National Technical University of Ukraine "IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Department of Computer Engineering

OBJECT-ORIENTED PROGRAMMING

Syllabus

Requisites of the Cource

Cycle of Higher Education	***First cycle of higher education (Bachelor's degree)***
Field of study	*12 Information Technologies*
Spciality	*123 Computer Engineering*
Education Program	*Computer Systems and Networks*
Type of Course	*Normative*
Mode of Studies	*Full-time*
Year of studies, semester	*First year, second semester*
ECTS workload	*5 credits (ECTS), Time allotment - 150 hours, including 54 hours of classroom work, and 96 hours of self-study.*
Testing and assessment	*Final Test*
Course Schedule	*1,5 classes per week by the timetable http://rozklad.kpi.ua/*
Language of Instruction	*English*
Course Instructor	*Lecturer and Lab teacher: Associate Professor, Ph.D, Victor Porev v_porev@ukr.net*
Access to the Course	https://drive.google.com/drive/folders/1oX42kspBl2ymt6cPaH5YgsAlGsdBb82x?usp=sharing

Outline of the Course

1. Course description, goals, objectives and learning outcomes

The purpose of studying the discipline is to form students' abilities and skills
- master the object-oriented approach for the development and maintenance of software systems;
- understand the methodology of object-oriented design, master it and use it throughout the software life cycle;
- develop software using modern software tools

As a result of studying this discipline, the following learning outcomes are achieved
- be able to apply knowledge to identify, formulate and solve technical problems of the specialty, using methods that are most suitable for achieving goals;
- be able to develop software for embedded and distributed applications, mobile and hybrid systems, calculate, operate, typical for the specialty equipment;
- ability to develop, adapt, use software to improve the efficiency of high-performance computer systems;
- be able to evaluate the results obtained and defend the decisions with arguments

and

knowledge of:
- object-oriented programming language;
- patterns of object-oriented design;
- of tools and integrated environments for software development

skills:
- analyze software requirements;
- design software architecture;
- use patterns of object-oriented design;
- perform refactoring of program code;
- develop and debug software;
- use the necessary tools and integrated environments to solve problems

experience:
- software development;
- work with computer tools and integrated environments.

2. Prerequisites and post-requisites of the course (the place of the course in the scheme of studies in accordance with curriculum)

To successfully master the discipline "Object-Oriented Programming" in accordance with the educational program, you must first master the knowledge of the disciplines: "Programming", "Data Structures and Algorithms", "Discrete Mathematics".

Competences, knowledge and skills acquired in the study of the discipline "Object-Oriented Programming" can be used to study the disciplines "Software Engineering", "Course work of software engineering", "Parallel programming".

3 Content of the Coarse

The list of the main topics included in the program of studying the discipline "Object-oriented programming":

Section 1. Introduction to the course of OOP

Topic 1.1. An overview of the basic concepts of software development

Section 2. Introduction to the C ++ language of object-oriented programming

Topic 2.1. Basic elements of the C ++ language

Topic 2.2. C ++ preprocessor. Modularity of programs

Topic 2.3. C ++ classes. Encapsulation. Inheritance. Polymorphism

Topic 2.4. C ++ classes. Multiple inheritance

Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system

Section 3. Relationships of classes and operations with objects

Topic 3.1. UML diagrams

Topic 3.2. Nested and local classes

Topic 3.3. Operations with objects

Section 4. Generalized programming elements and standard C ++ libraries

Topic 4.1. C ++ templates

Topic 4.2. Standard C ++ libraries. Containers

Topic 4.3. Standard C ++ libraries. Algorithms, functional objects

Section 5. Organization of interaction of classes and objects

Topic 5.1. Callback functions

Topic 5.2. Interfaces

Topic 5.3. Static members

Section 6. Patterns of object-oriented design

Topic 6.1. The concept of pattern design. Singleton pattern

Topic 6.2. Varieties of Factory Patterns

Topic 6.3. Facade Patterns, Adapter, Dependency Injection, Bridge

Topic 6.4. Patterns Decorator, Observer, Visitor

Section 7. Features of object-oriented design

Topic 7.1. Object-oriented approach vs functional-procedural

Topic 7.2. SOLID principles

Topic 7.3. Refactoring

4 Coursebooks and teaching resources

Main:

