

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 25: Patrones de Comportamiento (parte 1)

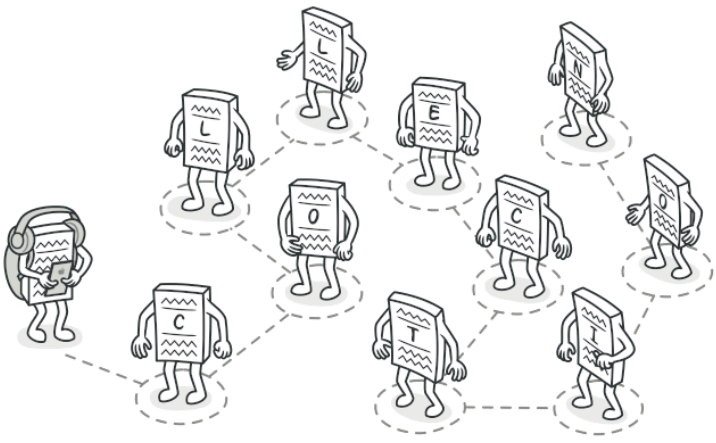
Francisco Gazitúa Requena (fco_gazitua_requena@uc.cl)

IIC2113 Diseño Detallado de Software

4 de Noviembre, 2025

Iterator

Iterator



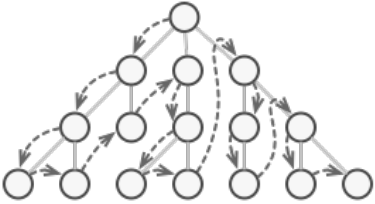
Iterator

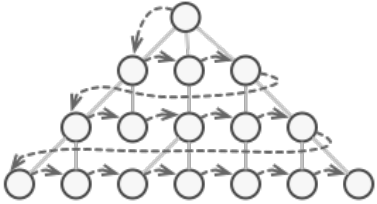




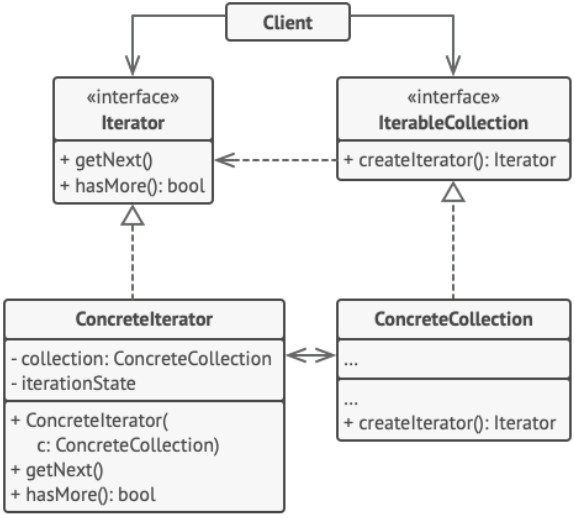


Iterator

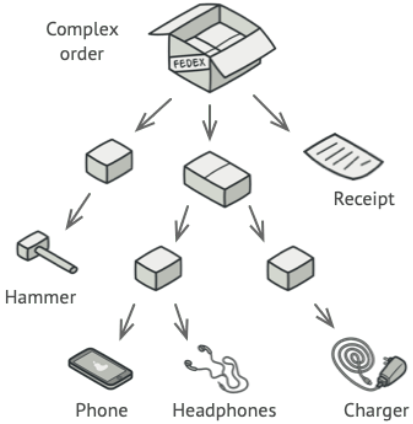




Iterator


Client
«interface»
Iterator
+ getNext()
+ hasMore(): bool
«interface»
IterableCollection
+ createIterator(): Iterator
ConcreteIterator
- collection: ConcreteCollection
- iterationState
+ ConcreteIterator(
c: ConcreteCollection)
+ getNext()
+ hasMore(): bool
ConcreteCollection
...
...
+ createIterator(): Iterator

Iterator


Complex order
FEDEX
Receipt
Hammer
Phone
Headphones
Charger

Iterator

Primero creamos la interfaz del iterador:

```
2 public interface Iterator {
3     Component GetNext();
4     bool hasMore();
5 }
```

