Основни принципи на обектно ориентираното програмиране

# Въведение

Двете основни програмни парадигми[[1]](#footnote-1)(paradigms) са императивната и декларативната. Основната разлика между двете е, че при императивното програмиране програмистът задава конкретни инструкции как да работи програмата, а при декларативното програмистът задава правила, от които компилаторът сам прави програма (това е доста съкратено обяснение!). Обектно ориентираното програмиране спада към императивното програмиране (програмистът казва къде са циклите, какви са променливите, коя функция да се извика).

На кратко при обектно ориентираното програмиране програмата представлява взаимодействие между обекти наречени класове (обекти са по-точно инстанции на класове). Всеки клас съдържа данни и функции, които оперират върху данните. Всеки клас има интерфейс, през който комуникира с „външния свят“(те. с другите класове).

# ООП – основни принципи

Основните принципи в обектно ориентираното програмиране са:

1. Абстракция (abstraction) – детайлите на имплементацията са скрити и програмистът е предоставил лесен за разбиране от потребителя интерфейс.
2. Капсулация (encapsulation) – всеки клас има публичен интерфейс само и единствено, през който може да комуникира с външния свят.
3. Наследяване (inheritance) – позволява преизползване на код. Ако имаме няколко различни класа, които трябва да съдържат общи функции или данни, може да извлечем общите неща в отделен клас (клас родител), който после другите класове да наследят.
4. Полиморфизъм (polymorphism) – позволява да имаме един общ интерфейс, зад който може да стоят обекти с една и съща функция, но различна имплементация на тази функция, като това става по време на изпълнение на програмата.

## Обекти (класове)[[2]](#footnote-2)

При ООП кодът на една програма се разделя в класове, като всеки клас изпълнява някаква определена функция. Това улеснява работата в екип, всеки член на екипа е отговорен за направата на някой клас и в общия случай не е нужно да знае другите как класовете на другите работят отвътре. Нужно е само да знае интерфейса на класа и как да го използва. Това разбиване позволява и отделните части на програмата да се тестват по отделно. Това също прави програмата доста по-модулна: може да променим отвътре даден клас без да чупим цялата програма.

## Абстракция

Всеки клас предоставя интерфейс, с който може да общува с други класове, като другите класове не е нужно да знаят тънките детайли вътре в него. Всъщност абстракцията е в основата на човешкото мислене.

Отключването на врата е пример за абстракция. Всички знаем, че като сложим точно определен ключ в ключалката и го завъртим, ще е трудно за някой крадец да влезе. Не ни е нужно да знаем дали ключалката е disc detainer lock, pin tumbler lock или друг вид.

За пример свързан с програмирането нека вземем класа std::set от STL на C++. Това е клас контейнер, за различни елементи от един и същи тип, които са сравними. Обикновено структурата данни, която е в std::set е двоично дърво за търсене[[3]](#footnote-3), но това не е нужно да го знаем, за да ползваме класа.

## Капсулация

Отново класът предоставя интерфейс, с който да общува с външния свят. Това не позволява някой неволно да бърка в класа и да променя състоянието му (данните му), защото в общия случай това води до невалиден/счупен обект. Всеки клас сам се грижи да е в добро състояние.

## Наследяване

Позволява преизползването на код. Често правим отделен клас за кода, който ползваме на няколко места, и наследяваме този клас. Класа, от който наследяваме наричаме базов клас/клас родител (base class/parent class/superclass), а наследяващия клас: клас наследник (child class/subclass). Наследяването се ползва при полиморфизма.

Имаме и други начини да преизползваме код: функции и член данни. Важно е да преценим дали алтернативните начини не са по-добър вариант от наследяването.

## Полиморфизъм

Не пиша обяснение, защото още не разбирам напълно как и защо се използва. Все пак по-долу има пример за ползване на полиморфизъм в C++.

1 Класически пример за полиморфизъм в C++



2 Изходът на горния пример.



1. Парадигма: модел, начин на мислене конкретно в някоя научна дисциплина. [↑](#footnote-ref-1)
2. Тук използвам обекти и класове взаимно заменяемо, но клас е шаблон за правене на обекти (да не се бърка с template), а обект е нещото, което „живее“ в паметта на компютъра по време на изпълнение на програмата. Също обект от даден клас наричаме инстанция на този клас. [↑](#footnote-ref-2)
3. <https://cplusplus.com/reference/set/set/> [↑](#footnote-ref-3)