Modelowanie i Analiza Systemów informacyjnych (MAS)

Studia stacjonarne i zaoczne. Wydział informatyki, Wydział zarządzania

wersja z 2026-03-09

Wykładowca: **dr inż. Mariusz Trzaska**
(*mtrzaska@pjwstk.edu.pl*, *http://www.mtrzaska.com*)

1. 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 nieistniejących 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).

2. Plan zajęć

2.1. Plan zajęć (studia dzienne, 15 zajęć)

Nr	Wykład	Ćwiczenia
1	Projektowanie i modelowanie architektury systemu	• Krótkie omówienie założeń przedmiotu oraz projektu • Ćwiczenia z analizy dynamicznej. Ćwiczenia polegają na rysowaniu diagramów na podstawie wymagań. Można skorzystać z dedykowanego narzędzia CASE lub tradycyjnej tablicy.
2	Wybrane konstrukcje obiektowych języków programowania (1)	Ćwiczenia z wybranych konstrukcji obiektowych języków programowania. Ćwiczenia polegają na tworzeniu programów z wykorzystaniem różnych konstrukcji. Ich rodzaj oraz zakres jest wyznaczony stopniem zaawansowania

		danej grupy (jest ustalany przez prowadzącego ćwiczenia) i może być oparty na zawartości wykładu.
3	Wybrane konstrukcje obiektowych języków programowania (2)	Ćwiczenia z wybranych konstrukcji obiektowych języków programowania (2).
4	Wykorzystanie klas w obiektowych językach programowania	Ćwiczenia z wykorzystania klas w obiektowych językach programowania.
5	Realizacja asocjacji w obiektowych językach programowania	• Ćwiczenia z realizacji asocjacji w obiektowych językach programowania (1). • **Odbiór mini projektu MP1** (materiał z wykładu dotyczącego klas)
6	Realizacja asocjacji w obiektowych językach programowania (2)	Ćwiczenia z realizacji asocjacji w obiektowych językach programowania (2).
7	Realizacja różnych modeli dziedziczenia w obiektowych językach programowania	• Ćwiczenia z realizacji różnych modeli dziedziczenia w obiektowych językach programowania. • **Odbiór mini projektu MP2** (materiał z wykładu dotyczącego asocjacji)
8	Realizacja pozostałych, wybranych konstrukcji UML w obiektowych językach programowania	Ćwiczenia z realizacji różnych konstrukcji UML w obiektowych językach programowania (ograniczenia).
9	Wykorzystanie modelu relacyjnego w obiektowych językach programowania	• **Odbiór mini projektu MP3** (materiał z wykładu dotyczącego dziedziczenia) • Ćwiczenia z wykorzystania modelu relacyjnego w obiektowych językach programowania.
10	Wykorzystanie modelu relacyjnego w obiektowych językach programowania (2)	Ćwiczenia z wykorzystania modelu relacyjnego w obiektowych językach programowania.
11	Użyteczność (*usability*) graficznych interfejsów użytkownika	• **Odbiór mini projektu MP4** (materiał z wykładu dotyczącego modelu relacyjnego) • Prace związane z projektem końcowym.
12	Projektowanie i	• Ćwiczenia z implementacji

	implementowanie graficznych interfejsów użytkownika	graficznych interfejsów użytkownika (w tym wykorzystania edytora GUI). • Prace związane z projektem końcowym.
13	Projektowanie i implementowanie graficznych interfejsów użytkownika (2)	• Ćwiczenia z implementacji graficznych interfejsów użytkownika (w tym wykorzystania edytora GUI). • Prace związane z projektem końcowym. • **Ostateczny termin wysłania dokumentacji projektowej**. Nie ma możliwości późniejszego oddawania dokumentacji.
14	Projektowanie i implementowanie graficznych interfejsów użytkownika (3)	**Odbiór implementacji projektów końcowych**.
15	Projektowanie i implementowanie graficznych interfejsów użytkownika (4)	Wszystkie sprawy związane z zaliczeniem ćwiczeń (w tym zaliczenia poprawkowe).