1. Lectures
   https://drive.google.com/drive/folders/1yJUO8ALYikkR-RyJlegD6iEuEix52CGI?usp=sharing
2. Lab works
   https://drive.google.com/drive/folders/136iMz4R1lCSeYl8rOPldsigWXxGZYKB9?usp=sharing

Additional:

3. Bjarne Stroustrup. The C++ Programming Language 5th edition. ISBN-13: 978-1691196005
4. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. ISBN-13: 978-0201633610
5. Gerbert Schildt. C++: A Beginner's Guide. ISBN-13: 978-0072232158, 2012, 542 p.
6. Gerbert Schildt. Java: The Complete Reference, Eleventh Edition. ISBN-13: 978-1260440232, 2018, 1248 p.
7. Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts. Refactoring: Improving the Design of Existing Code. ISBN-13: 978-0201485677
8. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language 3rd Edition. ISBN-13: 978-0321193681
9. Microsoft. Overview of Windows Programming in C++. https://docs.microsoft.com/en-us/cpp/windows
10. Samuel Oloruntoba. SOLID: The First 5 Principles of Object Oriented Design. https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
11. Working Draft, Standard for Programming. Language C++. ISO/IEC N4582, 1514 pp. https://www.open-std.org/jtcl/sc22/wg21/docs/papers/2016/n4582.pdf

Educational content

5 Methodology

The educational content of the discipline consists of 18 lectures and 6 lab works

Lectures

Lecture 1

Topic 1.1. An overview of the basic concepts of software development
Varieties of programming paradigms. Program execution environments. Build a message-driven program
Tasks on self-study:
1. Varieties and examples of modern interpreters and compilers. Integrated application development environments
2. Software implementation of message handlers
3. Features of application programming for Windows and Android

4. Examples of APIs
Topic 2.1. Basic elements of the C ++ language
Alphabet, operations, operators. Conditional operators, loop operators. Simple types. Pointers. Arrays, structures. Functions: announcement, definition, call. The main function of the program and its arguments
Tasks on self-study:
1. The main function for Windows programs
2. Creating a project of several modules

Lecture 2

Topic 2.2. C ++ preprocessor. Modularity of programs
The role of the preprocessor. Macros. Features of the program code of modules on C ++. Module title. Hiding data and functions in modules. Connections between modules, hierarchy of modules.
Self-study task: Display of module hierarchy

Lecture 3

Topic 2.3. C ++ classes. Encapsulation. Inheritance. Polymorphism
Announcement of a class, definition of its members, creation of copies of objects. Restricting access to class members. Inheritance. Hierarchies of classes. Virtual functions and abstract classes. The concept and implementation of polymorphism.
Tasks on self-study:
1. Dynamic objects. Arrays of objects
2. Classes and modularity. Two roles of classes

Lecture 4

Topic 2.4. C ++ classes. Multiple inheritance
The concept of multiple inheritance. Examples of multiple inheritance. Rhombic inheritance. Virtual base classes. Problems of realization of multiple inheritance
Tasks on self-study:
1. Polymorphism in multiple inheritance
2. The need to use multiple inheritance, the possibility of abandoning it.
2. Multiple inheritance in other programming languages
Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system
The concept of API and software development capabilities. Windows messages and their role in operational system Windows. Dialog windows are modal and non-modal. Window procedure.
Tasks on self-study:
1. Programming of modal dialogue windows
2. Programming non-modal dialog boxes. Problems of organizing the work of many non-modal windows.

