COURSE DESCRIPTION CARD - SYLLABUS

Course name
Object-oriented Programming Languages [S1Teleinf1>JPO]

Course

Field of study
Teleinformatics

Year/Semester
2/4

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 inż. Paweł Sroka
pawel.sroka@put.poznan.pl

dr inż. Marek Michalski
marek.michalski@put.poznan.pl

dr hab. inż. Remigiusz Rajewski
remigiusz.rajewski@put.poznan.pl

Lecturers

Prerequisites

Student knows and understands - to an advanced extent - selected facts, objects and phenomena and also methods and theories explaining the complex relationships between them, constituting the basic general knowledge in the field of mathematics. Student can acquire information from literature, databases and other sources; he/she can integrate and interpret the obtained information, as well as draw conclusions and formulate and justify opinions; he/she is able to formulate and solve complex and unusual problems and also perform tasks in conditions that are not fully predictable by means of the proper selection of sources and information from these sources, evaluation, critical analysis and synthesis of this information. Student is ready to critically evaluate his/her knowledge, and also recognize the importance of knowledge in solving cognitive and practical problems

Course objective

Detailed familiarisation with the style of object-oriented programming. Mastering the skills of using object-oriented language constructs, including: - designing and implementing complex algorithms in C++ and Java languages, - implementing the principles of effective object-oriented programming (SOLID). Mastering the rules of choosing the style and programming language to the nature of the task.

Course-related learning outcomes

Knowledge:
1. Student can develop documentation on the engineering task implementation and prepare a text containing a
discussion of the results of this task; he/she can communicate using specialised terminology.
2. Student can design algorithms using basic algorithmic techniques, analyse their complexity and evaluate them.
3. Student can use programming environments and platforms for coding, running and testing simple programs in
imperative, object-oriented and declarative languages; he/she is able to use analytical, simulation and experimental methods in identifying and formulating engineering task specifications and solving the tasks.

Skills:
1. Student has an organised and theoretically grounded knowledge in the fields of: basic algorithms and their
analysing, techniques of designing algorithms, abstract data structures and their implementation, computationally
hard problems.
2. Student knows and understands - to an advanced extent - selected facts, objects and phenomena and also
methods and theories explaining the complex relationships between them, constituting the basic general knowledge in the field of programming.

Social competences:
1. Student is aware of the importance and understands non-technical aspects and effects of IT engineer’s activity
and the related responsibility for the decisions he/she makes; he/she is ready to care for the heritage and traditions of the profession

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning outcomes presented above are verified as follows:
Lecture: written or oral exam.
Labs: credit based on tests, programming activity in class, programs written as part of homework, or (optional) performing an individual project task (specification of requirements, application project, implementation in C++ or
Java language, documentation).
More than 50% points are necessary for passing exam and labs.

Programme content

Lecture (general content)::
Different styles of programming and their classification. Goals and rules of object modelling. The Unified Modelling
Language - the most often used structural and behavioural diagrams.
Basic paradigms of object-oriented programming (encapsulation, inheritance, polymorphism) and their implementation in C++ language.Handling errors and exceptions in object-oriented languages. Function overloading and operator overloading. Input/output libraries in C++. Dynamic storage management in object-
oriented languages and systems. Generic programming (templates), the Standard Template Library. Rules of
multi-thread programming. Regular expressions and the Boost.Regex library.

Basic elements of Java programming: byte code, class and object implementation, input/output implementation,
packages, interfaces, collections, exceptions and their handling, multi-thread programming, deprecated applets.
SOLID - five principles of effective object-oriented programming.
Labs (general content):
Designing and implementing algorithms in C++ and Java languages.
Lecture (detailed content - as above, without changes)
Different styles of programming and their classification. Goals and rules of object modelling. The Unified Modelling
Language - the most often used structural and behavioural diagrams.
Basic paradigms of object-oriented programming (encapsulation, inheritance, polymorphism) and their implementation in C++ language.Handling errors and exceptions in object-oriented languages. Function overloading and operator overloading. Input/output libraries in C++. Dynamic storage management in object-
oriented languages and systems. Generic programming (templates), the Standard Template Library. Rules of
multi-thread programming. Regular expressions and the Boost.Regex library.
Basic elements of Java programming: byte code, class and object implementation, input/output implementation,
packages, interfaces, collections, exceptions and their handling, multi-thread programming, deprecated applets.
SOLID - five principles of effective object-oriented programming.
Labs (detailed content):
Designing and implementing algorithms in C++ language and Java language.. Using the IDE MS Visual Studio
and the IDE distributed with the Eclipse platform .
4
Designing and implementing algorithms in Java language. Using the integrated development environment distributed with the platform Eclipse.

Course topics

none

Teaching methods

Lecture:
- oral and written presentation including running of sample programs (with elements of discussion)
Labs:
- introduction to the topic (short lecture), then individual problems solving, evaluation and correction of solutions,
- presentation of individual student projects in two stages (seminar method):
- specification of requirements for an application, application design (UML diagrams),
- presentation of the application in action, implementation details

Bibliography

Basic:
1. Stroustrup B., Język C++. Kompendium wiedzy. Wydanie IV, Helion, 2014.
2. Grębosz J., Opus magnum C++11. Programowanie w języku C++. Tom 1-3, Helion, 2020.
3. Prata S., Język C++. Szkoła programowania. Wydanie VI, Helion, 2012.
4. Schildt H., Java. Przewodnik dla początkujących. Wydanie VIII, Helion, 2022.

Additional:
1. Prata S., Język C. Szkoła programowania. Wydanie VI, Helion, 2016.
2. Stroustrup B., Programowanie. Teoria i praktyka z wykorzystaniem C++, Wydanie III, Helion, 2020.
3. Schildt H., Java. Kompendium programisty. Wydanie XI, Helion, 2020.
4. Eckel B., Thinking in Java. Edycja polska. Wydanie VI, Helion, 2017.

Breakdown of average student's workload

	Hours	ECTS
Total workload	120	5,00
Classes requiring direct contact with the teacher	64	3,00
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	56	2,00