2.2. Plan zajęć (studia zaoczne, 8 zjazdów)

Nr	Wykład	Ćwiczenia
1	Wybrane konstrukcje obiektowych języków programowania	• Krótkie omówienie założeń przedmiotu oraz projektu • Ćwiczenia z analizy dynamicznej.
2	Wykorzystanie klas w obiektowych językach programowania	Ćwiczenia z wybranych konstrukcji obiektowych języków programowania.
3	Realizacja asocjacji w obiektowych językach programowania	Ćwiczenia z wykorzystania klas w obiektowych językach programowania.
4	Realizacja asocjacji w obiektowych językach programowania (2)	• Ćwiczenia z realizacji asocjacji w obiektowych językach programowania (1). • **Odbiór mini-projektu MP1.**
5	Realizacja różnych modeli dziedziczenia w obiektowych językach programowania	Ćwiczenia z realizacji asocjacji w obiektowych językach programowania (2).

6	Realizacja pozostałych, wybranych konstrukcji UML w obiektowych językach programowania	• Ćwiczenia z realizacji różnych modeli dziedziczenia w obiektowych językach programowania. • **Odbiór mini-projektu MP2.**
7	Użyteczność (*usability*) oraz implementacja graficznych interfejsów użytkownika	• Ćwiczenia z tworzenia GUI (m.in. korzystając z edytora). • **Odbiór mini-projektu MP3.** • **Ostateczny termin wysłania dokumentacji projektowej**. Nie ma możliwości późniejszego oddawania dokumentacji.
8	Wykorzystanie modelu relacyjnego w obiektowych językach programowania	**Odbiór projektów końcowych.**

3. Mini-projekty

Ich celem jest praktyczne sprawdzenie zrozumienia sposobów implementacji poszczególnych konstrukcji z modelu pojęciowego (klasy, asocjacje, ekstensja, itp.) oraz współpracy z modelem relacyjnym. Dodatkowo można go 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) występujące w modelu pojęciowym. Każdy MP musi być wykonany jako **aplikacja konsolowa** i zawierać przykładowe dane i/lub metody ilustrujące jego poprawną pracę (zlokalizowane w metodzie `main()`). Terminy obron: patrz załączony plan zajęć. W ich trakcie można się spodziewać pytań dotyczących realizowanych zagadnień oraz sposobu implementacji. Istnieje możliwość oddawania mini projektów na późniejszych zajęciach, ale wiąże się to z 50% redukcją punktacji.

W mini-projektach, należy zaproponować własne przypadki biznesowe, tzn. nie wolno wykorzystywać przykładów z wykładów, książki, itp. Elementy MP, które podlegają ocenie:

Studia dzienne (15 zajęć)			
MP 01 Klasy, atrybuty	**MP 02** Asocjacje	**MP 03** Dziedziczenie	**MP 04** Model relacyjny[1]
• Ekstensja • Ekst. - trwałość • Atr. złożony • Atr. opcjonalny • Atr. powt.	• „Zwykła" • Z atrybutem • Kwalifikowana • Kompozycja *(w każdym*	• Klasa abstrakcyjna i polimorficzne wołanie metod • Overlapping • Wielodziedziczenie	• MR - klasy • MR – asocjacje (1:* lub *:*) • MR - dziedzicz. *Należy stworzyć działający program korzystający z bazy danych i realizujący*

[1] Model relacyjny (MR) nie obowiązuje Wydziału Zarządzania Informacją (WZI).

