



DISCIPLINE ACADEMIC SHEET
ACADEMIC YEAR 2025 – 2026

1. PROGRAMME DATA

1.1. Higher Education Institution	UNIVERSITY OF CRAIOVA
1.2. Faculty	Automation, Computers and Electronics
1.3. Department	Computers and Information Technology
1.4. Field of Study	Computers and Information Technology
1.5. Study level	L(License / Undergraduate)
1.6. Organizational form	IF
1.7. Study Program (name/code) /Qualification	Computers / L2060101010

2. DISCIPLINE DATA

2.1 Discipline Name	Object Oriented Applications Design						
2.2 Course Activities Holder	conf.dr.ing. Stefan Udristoiu						
2.3. Practical Activities Holder	conf.dr.ing. Stefan Udristoiu						
2.4. Study Year	2	2.5. Semester	2	2.6. Evaluation type	E	2.7. Discipline regime	DOB

3. ESTIMATED TOTAL TIME (hours per semester of teaching activities)

3.1. Number of hours per week	5	in which: 3.2 course	3	3.3. sem./lab./pr.	0/2/0
3.4. Total hours of curriculum	70	in which: 3.5 course	42	3.6. sem./lab./pr.	0/28/0
3.7. Time distribution - hours/week					hours
Study after manual, course support, bibliography and notes					10
Additional documentation in library, on specialized electronic platforms and field					5
Training seminars / labs, homework, portfolios and essays					6
Tutorial					6
Examinations					2
Other activities.....					2
Total hours per individual activities					30
3.8. Total hours per semester					100
3.9. Number of credits					4

4. PRECONDITIONS (where appropriate)

4.1. of curriculum	The prerequisites for this discipline are: Computer Programming, Numerical Methods, Computer Programming - Programming Techniques, Mathematical Analysis, Logical Design 1&2, Linear Algebra and Geometry, Data Structures, Object Oriented Programming.
4.2. of competence	The student is required to have fundamental knowledge about programming, programming techniques, data structures and object oriented programming.

5. CONDITIONS (where appropriate)

5.1. of the course	• Blackboard, videoprojector helpful but not mandatory.
5.2. of seminar/ laboratory/project	• Computers having Eclipse IDE.

6. COURSE OBJECTIVES – expected learning outcomes that are supported by completing and passing the course

Knowledge	Studentul/absolventul descrie, identifică, sumarizează concepte și metode elementare privitoare la limbaje de programare, medii de programare, tehnici de programare, baze de date. inteligență artificială și inginerie software și modul lor de aplicare în probleme concrete.
Skills (Abilities)	Studentul/absolventul alege și explică concepte proprii specifice proiectării algoritmilor, programării orientate pe obiecte, programării logice și funcționale. Studentul/absolventul specifică cerințe, analizează, elaborează, dezvoltă și testează sisteme software în limbaje de programare de uz general (C, etc.) și /sau obiect-orientate (C++, Java, etc.), aplicând elementele specifice ingineriei software. Studentul/absolventul proiectează, implementează, dezvoltă și /sau gestionează aplicații ce includ diverse tipuri de baze de date. Studentul înțelege conceptele principale care stau la baza sistemelor de traducere și compilarea codului sursă. Contribuie la dezvoltarea și îmbunătățirea uneltelor specifice limbajelor de programare.
Responsibility and autonomy	Studentul/absolventul are o comportare onorabilă, responsabilă, etică, în spiritul legii pentru a asigura reputația profesiei

