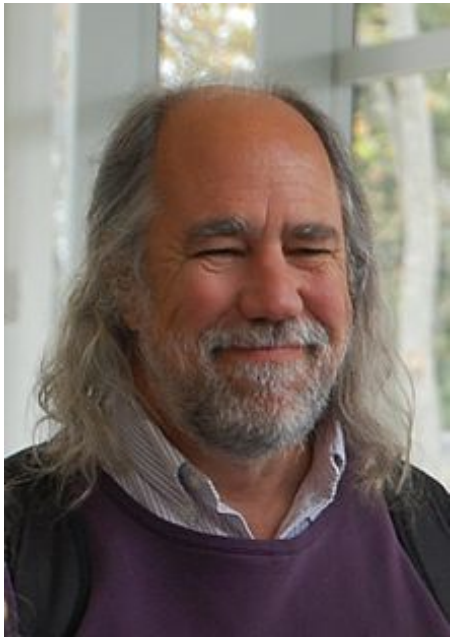
UML - zastosowania

UML znajduje zastosowanie w następujących obszarach:

- Systemy informatyczne dla przedsiębiorstw
- Bankowość i usługi finansowe
- Telekomunikacja
- Obronność, np. lotnicze systemy bojowe
- Transport
- Sprzedaż
- Nauka i badania
- Rozproszone usług internetowych
- itp.

Twórcy UML

Grady Booch



Źródło: https://en.wikipedia.org/wiki/Grady_Booch

Ivar Jacobson

Źródło: https://en.wikipedia.org/wiki/Ivar_Jacobson

James Rumbaugh

Źródło: https://en.wikipedia.org/wiki/Grady_Booch

Twórcy UML

- **Grady Booch**. Główny szef Rational Corp oraz redaktor magazynu Software Development. Opracował metodę obiektową *Object Oriented Analysis and Design* (OOAD) zwaną metodą Boocha. Więcej informacji: `https://en.wikipedia.org/wiki/Grady_Booch`
- **Ivar Jacobson**. Szwedzki naukowiec, informatyk i inżynier oprogramowania, znany jako główny twórca *Object Oriented Software Engineering* (OOSE) oraz RUP (*Objectory, Rational Unified Process*). Więcej informacji: `https://en.wikipedia.org/wiki/Ivar_Jacobson`
- **James Rumbaugh**. Amerykański naukowiec, informatyk i specjalista metodologii obiektowych. Znany jest jako twórca *Object Modeling Technique* (OMT). Więcej informacji: `https://en.wikipedia.org/wiki/James_Rumbaugh`

Historia UML I

Wyczerpujący materiał dotyczący historii UML można znaleźć w książce: James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual. Second Edition*. Addison Wesley, 2005.
Oto wybrane informacje:

- połowa lat 1990 - istnieje ponad 50 konkurencyjnych metod obiektowych analizy i projektowania oprogramowania. Powszechnie stosowane metody obiektowe to metody Grady Boocha (metoda OOAD), James Rumbaugha (metoda OMT) oraz Ivara Jacobsona (metoda OOSE):

  Opracowanie ujednoliconego języka modelowania wydaje się niezbędne.

Historia UML II

- 1994, 1995 - Grady Booch, Ivar Jacobson oraz James Rumbaugh rozpoczynają pracę w Rational Software Corporation
- Październik 1994 - firma Rational Software Corporation (od lutego 2003 część IBM) rozpoczęła oficjalne prace nad UML.
- 1995 - opublikowanie roboczej wersji UML 0.8
- 1996 - opublikowanie wersji UML 0.9; integracja metod obiektowych Boocha, metody OMT (ang. Object Modeling Technique, J. Rumbaugh), metody OOSE (ang. Object-Oriented Software Engineering, Ivar Jacobsen) oraz elementów innych istniejących metod obiektowych.