Iterator

Luego definimos una primera forma de iterar el árbol:

```
7  public class DFSIterator:Iterator {
8      private Stack<Component> stack = new Stack<Component>();
9
10     public DFSIterator(Component root)
11         => stack.Push(root);
12
13     public bool hasMore()
14         => stack.Any();
15
16     public Component GetNext() {
17         Component node = stack.Pop();
18         foreach (Component child in node.GetChildren().Reverse())
19             stack.Push(child);
20         return node;
21     }
22 }
```

Iterator

... y una segunda forma de iterar el árbol:

```
24 public class BFSIterator:Iterator {
25     private Queue<Component> queue = new Queue<Component>();
26
27     public BFSIterator(Component root)
28         => queue.Enqueue(root);
29
30     public bool hasMore()
31         => queue.Any();
32
33     public Component GetNext() {
34         Component node = queue.Dequeue();
35         foreach (Component child in node.GetChildren())
36             queue.Enqueue(child);
37         return node;
38     }
39 }
```

Iterator

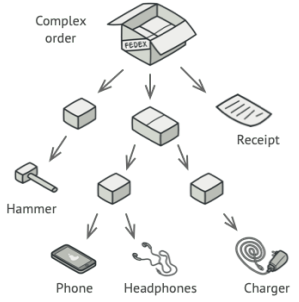
Como resultado, ahora este método puede mostrar el detalle de distintas formas:

```
41 public void ShowDetails(Iterator iterator) {
42     while (iterator.hasMore())
43         Console.WriteLine(iterator.GetNext());
44 }
```

Iterator

Como resultado, ahora este método puede mostrar el detalle de distintas formas:

```
41 public void ShowDetails(Iterator iterator) {
42     while (iterator.hasMore())
43         Console.WriteLine(iterator.GetNext());
44 }
```



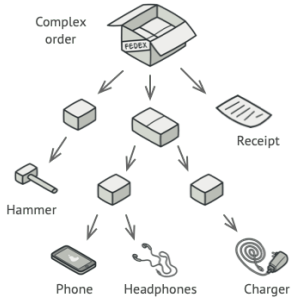

```
46 ShowDetails(new DFSIterator(completeOrder));
```

```
2  I am a box. My price is $883USD.
3  I am a box. My price is $9USD.
4  I am a hammer. My price is $9USD.
5  I am a box. My price is $873USD.
6  I am a box. My price is $859USD.
7  I am a phone. My price is $829USD.
8  I am a headphones. My price is $30USD.
9  I am a box. My price is $14USD.
10 I am a charger. My price is $14USD.
11 I am a receipt. My price is $1USD.
```

Iterator

Como resultado, ahora este método puede mostrar el detalle de distintas formas:

```
41 public void ShowDetails(Iterator iterator) {
42     while (iterator.hasMore())
43         Console.WriteLine(iterator.GetNext());
44 }
```

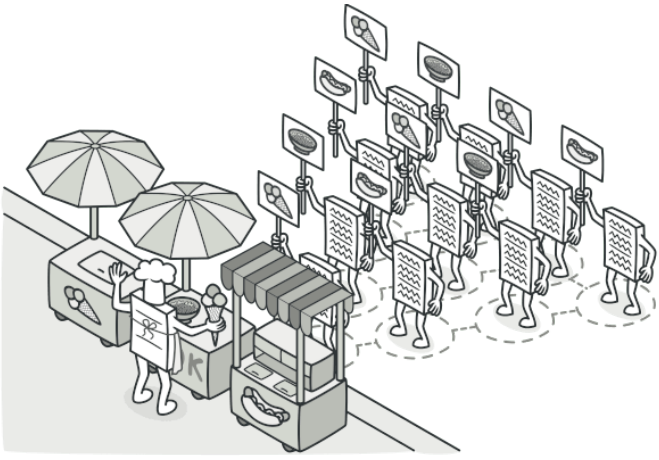



```
47 ShowDetails(new BFSIterator(completeOrder));
```

```
13 I am a box. My price is $883USD.
14 I am a box. My price is $9USD.
15 I am a box. My price is $873USD.
16 I am a receipt. My price is $1USD.
17 I am a hammer. My price is $9USD.
18 I am a box. My price is $859USD.
19 I am a box. My price is $14USD.
20 I am a phone. My price is $829USD.
21 I am a headphones. My price is $30USD.
22 I am a charger. My price is $14USD.
```