7. CONTENT

7.1. Course	The method of implementation	Teaching methods	Allocated time (hours)
1. Introduction in Object Oriented Design. Objectives (reducing complexity). KISS. YAGNI . DRY. Open Close Principle.	face to face (week 1)	Explaining, Demonstrating, Collaborating	3
2. Classes and objects. Encapsulation (bundling and information hiding, accessors, smells for broken encapsulation, how to refactor.) . Defining classes in Java. Creating objects. Object references and garbage collector.	face to face (week 2)	Explaining, Demonstrating, Collaborating	3
3. Rules about Class Methods. Method	face to face (week 3)	Explaining, Demonstrating,	3

cohesion (Definitions and classification, identification, how to refactor). Cyclomatic Complexity. Rules about parameters.		Collaborating	
4. Class cohesion. Single Responsibility Principle. LCOM4. How to refactor in order to obtain class cohesion.	face to face (week 4)	Explaining, Demonstrating, Collaborating	3
5. Class Coupling. Abstractions and OCP. Dependency Inversion Principle. Defining, implementing and extending interfaces in Java	face to face (week 5)	Explaining, Demonstrating, Collaborating	3
6. Interface Segregation Principle. Law of Demeter (one accepted form, two smells, how to refactor). Dependency Injection Frameworks	face to face (week 6)	Explaining, Demonstrating, Collaborating	3
7. Ottinger's Rules	face to face (week 7)	Explaining, Demonstrating, Collaborating	3
8. Inheritance. Inheritance in Java Extending behaviour. Favor composition over inheritance (justification, implementation, Composite, Decorator Design Pattern. Adapter Design Pattern. Facade Design Pattern.). Polymorphism and OCP. Liskov Substitution Principles. Abstract Classes in Java.	face to face (week 8)	Explaining, Demonstrating, Collaborating	3
9. Polymorphism Smells (how to refactor, Template Design Pattern, State Pattern). Strategy Pattern (definition, implementation, when to use). Visitor Design Pattern and alternatives to this design pattern.	face to face (week 9)	Explaining, Demonstrating, Collaborating	3
10. Exception in Java. Use of exceptions.	face to face (week 10)	Explaining, Demonstrating, Collaborating	3

11. GUI Architectures. Introduction in Swing.	face to face (week 11)	Explaining, Demonstrating, Collaborating	3
12. Observer/Observable Design Pattern. Command Pattern. Model-View-Controller. Model-View-Presenter	face to face (week 12)	Explaining, Demonstrating, Collaborating	3
13. Object-Relational Patterns. Repository Design Pattern. Data Mapper. Unit of Work. Identity Map. Lazy Load (Proxy). Abstract Factory. ORM	face to face (week 13)	Explaining, Demonstrating, Collaborating	3
14. Comments. Improving performace vs. Design	face to face (week 14)	Explaining, Demonstrating, Collaborating	3

Bibliography:

1. * - Wikipedia.
2. * - <http://www.objectmentor.com>.
3. Beck, Kent - Test Driven Development: By Example, Addison-Wesley, 2002.
4. Evans, Eric - Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2004.
5. Freeman, Eric & Sierra, Kathy & Bates, Bert - Head First Design Patterns, O'Reilly, 2004.
6. Fowler, Martin & Beck, Kent - Refactoring: improving the design of existing code, Addison-Wesley, 2018.
7. Fowler, Martin & Rice, David & Foemmel, Matthew – Patterns of Enterprise Application Architecture, Addison-Wesley, 2002.
8. Gamma, Erich & Johnson, Ralph & Vlissides, John & Hel, Richard - Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.
9. Henney, Kevlin - 97 Things Every Programmer Should Know: Collective Wisdom from the Experts, O'Reilly, 2010.
10. Hunt, Andrew & Thomas, David - The Pragmatic Programmer: From Journeyman to Master, Addison-Wesley Professional, 1999.
11. Martin, Robert C. - Agile Software Development, Principles, Patterns, and Practices, Prentice Hall, 2009.
12. Martin, Robert C. - Clean code: a handbook of agile software craftsmanship, Prentice Hall, 2009.
13. Martin, Robert C. - Clean Architecture: A Craftsman's Guide to Software Structure and Design, O'REILLY, 2017
14. McConnell, Steve - Code complete, Microsoft Press, 2004.
15. Meyer, Bertrand - Touch of Class: Learning to Program Well with Objects and Contracts, Springer, 2009.
16. Ottinger, Tim - Ottinger's Rules for Variable and Class Naming, <http://www.objectmentor.com/resources/articles/naming.htm>