Historia UML III

- 1996 - powstaje konsorcjum firm (HewlettPackard, IBM, Microsoft, Oracle, Rational SC). Wynikiem współpracy staje się UML 1.0.
- 1997 - UML 1.0 zostaje przekazany grupie Object Management Group (OMG), która do dzisiaj zajmuje się jego rozwojem.
- Kolejne lata - OMG wypracowuje wersje 1.1, 1.2, 1.3, 1.4, 1.4.2.
  - Wersja 1.4.2 została poddana standaryzacji ISO/IEC 19501 i jest ostatnią wersję z gałęzi 1.x oznaczoną numerem 1.5.
- Czerwiec 2005 - OMG publikuje UML 2.0 łącząc wysiłki ponad stu organizacji.
- Sierpień 2007 - OMG wydaje wersje 2.1.1

Historia UML IV

- Luty 2009 - OMG publikuje UML 2.2.
- Maj 2010 - OMG publikuje UML 2.3.
- Sierpień 2011 - OMG publikuje UML 2.4.1. zostaje znormalizowana (ISO/IEC 19505-1 i 19505-2).
- Październik 2012 - OMG publikuje UML 2.5 jako wersje "In process".
- Czerwiec 2015 - OMG publikuje oficjalną wersję UML 2.5
- Grudzień 2017 - OMG publikuje oficjalną wersję UML 2.5.1

Model UML systemu

- Model UML systemu jest wyrażany przy pomocy **diagramów** przedstawiających rozmaite części i aspekty modelu.
- Diagramy różnią się:
  - *rodzajem* - różne typy diagramów odpowiadają różnym sposobom widzenia systemu
  - *stopniem szczegółowości* - każdy diagram tworzony jest w konkretnym celeu w konkretnej fazie rozwijania oprogramowania; inny poziom szczegółowości zawierać będzie konsultowany z użytkownikami diagram z fazy określania wymagań, a inny diagram mający być szczegółową specyfikacją elementu systemu, przeznaczony do automatycznej generacji kodu na jego podstawie.