• Atr. klasowy • Atr. pochodny • Met. klasowa • Przesłonięcie, przeciążenie	*przypadku: liczności 1-* lub *-* oraz automatyczne tworzenie poł. zwrotnego)*	• Wieloaspektowe • Dynamiczne	*powyższe konstrukcje. Trzeba skorzystać z oprogramowania typu ORM, np. Hibernate, Entity Framework.*

Studia zaoczne (8 zjazdów)		
MP 01 Klasy, atrybuty	**MP 02** Asocjacje	**MP 03** Dziedziczenie
• Ekstensja • Ekst. - trwałość • Atr. złożony • Atr. opcjonalny • Atr. powt. • Atr. klasowy • Atr. pochodny • Met. klasowa • Przesłonięcie, przeciążenie	• „Zwykła” • Z atrybutem • Kwalifikowana • Kompozycja *(w każdym przypadku: liczności 1-* lub *-* oraz automatyczne tworzenie poł. zwrotnego)*	• Klasa abstr/ polimorfizm • Overlapping • Wielodziedziczenie • Wieloaspektowe • Dynamiczne

Tematyka MPx może, ale nie musi, być związana z tematyką projektu końcowego.

Nie wolno łączyć ze sobą przykładów dla poszczególnych konstrukcji, np. każdy rodzaj atrybutu musi mieć swój własny biznesowy przykład.

W związku z procedurą antyplagiatową (patrz dalej), przed indywidualną obroną, implementacja MPx musi być wgrana do odpowiedniego folderu w systemie Gakko/EduX (jako archiwum „*MPx_Nazwisko_Imię_NrIndeksu.zip*” zawierające wyłącznie kody źródłowe, bez plików binarnych, bibliotek itp.).

Mini-projekty są integralną częścią składową oceny końcowej (patrz punkt 5.1). W związku z tym warto je dobrze wykonać.

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)[2] 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

4.1.1. Dokumentacja może, ale nie musi, być oparta na projekcie z przedmiotu PRI.

[2] Dla Wydziału Zarządzania Informacją (WZI) obowiązuje limit 8 – 15 klas.

4.1.2. Podstawą projektu są wymagania użytkownika opisane w postaci „historyjki” w punktach.

4.1.3. Zgodnie z nimi należy opracować wymagania funkcjonalne (jako diagramy przypadków użycia i ewentualnie w postaci tekstowej) oraz niefunkcjonalne (tekst w punktach).

4.1.4. Istotnym składnikiem wymagań jest również analityczny diagram klas. Jako, że jest podstawą do dalszych prac, warto zadbać aby nie zawierał on błędów. Bez poprawnego analitycznego diagramu klas nie ma możliwości przygotowania odpowiedniego projektowego diagramu klas.

4.1.5. Bardziej szczegółowo należy omówić przynajmniej jeden nietrywialny przypadek użycia (odwołujący się do innego przypadku użycia). Scenariusz, dla tego przypadku, powinien być sporządzony zarówno z wykorzystaniem języka naturalnego, jak i diagramów aktywności.

4.1.6. Projekt interfejsu użytkownika: w oparciu o wybrany, nietrywialny przypadek użycia powinien być sporządzony projekt interfejsu użytkownika (zgodnie z wytycznymi podanymi na wykładzie).

4.1.7. Dla wybranego przypadku użycia należy przeprowadzić analizę dynamiczną (czyli wykorzystać diagramy: aktywności i stanu[3]). Analiza dynamiczna powinna zakończyć się jawnym omówieniem jej skutków. „Skutki” analizy dynamicznej (np. nowe atrybuty, nowe asocjacje, itp.) powinny być umieszczone na implementacyjnym (projektowym) diagramie klas. Projekt musi wykorzystywać wszystkie wymienione rodzaje diagramów.

4.1.8. Należy dołączyć elementy specyfikujące podjęte decyzje projektowe/implementacyjne (na przykładach, w oparciu o odpowiednie fragmenty diagramu), np. sposób realizacji ekstensji klasy, dziedziczenia itp.

4.1.9. Na podstawie analitycznego diagramu klas, tworzony jest projektowy (implementacyjny) diagram klas. Może zawierać **tylko** elementy wynikającego z diagramu analitycznego.

a. jest uszczegółowiony,

b. wszystkie konstrukcje nie istniejące w wybranym języku programowania są zamienione (zgodnie z podjętymi decyzjami projektowymi),

c. ewentualnie jest uzupełniony o metody wynikłe z analizy dynamicznej.

