COURSE DESCRIPTION CARD - SYLLABUS

Course name
Object-Oriented Programming

Course

Field of study
Computing

Year/Semester
2/3

Area of study (specialization)

Profile of study
general academic

Level of study
First-cycle studies

Course offered in
Polish

Form of study
full-time

Requirements
compulsory

Number of hours

Lecture
16

Laboratory classes

Other (e.g. online)

Tutorials

Projects/seminars
30

Number of credit points

3

Lecturers

Responsible for the course/lecturer:
dr inż. Tomasz Koszlajda
email: Tomasz.Koszlajda@cs.put.poznan.pl
tel. 61 665 2960
Faculty of Computing and Telecommunications
Piotrowo 2, 60-965 Poznań

Responsible for the course/lecturer:
dr hab. inż. Dariusz Brzeziński, prof. PP
email: Dariusz.Brzeziński@cs.put.poznan.pl
tel. 61 665 3057
Faculty of Computing and Telecommunications
Piotrowo 2, 60-965 Poznań

Prerequisites

A student starting this subject should have the basic knowledge obtained in the subjects: Introduction to Computing and Algorithms and Data Structures.

He should have the ability to solve basic problems with algorithm specifications, to write, modify and test computer programs independently, as well as the ability to obtain information from indicated sources.

He should also understand the necessity to expand his competencies and have a willingness to cooperate as part of a team. In addition, in terms of social competence, the student must present such attitudes as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture and respect for other people.

Course objective

1 Teaching students the principles of creating universal software modules suitable for reusable in various programming projects and easy to develop and maintain, by applying the unique solutions available in object-oriented languages that favor the creation of computer programs with such characteristics. In addition, the goal is to teach students to create their own semantically rich and versatile abstract data types.

2.Developing in students the ability to design and create computer systems with a correct architecture, i.e., one that is characterized by cohesiveness of the constituent program modules and loose relationships between these modules.

3. Developing in students the ability to communicate during the independent creation of computer program modules to be composed into a single whole. In addition, obtaining skills to find optimal, ready and available components for use in their own complex computer programs.

Course-related learning outcomes

Knowledge

has a structured, theoretically supported general knowledge of algorithms, programming languages and paradigms, and software engineering, (K1st_W4)

knows and understands generic classes, exception handling in object-oriented languages and is able to apply these mechanisms to create reusable universal software modules,

guaranteeing the construction of computer programs of high quality, (K1st_W6)

knows and understands the principles of construction of computer programs that process persistent objects stored in the database, and the principles of construction of programs with complex multifaceted architecture. (K1st_W6)

Knows the basic methods, techniques and tools used in solving simple tasks

information technology, algorithms and problems, construction of computer systems, implementation of programming languages and software engineering, (K1st_W7)

has the knowledge necessary to transform object models of fragments of reality into selected object-oriented languages, (K1st_W7)

has the knowledge necessary for object-oriented modeling and analysis of non-small fragments of the "real world" related to various fields of application, (K1st_W7)

knows and understands the syntax and semantics of basic and complex object-oriented mechanisms, such as:

classes, objects, interfaces and object implementation, object encapsulation, class inheritance and subtype relation, polymorphic variables and substitutions, dynamic binding, (K1st_W7)

Skills

has the ability to develop computer programs of high quality in accordance with the criteria defined in ISO standards, and in particular characterized by: reliability, ease of maintenance and development, flexibility, ease of testing, portability and ease of code sharing, (K1st_U9)

is able to create an object model of a simple system (e.g. in the UML language) (K1st_U10)

is able to choose a programming language suitable for a given programming task (K1st_U10)

is able - according to a given specification - to design and implement a simple information system, using appropriate methods, techniques and tools (K1st_U11)

has the ability to formulate classes and program them using at least two popular tools, i.e., object-oriented programming languages: C++ and Java, (K1st_U11)

is able to work in a group (K1st_U18)

Social competences

understands that in programming languages are dynamically developing and some of the skills related to programming become obsolete very quickly (K1st_K1)

Is aware of the importance of correct modeling of reality in solving engineering problems and knows examples and understands the causes of malfunctioning information systems (K1st_K2)

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Formative assessment:

(a) for lectures:

- activity during lectures

b) in terms of project classes:

- on the basis of the evaluation of the current progress of the tasks - exercises and the final credit project

Summative assessment:

a) in terms of lectures, verification of the established learning outcomes is realized by:

- evaluation of knowledge and skills demonstrated in the written assessment

b) in the scope of project classes, verification of the assumed educational effects is realized by:

- evaluation of the realization of final projects

Programme content

The program of lectures on the subject includes the following topics:

The rationale of object-oriented programming derived from the analysis of the sources of the software crisis. The idea of a new programming paradigm that supports the creation of high-quality programs.

