COURSE DESCRIPTION CARD - SYLLABUS

Course name
Object programming [N1Inf1>POB]

Course

Field of study Computing	Year/Semester 2/3
Area of study (specialization) –	Profile of study general academic
Level of study first-cycle	Course offered in Polish
Form of study part-time	Requirements compulsory

Number of hours

Lecture 12	Laboratory classes 0	Other 0
Tutorials 0	Projects/seminars 16	

Number of credit points

3,00

Coordinators

dr inż. Tomasz Koszlajda
tomasz.koszlajda@put.poznan.pl

Lecturers

Prerequisites

A student beginning this course should have basic knowledge gained in the courses: Introduction to Computing and Algorithms and Data Structures. They should have the ability to solve basic algorithmic problems, to write, modify and test computer programs on their own, and the ability to obtain information from indicated sources. They should also understand the need to broaden their competences and be ready to start cooperation within the team. Moreover, in the scope of social competences, the student must present such attitudes as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture and respect for other people.

Course objective

1) Teach students the principles of creating universal software modules suitable for multiple use in various programming projects and easy to develop and maintain, by using unique solutions available in object-oriented languages, which promote the development of computer programs with such features. Furthermore, the aim is to teach students to create their own semantically rich and universal abstract data types. 2. Develop students" skills in designing and creating information systems with correct architecture, i.e. one that is characterized by cohesiveness of program modules" components and loose coupling between them. 3) Developing in students the ability to communicate while independently creating computer program modules to be composed into one whole. Moreover, acquiring the ability to search for optimal, ready- made and available components to be used in their own complex computer programs.

Course-related learning outcomes

Knowledge:
1. has structured, theoretically backed up general knowledge of algorithms, languages and paradigms of programming and software engineering, (K1st_W4)
2. knows and understands the generic classes, exception handling in object-oriented languages and is able to apply these mechanisms to create reusable universal software modules that guarantee the construction of high quality computer programs, (K1st_W6)
3. knows and understands the rules of construction of computer programs processing persistent objects stored in the database and the rules of construction of programs with a complex multi-faceted architecture. (K1st_W6)
4. knows basic methods, techniques and tools used in solving simple computing tasks, algorithms and problems, building computer systems, implementing programming languages and software engineering, (K1st_W7)
5. has the knowledge necessary to transform object-oriented models of reality fragments into selected object-oriented languages, (K1st_W7)
6. has the knowledge necessary for modelling and object-oriented analysis of small fragments of the "real world" related to different areas of application, (K1st_W7)
7. knows and understands the syntax and semantics of basic and complex object mechanisms such as: classes, objects, interfaces and implementation of objects, object encapsulation, class inheritance and subtype relationship, polymorphic variables and substitutions, dynamic binding, (K1st_W7)

Skills:
1. has the ability to develop high quality computer programs that comply with the criteria defined in ISO standards, and in particular are characterised by: reliability, ease of maintenance and development, flexibility, ease of testing, portability and ease of code sharing, (K1st_U9)
2. can create an object-oriented model of a simple system (e.g. in UML) (K1st_U10)
3. is able to select the programming language appropriate to the programming task (K1st_U10)
4. is able, according to a given specification, to design and implement a simple information system using appropriate methods, techniques and tools (K1st_U11)
5. has the ability to define classes and implement them using at least two popular tools, i.e. object-oriented programming languages: C++ and Java, (K1st_U11)
6. can work in a group (K1st_U18)

Social competences:
1. understands that programming languages are evolving rapidly and that some programming skills are quickly becoming obsolete (K1st_K1)
2. is aware of the importance of correct modeling of reality in solving engineering problems; knows examples of faulty information systems and understands the causes of their failure (K1st_K2)

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning outcomes presented above are verified as follows:
Formative evaluation:
(a) as for lectures:
- activity during lectures
(b) as for project classes:
- on the basis of the evaluation of the current progress on tasks and the final project

