# Design and Analysis of Information Systems (MAS)

*Updated: 2022-04-19*

Lecturer: **Mariusz Trzaska**, Ph. D. (*mtrzaska@pjwstk.edu.pl*, *http://www.mtrzaska.com*)
Classes: Emil Wcisło (*ewcislo@pjwstk.edu.pl*)

## 1. Introduction

The “Design and Analysis of Information Systems” course is devoted to practising the skills that are necessary for transition from a conceptual model (the product of the analysis and requirements specification phase in software development life cycle) into a specific implementation environment, either object-oriented or relational. Students learn some conceptual features that have no direct counterparts in a chosen implementation environment (actually Java or MS C#). Some rules regarding designing (usability) and implementing user interfaces and utilization of software frameworks are also presented. The lectures are supported by the implementation including data management, application’s behaviour and creating simple GUIs. Requirement specification and static analysis should be done along with teaching the course “Object Modelling Techniques in Software Analysis” (PRI). The size of a project, that each student has to do on their own, is limited to between 10 and 15 UML classes.

## 2. Schedule

| No. # | Lecture | Excercise |
|---|---|---|
| 1 | Designing and Modeling of System Architecture | • A brief discussion of the subject and the objectives of the project.<br>• Exercises of dynamic analysis.<br>• Class involves drawing diagrams on the basis of the requirements. You can use the dedicated CASE tool or traditional whiteboard. |
| 2 | Selected Constructs of Object-oriented Programming Languages | Exercising of the selected constructs of object-oriented programming language.<br>Class involves creating programs using various designs. The nature and scope is defined by level of the group (which is determined by the teacher) and can be based on the contents of the lecture. |
| 3 | Selected Constructs of Object-oriented Programming Languages (2) | Exercising of the selected constructs of object-oriented programming language. |
| 4 | Using Classes in Object-oriented Programming Languages | Exercising with the use of classes in object-oriented programming languages. |

| | | |
|---|---|---|
| 5 | Associations in Object-oriented Programming Languages | • Exercising of implementing association in object-oriented programming languages (1).<br>• **Evaluation of the mini project MP1** (lecture material for classes) |
| 6 | Associations in Object-oriented Programming Languages (2) | Exercising of implementing association in object-oriented programming languages (2). |
| 7 | Inheritance in Object-oriented Programming Languages | • Exercising the implementation of different models of inheritance in object-oriented programming languages.<br>• **Evaluation of a mini project MP2** (lecture material for associations) |
| 8 | Implementation of Other UML Constructs in Object-oriented Programming Languages | • Exercising the implementation of various UML structures in object-oriented programming languages.<br>• **Evaluation of a mini project MP3** (lecture material for inheritance) |
| 9 | Relational Model in Object-oriented Programming Languages | • **Evaluation of a mini project MP4** (lecture material for constraints)<br>• Exercising the implementation of the relational model in object-oriented programming languages |
| 10 | Relational Model in Object-oriented Programming Languages (2) | • Exercising the implementation of the relational model in object-oriented programming languages |
| 11 | Usability of Graphical User Interfaces | • **Evaluation of a mini project MP5 (**lecture material for relational model)<br>• Work on the final project. |
| 12 | Design and Implementation of Graphical User Interfaces | • Exercising the implementation of graphical user interfaces (including the use of a dedicated GUI editor).<br>• Work on the final project. |
| 13 | Design and Implementation of Graphical User Interfaces (2) | • Exercising the implementation of graphical user interfaces (including the use of a dedicated GUI editor).<br>• Work on the final project.<br>• **The deadline for sending documentation. There is no possibility of sending the documentation later.** |
| 14 | Design and Implementation of | **Evaluation of final project implementation.** |

| | | |
|---|---|---|
| | Graphical User Interfaces (3) | |
| 15 | Design and Implementation of Graphical User Interfaces (4) | All matters relating to the final evaluation of the classes. |

3. Mini-projects

Their goal is to test the understanding the practical ways to implement particular constructions of the conceptual model (classes, associations, an extent, etc.) and their cooperation with the relational model. In addition, they can be treated as a "cornerstone" of the final project (which later will be supplemented with other required elements, e. g. GUI). You need to implement various business constructs (all the technical information such as the number of objects in extent, setters / getters do not apply), which are present in the conceptual model. Every MP must be implemented as a **console application** and contain examples of data and / or methods that illustrates its proper operation (localized in the `main()` method). Projects evaluation schedule – see attached classes schedule. During the evaluation you can expect questions about ongoing issues and implementation approaches. Your late mini projects can be evaluated during next classes, but it is associated with a 50% reduction in the score.