Lecture 5

Topic 2.5. Programming features for the Windows operating platform. Window classes. Child windows. Object orientation of Windows.
Tasks on self-study:
1. Possibilities of conveniences of programming of functions of windows. MessageCrackers and others
2. Windows programs and OOP windows
3. Possibilities of using C ++ classes for window programming

Lecture 6

Topic 3.1. UML diagrams

Possibilities of describing systems with UML diagrams. The main types, varieties of diagrams. Use-case Diagrams and Class Diagrams. Types of relations of classes and objects. Dependence, association, aggregation, composition, generalization (inheritance).
Tasks on self-study:
1. Tools for creating and drawing UML diagrams
2. Making of class diagrams in an integrated software development environment
3. Using UML diagrams to document software systems

Lecture 7

Topic 3.2. Nested and local classes
The concept of nested classes. Restrict access to members of the nested class and the span class. Local classes. Features of such classes in C ++ and in other object-oriented programming languages.
Self-study task: Differences in programming nested and local C ++ and Java classes

Lecture 8

Topic 3.3. Operations with objects
Types of operations on objects as a whole. Copying, assigning objects, adding objects. Copy constructor. Operator overload
Tasks on self-study:
1. Friendly functions
2. Global overload operators
3. Features of the implementation of overload operators C ++ and other languages

Lecture 9

Topic 3.3. Operations with objects
Transfer an object as a function parameter. Object as a result of a function call.
Overview of the possibilities of execution on objects in C ++
Tasks on self-study:
1. Features of the use of local and dynamic objects in operations on objects
2. Differences between objects references from pointers to objects

Lecture 10

Topic 4.1. C ++ templates
Template functions and template classes. Specialization templates. Expediency of templates.
Tasks on self-study:
1. The concept of generalized programming in C ++
2. Generalized programming in Java, C # and other programming languages
Topic 4.2. Standard C ++ libraries. Containers
Evolution of standard C and C ++ libraries. The composition of the standard C ++ library. Standard Template Library (STL). Types of library classes. Class complex. Class string.
Tasks on self-study:
1. Using the string class for different types of character encoding
2. Convert string data to char, wchar and vice versa

Lecture 11

Topic 4.2. Standard C ++ libraries. Containers
The concept of container class. Requirements for custom classes as container elements. Classes vector, list, map
Tasks on self-study:

1. Use of standard containers for storage of graphic objects of Shape classes from laboratory works
2. Classes array, multimap and their possible use
Topic 4.3. Standard C ++ libraries. Algorithms, functional objects
The concept of a library of algorithm templates. Template classes of algorithms and their members. Iterators. List of standard algorithms. Algorithms count_if, find_if, for_each, remove.
Functional objects. Classes of standard functional objects. Examples of the use of functional objects.
Tasks on self-study:
1. Evolution of algorithm templates in different standards C ++ (C ++ 11, C ++ 14 and others)
2. Evolution of functional objects in different standards C ++ (C ++ 11, C ++ 14 and others)

Lecture 12

Topic 5.1. Callback functions
The need to organize a callback function. Examples of callback functions. Possibilities of callback programming in C ++. Callback functions and use of the Windows API. Organization of connections between modules using callback. Possibilities of realization of callback technik by C ++ classes.
Tasks on self-study:
1. Features of the CALLBACK window procedures in Windows.
2. Affinity of callback and functional objects.
Topic 5.2. Interfaces
Interface class declaration. Features of interface class members. Using interface classes. Features of implementation of classes-interfaces in C ++. Using the interface keyword in various C ++ implementations. Expediency of interface classes in program architecture.
Tasks on self-study:
1. The correlation of interfaces and abstract classes
2. Features of interfaces in Java, C# and other programming languages

Lecture 13

Topic 5.3. Static members
Features of static local variables. Static members of class. Expediency of using static class members. Static class member functions.
Tasks on self-study:
1. Several contexts of using the word static
Topic 6.1. The concept of pattern design. Singleton pattern
The concept of pattern design and programming. Expediency of patterns. Book Design Patterns. Classification of patterns. The Singleton pattern and its implementations are the classic Meersa Singleton.
Tasks on self-study:
1. Possibilities of programming the Singleton pattern for the editor of objects in laboratory work
2. Problems of implementation of Singleton pattern in C ++ for multithreaded applications

Lecture 14

Topic 6.2. Varieties of Factory Patterns
Pattern Simple Factory. The expediency of such a pattern. Pattern Factory Method. Pattern Abstract Factory.
Task on self-study: The feasibility of using patterns Factory Method and Abstract Factory.