Summative evaluation:
a) as for lectures, the verification of the assumed learning outcomes is carried out by:
- assessment of knowledge and skills shown on the written test
- discussion of the results of the test
b) as for project classes, the verification of the assumed learning outcomes is carried out by:
- evaluation of the final project

Programme content

The programme of lectures in the course includes the following topics:
The premises of object-oriented programming resulting from the analysis of software crisis roots. The idea of a new programming paradigm, which supports the creation of high quality programs. Searching for an optimal programming language and methodologies appropriate for building universal software modules for multiple use. Relationship of the object-oriented paradigm with software engineering. Quality metrics for software architecture: cohesion and independence of software modules. Programming languages with expandable system of data types. Implementation of the concept of abstract data types. Brief presentation of the history of object-oriented languages.
Modelling and object analysis with CRC cards and UML language for class and object diagrams and collaboration diagrams. Getting to know the basic constructors of an object model: class, object, class variables and operations, generalization relationships, relationships between classes. Examples of simple models of reality fragments. Methodologies for modeling and analysis. Transformation of object diagrams into object-oriented programming languages.
Definitions of basic object concepts: object, attributes (variables) of an object, object methods, transmission of messages triggering calls of object methods, class interfaces, objects as class instances, Examples of defining classes including: definitions of class constructors and destructors, operator overloading, class variables and methods. Encapsulation of classes as a mechanism for limiting relationships between software modules. Friendship relationship between classes. Dual view of the class as a new data type and as a software module. Comparison of solutions to simple problems in a functional and object oriented way. Implementation of complex objects and relationships between objects. Getting to know the types of operators to copy complex objects. Architecture of an object virtual machine.
Class inheritance and subtype relationship between classes. Definition of new features of derived classes, hiding of methods and variables, covariant redeclaration of variables and methods and implementation of abstract classes. Overview of class inheritance network topology in different programming languages. Virtual inheritance in C++ language. Inheritance of class constructors and destructors. Methods of applying the class inheritance mechanism.
Subtype relationships between classes. Defining polymorphic variables and polymorphic substitutions. Increasing versatility and flexibility of classes by using late message binding. Implementation and application examples of the late binding mechanism. Late binding and reflection mechanisms of data types. Dynamic projection of data types.
Further increase of class versatility by defining generic classes. Creating universal programs while maintaining strong data typing. Application limits of generic classes: limited and unlimited genericity. Typical examples of generic classes. Class templates in C++ language.
Creating reliable computer programs. Code security levels. Basic strategies of creating programs resistant to errors and exceptions. Methodologies and techniques of exception handling in object oriented languages. Defining and reporting exceptions. Capturing exceptions and their handling. Examples of applications of exception handling. Validity of exception handling for reusable software modules.
Methodology for creating software that meets its specification. Formal specification of semantics of abstract data types. Contract programming: analysis and programming of class axioms and start and end conditions of methods. Defining and applying for assertions in object oriented languages.
Ensuring persistence of objects by storing them in a database. Functionality of system software for object-relational mapping (OR/M). Methods of correct mapping of a network of classes into a relational database scheme.
Cases of functionally complex programs with intersecting aspects. Extension of object-oriented languages with explicit definition of aspects. Transfer of control between intersecting aspects. Examples of using aspect languages.
The above issues are illustrated with examples in various object-oriented programming languages, including C++ and Java.
Within the framework of the project activities, students will become familiar with two object-oriented

programming languages in depth: C++ and Java. The exercises consist in creating programs containing basic structures of object-oriented languages presented during the lectures. In addition, system libraries are used for: collections, GUI, streams, multithreading and serialization of object state. The familiarization with each of the two programming languages ends with independent realization of small projects including object analysis and program implementation.
Some of the aforementioned course contents will be realized as part of the student"s own work.

