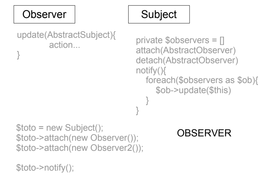
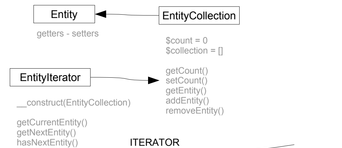
OBSERVER



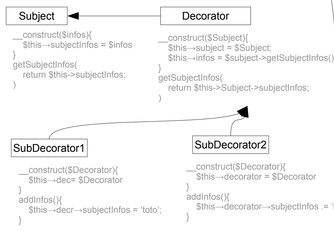

```
$toto = new Subject();
$toto->attach(new Observer());
$toto->attach(new Observer2());

$toto->notify();
```

ITERATOR



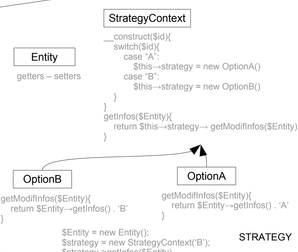
DECORATOR

subject is left untouched
the Decorator holds all the infos and gets modified
the SubDecorators don't return anything but modify the Decorator




```
$subject    = new Subject('infos');
$decorator = new Decorator($subject);
$subDec1  = new SubDecorator1($decorator);
$subDec2  = new SubDecorator2($decorator);

$subDec1→addInfos();
$subDec2→addInfos();

$decorator→getSubjectInfos();
```

STRATEGY

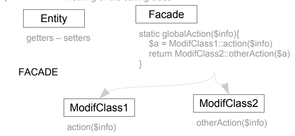



```
$Entity = new Entity();
$strategy = new StrategyContext('B');
$strategy->getInfos($Entity)
```

only the strategy class is called

FACADE

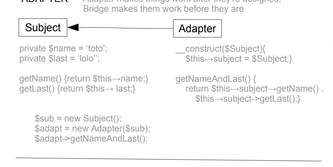
the complex subsystem will know
nothing of the calling class




```
$Entity = new Entity();
$infos = Facade::globalAction($Entity->getInfo())
```

ADAPTER

creates a class from another one with
the exact method you need.
Make class work with others without changing their code
Adapter makes things work after they're designed;
Bridge makes them work before they are




```
$sub = new Subject();
$adapt = new Adapter($sub);
$adapt->getNameAndLast();
```

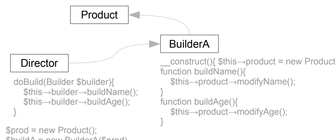
FACTORY METHOD

a class creates the object you want to use


```
$John = Factory::create('John', 'Doe');
echo $John→getNameAndAge();
```

BUILDER

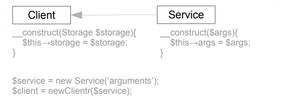
Uses FACTORY METHOD behind the scene
Separates the construction




```
$prod = new Product();
$buildA = new BuilderA($prod)
$dir = new Director()
$dir→doBuild($buildA)
```

SIMPLE DEPENDENCY INJECTION

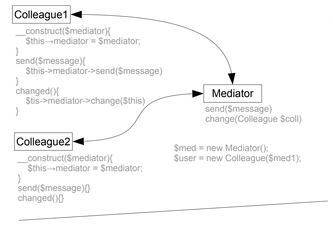
A dependency is an object that can be used (a service)
An injection is the passing of a dependency to a dependent
object (a client) that would use it




```
$service = new Service('arguments');
$client = newClientr($service);
```

MEDIATOR

A Mediator is responsible for controlling and coordinating the
interactions of a group of objects.
The Mediator serves as an intermediary that keeps objects
from referring to each other explicitly
Colleagues should not know the next Collegue
Each Collegue class knows its mediator object

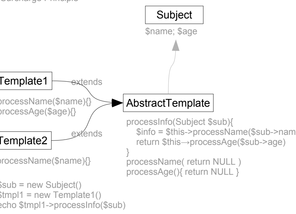



```
$med = new Mediator();
$user = new Colleague($med1);
```

DEPENDENCY INJECTION +
LAZY INSTANTIATION = service container
service container-->2 variables référencent meme instance du service
BUILDER
ABSTRACT FACTORY
COMPOSITE
MEDIATOR
TEMPLATE METHOD
Making every filter referencing the next one would build a Chain of Responsibili

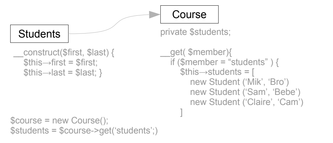
TEMPLATE METHOD

Surcharge Principle




```
$sub = new Subject()
$tmpl1 = new Template1()
echo $tmpl1->processInfo($sub)
```

LAZY INSTATIATION




```
$course = new Course();
$students = $course->get('students';)

// __get est utilisé pour lire des données
inaccessibles
```