Wszystkie wykłady

Spis treści:

Mini-projekty
Projekt
Terminy
Zaliczenie ćwiczeń
Plagiat
Egzamin
Materiały uzupełniające
Narzędzia implementacyjne

# Wprowadzenie do przedmiotu MAS (Modelowanie i Analiza Systemów)

*Autor: dr inż. Mariusz Trzaska*

## 1. Uzasadnienie

Współczesne języki programowania są w większości oparte o paradygmat obiektowy. W związku z tym jego praktyczne zrozumienie jest niezbędną podstawą do pracy w tym zawodzie. Co więcej, programiści często tworzą systemy w oparciu o materiały analityczne. A te w dużej mierze wykorzystują obiektowe notacje graficzne (np. UML). W trakcie tej i następnych lekcji, pogłębisz swoją wiedzę teoretyczną, ale przede wszystkim poznasz sposoby implementacji różnych pojęć z obiektowości. Dzięki temu będziesz w stanie projektować oraz implementować nowoczesne aplikacje biznesowe.

## 2. Wprowadzenie

Przedmiot poświęcony jest osadzeniu modelu pojęciowego dziedziny problemowej, efektu fazy analizy i specyfikacji wymagań, w konkretnym środowisku implementacyjnym (zarówno obiektowym jak i relacyjnym). Studenci poznają sposoby realizacji konstrukcji, niezbędnych do osadzenia modelu, a nie istniejących w wybranym języku programowania. Dyskutowane są również elementy związane z użytecznością (w tym jej testowaniem) graficznych interfejsów użytkownika. Wiedza teoretyczna poparta jest praktyczną implementacją struktury danych, logiki biznesowej oraz prostych i zaawansowanych graficznych interfejsów użytkownika (m. in. przy wykorzystaniu dedykowanych edytorów). Każdy ze studentów jest zobowiązany do przeprowadzenia analizy dynamicznej oraz wykonania prac projektowych i implementacyjnych, w oparciu o indywidualne wymagania użytkownika. Specyfikacja wymagań i analiza statyczna powinny być przeprowadzone w trakcie nauczania przedmiotu Projektowanie Systemów Informacyjnych (PRI). Analogicznie studenci przystępujący do kursu MAS powinni umieć programować w jakimś obiektowym języku (domyślnie Java).

## 3. Mini-projekty

Celem mini-projektów jest praktyczne sprawdzenie zrozumienia sposobów implementacji poszczególnych konstrukcji z modelu pojęciowego (klasy, asocjacje, ekstensja, itp.) oraz współpracy z modelem relacyjnym. Dodatkowo można je potraktować jako „fundament” projektu końcowego (który później zostanie uzupełniony o odpowiednie elementy, m. in. GUI). Należy zaimplementować różne biznesowe konstrukcje (odpadają wszelkie techniczne informacje jak np. liczba obiektów w ekstensji, settery/gettery, wyświetlanie obiektów, itp.) występujące w modelu pojęciowym. Nie wolno łączyć przykładów, tzn. każda konstrukcja musi mieć swój własny przykład biznesowy. Zakres ocenianych zagadnień jest przedstawiony w poniższej tabeli:

| MP1<br>Klasy, atrybuty | MP2<br>Asocjacje | MP3<br>Dziedziczenie | MP4<br>Ograniczenia | MP5<br>Model relacyjny (MR) |
|---|---|---|---|---|
| • Ekstensja<br>• Ekstensja - trwałość<br>• Atrybut | • Binarna<br>• Z atrybutem<br>• Kwalifikowana<br>• Kompozycja | • Disjoint<br>• Klasa abstrakcyjna<br>• Polimorficzne wołanie metody | • Atrybutów<br>• Unique<br>• Subset<br>• Ordered | • MR - klasy<br>• MR - asocjacje (liczności 1:* |

| | | | | |
|---|---|---|---|---|
| złożony<br>• Atrybut opcjonalny<br>• Atrybut powtarzalny<br>• Atrybut klasowy<br>• Atrybut pochodny<br>• Metoda klasowa<br>• Przesłonięcie<br>• Przeciążenie | Każda z asocjacji musi mieć liczność co najmniej 1-* oraz automatyczne tworzenie połączenia zwrotnego. | • Overlapping<br>• Wielodziedziczenie<br>• Wieloaspektowe<br>• Dynamiczne | • Bag<br>• Xor<br>• Ograniczenie Własne | lub *:*)<br>• MR - dziedziczenie<br><br>Należy stworzyć działający program korzystający z bazy danych i realizujący powyższe konstrukcje. Można korzystać z dowolnych narzędzi, m. in. ORM-ów, np. Hibernate, Entity Framework. W archiwum powinny znaleźć się tylko pliki źródłowe (bez binarnych/bibliotek) ewentualnie razem z mapującymi (dla ORM-ów). Oczywiście należy załączyć odpowiednie przykłady użycia. |