Visitor

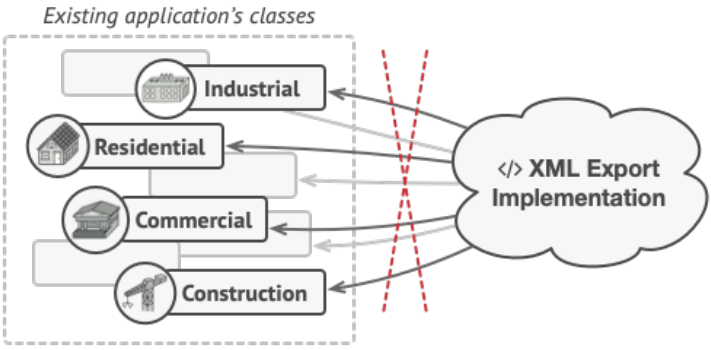
Visitor



Visitor



Visitor


Existing application's classes
Industrial
Residential
Commercial
Construction
</> XML Export
Implementation

Visitor

```
2 public class ExportVisitor {
3     public void ExportCity(City c) { /* ... */ }
4     public void ExportIndustry(Industry f) { /* ... */ }
5     public void ExportSightSeeing(SightSeeing ss) { /* ... */ }
6 }
```

Visitor

```
2  public class ExportVisitor {
3      public void ExportCity(City c) { /* ... */ }
4      public void ExportIndustry(Industry f) { /* ... */ }
5      public void ExportSightSeeing(SightSeeing ss) { /* ... */ }
6  }
```

```
8  foreach (var node in graph) {
9      if(node is City)
10         exportVisitor.ExportCity((City)node);
11     if(node is Industry)
12         exportVisitor.ExportIndustry((Industry)node);
13     if(node is SightSeeing)
14         exportVisitor.ExportSightSeeing((SightSeeing)node);
15     /* ... */
16 }
```

Visitor

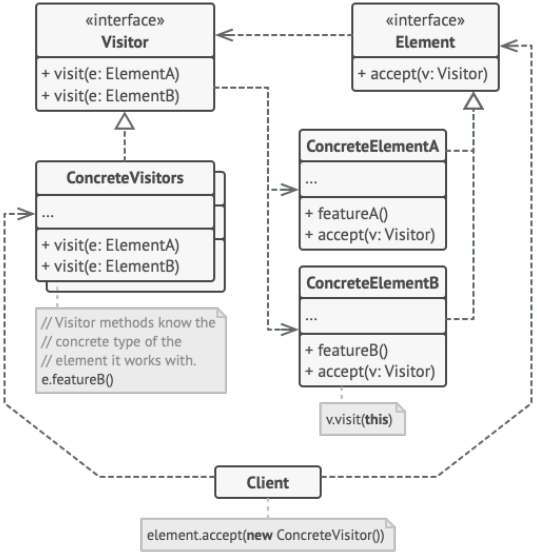
```
18 public class SightSeeing: Node {
19     public override void AcceptVisitor(ExportVisitor visitor)
20         => visitor.ExportSightSeeing(this);
21     /* ... */
22 }
23 public class Industry: Node {
24     public override void AcceptVisitor(ExportVisitor visitor)
25         => visitor.ExportIndustry(this);
26     /* ... */
27 }
28 public class City: Node {
29     public override void AcceptVisitor(ExportVisitor visitor)
30         => visitor.ExportCity(this);
31     /* ... */
32 }
```

Visitor

```
18 public class SightSeeing: Node {
19     public override void AcceptVisitor(ExportVisitor visitor)
20         => visitor.ExportSightSeeing(this);
21     /* ... */
22 }
23 public class Industry: Node {
24     public override void AcceptVisitor(ExportVisitor visitor)
25         => visitor.ExportIndustry(this);
26     /* ... */
27 }
28 public class City: Node {
29     public override void AcceptVisitor(ExportVisitor visitor)
30         => visitor.ExportCity(this);
31     /* ... */
32 }
```

```
34 foreach (var node in graph)
35     node.AcceptVisitor(exportVisitor);
```

