SYLLABUS

1. Data about the program of study

1.1 Institution	The Technical University of Cluj-Napoca
1.2 Faculty	Faculty of Automation and Computer Science
1.3 Department	Computer Science
1.4 Field of study	Computer Science and Information Technology
1.5 Cycle of study	Bachelor of Science
1.6 Program of study / Qualification	Computer science / Engineer
1.7 Form of education	Full time

2. Data about the subject

2.1 Subject name	***Object Oriented Programming***			Subject code	**18**
2.2 Course responsible / lecturer	Lect. dr. eng. Joldoș Marius - Marius.Joldos@cs.utcluj.ro				
2.3 Teachers in charge of seminars / Laboratory / project	Lect. dr. eng. Joldoș Marius - Marius.Joldos@cs.utcluj.ro				
2.4 Year of study	II	2.5 Semester	3	2.6 Type of assessment (E - exam, C - colloquium, V – verification)	E
2.7 Subject category	Formative category: DF – fundamental, DS – speciality, DC – complementary				DF
	Optionality: DOB – mandatory, DOP – optional (alternative), DFA – optional (free choice)				DOB

3. Estimated total time

3.1 Number of hours per week	4	of which:	Course	2	Seminars	-	Laboratory	2	Project	-
3.2 Number of hours per semester	56	of which:	Course	28	Seminars	-	Laboratory	28	Project	-
3.3 Individual study:										
(a) Manual, lecture material and notes, bibliography										25
(b) Supplementary study in the library, online and in the field										17
(c) Preparation for seminars/laboratory works, homework, reports, portfolios, essays										16
(d) Tutoring										6
(e) Exams and tests										5
(f) Other activities:										-

3.4 Total hours of individual study (suma (3.3(a)...3.3(f)))	69
3.5 Total hours per semester (3.2+3.4)	125
3.6 Number of credit points	5

4. Pre-requisites (where appropriate)

4.1 Curriculum	Computer Programming course
4.2 Competence	Use of a procedural programming language such as C

5. Requirements (where appropriate)

5.1. For the course	
5.2. For the applications	

6. Specific competence

6.1 Professional competences	**C2 –** Designing hardware, software and communication components • **C2.1 –** Describing the structure and functioning of computational, communication and software components and systems • **C2.2 –** Explaining the role, interaction and functioning of hardware, software and communication components • **C2.3 –** Building the hardware and software components of some computing systems using algorithms, design methods, protocols, languages, data structures, and technologies • **C2.4 -** Metric based evaluation of functional and non-functional characteristics of computing systems • **C2.5 -** Implementation of hardware, software and communication components
6.2 Cross competences	N/A

7. Learning outcomes

Knowledge	Describe, identify, and summarize the transition from procedural execution to object-oriented architecture, specifically regarding the abstraction layer between high-level domain logic and the JVM. This involves a fundamental understanding of managed memory (Stack vs. Heap), the mechanics of object lifecycle and identity, and the implementation of dynamic binding to manage system behavior. By mastering these principles through Java, the student gains the theoretical basis necessary to apply systematic logic to concrete problems, such as data encapsulation for persistence, resource resilience through exception handling, and the architecture of modular, decoupled systems via interfaces and domain modeling.
Abilities/ Skills	Analyze systems by mapping abstract domain models to the managed runtime environment of the Java Virtual Machine (JVM). It facilitates the capacity to design and implement decoupled, modular software solutions through interfaces and inheritance, ensuring scalable architecture and data integrity via strict encapsulation. Furthermore, it establishes the technical proficiency required to diagnose and debug complex systemic behaviors, ranging from runtime exceptions and logic flaws in polymorphic dispatch to the optimization of object lifecycles and memory usage, thereby ensuring the functional reliability and architectural robustness of integrated software systems.
Responsibility and Autonomy	The course fosters the autonomy required to independently architect object-oriented systems, ensuring the systematic transition from abstract domain models to functional, decoupled code. It develops the responsibility to adhere to strict architectural specifications and deadlines, requiring a clear and concise description of class hierarchies, object interactions, and system behavior in both written documentation and the presentation of software design. By taking full ownership of the unit testing and systemic debugging phases, the individual ensures the robustness and integrity of their own work, establishing a professional foundation for the disciplined development and documentation of modular software components in integrated information systems.

