COURSE DESCRIPTION CARD - SYLLABUS

Course name

Object programming

Course

Field of study	Year/Semester
Artificial intelligence	2/3
Area of study (specialization)	Profile of study
	general academic
Level of study	Course offered in
First-cycle studies	English
Form of study	Requirements
full-time	compulsory

Number of hours

Lecture	Laboratory classes	Other (e.g. online)
15	30	
Tutorials	Projects/seminars	

Number of credit points

4

Lecturers

Responsible for the course/lecturer:

Dariusz Brzezinski, Ph.D.

email: Dariusz.Brzezinski@cs.put.poznan.pl

tel: 61 665 3057

Faculty of Computing and Telecommunications

ul. Piotrowo 2, 60-965 Poznan

Responsible for the course/lecturer:

Prerequisites

The student should have basic knowledge of algorithmics and data structures. Students are also expected to be capable of solving basic programming problems, as well as writing, testing, and modyfying existing code. The student should also be capable of finding information on his own and be willing to work as part of a team. Finally, it is expected that the student resembles an attitude of honesty, responsibility, perseverence, creativity, and that of respect for other people.

Course objective

The goal of the course is to teach students how to model and create reusable, easily-maintainable software, by using tools provided by object-oriented programming languages. Moreover, the students will learn how to create and use custom data types, how to model software systems based on clean code principles, and how to communicate their work to other programmers.

Course-related learning outcomes

Knowledge

1. The student has basic, theoretically grounded knowledge from the fields of programming languages, programming paradigms, and software engineering.

2. Has basic knowledge of the processes and life cycles that occur in software systems.

Skills

1. The student has basic skills concerning the assessment of the computational complexity of algorithms, programming with popular languages, and using operating systems.

2. Can design (following a pre-defined specification) and create an IT system by first selecting and then using the available methods, techniques and computer tools, in particular by using object-oriented programming languages.

3. Can adapt existing algorithms, as well as formulate and implement novel algorithms, by using at least one software development tool.

Social competences

1. The student understands that programming languages, programming paradigms, and software engineering best practices are part of an ongoing field of research, and that one must keep learning to be up to date with the state-of-the-art.

2. Knows the impact that programming paradigms (including object-oriented programming) can have on solving practical tasks in companies, and its potential effect on entire societies.

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Lecture: a written test involving short answer questions, simple programming assignments, modeling assignments. At least 50% of points are required to pass.

Labs: an evaluation based on two programming projects.

Programme content

Lecture:

1. What is object(-oriented) programming and when to use it. Object-oriented versus other programming paradigms. Basic properties of object-oriented programming languages.

2. Object analysis. Object-oriented modeling through UML class diagrams: inheritance, implementaction, aggregation, composition, association. Other UML diagrams.

3. SOLID, the five object-oriented design principles: (S)ingle-responsibility, (O)pen-closed, (L)iskov substitution, (I)nterface segregation, (D)ependency inversion.

4. Inheritance, abstract classes, generic types: implementations in different languages and use cases.

5. Selected software design patterns with examples in object-oriented programming languages.

6. Extensions to object-oriented programming languages: reflection, anonymous types, lambda expressions, covariance and countervariance, asynchronous programming, auto and dynamic typing, declarative programming.

7. Contract-based programming.

8. Aspect-based programming and future directions.

Laboratory:

1. Introductory lab: object-oriented programming principles, simple UML diagrams.

2. C++ - pointers, classes, inheritance, abstraction, and polymorphism.

3. C++ - operators, parent classes, friends.

4. C++ - templates, promisses, excpetions.

5. Introduction to Java.

6. Inheritance in Java.

7. Collections.

8. Programming graphical user interfaces (Swing and JavaFX).

9. Threads and concurrency control.

10. Input/output streams and serialization.

11. Exceptions and debugging.

12. Generic types in Java.

13. Extensions to object-oriented programming languages (Java, C++, C#).

14. Profilers and code optimization.

15. Project evaluations.

Teaching methods

Lecture: multimedia presentations, whiteboard examples, brainstorming.

Lab: multimedia presentations, discussions, whiteboard examples, programming assignments, teamwork.

Bibliography

Basic

Thinking in Java, Bruce Eckel, Pearson, 2006

Clean Code: A Handbook for Beginners to Learn How to Become a Better Programmer, Robert C. Martin, Independently published, 2019

C++ Programming Language, Bjarne Stroustrup, Addison-Wesley Professional, 2013

Additional

The Agile Samurai. How Agile Masters Deliver Great Software, Jonathan Rasmusson, Pragmatic Programmers, 2017

Breakdown of average student's workload

	Hours	ECTS
Total workload	100	4,0
Classes requiring direct contact with the teacher	45	2,0
Student's own work (literature studies, preparation for laboratory classes, preparation for exam, project preparation) [1]	55	2,0

[1] delete or add other activities as appropriate