Mini-projekty są częścią składową oceny końcowej. W związku z tym warto je dobrze wykonać. Oddajemy je w postaci archiwum 7z/ZIP/RAR zawierającego (prosta **aplikacja konsolowa**):

1. kody źródłowe z komentarzami (opisującymi zrealizowane konstrukcje, API oraz poszczególne kroki implementacji). Należy przestrzegać wytycznych dla używanej platformy, np. Java wymaga oddzielnych plików dla każdej publicznej klasy. Nie trzeba dołączać plików binarnych (bibliotek, obrazków, itp.);
2. **jedną klasę (plik)** *Main.java* (*main.cs*, itd.) z metodą `main` zawierającą skomentowane przykłady wykorzystania zrealizowanych konstrukcji, np.:

   ```
   1 pracownik1.setAdres(adres2); //atrybut złożony
   ```

   **Tylko** skomentowane elementy zaprezentowane w w/w klasie będą oceniane. Projekty bez tej klasy **nie będą oceniane**;

Plik archiwum (*MPx_Nazwisko_Imię_NrIndeksu.zip* gdzie x jest numerem mini-projektu) należy umieścić w odpowiednim folderze systemu EDU. Ostateczny termin – patrz dalej.

## 4. Projekt

Projekt składa się z dwóch części: dokumentacyjnej oraz implementacyjnej. Jego tematyka powinna być tak dobrana, aby dało się stworzyć odpowiednią liczbę (12 - 15) sensownych klas biznesowych. W związku z tym raczej odpadają wszelkie aplikacje narzędziowe, systemowe, odtwarzacze multimediów, itp. W razie wątpliwości należy skonsultować się z prowadzącym ćwiczenia.

### 4.1 Dokumentacja projektowa

1. Dokumentacja zawiera część „starą”, czyli tą, która została wyprodukowana na przedmiocie PRI (w postaci załącznika) oraz dokumentację „nową”. Dokumentacja „nowa” powiela schemat dokumentacji „starej”, ale dochodzi tu też trochę nowych elementów, o których poniżej. Osoby, które nie posiadają „starego” projektu z PRI muszą opracować co najmniej: wymagania użytkownika (jako „historyjkę”), diagram przypadków użycia oraz analityczny diagram klas.
2. Przypadki użycia: „Nowa” dokumentacja przypadków użycia powinna zawierać oprócz diagramu powielonego ze „starej” dokumentacji szczegółowy opis jednego nietrywialnego (odwołującego się do innego przypadku użycia) przypadku użycia (zgodnie z materiałem podawanym na wykładzie z PRI). Scenariusz, dla tego przypadku, powinien być sporządzony zarówno z wykorzystaniem języka naturalnego, jak i diagramów aktywności.
3. Projekt interfejsu użytkownika: W oparciu o wybrany, nietrywialny przypadek użycia powinien być sporządzony projekt interfejsu użytkownika (zgodnie z wytycznymi podanymi na wykładzie).
4. Dla wybranego przypadku użycia należy przeprowadzić analizę dynamiczną (czyli wykorzystać diagramy: aktywności i stanu ). Analiza dynamiczna powinna zakończyć się nie tylko pojawieniem się metod na diagramie klas, ale też jawnym omówieniem jej skutków. „Skutki” analizy dynamicznej (być może nie tylko metody, ale też nowe atrybuty, nowe asocjacje, itp.) powinny być umieszczone na diagramie klas przerysowanym ze „starego” projektu i w miarę możliwości wyróżnione innym kolorem. Projekt musi wykorzystywać wszystkie wymienione rodzaje diagramów.
5. Przed „ostatecznym” diagramem klas, stanowiącym podstawę do implementacji, należy koniecznie dołączyć elementy specyfikujące podjęte decyzje projektowe (na przykładach, w oparciu o odpowiednie fragmenty diagramu), np. sposób realizacji ekstensji klasy.
6. Podsumowując, „ostateczny” (projektowy/implementacyjny) diagram klas różni się od diagramu dostarczonego na PRI w następujących punktach:
    1. jest uszczegółowiony,
    2. wszystkie konstrukcje nie istniejące w danym języku programowania są zamienione (zgodnie z podjętymi decyzjami projektowymi),
    3. jest uzupełniony o metody wynikłe z analizy dynamicznej.
7. Należy zadbać o **czytelność całej dokumentacji**, a szczególnie diagramów (zalecany jest jakiś format wektorowy (np. *Enhanced Metafile*). W tym celu warto upewnić się, że:
    1. prawidłowo stosujemy notację UML (a nie „podobne” elementy graficzne),
    2. każdy z nich jest prawidłowo podpisany,
    3. został przygotowany w odpowiednim narzędziu (np. UMLet),
    4. stosujemy rozmiar czcionek niewymagający nadmiernego powiększania (**czytelny diagram A4 w widoku 100%, np. na wydrukowanej stronie**; zwiększ czcionkę na diagramie w razie potrzeby),
    5. nie dzielimy diagramów na części,
    6. elementy (np. klasy) są właściwe rozplanowane, m. in. dziedziczenie w pionie, asocjacje w poziomie, unikamy przecinania linii, asocjacje mają nazwy i liczności.
8. Dokumentację projektową MAS oddajemy w postaci pojedynczego pliku PDF (nazwa: *MAS_Projekt_Nazwisko_Imię_NrIndeksu.PDF*) umieszczonego w odpowiednim folderze systemu EDU. Ostateczny termin – patrz dalej.
9. Podsumowanie zawartości dokumentacji projektowej (PRI + MAS):
    1. Wymagania użytkownika
    2. Diagram przypadków użycia
    3. Diagram klas analityczny
    4. Diagram klas projektowy
    5. Scenariusz przypadku użycia (jako tekst w punktach)
    6. Diagram aktywności dla przypadku użycia
    7. Diagram stanu dla klasy
    8. Projekt GUI
    9. Omówienie decyzji projektowych i skutków analizy dynamicznej