7.2. Seminar/laboratory/project	The method of implementation	Teaching methods	Allocated time (hours)
1. Introduction in Eclipse	face to face (week 1)	Problem solving. Teacher's constant feedback.	2

2. Defining classes and methods. Refactor broken encapsulation.	face to face (week 2)	Problem solving. Teacher's constant feedback.	2
3. Compute cyclomatic complexity. Refactor methods.	face to face (week 3)	Problem solving. Teacher's constant feedback.	2
4. Compute LCOM4. Improve class cohesion.	face to face (week 4)	Problem solving. Teacher's constant feedback.	2
5. Implement interfaces. Use dependency inversion.	face to face (week 5-6)	Problem solving. Teacher's constant feedback.	4
6. Implement Composite, Decorator, Adapter.	face to face (week 7)	Problem solving. Teacher's constant feedback.	2
7. Implement/ refactor to polymorphism. Implement Template Method.	face to face (week 8)	Problem solving. Teacher's constant feedback.	2
8. Implement state and strategy pattern.	face to face (week 9)	Problem solving. Teacher's constant feedback.	2
9. Introduction to Swing.	face to face (week 10-11)	Problem solving. Teacher's constant feedback.	4
10. Command Design Pattern	face to face (week 12)	Problem solving. Teacher's constant feedback.	2
11. MVC	face to face (week 13)	Problem solving. Teacher's constant feedback.	2
12. MVP	face to face (week 14)	Problem solving. Teacher's constant feedback.	2
Bibliography:			
1. * - https://docs.oracle.com/javase/tutorial/			
2. Horstmann, Cay S. - Core Java Volume I--Fundamentals, 9th Edition, Prentice Hall, 2012.			

8. Correlating the content of the discipline with the expectations of representatives from the epistemic community, professional associations and representative employers in the field related to the program

The content of the discipline is constantly updated based on my experience as a software developer and software architect (30 years of experience working for Novus, MuseGlobal, Netdania and IT SIX) and from continuous feedback from fellow programmers from the majority of the software companies based in Craiova. Also for the for 11 years I was responsible together with IT SIX for their Software Engineering Academy.

9. EVALUATION

Activity Type	9.1. Evaluation criteria	9.2. Evaluation methods	9.3. Final mark weight
9.4. Course	Level and quality of student interaction	Asking constantly for solutions. Inserting	The final mark represents the

	and feedback during classes. Problem solving skills.	defects in solutions and asking students to discover them and correcting them. Final writing exams consisting of problems covering the most important chapters.	average of the maximum mark obtained for each chapter during the semester (laboratory and course activities result) and the final exam.
9.5. Seminar/laboratory/project	Problem solving skills and the level of understanding of theoretical concepts. The level of work and constancy is also recorded.	During the entire semester, students are required to solve problems and to present them, responding to a series of “What if?” and “Why” questions.	The final mark represents the average of the maximum mark obtained for each chapter during the semester (laboratory and course activities result) and the final exam.
9.6. Minimum standard of performance			
The minimum standard of performance consists of abilities to solve problems which require starting and waiting processes, creating and joining threads which performs activities that do not require synchronizations and communicating simple messages using pipes. They can solve those problems during laboratories (or seminar activities) or at the final exam.			

Date of Completion	Signature of the Course Activities Holder,	Signature of the Practical Activities Holder,
23.08.2025	Conf.dr.ing. Stefan Udristoiu	Conf.dr.ing. Stefan Udristoiu

Date of Approval in the Department	Signature of the Department Director,
25.08.2025	Ș.I.dr.ing. Nicolae ENESCU

Date of Approval in the Faculty	Signature of the Dean,
	Prof.dr.ing. Dorin ȘENDRESCU