MP elements to be assessed[1]:

| **MP 01**<br>Classes, attributes | **MP 02**<br>Associations | **MP 03**<br>Inheritance | **MP 04**<br>Constraints | **MP 05**<br>Relational model[2] |
|---|---|---|---|---|
| • A class extent<br>• A class extent – persistence<br>• A complex attribute<br>• An optional attribute<br>• A multi-value attribute<br>• A class attribute<br>• A derrived attribute<br>• A class method<br>• Method overriding and overloading | • "Basic"<br>• With an attribute<br>• Qualified<br>• Composition<br>In every case: cardinality 1-* or *-* and automatic creation of a reverse connection. | • An abstract class and polymorphic method invocation<br>• An overlapping inheritance<br>• A multi-inheritance<br>• A multi-aspect inheritance<br>• A dynamic inheritance | • For an attribute<br>• Unique<br>• Subset<br>• Ordered<br>• Bag<br>• Xor<br>• Custom business constraint | • RM - classes<br>• RM – associations (1-* or *-*)<br>• RM - inheritance.<br>You should create a program that uses a database and implements the above structures. You can use any tools, e.g. ORM-s, for example: Hibernate, Entity Framework. |

The topics of the MPx could be different from a topic of the final project.

[1] You should propose your own business cases, i. e. do not use examples from lectures, books, etc.
[2] Relational model (MR) is not applicable for students of Department of Information Management (WZI).

It is not allowed to combine different constructs in the same example, e.g. each kind of an attribute should have its own business example.

Mini-projects are an integral part of the final evaluation (see section 5.1). Thus they should be well prepared.

4. The final project consists of two parts: a documentation and an implementation. Its topic should be complex enough to create the appropriate number (12 - 15) meaningful business classes. Because of this, topics like tool applications, system applications, media players, etc. are not suitable. If you have doubts you should consult the teacher.

   4.1. Documentation

   4.1.1. The documentation includes the "old" part, i. e. the one that has been created on the PRI class (included as an attachment) and the “new” documentation. The “new” documentation copies the “old” documentation scheme, but there are some new elements, which are mentioned below. Those who do not have the "old" project of the PRI must develop at least: requirements (as the "story"), the use case diagram and analytical class diagram.

   4.1.2. Use cases: „New” documentation of use cases should include detailed description of one non-trivial use case in addition to “old” documentation. The use case should refer to another use case. The scenario for this use case should be made using both natural language and activity diagram.

   4.1.3. User interface design: based on the selected non-trivial use case, user interface design should be made (according to the guidelines given in the lecture).

   4.1.4. For selected use case a dynamic analysis should be performed (i.e., UML diagrams of activity, interaction and state should be provided). Dynamic analysis should be completed not only by the appearance of the methods in the class diagram, but also a discussion of its implications. “The effects” of the dynamic analysis (perhaps not only new methods, but also new attributes, new associations, etc.) should be placed on a class diagram redrawn from the "old" project and, where possible, highlighted with a different color. The project must use all the kinds of diagrams mentioned above.

   4.1.5. Before the "final" class diagram, which forms the basis for implementation, it is necessary to include elements specifying the design decisions (for example, based on the relevant parts of the diagram), e.g. how to implement a class extent.

   4.1.6. In summary, the "final" (design) class diagram differs from the diagram provided on the PRI in the following sections:

   - Has more details
   - All structures which do not exist in a given programming language are transformed (according to the design decisions)
   - Is supplemented with methods resulting from dynamic analysis

   4.1.7. You should take care of **the readability of the entire documentation**, especially diagrams (some vector format is recommended, e. g. *Enhanced Metafile*). Hence, you should ensure that:

   - each of them is properly described/entitled,
   - diagrams are prepared in a suitable tool (e. g. UMLet),
   - you use fonts which are big enough (100% A4 view should be readable),

- layout is clear, e.g. draw inheritance as vertical lines and associations as horizontal ones, avoid crossing lines, etc.