Visitor

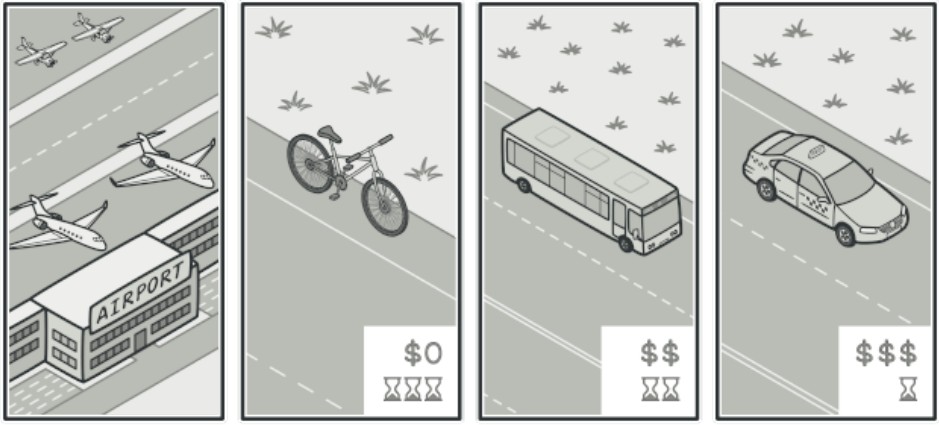

«interface»
Visitor
+ visit(e: ElementA)
+ visit(e: ElementB)
«interface»
Element
+ accept(v: Visitor)
ConcreteVisitors
...
+ visit(e: ElementA)
+ visit(e: ElementB)
ConcreteElementA
...
+ featureA()
+ accept(v: Visitor)
ConcreteElementB
...
+ featureB()
+ accept(v: Visitor)
// Visitor methods know the
// concrete type of the
// element it works with.
e.featureB()
v.visit(this)
Client
element.accept(new ConcreteVisitor())