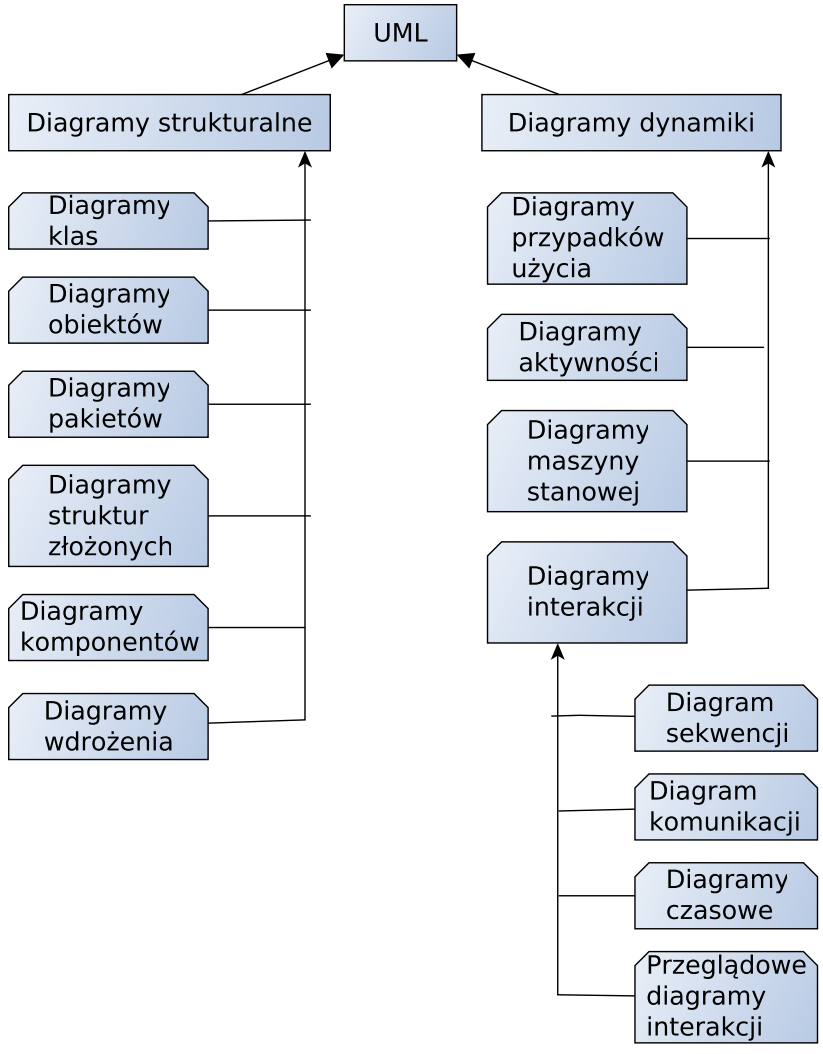
Model UML systemu

Diagramy UML pozwalają na ilustrację rozmaitych aspektów systemu:

- **struktury** - diagramy definiujące wyłącznie statyczne aspekty systemu.
- **zachowania** - diagramy definiujące wyłącznie dynamiczne aspekty systemu.
- **zachowania z uwzględnieniem struktury** - diagramy ilustrujące łącznie aspekty dynamiczne i statyczne.

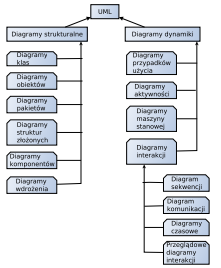
Diagramy UML - obecna specyfikacja

Obecna specyfikacja UML wyróżnia 13 rodzajów diagramów w następującej hierarchii:



Diagramy UML - obecna specyfikacja

Obecna specyfikacja UML wyróżnia 13 rodzajów diagramów w następującej hierarchii:

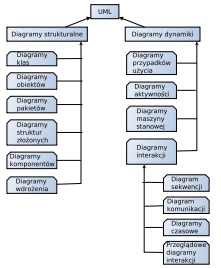


Omówione będą następujące diagramy:

- Diagramy strukturalne (ang. Structure Diagram): Diagram klas (ang. class diagram), Diagram obiektów (ang. object diagram), Diagram pakietów (ang. package diagram), Diagram struktur złożonych (ang. composite structure diagram), Diagram komponentów (ang. component diagram), Diagram wdrożenia (ang. deployment diagram).

Diagramy UML - obecna specyfikacja

Obecna specyfikacja UML wyróżnia 13 rodzajów diagramów w następującej hierarchii:



- Diagramy dynamiki/zachowania (ang. Behavior Diagram): Diagram przypadków użycia (ang. use case diagram); Diagram aktywności (czynności) (ang. activity diagram); Diagramy maszyny stanowej (ang. state machine diagram); Diagramy interakcji (ang. interaction diagram) - diagram sekwencji (ang. sequence diagram), diagram komunikacji (ang. communication diagram), diagramy czasowe (ang. timing diagram), przeglądowe diagramy interakcji (ang. interaction overview diagrams).

Elementy składowe specyfikacji UML

Istnieją cztery części specyfikacji UML 2.x:

- **Superstruktura** - definiuje notację i semantykę dla diagramów i elementów ich modelu. Istotna dla każdego, kto modeluje w UML'u i chce, aby model był poprawnie rozumiany.
- **Infrastruktura** - definiuje metamodel języka UML, na którym opiera się superstruktura. Ważna przede wszystkim dla konstruktorów oprogramowania do modelowania.
- **Język OCL** (ang. Object Constraint Language) - określa zasady działania elementów modelu.
- **Interakcja diagramów UML** - definiuje współdziałanie pomiędzy układami diagramów UML 2.x.

Perspektywa 4+1 I

Autorzy UML rozróżniają pięć perspektyw spojrzenia na system informatyczny i przyporządkowują im odpowiednie rodzaje diagramów UML:

- **perspektywa przypadków użycia** (zakres i funkcjonalość systemu) - opisuje funkcjonalność, jaką powinien dostarczać system, widzianą przez jego użytkowników, czyli opisuje zachowanie systemu obserwowane z zewnątrz; diagramy przypadków użycia, diagramy pakietów.
- **perspektywa projektowa** (logiczna, budowa systemu) - opisuje sposób realizacji funkcjonalności, strukturę systemu widzianą przez projektanta (tj. klasy, interfejsy, kooperacje); diagramy klas, obiektów, pakietów, struktur złożonych.