4.1.8. Documentation should be delivered **in a single PDF file** (*MAS_Group_Lastname_Firstname_StudentNo.pdf*) sent to the email address of the teacher. Deadline - see the attached schedule.

4.1.9. A summary of the contents of the project documentation (PRI + MAS):

- User requirements
- The use case diagram
- The class diagram - analytical
- The class diagram - design
- The scenario of selected use case (as text)
- The activity diagram for picked use case
- The state diagram for selected class
- The interaction (sequence) diagram for selected use case
- The GUI design
- The discussion of design decisions and the effect of dynamic analysis

4.1.10. The evaluation of the documentation will be issued in accordance with the table below (prerequisite: meeting the requirements of paragraph 4.1):

| **Criterion** | **Max points** |
|---|---|
| The complexity of the business domain | 10 |
| Documenting the use case(s) (scenario and diagram) | 10 |
| Correctness and complexity of the design class diagram | 25 |
| Correctness and complexity of the interaction diagram | 10 |
| Correctness and complexity of the activity diagram | 10 |
| Correctness and complexity of the state diagram | 10 |
| The GUI design | 10 |
| Discussion of the design decisions | 10 |
| Readability and organization of the document | 5 |
| *Total* | *100* |

4.2. Implementation

4.2.1. The whole structure (all the classes with appropriate associations).

4.2.2. Methods required to implement the use case (or cases).

4.2.3. Elements of the graphical user interface (GUI), which are necessary to present a working implementation of the selected use case. Each project **must have** a GUI.

4.2.4. **Minimum implementation of the GUI must** involve an interaction of two classes connected with an association (required target cardinality: many), for

example: there are two classes: an *Employee* and a *Company*; a widget (capable of showing many items, e.g. *ListBox)* contains a list of companies, after clicking on any item it should display another widget (capable of showing many items, e.g. *ListBox*) which contains a list of its employees retrieved **using a defined association** (usually it means no SQL queries). GUI implementation, which **only** creates connections between objects and does not allow for the above interaction, is **not enough to pass the project**. Similarly, solutions e.g. with a single widget, a *TextBox*, target cardinality "1" or just filtering the extent (instead of using a previously defined association) are insufficient.

4.2.5. The implementation must contain **sample data** showing the correct operation.

4.2.6. Pay attention to quality, ergonomics and usability of GUI (e.g., windows scaling, colors, philosophy of actions) - it is an important component of the final assessment. Design and implementation of GUI (you can use dedicated editors) shall be in accordance with the principles of usability, described at the lecture.

4.2.7. All data stored in the system must be persistent (e.g. file, database, dedicated library, etc.).

4.2.8. Implementation of the project will be individually evaluated during the exercises (see below). Therefore, it does not need to be send in any persistent form.

4.2.9. Language of the implementation can be Java, C# or C++. Other languages should be agreed with the teacher.

4.2.10. The grade for the implementation will be issued in accordance with the following table (prerequisite: meeting the requirements of Section. 4.2):

| **Criterion** | **Max points** |
|---|---|
| Difficulty of the task | 10 |
| The scope and the correctness of the realized functionality | 15 |
| The scope and correctness of completed object-oriented constructs | 25 |
| Code quality (names, comments, *JavaDoc*, etc.) | 5 |
| The elegance of implemented solutions | 15 |
| The way of the persistence implementation | 10 |
| The implementation of the GUI (including usability/ ergonomics) | 20 |
| *Total* | *100* |

The final project does not have to include all constructs from mini-projects.

4.3. **Each** project will be evaluated individually. During the evaluation, you can expect detailed questions about the way of the implementation, e. g. *"What will be if…"*, *"Why it's done this way…"*, *"Please make the following modifications..."*. People who implemented the projects personally should not have any problems with answering the above questions. Lack of ability to answer above questions will result in negative grade of the classes.