Lecture 15

Topic 6.3. Facade Patterns, Adapter, Dependency Injection, Bridge

The expediency of creating the facade of the software system. Assignment and implementation of the Adapter pattern. Patern Bridge.
Tasks on self-study:
1. Why separate the user of the software system from the internal classes?
2. What is the Injection technique based on?
3. Differences of the Bridge pattern from the Strategy pattern

Lecture 16

Topic 6.4. Patterns Decorator, Observer, Visitor
The main purpose of the pattern is Decorator. Observer pattern and its components. Implementation of monitoring the state of several objects. Visitor pattern and possibilities to expand the functionality of the class system.
Tasks on self-study:
1. Features of programming classes according to the pattern Decorator.
2. The role of the Visitor pattern in overcoming the Expression Problem
Topic 7.1. Object-oriented approach vs functional-procedural
An example to illustrate the relationship between object-oriented and functional-procedural approaches. Duality. Expression Problem
Task on self-study: Analyze the capabilities and means of object-oriented programming of messages in applications for Windows and other operating platforms.

Lecture 17

Topic 7.2. SOLID principles
Features of bad projects. Overview of the principles that are part of SOLID: Single responsibility, Open / Closed, Liskov, Interface segregation, Dependency inversion. Examples of program code.
Tasks on self-study:
1. Antipatern God object
2. Dependensy inversion technique for managing dependencies of software system modules

Lecture 18

Topic 7.3. Refactoring
The concept of refactoring. Iterative development and improvement of code. Classification of methods and techniques of refactoring. Examples of refactoring.
Tasks on self-study:
1. Replacement switch by polymorphism
2. Analyze the possibilities of refactoring the code of laboratory work of the object editor
Module control work

Laboratory works

Laboratory work 1

Get acquainted with the Microsoft Visual Studio software development environment and compile modular program projects in C ++

Objective: to get the first skills of creating programs for Windows based on Win32 API projects for Visual C ++ and learn modular programming in C ++.

To perform the work requires the study of individual information on the following topics:

- Topic 1.1. An overview of the basic concepts of software development. In particular, issues related to the construction of programs managed by graphical user interface messages
- Topic 2.2. C ++ preprocessor. Modularity of programs. In particular, you need to learn how to compose multi-module projects from several files in a C ++ software development environment.

- Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system

You also need to learn how to develop software loosely coupled component modules with minimal outsourcing of the details of the implementation of each module. It is necessary to develop the simplest interface of interaction between modules

Laboratory work 2

Development of a graphical object editor in C ++

The aim of the work is to gain the ability and skills to use encapsulation, type abstraction, inheritance and polymorphism based on C ++ classes, programming a simple graphic editor in an object-oriented style.

To perform the work requires the study of individual information on the following topics:

- Topic 2.3. C ++ classes. Encapsulation. Inheritance. Polymorphism
- Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system

It is necessary to master the question of creating a hierarchy of classes for the description of graphic objects and to provide message processing for writing an array of objects and polymorphic mapping

Laboratory work 3

Development of the user interface on C ++

The purpose of the work is to gain the ability and skills to use encapsulation, type abstraction, inheritance and polymorphism based on C ++ classes, programming a graphical user interface. In addition to learning the features of programming applications with a graphical interface, message-driven, students learn to document programs with UML diagrams, in particular, master class diagrams.

To perform the work requires the study of individual information on the following topics:

- Topic 2.3. C ++ classes. Encapsulation. Inheritance. Polymorphism
- Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system
- Topic 3.1. UML diagrams

Laboratory work 4

Improving the code structure of the graphical object editor in C ++

The purpose of the work is to gain the ability to design classes by upgrading the code of the graphic editor in an object-oriented style to ensure the convenient addition of new types of objects. In fact, refactoring is performed to perform the work. Students gain certain skills in developing complex projects through iterative improvement.

To perform the work requires the study of individual information on the following topics:

- Topic 2.3. C ++ classes. Encapsulation. Inheritance. Polymorphism
- Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system
- Topic 3.1. UML diagrams
- Topic 7.3. Refactoring

Laboratory work 5

Development of a multi-window user interface for a graphical object editor

The purpose of the work is to gain the ability and skills to program the multi-window program interface in C ++ in an object-oriented style.