Course topics

Lecture topics include the following:
Object-oriented programming premises resulting from the analysis of the sources of the software crisis. The concept 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 universal, reusable program modules. The relationship between the object-oriented paradigm and software engineering. Quality metrics for computer program architecture: cohesion and independence of program modules. Programming languages with an extensible 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 for class and object diagrams, as well as collaboration diagrams. Learning the basic constructs of the object model: class, object, class variables and operations, generalization relationships, and relationships between classes. Examples of simple models of fragments of reality. Methodologies for modeling and analysis. Transforming object-oriented diagrams into object-oriented programming languages.
Definitions of basic object-oriented concepts: objects, object attributes (variables), object methods, message sending triggering object method calls, class interfaces, objects as class instances. Examples of class definitions, including: definitions of class constructors and destructors, overloaded operators, class variables, and class methods. The hermetic nature of class implementations as a mechanism for limiting relationships between software modules. The friendship relationship between classes. A dual perspective on a class as a new data type and as a software module. Comparison of functional and object-oriented solutions to simple problems. Implementation of complex objects and relationships between objects. Understanding the types of operators for copying complex objects. Architecture of an object-oriented virtual machine.
Class inheritance and the subtype relationship between classes. Defining new features of derived classes, overriding methods and variables, covariant redeclaration of variables and methods, and implementing abstract classes. A review of the topology of class inheritance networks in various programming languages. Virtual inheritance in C++. Inheritance of class constructors and destructors. Methodologies for applying the class inheritance mechanism. Subtype relationships between classes. Defining polymorphic variables and polymorphic substitution. Increasing the universality and flexibility of classes by using late message binding. Implementation and examples of late binding. Late binding and data type reflection mechanisms. Dynamic data type casting. Further increasing the degree of universality of classes by defining generic classes. Creating universal programs while maintaining strong typing. The limits of generic class applicability: restricted and unrestricted genericity. Typical examples of generic classes. Class patterns in C++.
Creating reliable computer programs. Code security levels. Basic strategies for creating programs that are error- and exception-tolerant. Exception-handling methodologies and techniques in object-oriented languages. Defining and throwing exceptions. Catching and handling exceptions. Examples of exception handling applications. The importance of exception handling for reusable software modules.

A methodology for creating software that conforms to its specifications. Formal specification of the semantics of abstract data types. Programming by contract: analyzing and programming class axioms and method start and end conditions. Defining and applying assertions in object-oriented languages.

Ensuring object persistence by storing them in a database. System software functionality for object-relational mapping (OR/M). Methodologies for correctly mapping class networks to a relational database schema. Cases of functionally complex programs with intersecting aspects. Extending object-oriented languages with explicit aspect definition. Passing control between intersecting aspects. Examples of aspect-oriented languages. The above topics are illustrated with examples in object-oriented programming languages: C++, Java, C#, and Eiffel.
During the laboratory sessions, students will become thoroughly familiar with two object-oriented programming languages: C++ and Java. The exercises involve independent development of programs incorporating the basic object-oriented language constructs presented in lectures. Additionally, system

libraries are explored, including collections, graphical interfaces, streams, multithreading, and object state serialization. The coursework in each of the two programming languages culminates in the independent completion of small projects involving object-oriented analysis and program implementation.

Some of the above-mentioned program content is implemented through student work. Teaching methods:
1. Lecture: multimedia presentation, presentation illustrated with examples on the board, multimedia presentation
2. Laboratory exercises: practical exercises, discussion, teamwork, multimedia presentation, case studies, demonstration, brainstorming

Teaching methods

Lecture: multimedia presentation, presentation illustrated with examples given on the board, multimedia show.
Project classes: practical exercises, discussion, teamwork, multimedia show, case study, demonstration, brainstorming.

Bibliography

Basic
1. Programowanie zorientowane obiektowo, Bertrand Meyer, 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. https://docs.oracle.com/javase/specs/
3. https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf

Breakdown of average student's workload

	Hours	ECTS
Total workload	75	3,00
Classes requiring direct contact with the teacher	28	1,50
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	47	1,50