8. Discipline objective (as results from the *key competences gained*)

7.1 General objective	To learn a rigorous approach of object-oriented concepts using Java as an example language
7.2 Specific objectives	• To transition from imperative to object-oriented programming, prioritizing object state and behavior over procedural logic. • To design modular systems using classes and encapsulation as fundamental building blocks for data integrity. • To develop reusable and extensible code by implementing inheritance, polymorphism, interfaces, and generics. • To apply structural best practices, such as choosing between inheritance and composition, to build maintainable class hierarchies. • To implement robust software through structured exception handling, automated testing, and code refactoring. • To manage complex data and external resources using the Collections framework, I/O streams, and third-party APIs. • To utilize class and object diagrams as visual aids to plan and document object relationships and system architecture.

9. Contents

8.1 Lectures	Hours	Teaching methods	Notes
Paradigm Shift: From Procedures to Objects	2	Lectures, demos and discussions.	Uses a video-projector.
Behavioral Modeling & Logic	2		
Classes & Encapsulation	2		
Object Identity & Lifecycle	2		
Inheritance & Polymorphism	2		
Interfaces	2		
Composition vs. Inheritance	2		
Exceptions	2		
Collections & Generics	2		
Strings & I/O	2		
Modular Program Design & Package Management	2		
Testing, Logging & simple GUI	2		
Domain Modeling & System Architecture	2		
Review	2		

Bibliography:

1. Paul & Harvey Deitel, Java. How to Program (Early Objects), Tenth Edition, Prentice Hall, 2015
2. Bruce Eckel, Thinking in Java, Fourth Edition, Prentice Hall PTR, 2006 (downloadable for free from the Web).
3. David J. Barnes & Michael Kölling, Objects First with Java. A Practical Introduction using BlueJ, Sixth Edition, Pearson Education, 2017
4. Oracle Java Tutorials (freely downloadable from the Web)

8.2 Applications - Seminars / Laboratory / Project	Hours	Teaching methods	Notes
Using Java IDEs. Primitive Types in Java	2	Tutoring, discussions, and assisted program development.	PCs equipped with Java SDK and IDEs (BlueJ, IntelliJIdea).
Implementing small algorithms. Debugging basics	2		
Implementing simple classes. UML to code exercises.	2		
Multi class mini project. Static vs. instance exercises	2		
Class hierarchies. Polymorphism exercises	2		
Interface based design. Strategy like patterns (informal)	2		
Refactoring inheritance to composition. Domain modeling exercises	2		
Exception-safe code. Testing exception behavior	2		
Collections exercises. Small data processing tasks	2		
File parsing tasks. Mini data processing project	2		
Start of semester miniproject. Requirements and UML	2		
Test driven exercises. Refactoring tasks	2		
Finalization of semester miniproject	2		
Laboratory test	2		

Bibliography

1. Course Moodle site available at: https://moodle.cs.utcluj.ro/

**Se vor preciza, după caz: tematica seminariilor, lucrările de laborator, tematica și etapele proiectului.*

10. Bridging course contents with the expectations of the representatives of the community, professional associations and employers in the field

The contents of the course is in accordance with the ACM Computer Science Curricula recommendations Java programming language is the most widely used language.

11. Evaluation

Activity type	Assessment criteria	Assessment methods	Weight in the final grade
Course	Ability to solve problems using the object orientated paradigm	Three in-class tests (T) + Final Written exam (W)	60% = 50% W + 10% T
Seminar	-	-	-
Laboratory	Quality of laboratory applications and evaluation of the laboratory tests	Analysis and evaluation of the tests and assignments	40%
Project	-	-	-
Minimum standard of performance: Grade calculus: 40% laboratory + 60% exams and tests Conditions for participating in the final exam: Laboratory ≥ 5 Conditions for promotion: grade ≥ 5			

Date of filling in:	**Responsible**	**Title, First name Last name**	**Signature**
	Course	Lect.dr.eng. Marius JOLDOȘ	
	Applications	Lect.dr.eng. Marius JOLDOȘ	

Date of approval in the department	Head of department, Prof.dr.eng. Rodica Potolea
Date of approval in the Faculty Council	Dean, Prof.dr.eng. Vlad Muresan