Strategy

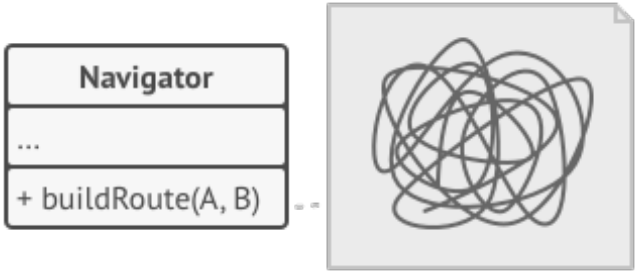
Strategy



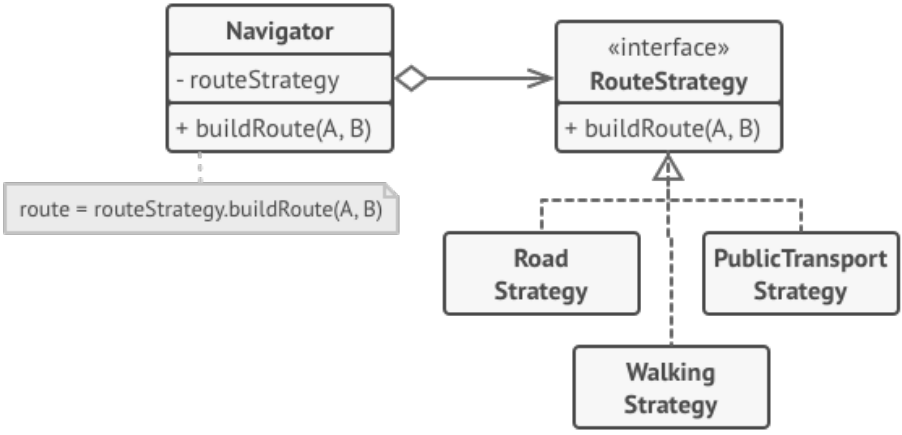
Strategy



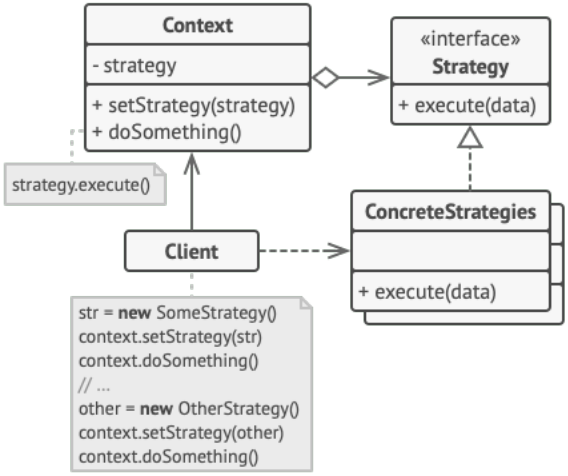
Strategy



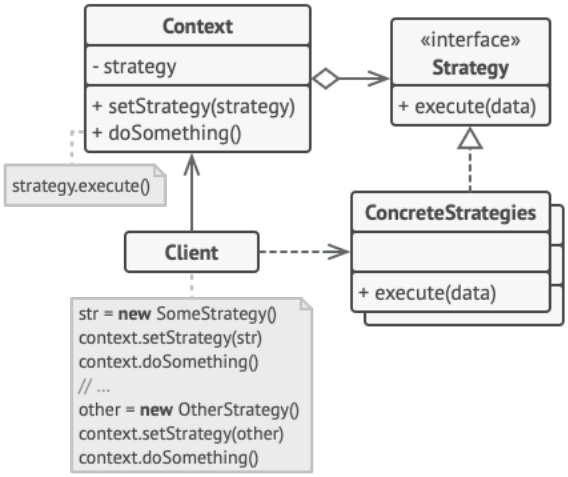
Strategy


Navigator
- routeStrategy
+ buildRoute(A, B)
route = routeStrategy.buildRoute(A, B)
«interface»
RouteStrategy
+ buildRoute(A, B)
Road Strategy
PublicTransport Strategy
Walking Strategy

Strategy

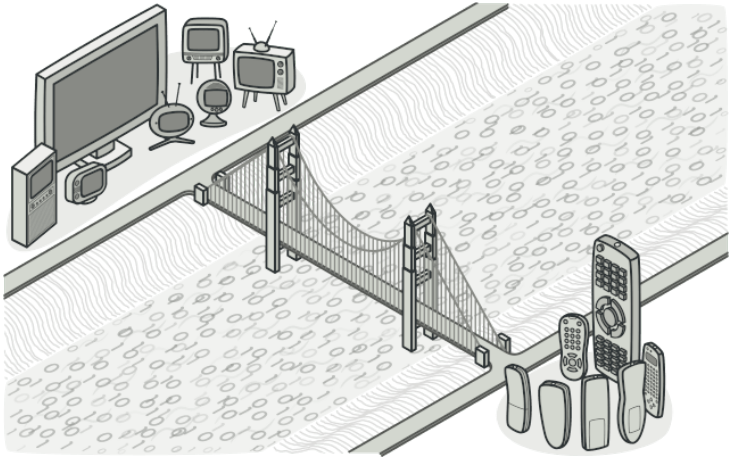

Context
- strategy
+ setStrategy(strategy)
+ doSomething()
«interface»
Strategy
+ execute(data)
strategy.execute()
ConcreteStrategies
+ execute(data)
Client
str = new SomeStrategy()
context.setStrategy(str)
context.doSomething()
// ...
other = new OtherStrategy()
context.setStrategy(other)
context.doSomething()