The search for an optimal programming language and methodologies appropriate for building reusable universal software modules. The relationship of the object-oriented paradigm to software engineering. Quality metrics of computer program architecture: cohesion and independence of of software modules. Programming languages with an expandable data type system. Implementation of the concept of abstract data types. A brief overview of the history of object-oriented languages.

Object-oriented modeling and analysis using CRC cards and UML language for class and object diagrams and collaboration diagrams. Learning the basic constructs of the object-oriented model: class, object, class variables and operations, generalization relationships, relationships between classes. Examples of simple models of fragments of reality. Methodologies for modeling and analysis. Transformation of object-oriented diagrams to object-oriented programming languages.

Definitions of basic object-oriented concepts: object, object attributes (variables), object methods, sending messages that trigger object method calls, class interfaces, objects as instances of classes, Examples of class definition including: definitions of class constructors and destructors, overloaded operators, class variables and methods. Encapsulation of class implementations as a mechanism for limiting the relationship between software modules. Friendship relationship between classes. A dual view of the class as a new data type and as a program module. Comparison of solutions of simple problems in a functional and object-oriented manner. Implementation of complex objects and relationships between objects. Learning about the types of copy operators of complex objects. Architecture of an object-oriented virtual machine.

Inheritance of classes and subtype relationship between classes. Definition of new properties of derived classes, overriding methods and fields, covariant redeclaration of variables and methods, and implementation of abstract classes. Overview of class inheritance network topologies in various programming languages. Virtual inheritance in C++. Inheritance of constructors and destructors of classes. Methodologies for applying the mechanism of class inheritance.

Subtype relationships between classes. Defining polymorphic variables and polymorphic substitutions. Increasing the versatility and flexibility of classes by using late message binding.

Implementation and examples of use of the late binding mechanism. Late binding, and reflection mechanisms of data types. Dynamic casting of data types.

Further increasing the degree of universality of classes by defining generic classes. Creating universal programs while maintaining strong data typing. Limits of applicability of generic classes: bounded and unbounded generics. Typical examples of generic classes. Class templates in the C++ language.

Developing reliable computer programs. Levels of code security. Basic strategies for creating programs resistant to errors and exceptions. Methodologies and techniques of exception handling in object-oriented languages. Defining and reporting exceptions. Exception catching and exception handling. Examples of exception handling applications. Importance of exception handling for programming modules intended for reuse.

Methodologies for developing software that conforms to its specification. Formal specification of semantics of abstract data types. Programming by contract: analysis and programming axioms of classes and initial and final conditions of methods. Definition and applications for assertions in object-oriented languages.

Ensuring the durability of objects by storing them in a database. Functionality of system software for object-relational mapping (OR/M). Methodologies of correct mapping of a network of classes into a relational database schema. Cases of functionally complex programs with intersecting aspects. Extension of object-oriented languages to explicitly define aspects. Transfer of control between intersecting aspects. Examples of application of aspect-oriented languages.

The above topics are illustrated with examples in object-oriented programming languages: C++, Java, C# and Eiffel.

During the project classes, students will learn in depth about two object-oriented languages programming: C++ and Java. Exercises involve independent development of programs incorporating the basic constructs of the object-oriented languages presented in lectures. In addition, system libraries are reworked in terms of: collections, graphical interface, streams, multithreading and serialization of object state. Familiarization with each of the two programming languages culminates in independent preparation of small projects involving object-oriented analysis and implementation of programs.

Some of the above-mentioned curricular content is carried out as part of the student's own work.

Teaching methods

Lecture: multimedia presentation, illustrated by examples given on the blackboard.

Project classes: programming tasks, discussion, multimedia demonstration, case study, demonstration, brainstorming.

Bibliography

Basic

1. Programowanie zorientowane obiektowo, Bertrand Mayer, Helion, Warszawa, 2005

2. Metody obiektowe w teorii i praktyce, Ian Graham, WNT, Warszawa, 2004

3. Smalltalk-80: The Language and its Implementation, Goldberg A.J., A.D.Robson, Addison-Wesley, 1983

4. Język C++, Bjarne Stroustrup, WNT, Warszawa, 1994

5. The Java(TM) Programming Language (3rd Edition), Ken Arnold, James Gosling, David Holmes, Addison Wesley Professional, 2000

6. Programowanie obiektowe, Peter Coad, Edward Yourdon, Read Me, 1994

7. Analiza obiektowa, Peter Coad, Edward Yourdon, Read Me, 1994

8. Nowoczesne projektowanie w C++, Andrei Alexandrescu, WNT, 2005

Additional

1. Simula Begin, Graham M. Birtwistle, O.J. Dahl, B. Myhrhaug, K. Nygaard, 1973

2. http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf

3. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Breakdown of average student's workload

	Hours	ECTS
Total workload	75	3,0
Classes requiring direct contact with the teacher	46	2,0
Student's own work (literature studies, preparation for the project classes, preparation for the test, preparation of the project) [1]	29	1,0

[1] delete or add other activities as appropriate