УДК 811.111: 004.432.2

Matsak I., Vanik I.

Object-oriented Programming

Belarusian National Technical University
Minsk, Belarus

Object-oriented programming combines a group of data attributes with functions or methods into a unit called an "object." Typically, OOP languages are class-based, which means that a class defines the data attributes and functions as a blueprint for creating objects, which are instances of the class. Multiple independent objects may be instantiated or represented from the same class and interact with each other in complex ways.

A simple example would be a class representing a person. The person class would contain attributes to represent information such as the person's age, name, height, etc. The class definition might also contain functions such as "sayMyName" which would simply print that person's name to the screen. A family could be constructed by instantiating person objects from the class for each member of the family. Each person object would contain different data attributes since each person is unique [1].

This programming style is pervasive in popular programming languages such as Java, C++, Python, JavaScript and C# among others. By defining sets of classes that represent and encapsulate objects in a program, the classes can be organized into modules, improving the structure and organization of software programs. Thus, developers often use OOP as a tool when they need to create complex programs since they are easier to perceive in terms of classes and their relationships.

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance, and polymorphism. Even if these concepts seem incredibly complex, understanding the general framework of how they work will help you understand the basics of an OOP computer program. Below, we outline these four basic principles and what they entail.

The word, “encapsulate,” means to enclose something. Just like a pill "encapsulates" or contains the medication inside of its coating, the principle of encapsulation works in a similar way in OOP: by forming a protective barrier around the information contained within a class from the rest of the code.

In OOP, we encapsulate by binding the data and functions which operate on that data into a single unit, the class. By doing so, we can hide private details of a class from the outside world and only expose functionality that is important for interfacing with it [2].

Often, it’s easier to reason and design a program when you can separate the interface of a class from its implementation, and focus on the interface. It looks like a “black box,” where it’s not important to understand the gory inner workings in order to reap the benefits of using it. This process is called “abstraction” in OOP, because we are abstracting away the complex implementation details of a class and only presenting a clean and easy-to-use interface via the class member functions. Carefully used, abstraction helps isolate the impact of changes made to the code, so that if something goes wrong, the change will only affect the implementation details of a class and not the outside code.

Object-oriented languages that support classes almost always support the notion of “inheritance.” Classes can be organized into hierarchies, where a class might have one or more parent or child classes. Therefore, if a class inherits from another class, it automatically obtains a lot of the same functionality and properties from that class and can be

extended to contain separate code and data. A nice feature of inheritance is that it often leads to good code reuse since a parent class functions don't need to be re-defined in any of its child classes.

In OOP, polymorphism allows for the uniform treatment of classes in a hierarchy. Therefore, calling code only needs to be written to handle objects from the root of the hierarchy, and any object instantiated by any child class in the hierarchy will be handled in the same way.

Because derived objects share the same interface as their parents, the calling code can call any function in that class interface. At run-time, the appropriate function will be called depending on the type of object passed leading to possibly different behaviors.

Object Oriented programming requires thinking about the structure of the program and planning at the beginning of coding. Looking at how to break up the requirements into simple, reusable classes that can be used to blueprint instances of objects. Overall, implementing OOP allows for better data structures and reusability, saving time in the long run [3].

References:

1. SOLID Rules in Object-Oriented Programming [Electronic resource]. – Mode of access: https://community-z.com/communities/epam-poland/articles/1190. – Date of access: 14.11.2021.
2. The four principles of OOP [Electronic resource]. – Mode of access:https://seramasumi.github.io/docs/Tech%20Knowledge/The%20four%20principles%20of%20OOP.html. – Date of access: 14.11.2021.
3. OOP Explained in Depth [Electronic resource]. – Mode of access: https://www.educative.io/blog/object-oriented-programming – Date of access: 14.11.2021.