Design Patterns

# D:\4. Documents\Bressan\Books\Design Patterns\DesignPatterns-table.png

# Strategy

**“Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.”**

**Description:**

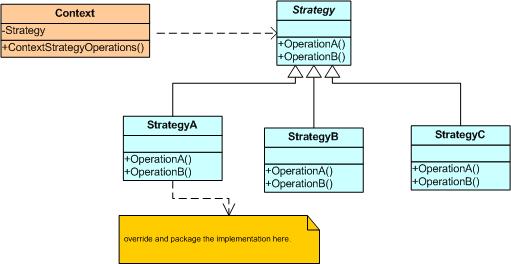
* Like a **Command** design pattern encapsulates some kind of behavior separated in a class.
* Work with different algorithms with different behavior but each of them do same work.
* Includes methods which to be implemented from it inheritors.
* The **strategy** pattern uses composition instead of inheritance.
* Composition means to pass “behavior” as a parameter of method.
* Encapsulates an algorithm inside a class.
* Making each algorithm replaceable by others.
* All the algorithms can work with the same data transparently.
* The client can transparently work with each algorithm.

Examples:

Checker system in BgCoder: Check(ICheckerStrategy checker) - TrimChecker, SortChecker, ExactChecker

Selection of appropriate sorting/searching algorithm - applied as a class implements interface ISortStrategy (e.g.)

UML Diagram:



Some additional information on the Strategy pattern:

[Wikipedia - Strategy Pattern](http://en.wikipedia.org/wiki/Strategy_pattern).

[Data & Object Factory - Strategy Pattern Overview](http://www.dofactory.com/Patterns/PatternStrategy.aspx).

# Facade

“**Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.**”

**Description:**

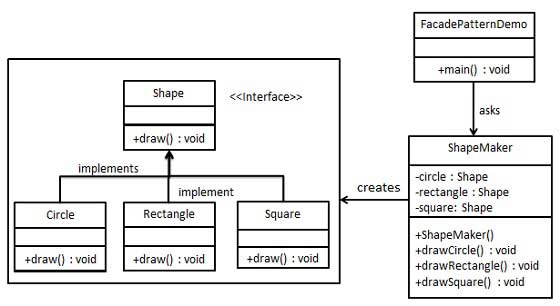
* To separate given functionality and to simplify/avoid subset of classes when want to create/use object.
* To deliver convenient interface from higher level to group of subsystems or single complex subsystem.
* Implementing this pattern our clients can use a simple interface, but if they really need to, can still go straight to the subsystem classes themselves, as opposed to the **Adapter** pattern, where you only talked to the **Adapter**.

Examples:

In C#: File.ReadAllText() method hides complexity (open/dispose stream, reading byte streams, convert them to strings, etc.)

Façade pattern used in many Win32 API based classes to hide Win32 complexity

UML Diagram:



Some additional information on the Facade Pattern:

[Wikipedia - Facade Pattern](http://en.wikipedia.org/wiki/Facade_pattern).

[Data & Object Factory - Facade Pattern Overview](http://www.dofactory.com/Patterns/PatternFacade.aspx).

# Builder

**“Separate the construction of a complex object from its representation**

**so that the same construction process can create different representations.”**

**Description:**

* When the creational of objects made in steps.
* Creates an object in steps in specific order that object depends on many things.
* Defines a class understands the sequence of this steps.
* Separation of logic and data.

**Solve 3 types of problems:**

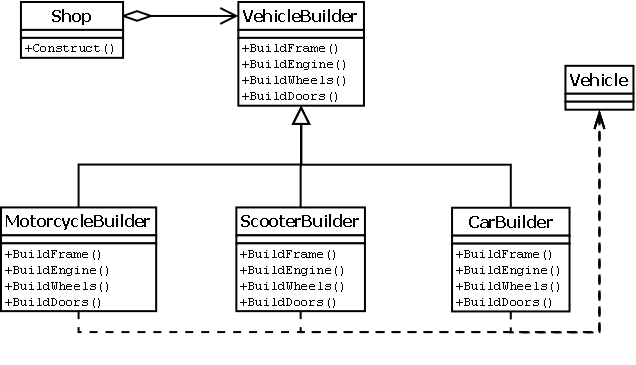
* + too many parameters
  + order dependent
  + different constructions

Examples:

Shop (Director) -> use specific -> Vehicle builder -> to -> Construct a vehicle

1. **Builder** is used by Director (who defines steps how to build a vehicle)
2. **Builder** is implemented by a concrete builder
3. Product is produced by the concrete builder

UML Diagram:



Some additional information on the Builder Pattern:

[Wikipedia - Builder Pattern](http://en.wikipedia.org/wiki/Builder_pattern).

[Data & Object Factory - Builder Pattern Overview](http://www.dofactory.com/Patterns/PatternBuilder.aspx).