4.1.10. Należy zadbać o **czytelność całej dokumentacji**, a szczególnie diagramów (zalecany jest jakiś format wektorowy (np. *Enhanced Metafile*) lub plik z bezstratną kompresją, np. PNG). W tym celu warto upewnić się, że:

[3] Diagram stanu robimy dla konkretnej klasy (a nie całego systemu, czy przypadku użycia), w miarę możliwości z analizowanego przypadku użycia.

a. prawidłowo stosujemy notację UML (a nie „podobne” elementy graficzne),
b. każdy diagram jest prawidłowo podpisany,
c. został przygotowany w odpowiednim narzędziu (np. UMLet),
d. na diagramach stosujemy **rozmiar czcionek** niewymagający nadmiernego powiększania (czytelny diagram A4 w widoku 100%, np. na wydrukowanej stronie; w razie potrzeby zwiększ czcionkę na diagramie, a nie cały diagram),
e. nie dzielimy diagramów na części,
f. elementy (np. klasy) są właściwe rozplanowane, m. in. dziedziczenie w pionie, asocjacje w poziomie, unikamy przecinania linii, asocjacje mają nazwy i liczności.

Uwaga: prace zawierające nieczytelne diagramy **nie będą sprawdzane**.

W stworzeniu czytelnych diagramów, pomocny może być **przygotowany poradnik**.

4.1.11. Dokumentację projektową oddajemy w postaci **pojedynczego** pliku PDF (*MAS_Grupa_Nazwisko_Imię_NrIndeksu.PDF*) wysłanego na adres mailowy prowadzącego ćwiczenia. Ostateczny termin – patrz załączony plan zajęć.

4.1.12. Podsumowanie zawartości dokumentacji projektowej:

a. Wymagania użytkownika („historyjka”)
b. Diagramy przypadków użycia (wymagania funkcjonalne)
c. Wymagania niefunkcjonalne jako lista
d. Diagram klas – analityczny
e. Diagram klas – projektowy
f. Scenariusz przypadku użycia (jako wypunktowany tekst)
g. Diagram aktywności dla przypadku użycia
h. Diagram stanu dla klasy
i. Projekt GUI
j. Omówienie decyzji projektowych i skutków analizy dynamicznej

4.1.13. 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***

4.2. Implementacja[4]

4.2.1. Cała struktura (wszystkie klasy z odpowiednimi asocjacjami, dziedziczeniami itp.),

4.2.2. Metody potrzebne do realizacji wybranego przypadku (lub przypadków) użycia,

4.2.3. Elementy graficznego interfejsu użytkownika (GUI), które są niezbędne do zaprezentowania pracującej implementacji z działającym wybranym przypadkiem użycia. Każdy projekt **musi** posiadać GUI.

4.2.4. **Minimalna implementacja GUI musi** obejmować interakcję co najmniej dwóch klas połączonych asocjacją (wymagana liczność docelowa: „wiele”),

a. Na diagramie mamy dwie klasy połączone asocjacją, np.: *Firma i Pracownik*; 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 asocjacji** (przeważnie oznacza to brak stosowania języka zapytań, w tym SQL-a).

b. Zaimplementowane GUI, które **tylko** tworzy połączenia pomiędzy obiektami i nie pozwala na w/w interakcję, **nie wystarczy do zaliczenia**.

c. 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).

4.2.5. Implementacja musi zawierać **przykładowe dane** ilustrujące poprawne działanie.

4.2.6. Należy zwrócić uwagę na jakość, ergonomię i użyteczność (*usability*) 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.

4.2.7. **Wszystkie** dane przechowywane w systemie muszą być trwałe (np. plik, baza danych, dedykowana biblioteka, itp.). Warto zwrócić uwagę na poprawność wyboru technologii (np. serializacja nie nadaje się dla

[4] Projekt dla WZI nie musi posiadać GUI. Zamiast tego można stworzyć metodę `main` ilustrującą działanie zrealizowanych konstrukcji (przykładowe dane i operacje na nich, np. stworzenie powiązań i wyświetlenie na konsoli powiązanych obiektów).

aplikacji webowych) oraz prawidłowość jej implementacji, np. serializacja wymaga jednego wspólnego pliku dla wszystkich ekstensji.