To perform the work requires the study of individual information on the following topics:

- Topic 2.3. C ++ classes. Encapsulation. Inheritance. Polymorphism
- Topic 2.4. C ++ classes. Multiple inheritance
- Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system
- Topic 6.4. Patterns Decorator, Observer, Visitor

Laboratory work 6

Build a software system from a variety of message-driven objects

Purpose: to gain the ability and skills to use the means of information exchange and to program the interaction of independently operating software components.

The main task will be to build an interface of messages between the objects-components of software systems. To do this, you must select a specific means of messaging, determine the format of the message, the means of its transmission and reception.

To perform the work requires the study of individual information on the following topics:

- Topic 2.5. Programming features for the Windows operating platform. Object orientation of the system
- Topic 3.1. UML diagrams
- Topic 6.4. Pattern Decorator. Observer, Visitor
- Topic 7.1. Object-oriented approach vs functional-procedural

Questions for Modular Control Work

1. Software development. Interpreters and compilers
2. Messages such as Message Driven Application. How are they programmed?
3. Programming paradigms
4. Modularity of C ++ programs. Modularity strategy
5. The structure of the source text of the module C ++
6. C ++ classes. Members. Advertisement. Definition
7. C ++ classes. Designer. Default constructor. Destructor
8. C ++ classes. Ways to create instances of objects
9. C ++ classes. Dynamic objects. Array of objects
10. Classes and modularity. Two roles of classes
11. Inheritance of classes. Concept. Examples
12. Possibilities of restricting access to members of the base class in derived classes
13. Virtual functions. Abstract classes
14. Polymorphism based on classes C ++. Example for an array of objects
15. The concept of multiple inheritance of C ++ classes
16. Rhombic inheritance
17. Virtual base classes
18. Polymorphism in multiple inheritance
19. Static class members and how they are used
20. Nested and local classes
21. Identification of types during program execution. RTTI
22. Overloading operators. Basic concepts. Limitation
23. Overloading operators. The operator as a member function of the class. Examples
24. Operator overload “=“
25. Overload of the "+" operator
26. Overload of the "+" operator for the case: number + object
27. Friendly functions. What are their benefits? Example
28. Operator overload []

29. Assigning objects: B = A
30. Initialization of one object by another object: classname B = A
31. Transfer of object through parameters of function: Func (obj)
32. Object as a result of the function: obj = Func ()
33. Copy Designer. Example
34. Overview of operations on objects: B = A, classname B = A, Func (obj), obj = Func (), classname * p = Func () and ways to solve problems
35. The concept of C ++ templates. Example
36. Template functions C ++. Specialization function templates. Example
37. Template classes C ++. Example of a template class
38. Standard C ++ library
39. Class complex. Example of use
40. Class string. Example of use
41. Class vector. Example of use
42. Class list. Example of use
43. Class map. Example of use
44. Standard algorithms of the C ++ template library
45. Count_if algorithm. Example of use
46. Find_if algorithm. Example of use
47. Algorithm for_each. Example of use
48. Remove algorithm. Example of use
49. Polymorphism. Rationale. Interpretation
50. Interface classes and why they are needed
51. Features of C -+ interface classes
52. The concept of Callback - functions. Justification of the need for such functions
53. Examples of Callback - functions
54. Implementation of Callback - functions using pointers
55. Implementation of CallBack-functions with virtual functions
56. UML diagrams
57. Diagram of precedents
58. Diagram of classes. Construction by means of Microsoft Visual Studio
59. Multiple inheritance and program structure
60. Replacement of multiple inheritance - what and how?
61. The main types of relationships between classes, objects in class diagrams
62. Relationship Dependence, Association on UML diagrams
63. Relationships Generalization, Aggregation, Composition on class diagrams
64. Composition vs Inheritance
65. The concept of design template. Review of patterns
66. Patern Odinak (Singleton). Known implementations of this pattern
67. Pattern Simple Factory
68. Patern Factory Method
69. Pattern Abstract Factory
70. Pattern Builder
71. Pattern Dependency Injection
72. Patern Façade
73. Pattern Adapter
74. Patern Bridge
75. Pattern Decorator
76. Pattern Observer

77. Visitor Pattern
78. Procedural approach vs Object-oriented. Expression Problem
79. Principles of object-oriented design and programming. SOLID principles
80. Demeter's law
81. Refactoring in OOP

6. Self-study

Students should consolidate the knowledge gained during lectures and deepen their knowledge for further study. In addition, independent work is required when performing laboratory work and calculation and graphic work.