4.4. The following aspects of the project will be evaluated (see also points 4.1.10, 4.2.10):

4.4.1. The difficulty of the task.

4.4.2. The realized scope of the functionality.

4.4.3. The quality of the code including comments, that allow to automatically generate API documentation.

4.4.4. The elegance of implemented solutions, including the GUI ergonomics

5. The credit of the exercises

Final evaluation of the exercise consists of the following components:

5.1. Points for mini-projects (no need to individually pass each of them): 20 + 20 + 20 + 16 + 24 = 100 pts.,

5.2. Evaluation of the project:

5.2.1. Grade for the implementation

5.2.2. Grade for the documentation

With each part (5.1, 5.2.1, 5.2.2) you must obtain a passing mark (50% or 3,0). Therefore, people, who for example pass MP, and does not pass the project **will not pass the classes**.

Additionally, you can earn up to 15 points for solving tasks described during lectures. These bonus points are added to the total number of points (see Para. 5.1), provided you have at least 50% of the points.

6. Deadlines

Deadlines for the various tasks (mini-projects, final project documentation, and implementation of the final project) are given in the table (point 2). Exceeding deadlines will result serious reduction in the mark, including up to failing the exercises.

There is no possibility of passing exercises at the end of the semester, conditional on the exam, retake session, etc.

7. The exam

MAS exam consists of two parts (total points are the foundation of the mark):

7.1. **Test.** You should evaluate each of the questions (Y / N). Correct answer is 2 point., Incorrect -2 point., No response - 0 points.

7.2. **Practical exercises**. You should name and briefly discuss how to implement the given structure drawn on the attached conceptual class diagram.

There is no exemption from the exam.

An example exam: http://www.mtrzaska.com/mas-egzamin.

8. Materials

8.1. Book (polish version): M. Trzaska: „Modelowanie i implementacja systemów informatycznych". Rok 2008. Wydawnictwo PJWSTK. Stron 299. ISBN 978-83-89244-71-3.

- Electronic version of the book (eBook works on platforms: Windows, iOS, Android, mobile devices and readers) in internet bookstore Ibuk: PDF version.
- Print version: PJATK bookstore.
- Fragments w PDF: http://www.mtrzaska.com/mas-ksiazka

8.2. General Information (may appear in newer versions).

http://www.mtrzaska.com/mas-informacje

8.3. The electronic version of the lectures:

http://www.mtrzaska.com/mas

Due to the complex nature of these issues, it is recommended to attend the lecture (regardless of the fact that these materials are available on-line).

8.4. Sample programming tasks

These programming tasks are only intended to guide programming material (learned during previous courses) and their solutions will not be assessed as part of the subject MAS (Modeling and Analysis of Information Systems). They can be used during the "*Selected constructions of object-oriented programming languages*" class. Students starting the MAS course should be able to solve the vast majority of the following tasks.

https://www.mtrzaska.com/mas-programming-tasks/

8.5. Free books on-line:

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

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

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

8.6. Implementation tools

Due to the fact that there is quite a large freedom to choose the technology of the project, there is no mandatory tools list. However, the following list contains tools, which can be useful:

- CASE tools (including diagramming editors):
  - UMLet (*open source*, multiplatform): https://www.umlet.com/
  - Visual Paradigm Community Edition: http://www.visual-paradigm.com (also on-line edition),
  - Lucid Charts https://lucid.app/ (also on-line edition),
  - 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/ (allows, for example, the creation of UML diagrams and source code generation)

  - MS Visio (ELMS license available for PJAIT students)
  - A comprehensive list of tools: http://en.wikipedia.org/wiki/List_of_UML_tools.
- IDE
  - IntelliJ IDEA (free *Community*): https://www.jetbrains.com/idea/
  - Eclipse for Java: http://www.eclipse.org/,
  - NetBeans for Java: http://www.netbeans.org/,
  - MS Visual Studio (ELMS license available for PJIIT students)
- GUI editors
  - Included in NetBeans;
  - For Eclipse: Jigloo SWT/Swing GUI Builder (http://www.cloudgarden.com/jigloo/);
  - For Eclipse: WindowBuilder Pro - free to use after the acquisition by Google (http://code.google.com/intl/pl/webtoolkit/tools/wbpro);
  - Included in MS Visual Studio.

If you have any doubts, please contact: *mtrzaska@pjwstk.edu.pl*