### 4.2 Implementacja projektu

1. Cała struktura (klasy z odpowiednimi powiązaniami),
2. Metody potrzebne do realizacji wybranego przypadku (lub przypadków) użycia,

3. Elementy graficznego interfejsu użytkownika (GUI), które są niezbędne do zaprezentowania pracującej implementacji z działającym wybranym przypadkiem użycia. Każdy projekt musi posiadać GUI.
4. Minimalna implementacja GUI **musi** obejmować interakcję co najmniej dwóch klas połączonych asocjacją (wymagana liczność docelowa: „wiele"), np. mamy dwie klasy: *Pracownik* i *Firma*; odpowiedni *widget* (wyświetlający wiele elementów, np. *ListBox*) zawiera listę firm, po kliknięciu w dowolną pozycję wyświetla się inny *widget* (wyświetlający wiele elementów, np. *ListBox*) zawierający listę jej pracowników pobraną **za pomocą zdefiniowanej aoscjacji** (przeważnie oznacza to brak stosowania SQL-a). Zaimplementowane GUI, które tylko tworzy połączenia pomiędzy obiektami i nie pozwala na w/w interakcję, **nie wystarczy do zaliczenia**. Analogicznie, **niewystarczające** są np. rozwiązania z jednym *widget*'em, *TextBox*'em, licznością docelową „1" lub filtrujące dane z ekstensji (zamiast korzystania z uprzednio zdefiniowanej asocjacji).
5. Implementacja musi zawierać **przykładowe dane** ilustrujące poprawne działanie.
6. Należy zwrócić uwagę na jakość i ergonomię GUI (np. skalowanie okien, kolorystyka, filozofia działania) – jest to ważny składnik oceny końcowej. Projekt i wykonanie GUI (można używać dedykowanych edytorów) powinno być zgodne z zasadami użyteczności (usability), podawanymi na wykładzie.
7. **Wszystkie** dane (a nie tylko te z przypadków użycia) przechowywane w systemie muszą być trwałe (np. plik, baza danych, dedykowana biblioteka, itp.).
8. Część implementacyjna projektu będzie indywidualnie odbierana w czasie zajęć (patrz dalej). Kody źródłowe trzeba wgrać do systemu Gakko zgodnie z harmonogramem.
9. Językiem implementacji może być Java, C# lub C++. Inne języki wymagają zgody prowadzącego ćwiczenia.
10. Każdy projekt będzie **indywidualnie odbierany w terminie** podanym w ogłoszeniu kursu (do kilku dni przed egzaminem). W trakcie odbioru można spodziewać się szczegółowych pytań dotyczących sposobu implementacji, np. *„co by było gdyby..."*, *„dlaczego jest to zrobione w ten sposób..."*, *„proszę dokonać następującej modyfikacji..."*. Osoby, które samodzielnie wykonały projekt nie powinny mieć problemów z udzieleniem odpowiedzi na powyższe pytania. Brak umiejętności odpowiedzi na powyższe pytania oznacza **brak zaliczenia ćwiczeń**.
11. Rozwiązania niespełniające ww. warunków, **nie zaliczą ćwiczeń**.
12. Kryteria oceny: patrz dalej.

