**Czym są wzorce projektowe?**

Wzorce projektowe to sprawdzone, przetestowane i powszechnie stosowane rozwiązania konkretnych problemów programistycznych. Są to również wskazówki i wytyczne mówiące o tym w jaki sposób podejść do danego zagadnienia.

Wzorce projektowe najczęściej są stosowane w programowaniu obiektowym i pokazują zależności między klasami i obiektami oraz korzyści wynikające z tych zależności. Stosowanie wzorców projektowych pozwala na pisanie lepszych, bardziej efektywnych, skalowalnych i mniej narażonych na błędy programów.

Korzyści ze wzorców projektowych:

* **Są one sprawdzonymi rozwiązaniami.** Z racji tego, iż są wielokrotnie używane i ciągle optymalizowane przez dużą liczbę programistów, możemy być po pewni, że będą dobrze działały.
* **Łatwo ich re-używać.** Wzorzec przedstawia rozwiązanie, które można dostosować do własnych potrzeb.
* **Są łatwe do zrozumienia.** Duże i trudne rozwiązania są przedstawione w dość przystępny sposób.
* **Ułatwiają komunikację w zespole.** Zamiast dokładnie opisywać w jaki sposób rozwiązaliśmy dany problem, możemy powiedzieć z jakiego wzorca skorzystaliśmy.
* **Zmniejszają potrzebę refaktoryzowania kodu.** Korzystając ze wzorca problem jest zazwyczaj rozwiązany w sposób prawidłowy.
* **Zmniejszają ilość kodu.** Rozwiązania prezentowane we wzorcach są najczęściej już wielokrotnie sprawdzone i maksymalnie zoptymalizowane.

## Skąd się wzięły wzorce projektowe?

Twórcą tego pojęcia jest [Christopher Alexander](https://en.wikipedia.org/wiki/Christopher_Alexander), amerykański architekt który postawił tezę, że funkcjonalność oraz inne cechy użytkowe i konstrukcyjne można zapisać w postaci uogólnionych rozwiązań. Niestety idee Alexandra nie odbiły się szerokim echem w dziedzinie architektury, jednak były impulsem dla rozwoju projektowania oprogramowania.

Przełomowym wydarzeniem było wydanie książki pt. “[*Design Patterns: Elements of Reusable Object-Oriented Software*](https://en.wikipedia.org/wiki/Design_Patterns)”, autorstwa tzw. *Gang Of Four*(*GoF*). Po raz pierwszy zebrano i sklasyfikowano wtedy 24 wzorce projektowe dotyczące konstrukcji, struktury i zachowania obiektów w systemach informatycznych.

## Rodzaje wzorców projektowych

Wzorce projektowe JavaScript można podzielić na trzy główne kategorie:

* **Konstrukcyjne**
* **Strukturalne**
* **Behawioralne**

## Wzorce konstrukcyjne (Creational Patterns)

Wzorce konstrukcyjne skupiają się na tym w jaki sposób możemy tworzyć nowe obiekty w zależności od aktualnych potrzeb.

Do wzorców konstrukcyjnych zaliczamy:

* [Constructor](https://frontstack.pl/constructor-design-pattern/)
* [Module](https://frontstack.pl/module-design-pattern)
* [Singleton](https://frontstack.pl/singleton-design-pattern/" \t "_blank)
* Simple Factory
* Factory Method
* Abstract factory
* Prototype
* Builder

## Wzorce strukturalne (Structural Patterns)

Nasze aplikacje niemalże nigdy nie składają się tylko z jednego rodzaju obiektu. Przeważnie mamy do czynienia z bardzo dużą liczbą różnego rodzaju obiektów, które dopiero gdy zaczną ze sobą prawidłowo *współpracować* stworzą kompletny system. W celu poprawnej komunikacji i bezkonfliktowej pracy poszczególnych części systemu będziemy stosować właśnie wzorce strukturalne. Wzorce te pomogą nam budować zależności między obiektami w taki sposób, że gdy zmienimy jeden fragment systemu, nie będzie to skutkowało koniecznością wprowadzania zmian w innych jego częściach.

Do wzorców strukturalnych zaliczamy:

* [Facade](https://frontstack.pl/facade-design-pattern/)
* [Flyweight](https://frontstack.pl/flyweight-design-pattern/" \t "_blank)
* [Adapter](https://frontstack.pl/adapter-design-pattern/)
* Decorator
* Composite
* Bridge
* Proxy

## Wzorce behawioralne (Behavioral Patterns)

Wzorce behawioralne opisują jak realizować podział obowiązków pomiędzy obiektami a także definiują metody komunikacji pomiędzy nimi. Jest to szczególnie pomocne, gdy chcemy mieć pewność, iż informacje w całym systemie są poprawnie synchronizowane między poszczególnymi komponentami.

Do wzorców behawioralnych zaliczamy:

* [Mediator](https://frontstack.pl/mediator-design-pattern/)
* [Observer](https://frontstack.pl/observer-design-pattern/)
* Command
* Chain of responsibility
* Interpreter
* Iterator
* Memento
* State
* Strategy
* Template method
* Visitor