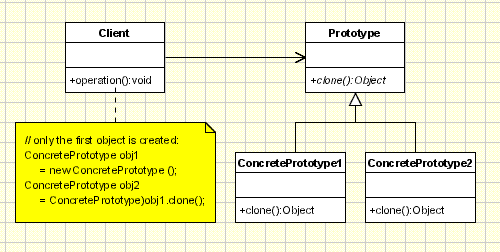
# Design patterns

## Creational pattern - Prototype

Prototype Design pattern-а се занимава с това да предостави функционалност, чрез която по-лесно можем на клонираме даден обект вместо да правим нов. Ако инстанцирането на един нов обект отнема много време, както и много ресурс, по-добрият вариант е да клонираме вече инициализиран обект. Protype е pattern-a, който позволява новият обект да клонира вече създаден, приемайки всички негови свойства. От тази гледна точка звучи много като Factory, но разликата се крие в това, че Factory-то никога не съдържа повече от един обект.

Предимствата, които предоставя Prototype са, че специфицира типа на обекта, за да го създаде, използвайки прототипната инстанция и че създаването на нов обект става чрез копирането на този прототип.

Prototype pattern-a се имплементира, като се използва абстрактен клас и само три типа класове, които пък правят имплементацията му още по-лесна.



Имплементацията се състои от три части: Client, който създава нов обект с помощта на прототипа, за да може да се клонира, Prototype, който пък декларира интерфейса, който му позволява да се самоклонира и ConcretePrototype (конкретните прототипи), които имплементират операцията за самоклониране.

Prototype pattern-a много често се използва в ситуации, в които трябва да се генерират много обекти, които много често сами по себе си се повтарят, като например игрите. Prototype е полезен в този случай, защото вместо да hardcode-ваме различните обекти, използвайки нова операция, главния метод CreateGame() ще бъде параметризиран с различни прототипни обекти, така че компизицията на картата да може много лесно да се променя със заместването на един прототипен обект с друг.

## Behavioral pattern – Strategy

Strategy pattern-a е възникнал поради случаите, в които различията между някои класове се състои само в поведението им. Именно и този проблем решава Strategy, като дава възможността да се избират различни алгоритми, които могат да се приложат в определена ситуация.

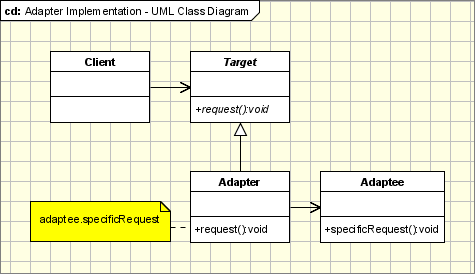
Целта на Strategy е да дефинира набор от алгоритми, капсулирайки всеки от тях, правейки ги непроменяеми, както и независими един от друг за клиента, от когото ще бъдат използвани.



Strategy се имплементира, като се дефинира един интерфейс IStrategy, който обикновенно поддържа всички алгоритми. Context-а пък използва този интерфейс, за да извика алгоритмите дефинирани от ConcreteStrategyA, B, C, който пък от своя страна си override-ват по свой си начин методите, имплементирани от Strategy интерфейса.

## Structural pattern – Adapter

Главната цел на Adapter-а е да преобразува даден интерфейс на клас в друг очакван интерфейс. По този начин той позволява на класовете да работят заедно, което иначе не би могло да се случи, поради несъответствие в интерфейсите.   
Следващата UML диаграма описва подробно Adapter Pattern-a.



От диаграмата лесно можем да забележим, че Target дефинира специфичен интерфейс, който клиента използва. Adapter-a адаптира интерфейса Adaptee към интерфейса на Target. Client пък играе ролята на връзка с обектите.

## 