Perspektywa 4+1 II

- **perspektywa procesowa** (dynamiczna, zachowanie)- zawiera podział systemu na procesy (czynności) i procesory (jednostki wykonawcze) oraz opisuje właściwości pozafunkcjonalne systemu i służy przede wszystkim programistom i integratorom; diagramy aktywności (czynności), maszyny stanowej, pakietów sekwencji, komunikacji, czasowe oraz przeglądowe diagramy interakcji.
- **perspektywa implementacyjna** (software) - opisuje poszczególne moduły i ich interfejsy wraz z zależnościami; perspektywa ta jest przeznaczona dla programisty (komponenty i pliki, zarządzanie konfiguracją); diagramy komponentów, diagramy pakietów.

Perspektywa 4+1 III

- **perspektywa wdrożeniowa** (rozlokowanie, sprzęt) - definiuje fizyczny podział elementów systemu i ich rozmieszczenie w infrastrukturze, czyli dotyczy fizycznej realizacji sprzętowej systemu; perspektywa taka służy integratorom i instalatorom systemu; diagramy wdrożenia, diagramy pakietów.

Diagram klas

- Diagramy klas przedstawiają statyczny widok modelu, lub jego części.
- Diagramy klas przedstawiają strukturę projektowanego systemu, lub jego części jako zbiór klas i interfesów wraz z ich atrybutami, funkcjami, ograniczeniami oraz powiązaniami między nimi.

Diagram klas

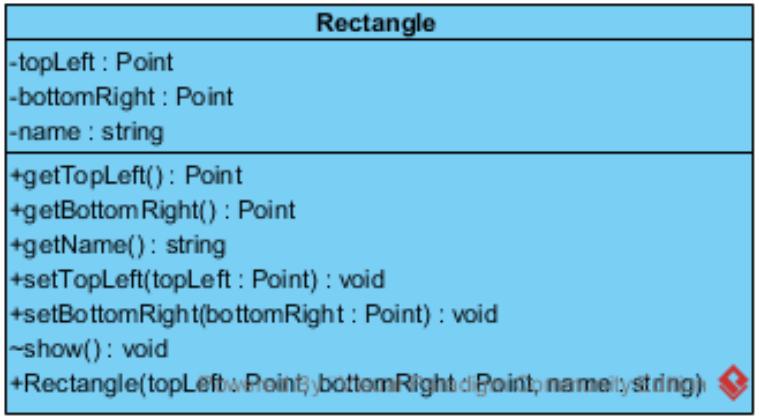
Definicja

- **Klasa jest elementem, który określa cechy (własności) i zachowanie, które obiekt jest w stanie wygenerować.**
- **Zachowanie opisane jest przy pomocy komunikatów wraz z operacjami, które są odpowiednie dla każdego komunikatu.**
- **Klasy mogą mieć również definicje ograniczeń, oznaczonych wartości i stereotypów.**

Notacja klas

Klasa jest reprezentowana przez prostokąt z wydzielonymi przedziałami na:

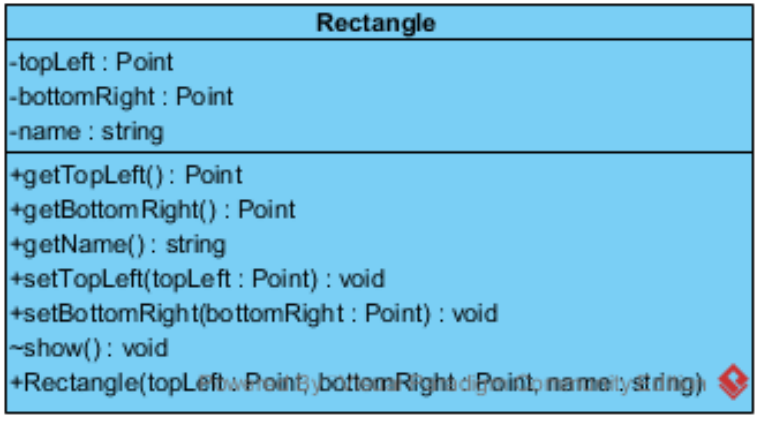
- nazwę
- atrybuty
- operacje (metody).



