# Structural design patterns

While working with a large application, you will usually encounter problems related to the complexity of a code.
This may disorient and complicate your work.
So, to minimise these problems, you should define the proper class structure and their connections to each other.

## What are structural design patterns?

**Structural design patterns** describe relations between the system's components.

Like any other design pattern, structural patterns solve the most common design problems.

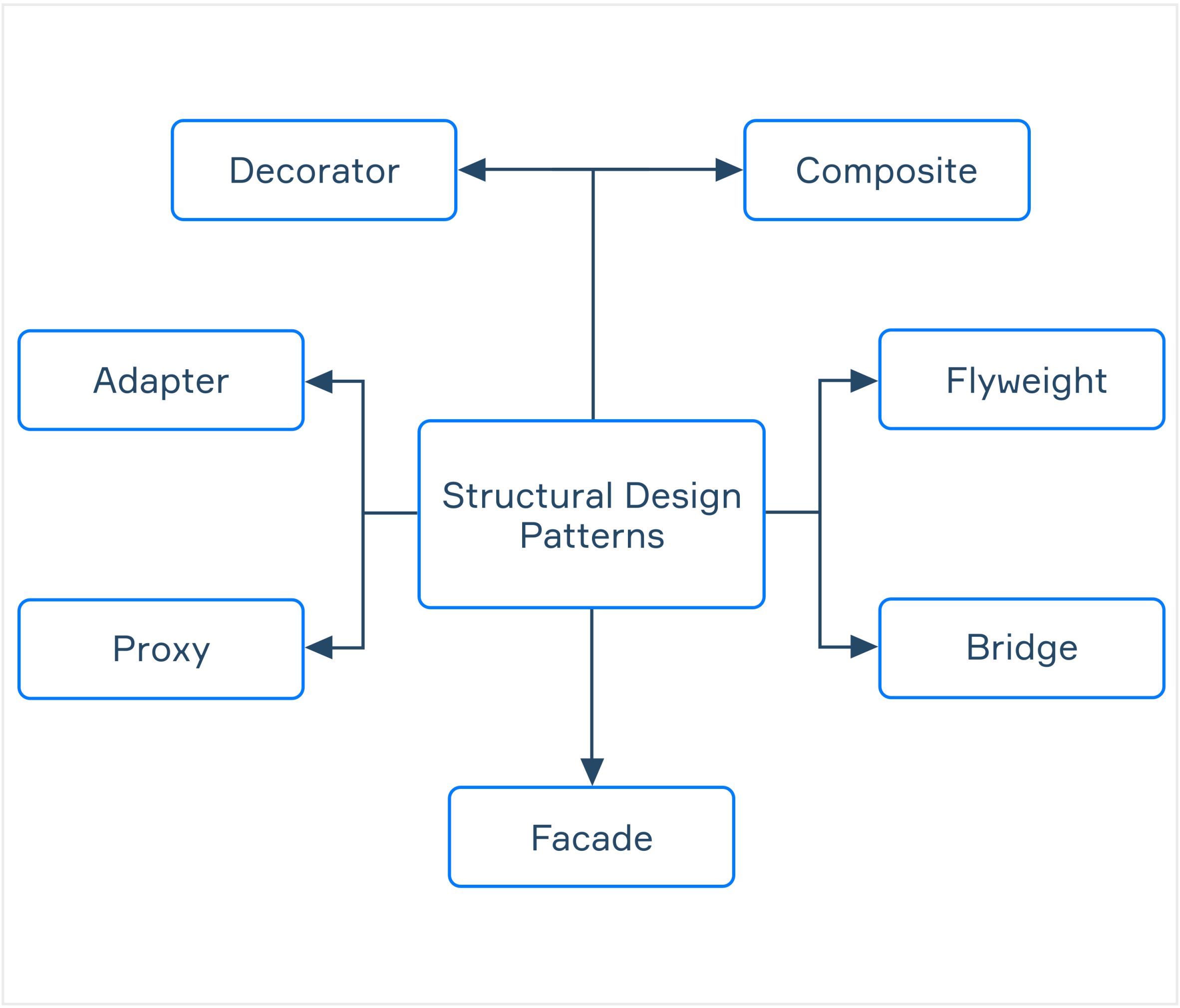
Through them, you can ease the application design process, by introducing solutions for recurring problems of component structuring.

Structural patterns simplify the structure by defining the relationships between objects.

Using these patterns, you will have a better understanding of whether we need to inherit, compose, or make and maintain any other relationships between system components.

## Types of structural patterns

There are 7 types of structural design patterns:



- **Adapter -** as the name suggests, this pattern focuses on connecting objects with incompatible interfaces.
- **Bridge -** this pattern separates abstraction and implementation of a class, allowing for their independent development.
- **Composite -** a pattern that structures objects in a hierarchical fashion so that a client can work with each one of them.
- **Decorator -** this pattern uses special wrapper objects for enhancement of object behaviour without original object modification.
- **Facade -** a pattern that provides an interface for a complex set of objects.
- **Flyweight -** a pattern that increases the number of objects you can fit into memory by sharing and reusing their common parts.
- **Proxy -** this pattern provides a substitute or placeholder for another

complex object.

A variety of these design patterns allow for the formation of a larger structure.

They define how each component should be structured so as to have very flexible interconnecting modules that can work together in a larger system.

### When to use structural patterns?

Here's the most basic description of these patterns applicability:

- **Adapter:** use this when you want to use some existing class with an incompatible interface.
- **Bridge:** use this pattern when you want to split larger classes and form structures that have several variants of some functionality.
- **Composite:** apply this pattern when you want your client to work with different elements of a structure uniformly.
- **Decorator:** use this pattern to add modifications to objects' behavior without interference with their code.
- **Facade:** use this pattern to provide a simple and straightforward interface to a complex system.
- **Flyweight:** apply this pattern to a project with a large number of objects to fit them all into your memory.
- **Proxy:** this pattern has various implementation variants, from controlling user access to maintaining heavy resources in the background.