## 5. Terminy

Szczegółowe informacje dotyczące terminów znajdują się w opisie kursu. Prace oddane po terminie będą miały obniżoną punktację (50% dla mini-projektów).

## 6. Zaliczenie ćwiczeń

Ocena końcowa z ćwiczeń składa się z następujących składowych:

1. Punkty za MPx (20 + 20 + 20 + 16 + 24 = 100 pkt.; w sumie trzeba uzyskać min. 50%)

   Za każde zagadnienie wymienione w tabeli dla poszczególnych MP, można otrzymać liczbę punktów obliczoną na podstawie wzoru: (Liczba punktów dla konkretnego MP) / (Liczba zagadnień w danym MP), np. dla MP1 za każdą konstrukcję można otrzymać 20 / 10 = 2 pkt. (razem: 10 x 2 pkt. = 20 pkt.)

   Liczba przyznanych punktów, za każde zagadnienie, będzie uwzględniała poprawność rozwiązania, wybór metody, czytelność kodu.

2. Ocena z projektu (należy uzyskać minimum 50% z każdej części):

    1. Ocena za implementację będzie wystawiana zgodnie z poniższą tabelą (warunek wstępny: spełnienie wymagań z pkt. 4.2):

| Kryterium | Maks. pkt. |
|---|---|
| Skomplikowanie, zakres i poprawność zrealizowanej funkcjonalności | 20 |
| Zakres i poprawność zrealizowanych konstrukcji z obiektowości | 25 |
| Jakość kodu (nazewnictwo, struktura, komentarze) | 5 |
| Elegancja zaimplementowanych rozwiązań | 15 |
| Sposób implementacji trwałości | 10 |
| Implementacja GUI (w tym użyteczność/ergonomia) | 20 |
| Prezentacja projektu | 5 |
| **Razem** | **100** |

Na podstawie powyższej punktacji wystawiana jest ocena w oparciu o następującą skalę: 50 pkt. → 3,0; 60 pkt. → 3,5; 70 → 4,0; 80 → 4,5; 90 → 5,0.

2. Ocena za dokumentację będzie wystawiana zgodnie z poniższą tabelą (warunek wstępny: spełnienie wymagań z pkt. 4.1):

| Kryterium | Maks. pkt. |
|---|---|
| Skomplikowanie dziedziny biznesowej | 10 |
| Udokumentowanie przypadku (ów) użycia (scenariusz i diagram) | 10 |
| Poprawność i złożoność projektowego diagramu klas | 35 |
| Poprawność i złożoność diagramu aktywności | 10 |
| Poprawność i złożoność diagramu stanu | 10 |
| Projekt GUI | 10 |
| Omówienie decyzji projektowych | 10 |
| Czytelność i organizacja dokumentu | 5 |
| **Razem** | **100** |

Na podstawie powyższej punktacji wystawiana jest ocena w oparciu o następującą skalę: 50 pkt. → 3,0; 60 pkt. → 3,5; 70 → 4,0; 80 → 4,5; 90 → 5,0.

Z każdej części trzeba otrzymać ocenę pozytywną. W związku z tym osoby, które np. zaliczą mini-projekty, a nie zaliczą projektu nie zaliczą ćwiczeń.

Dodatkowo można zdobyć 5% całości pkt. za rozwiązanie każdego z zadań podanych w wykładach. Te bonusowe punkty doliczane są do ogólnej liczby punktów pod warunkiem posiadania, co najmniej 50% pkt.

## 7. Plagiat

Wszystkie kody źródłowe podlegające ocenie (mini-projekty, projekt końcowy) będą przechodziły **procedurę antyplagiatową**. W przypadku wykrycia zapożyczeń z innych programów:

1. wszystkie programy, w których wykryto wspólny kod, **otrzymują 0 pkt.**;
2. **nie ma możliwości przysłania poprawionej wersji takiego programu**, co może prowadzić do **niezaliczenia ćwiczeń** bez szansy na ich poprawę w bieżącym semestrze;
3. nie będą prowadzone ustalenia, kto, od kogo kopiował kody źródłowe.