Notacja klas

Dostępność metod lub atrybutów:

- + **publiczna** - element jest widoczny z każdego miejsca w systemie
- # **chroniona** - element jest widoczny we własnej klasie i jej podklasach



- – **prywatna** - element jest widoczny tylko we wlasnej klasie
- ~ **publiczny wewnątrz pakietu** - element jest widoczny tylko wewnątrz własnego pakietu

Operacje (metody)

Nazwy operacji mogą wyglądać następująco:

```
[widoczność] nazwa [(parametry)]
[: typ wartości zwracanej] [{ustawienia}]
```

gdzie parametry:

```
nazwa [: typa parametru]
```

Poprawne nazwy metod to:

```
display
+display
+display()
+getPosition : Point
+getPosition(): Point
+setPosition(pos: Point)
+setPosition(pos: Point): void
```

Interfejs (Klasy abstrakcyjne)

Definicja

- Interfejs to klasa, która posiada jedną lub więcej metod (operacji) nieposiadających ciała, tzw. metod abstrakcyjnych (wirtualnych).
- Klasa, w której przynajmniej jedna metoda jest abstrakcyjna musi być zadeklarowana jako abstrakcyjna.
- Metody nieposiadające ciała są jedynie deklaracjami, zapowiedziami, że klasa dziedzicząca po interfejsie (klasie abstrakcyjnej) dostarczy ciała takiej metody, w przeciwnym razie sama też będzie klasą abstrakcyjną.
- Uwaga! nie można tworzyć instancji (obiektów) klas abstrakcyjnych.

Przykład klasy abstrakcyjnej w Java I

Oto dobrze znany kod *dr hab. Andrzeja Zbrzeznego, prof. UJD* z "Podstaw programowania w Javie".

```
import java.util.*;

public class TestOsoba {
 public static void main(String[] args) {
  Osoba[] ludzie = new Osoba[2];
  ludzie[0] = new Pracownik("Jan Kowalski", 50000);
  ludzie[1] = new Student("Maria Nowak", "informatyka");
  for (Osoba p : ludzie) {
    System.out.println(p.getNazwisko() + ": " + p.getOpis());
  }
 }
}

abstract class Osoba {
  public Osoba(String nazwisko){
   this.nazwisko = nazwisko;
  }
  public abstract String getOpis();
```