4.2.8. W kodzie nie należy umieszczać oczywistych komentarzy opisujących kod źródłowy.

4.2.9. Część implementacyjna projektu będzie indywidualnie odbierana w czasie zajęć (patrz dalej).

4.2.10. Językiem implementacji może być Java, C# lub C++. Inne języki wymagają zgody prowadzącego ćwiczenia.

4.2.11. 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 API)	5
Elegancja zaimplementowanych rozwiązań	15
Sposób implementacji trwałości	10
Implementacja GUI (w tym użyteczność/ergonomia)	20
Prezentacja projektu	5
Razem	***100***

Projekt końcowy nie musi zawierać wszystkich konstrukcji z MPx.

4.3. **Każdy** projekt będzie indywidualnie odbierany. 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**.

4.4. W związku z procedurą antyplagiatową (patrz dalej), przed indywidualną obroną, implementacja projektu musi być wgrana do odpowiedniego folderu w systemie Gakko/EduX (nazwa pliku: „*ProjectImpl_Nazwisko_Imię_Numer.zip*”; archiwum powinno zawierać wyłącznie kody źródłowe bez plików binarnych, bibliotek itp.).

5. Zaliczenie ćwiczeń

Ocena końcowa z ćwiczeń składa się z następujących składowych:

5.1. Punkty za mini-projekty (liczy się suma, nie trzeba indywidualnie zaliczać każdego elementu):

5.1.1. Studia dzienne (15 zajęć): 24 + 24 + 24 + 28 = 100 pkt.,

5.1.2. Studia zaoczne (8 zjazdów): 35 + 35 + 30 = 100 pkt.

5.2. Ocena za implementację projektu

5.3. Ocena za dokumentację projektu

Z każdej części 5.1, 5.2 i 5.3 trzeba otrzymać ocenę pozytywną. W związku z tym osoby, które np. zaliczą MP, a nie zaliczą dokumentacji projektu **nie zaliczą ćwiczeń**.

Dodatkowo można zdobyć do 15 pkt. za rozwiązanie zadań podanych w czasie wykładów. Te bonusowe punkty doliczane są do ogólnej liczby punktów (patrz pkt. 5.1) pod warunkiem posiadania co najmniej 50% pkt.

6. Plagiat

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

a. wszystkie programy, w których wykryto wspólny kod, **otrzymują 0 pkt.**;

b. **nie ma możliwości przysłania poprawionej wersji takiego programu**, co może prowadzić do **niezaliczenia ćwiczeń** bez szansy na ich poprawę w bieżącym semestrze;

c. nie będą prowadzone ustalenia, kto, od kogo kopiował kody źródłowe.

Przed obroną MPx oraz projektu końcowego, obowiązkowo należy je **wgrać do odpowiedniego folderu w systemie Gakko/EduX** (tylko pliki źródłowe). Terminy według ustaleń prowadzącego ćwiczenia.

7. Terminy

Terminy realizacji poszczególnych zadań (mini-projekty, dokumentacja projektu końcowego, implementacja projektu końcowego) są podane w tabeli (pkt. 2). Ich niedotrzymanie wiąże się z poważną redukcją oceny, aż do niezaliczenia włącznie.

Nie ma również możliwości zaliczania przedmiotu po zakończeniu semestru, warunkowo na egzaminie, w sesji poprawkowej, itp.

8. Egzamin

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

8.1. **Testowej**. Należy ocenić każde z pytań (T/N). Odpowiedź prawidłowa oznacza +2 pkt., błędna -2 pkt., brak odpowiedzi 0 pkt.

8.2. **Zadaniowej**. Należy nazwać oraz krótko omówić sposób implementacji zaznaczonych konstrukcji na otrzymanym diagramie klas.