## 8. Egzamin

Egzamin z MAS składa się z dwóch części (liczy się suma punktów):

1. Testowej. Należy ocenić każde z pytań (T/N). Odpowiedź prawidłowa oznacza +2 pkt., błędna -2 pkt., brak odpowiedzi 0 pkt.
2. Zadaniowej. Należy nazwać oraz krótko omówić sposób implementacji zaznaczonych konstrukcji na otrzymanym diagramie klas.

Nie ma zwolnień z egzaminu.

Przykładowy egzamin: http://www.mtrzaska.com/mas-egzamin.

## 9. Materiały uzupełniające

1. **Książka: M. Trzaska: „Modelowanie i implementacja systemów informatycznych". Wydawnictwo PJATK. Stron 299. ISBN 978-83-89244-71-3.**
   - Wersja drukowana: sklep PJWSTK.
   - Fragmenty w PDF: http://www.mtrzaska.com/mas-ksiazka
2. Wersja elektroniczna wykładów dla studiów stacjonarnych
3. Przykładowe zadania programistyczne
   Niniejsze zadania programistyczne mają na celu jedynie przypomnienie materiału dotyczącego programowania (przyswojonego w czasie wcześniejszych kursów) i ich rozwiązania nie będą oceniane w ramach przedmiotu MAS. Studenci przystępujący do kursu MAS, powinni umieć rozwiązać zdecydowaną większość z nich.
4. Bezpłatne książki on-line:
   - Bruce Eckel - Thinking in Java: http://www.mindview.net/Books/TIJ/
   - Allen B. Downey - How to Think Like a Computer Scientist: Java Version: http://www.greenteapress.com/thinkapjava/
   - Robert Sedgewick and Kevin Wayne - Introduction to Programming in Java: An Interdisciplinary Approach: http://introcs.cs.princeton.edu/home/
5. Wszystkie pozycje związane z modelowaniem pojęciowym (w oparciu o język UML) – odpowiedni spis można znaleźć w opisie przedmiotu Projektowanie Systemów Informacyjnych (PRI),
6. Pozycje dotyczące programowania w wybranym dla realizacji celów przedmiotu środowisku implementacji (aktualnie Java lub MS C#).

## 10. Narzędzia implementacyjne

Ze względu na fakt, iż istnieje dość duża dowolność wyboru technologii realizacji projektu, nie ma listy narzędzi obowiązkowych. Niemniej poniżej umieszczono listę narzędzi, które mogą być przydatne:

- Narzędzia CASE (edytory diagramów):
  - UMLet (*open-source*, wieloplatformowy): https://www.umlet.com//,
  - NetBeans for Java: http://www.netbeans.org/ (umożliwia m.in. tworzenie diagramów UML i generowanie kodu źródłowego)
  - Modelio (*open-source*, wieloplatformowy): https://www.modelio.org/
  - Visual Paradigm Community Edition: http://www.visualparadigm.com
  - ArgoUML: http://argouml.tigris.org/
  - MagicDraw Community Edition: http://www.magicdraw.com
  - StarUML: http://staruml.sourceforge.net/en/
  - MS Visio (dostępny na licencji ELMS dla studentów PJWSTK),
  - Obszerna lista narzędzi: http://en.wikipedia.org/wiki/List_of_UML_tools
- IDE
  - IntelliJ IDEA Community Edition: http://www.jetbrains.com/idea/
  - MS Visual Studio (darmowa edycja)
  - Eclipse for Java: http://www.eclipse.org/
  - NetBeans for Java: http://www.netbeans.org/
- Edytory GUI
  - wbudowany w NetBeans;
  - dla Eclipse: Jigloo SWT/Swing GUI Builder (http://www.cloudgarden.com/jigloo/);
  - dla Eclipse: WindowBuilder Pro po przejęciu przez Google jest darmowy (http://code.google.com/intl/pl/webtoolkit/tools/wbpro);
  - wbudowany w MS Visual Studio.

Większość pojęć teoretycznych poruszanych na przedmiocie MAS została dokładnie omówiona na przedmiocie PRI. Na MAS-ach zajmujemy się ich praktycznym wykorzystaniem (implementacją). W związku z tym osoby, które mają problemy z ich zrozumieniem powinny zacząć od przypomnienia sobie materiału z PRI.

Niemniej, w miarę możliwości, będziemy podawali i omawiali niezbędne definicje.

To samo dotyczy programowania w języku Java.

INFORMATYKA ZRÓWNOWAŻONEGO ROZWOJU
Kurs e-learningowy finansowany ze środków funduszy norweskich i funduszy EOG, pochodzących z Islandii, Liechtensteinu i Norwegii, oraz środków krajowych