COURSE DESCRIPTION CARD - SYLLABUS

Course name
Object-oriented programming [S1Bioinf1>POBKT]

Course

Field of study
Bioinformatics

Year/Semester
1/2

Area of study (specialization)
–

Profile of study
general academic

Level of study
first-cycle

Course offered in
Polish

Form of study
full-time

Requirements
compulsory

Number of hours

Lecture
30

Laboratory classes
30

Other
0

Tutorials
0

Projects/seminars
0

Number of credit points

5,00

Coordinators

dr hab. inż. Piotr Łukasiak prof. PP
piotr.lukasiak@put.poznan.pl

Lecturers

Prerequisites

The student starting this module should have basic knowledge of algorithms and programming languages. In terms of skills, proficiency is required in solving basic problems related to the specification of algorithms, independent writing, modification and testing of computer programs, along with the ability to obtain information from the indicated sources.

Course objective

The aim of the course is to teach the principles of creating universal program modules that are reusable in various programming projects and are easy to develop and maintain, through the use of unique algorithmic and programming solutions available in object-oriented languages, based on C ++ as an example. In addition, the aim is to teach students to create their own semantically rich and universal abstract data types, as well as to develop students" skills in designing and creating information systems with the correct architecture characterized by the cohesiveness of component program modules and relationships between these modules. An essential goal of the course is to educate students of communication skills during the independent development of computer program modules and to search for optimal components that can be used in their own complex computer programs.

Course-related learning outcomes

Knowledge:
The student knows and understands the principles of structured and object-oriented programming
The student knows and understands the basic methods, techniques and tools used in the process of solving bioinformatics tasks, mainly of an engineering nature
The student knows and understands the life cycle of information systems

Skills:
The student is able to obtain information from literature, databases and other properly selected sources, also in English
The student is able to integrate and interpret the obtained information, as well as draw conclusions and formulate and justify their opinions
The student is able to design and create computer software in accordance with the given specification, using appropriate methods, techniques and tools
The student is able to analyze the functionality and analyze the requirements of information systems
The student is able to independently acquire knowledge and improve his qualifications

Social competences:
The student is ready to learn throughout his life and improve his competences
The student is ready to cooperate and work in a group, assuming different roles in it
The student is ready to define priorities for the implementation of a task defined by himself or others
The student is ready to take responsibility for the decisions made

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning outcomes presented above are verified as follows:
The learning outcomes presented above are verified as follows:
Formative assessment:
a) in the field of lectures:
• activity during lectures
b) in the field of exercises:
• on the basis of the assessment of the current progress in the implementation of tasks and the final project
Summative assessment:
a) in the field of lectures, verification of the assumed learning outcomes is carried out by assessing the knowledge and skills shown on the written test, and discussing the results. The test consists of a set of open and closed questions regarding the skills and understanding of the elements of object-oriented programming on practical instances
b) in the field of laboratories, verification of the assumed learning outcomes is carried out by continuous assessment during each class (oral answers), rewarding the increase in the ability to use the learned rules, methods and program tools and evaluation of the implementation of final projects

Programme content

The lecture program includes
• premises of object-oriented programming,
• the idea of a new programming paradigm that supports the creation of high-quality programs.
• searching for the optimal programming language and methodologies appropriate for building universal program modules for multiple use
• relations of the object-oriented paradigm with software engineering
• quality metrics for computer program architecture: cohesion and independence of program modules
• implementation of the concept of abstract data types
• learning about the basic constructors of the object model: class, object, class variables and operations, generalization relationships, relationships between classes
• examples of simple models of fragments of reality
• definitions of basic object concepts: object, attributes (variables) of the object, methods of the object,
• sending messages triggering calls to methods of objects, class interfaces, objects as instances of classes
• examples of class definition including: definitions of class constructors and destructors, overloaded operators, class variables and methods

• hermetic implementation of classes as a mechanism for limiting relationships between program modules
• a friendship relationship between classes
• comparing solutions to simple problems in a functional and object-oriented way
• implementation of complex objects and relationships between objects
• class inheritance and subtype relationship between classes
• definition of new features of derived classes, overriding methods and variables, implementation of abstract classes
• virtual inheritance in C ++
• class constructor and destructor inheritance
• defining polymorphic variables and polymorphic substitutions
• implementation and examples of the use of the late binding mechanism
• dynamic data type casting
• increasing the degree of universality of classes by defining generic classes
• limits of applicability of generic classes: limited and unlimited genericity
• typical examples of generic classes
• C ++ class patterns
• creating reliable computer programs
• code security levels
• basic strategies for creating error- and exception-resistant programs
• methodology and techniques for handling exceptions in object-oriented languages
• defining and throwing exceptions, catching exceptions and handling them
• examples of the use of exception handling.

Course topics

The lecture program includes
• premises of object-oriented programming,
• the idea of a new programming paradigm that supports the creation of high-quality programs.
• searching for the optimal programming language and methodologies appropriate for building universal program modules for multiple use
• relations of the object-oriented paradigm with software engineering
• quality metrics for computer program architecture: cohesion and independence of program modules
• implementation of the concept of abstract data types
• learning about the basic constructors of the object model: class, object, class variables and operations, generalization relationships, relationships between classes
• examples of simple models of fragments of reality
• definitions of basic object concepts: object, attributes (variables) of the object, methods of the object,
• sending messages triggering calls to methods of objects, class interfaces, objects as instances of classes
• examples of class definition including: definitions of class constructors and destructors, overloaded operators, class variables and methods
• hermetic implementation of classes as a mechanism for limiting relationships between program modules
• a friendship relationship between classes
• comparing solutions to simple problems in a functional and object-oriented way
• implementation of complex objects and relationships between objects
• class inheritance and subtype relationship between classes
• definition of new features of derived classes, overriding methods and variables, implementation of abstract classes
• virtual inheritance in C ++
• class constructor and destructor inheritance
• defining polymorphic variables and polymorphic substitutions
• implementation and examples of the use of the late binding mechanism
• dynamic data type casting
• increasing the degree of universality of classes by defining generic classes
• limits of applicability of generic classes: limited and unlimited genericity
• typical examples of generic classes
• C ++ class patterns
• creating reliable computer programs

• code security levels
• basic strategies for creating error- and exception-resistant programs
• methodology and techniques for handling exceptions in object-oriented languages
• defining and throwing exceptions, catching exceptions and handling them
• examples of the use of exception handling.

Teaching methods

Lecture:
multimedia presentation, presentation illustrated with examples given on the board, solving problems, multimedia show
Laboratory exercises:
solving tasks, practical exercises, discussion, team work, multimedia show

Bibliography

Basic
1. Programowanie zorientowane obiektowo, Bertrand Meyer, Helion, Warszawa, 2005
2. Metody obiektowe w teorii i praktyce, Ian Graham, WNT, Warszawa, 2004
3. Język C++, Bjarne Stroustrup, WNT, Warszawa, 1994
4. Programowanie obiektowe, Peter Coad, Edward Yourdon, Read Me, 1994
5. Analiza obiektowa, Peter Coad, Edward Yourdon, Read Me, 1994
6. Nowoczesne projektowanie w C++, Andrei Alexandrescu, WNT, 2005
Additional
1. B. Eckel, Thinking in C++, HELION, 2002
2. Nicolai M. Josuttis, C++ Biblioteka standardowa, Podrecznik programisty

Breakdown of average student's workload

	Hours	ECTS
Total workload	125	5,00
Classes requiring direct contact with the teacher	60	2,50
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	65	2,50