Nie ma zwolnień z egzaminu.

Przykładowy egzamin: *https://www.mtrzaska.com/mas-egzamin/* .

9. Materiały

9.1. Informacje ogólne (mogą pojawiać się nowsze wersje).

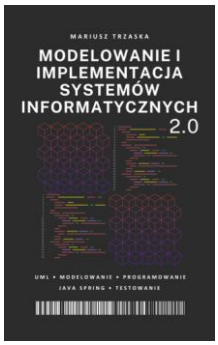
https://www.mtrzaska.com/tags/mas/

9.2. Wersja elektroniczna wykładów:

https://www.mtrzaska.com/tags/mas/

Ze względu na skomplikowanie omawianych zagadnień, **zaleca się uczęszczanie na wykład** (niezależnie od tego, że powyższe materiały są dostępne *on-line*)

9.3. **Nowa wersja książki** *M. Trzaska: „Modelowanie i implementacja systemów informatycznych 2.0”*. Stron 468. ISBN 978-83-976442-0-5.



- Wersja elektroniczna PDF (eBook działający na Windows, iOS, Android, urządzeniach mobilnych oraz czytnikach) w księgarni internetowej Empik (***więcej informacji***).
- Edycja ePub jest w przygotowaniu i powinna być wkrótce osiągalna.

9.4. 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 (Modelowanie i Analiza Systemów informacyjnych). Mogą być wykorzystane w czasie zajęć „*Wybrane konstrukcje obiektowych języków programowania*”. Studenci przystępujący do kursu MAS, powinni umieć rozwiązać zdecydowaną większość z nich.

https://www.mtrzaska.com/mas-zadania-programistyczne/

9.5. Bezpłatne książki on-line:

9.5.1. Bruce Eckel - Thinking in Java: http://www.mindview.net/Books/TIJ/

9.5.2. Allen B. Downey - How to Think Like a Computer Scientist: Java Version: http://www.greenteapress.com/thinkapjava/

9.5.3. Robert Sedgewick and Kevin Wayne - Introduction to Programming in Java: An Interdisciplinary Approach: http://introcs.cs.princeton.edu/home/

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 aplikacji, które mogą być przydatne:

- Narzędzia CASE (m.in. edytory diagramów):
  - UMLet (*open source*, wieloplatformowy): https://www.umlet.com/
  - Modelio (*open source*, wieloplatformowy): https://www.modelio.org/,

  - Visual Paradigm Community Edition: http://www.visual-paradigm.com (również wersja on-line),
  - Lucid Charts, https://lucid.app/ (również wersja on-line),
  - ArgoUML: http://argouml.tigris.org/,
  - MagicDraw Community Edition: http://www.magicdraw.com,
  - StarUML: http://staruml.sourceforge.net/en/,
  - NetBeans for Java: http://www.netbeans.org/ (umożliwia m.in. tworzenie diagramów UML i generowanie kodu źródłowego)
  - MS Visio (dostępny na licencji ELMS dla studentów PJATK),
  - Obszerna lista narzędzi: http://en.wikipedia.org/wiki/List_of_UML_tools.
- IDE
  - IntelliJ IDEA (darmowy *Community*): https://www.jetbrains.com/idea/
  - Eclipse for Java: http://www.eclipse.org/,
  - NetBeans for Java: http://www.netbeans.org/,
  - MS Visual Studio (dostępny na licencji ELMS dla studentów PJWSTK).
- Edytory GUI
  - wbudowany w IntelliJ IDEA, NetBeans;
  - dla Eclipse: Jigloo SWT/Swing GUI Builder (http://www.cloudgarden.com/jigloo/);
  - dla Eclipse: WindowBuilder Pro - po przejęciu przez Google darmowy (http://code.google.com/intl/pl/webtoolkit/tools/wbpro);
  - wbudowany w MS Visual Studio.

W razie wątpliwości proszę o kontakt: *mtrzaska@pjwstk.edu.pl*