Strategy

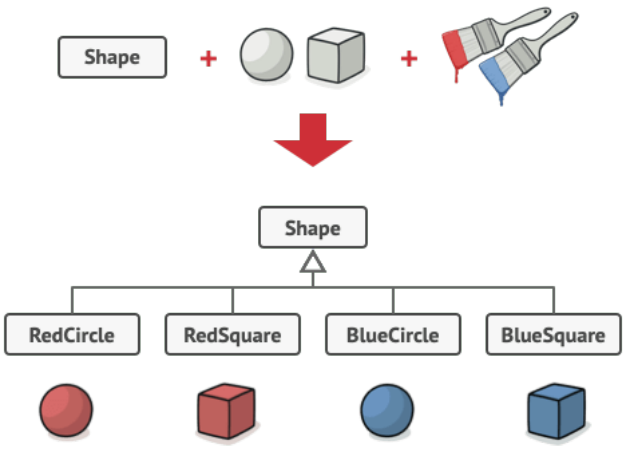


¿Les recuerda algún otro patrón? 🤔

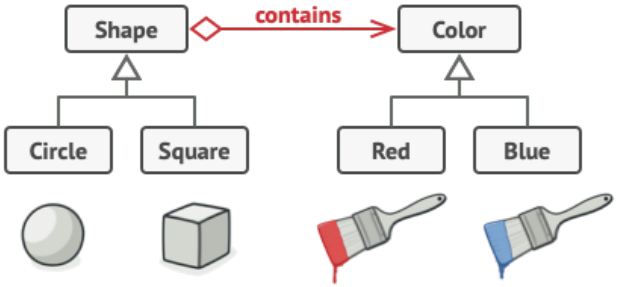
Strategy



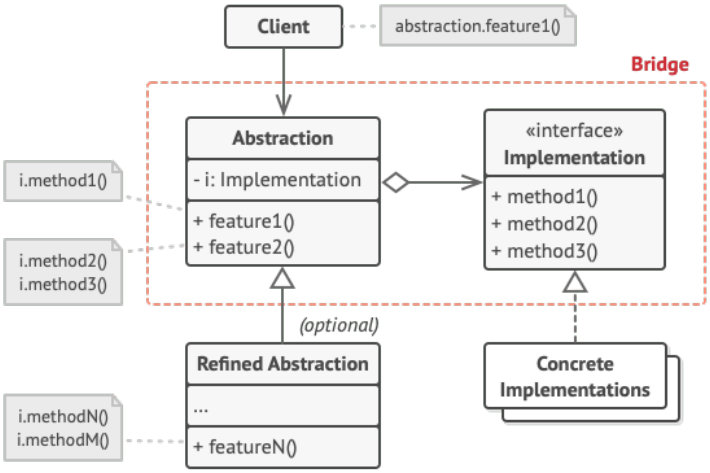
Strategy



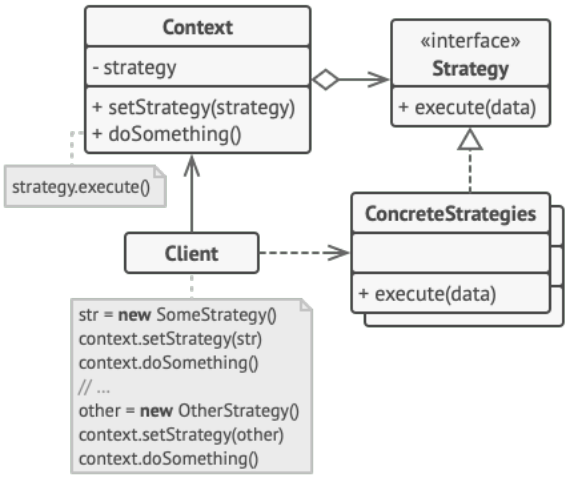
Strategy



Strategy

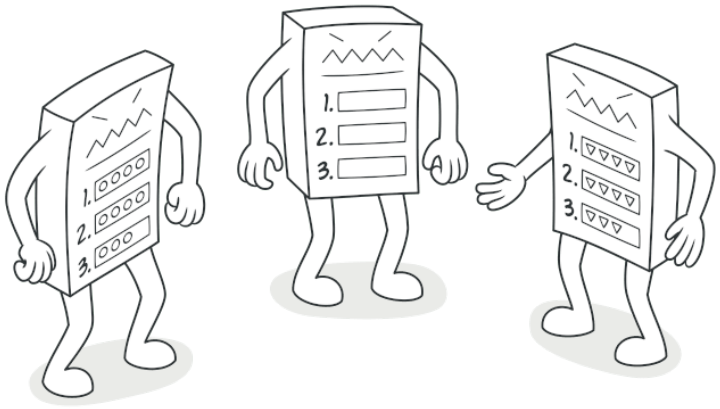

Client
abstraction.feature1()
Bridge
Abstraction
- i: Implementation
+ feature1()
+ feature2()
«interface»
Implementation
+ method1()
+ method2()
+ method3()
i.method1()
i.method2()
i.method3()
(optional)
Refined Abstraction
...
+ featureN()
Concrete
Implementations
i.methodN()
i.methodM()