Przykład klasy abstrakcyjnej w Java II

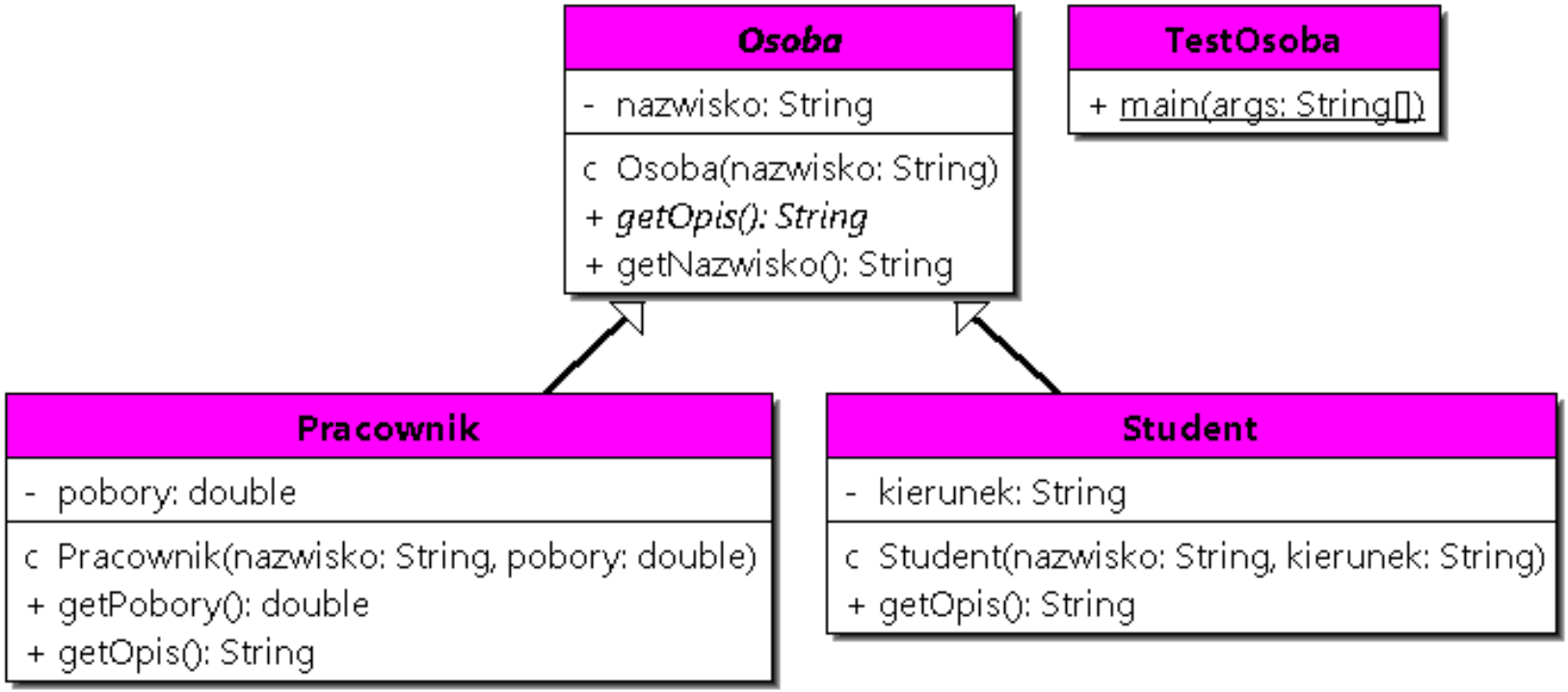
```
  public String getNazwisko() {
   return nazwisko;
  }
  private String nazwisko;
}

class Pracownik extends Osoba {
  public Pracownik(String nazwisko, double pobory) {
   super(nazwisko);
   this.pobory = pobory;
  }
  public double getPobory() {
   return pobory;
  }
  public String getOpis() {
   return String.format("pracownik z pensją %.2f zł", pobory);
  }
  private double pobory;
}
```

Przykład klasy abstrakcyjnej w Java III

```
class Student extends Osoba {
  public Student(String nazwisko, String kierunek){
    super(nazwisko);
    this.kierunek = kierunek;
  }
  public String getOpis() {
    return "kierunek studiów: " + kierunek;
  }
  private String kierunek;
}
```