The list of tasks on the self-study is presented in the information on Lectures

Policy and Assessment

7 Cource Policy

All students must attend lectures and laboratory classes - both in regular classroom learning (physical attendance) and distance learning (virtual attendance).

The results of laboratory work and individual tasks are drawn up in electronic format in the form of report files, executable files and other files. Such files should contain the results in accordance with the tasks, requirements and guidelines for each work. The work of the programs is checked by the teacher during the acceptance of works.

Works that are submitted in violation of deadlines without good reason are evaluated at a lower grade. The greater the delay, the lower the score.

All written works are checked for plagiarism. Plagiarism significantly reduces the evaluation, and significant borrowing of someone else's text can lead to unsatisfactory evaluation of the work. Write-offs during control works are forbidden (including with use of mobile devices).

8 Monitoring and Grading Policy

The final score ($\boldsymbol{R_D}$) of a student in the discipline "Object-Oriented Programming" consists of points that he gets for:

- 6 laboratory works ($\boldsymbol{R_{LAB}}$);
- modular control work ($\boldsymbol{R_{MCW}}$);
- exam ($\boldsymbol{R_E}$).

According to the "Regulations on the system of evaluation of learning outcomes in Igor Sikorsky KPI" (https://osvita.kpi.ua/sites/default/files/downloads/Pol_systema_ociniuvannia.pdf), approved by Order №1 / 273 of 14.09.2020, used a rating system type RSO-2, for which the rating consists of:

- starting assessment ($\boldsymbol{R_C}$) - assessment of activities during the semester: $\boldsymbol{R_C = R_{LAB} + R_{MCW}}$,
- examination grade $\boldsymbol{R_E}$

Thus, $\boldsymbol{R_D = R_C + R_E}$

Calculation of evaluation scales

Evaluation scale (maximum possible evaluation) of one laboratory work: 8 points.
Evaluation scale of 6 laboratory works: ***RLAB*** = 6 × 8 = 48 points
Scale of assessment of modular control work: ***RMCW*** = 12 points

Scale of initial estimation: ***RC = RLAB + RMCW*** = 60 points
Examination grade scale: ***RE*** = 40 points
Complete overall rating scale: ***R = RC + RE*** = 60 + 40 = 100 points
The minimum possible overall positive score is: ***RMIN*** = 0.6 × 100 = 60 points

Determining the conditions of admission to the exam

The condition for admission to the exam is the completion of tasks with a certain success during the semester, ie, the student must earn some non-zero starting rating during the semester ($R_C > 0$).

The minimum starting rating for admission to the exam (***RMINADM***) is determined from the assumption that having received the maximum possible grade on the exam (***RE***) the student will eventually have a minimum positive grade (***RMIN***). Then ***RMINADM = RMIN - RE*** = 60 - 40 = 20 points. Thus, the condition for admission to the exam will be R_C >= ***RMINADM***, ie, the value of the starting rating R_C must be at least 20 points.

Thus, subject to admission to the exam, the student as a result of the exam will have a final rating (R_D), which will be the sum of all grades obtained in points. The final rating is also translated into a grade on a university scale.

Table of translation of the final score in the university grading scale

Final score R_D	University grade
95...100	Excellent
85...94	Very Good
75...84	Good
65...74	Satisfactory
60...64	Sufficient
Below 60	Fail
Starting assessment (***Rc***) less than 20	Not admitted

Syllabus of the Course

Is designed by teacher Associate Professor, Ph.D, Victor Porev

Adopted by Department of Computing Engineering (protocol № 10 , 25 May 2022)

Approved by the Faculty Board of Methodology (protocol № 10 , 9 June 2022)