Strategy

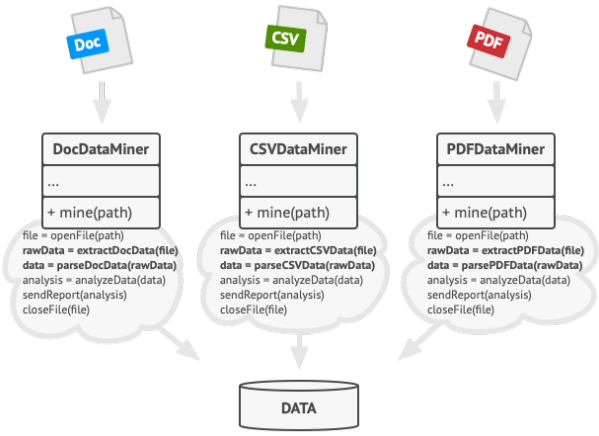

Context
- strategy
+ setStrategy(strategy)
+ doSomething()
«interface»
Strategy
+ execute(data)
strategy.execute()
Client
ConcreteStrategies
+ execute(data)
str = new SomeStrategy()
context.setStrategy(str)
context.doSomething()
// ...
other = new OtherStrategy()
context.setStrategy(other)
context.doSomething()

Template Method

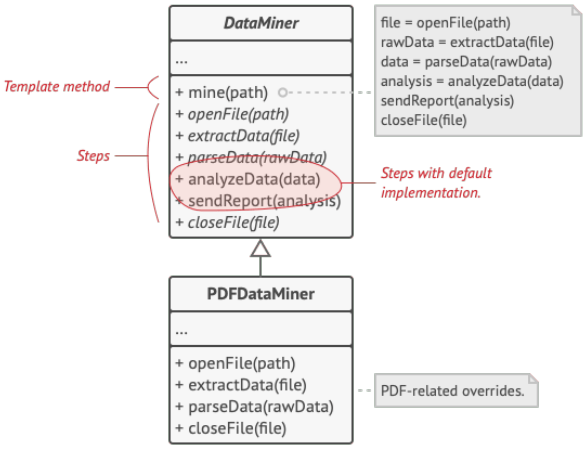
Template Method



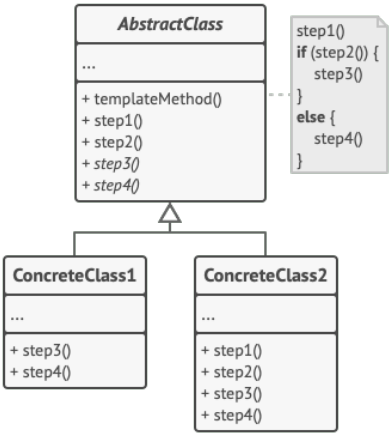
Template Method



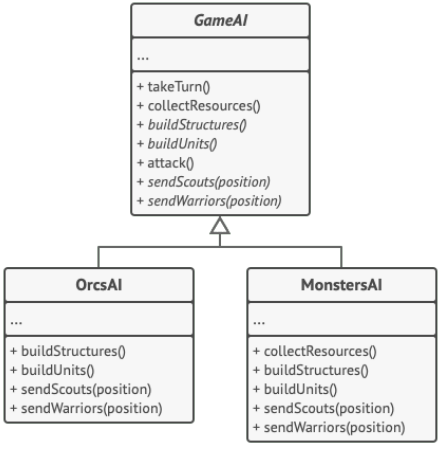
Template Method



Template Method



Template Method



Actividad