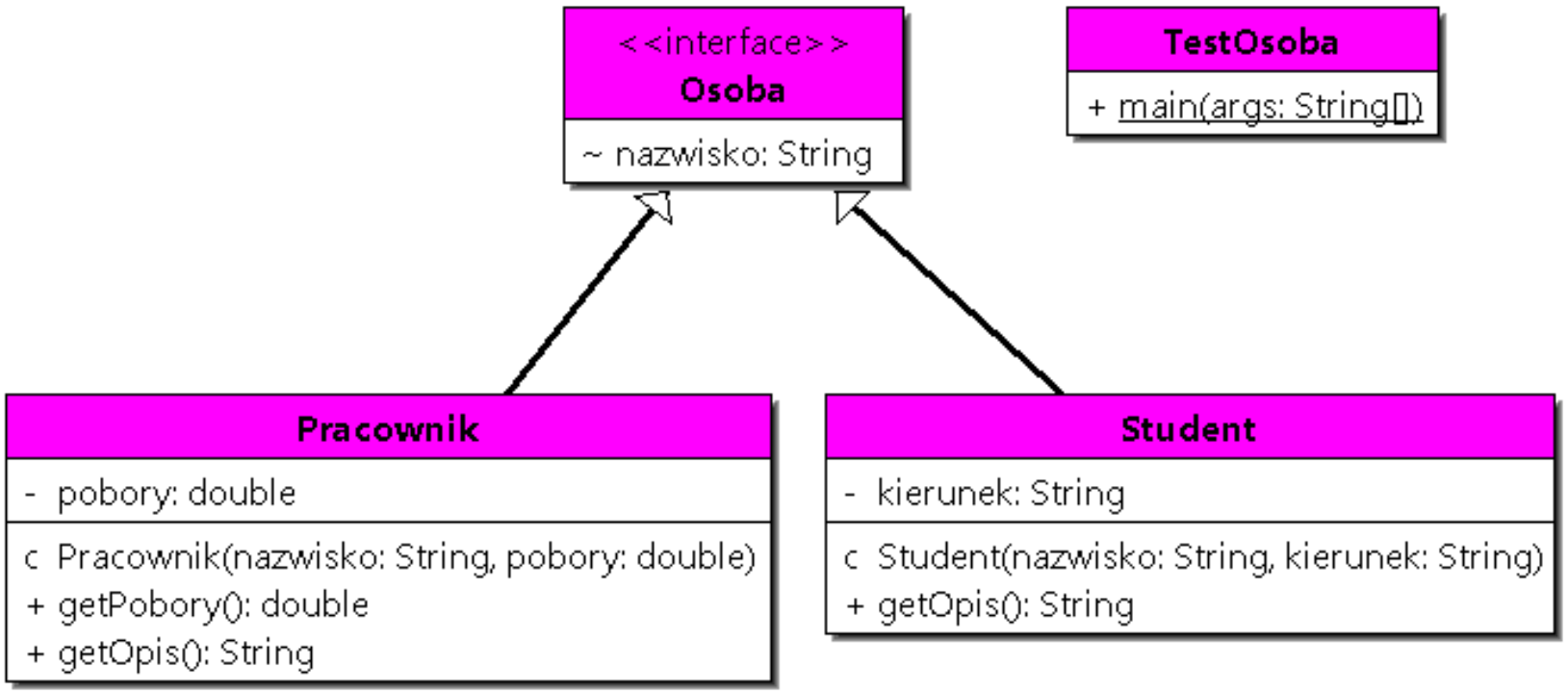
Interfejs (Klasy abstrakcyjne)

W UML-u klasy abstrakcyjne niewiele różnią się od normalnych klas. Jedyną widoczną różnica jest ich nazwa, napisana kursywą.



Interfejs

Klasy abstrakcyjne mogą być również wizualizowane z użyciem stereotypu «interface»



Interfejs

- Interfejs wymaga, aby klasa realizująca (u nas klasa Osoba) go dostarczyła implementacji wszystkich określonych w nim operacji. Co więcej, operacje te muszą w klasie mieć takie same nazwy jak w interfejsie.
- Połączenie pomiędzy Interfejsem a klasą realizującą przedstawiane jest na diagramie za pomocą strzałki z przerywaną linią i niezamalowanym grotem.
- W przypadku, gdy interfejs prezentowany jest w postaci kuli, związek realizacji pomiędzy klasą a interfejsem przedstawia się za pomocą linii ciągłej.

Związki między klasami

- Asocjacja (ang. Associations)
- Uogólnienie, dziedziczenie (ang. Generalizations)
- Agregacja (ang. Aggregations)
- Kompozycja (ang. Composite aggregation)
- Zagnieżdżenia (ang. Nestings)

Realizacja na kolejnym wykładzie ....