Patrones de Diseño

Benjamín Earle

Jerónimo Salazar

Objetivo

- Problema común
- Solución de calidad

Patrones

- Gang of Four
- 11/23 patrones

Pregunta 1 - 20 pts (Patrones de Diseño)

Un software de gestión bastante complejo incluye una componente destinada a generar reportes impresos. El problema es que el software debe manejar distintos formatos de reporte dependiendo si este está destinado a Chile, USA, Brasil, o alguno de otros 10 países donde se utiliza. Cada reporte incluye títulos de nivel 1, 2 y 3, párrafos, tablas y gráficos. Cada tipo de reporte incluye un set particular de dichos elementos.

a) (5 pts) ¿Cual es el patrón de diseño más apropiado para este problema ? Fundamente su respuesta

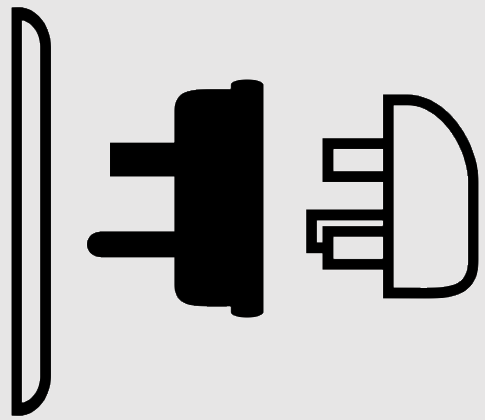
b) (10 pts) Haga un diagrama de clases UML que ilustre su solución para solo 2 tipos de reporte (Chile y USA) y 3 tipos de elementos (párrafos, tablas y gráficos). Escriba el código Ruby de las distintas clases del diagrama (solo lo mínimo necesario)

c) (5 pts) Escriba un trozo de código Ruby que permita demostrar la forma de usar lo anterior para generar los reportes de Chile y USA mencionados

Ventajas

- Soluciones probadas
- Terminología común
- Facilidad de modificar
- “Subirle el pelo” al código

Adaptador

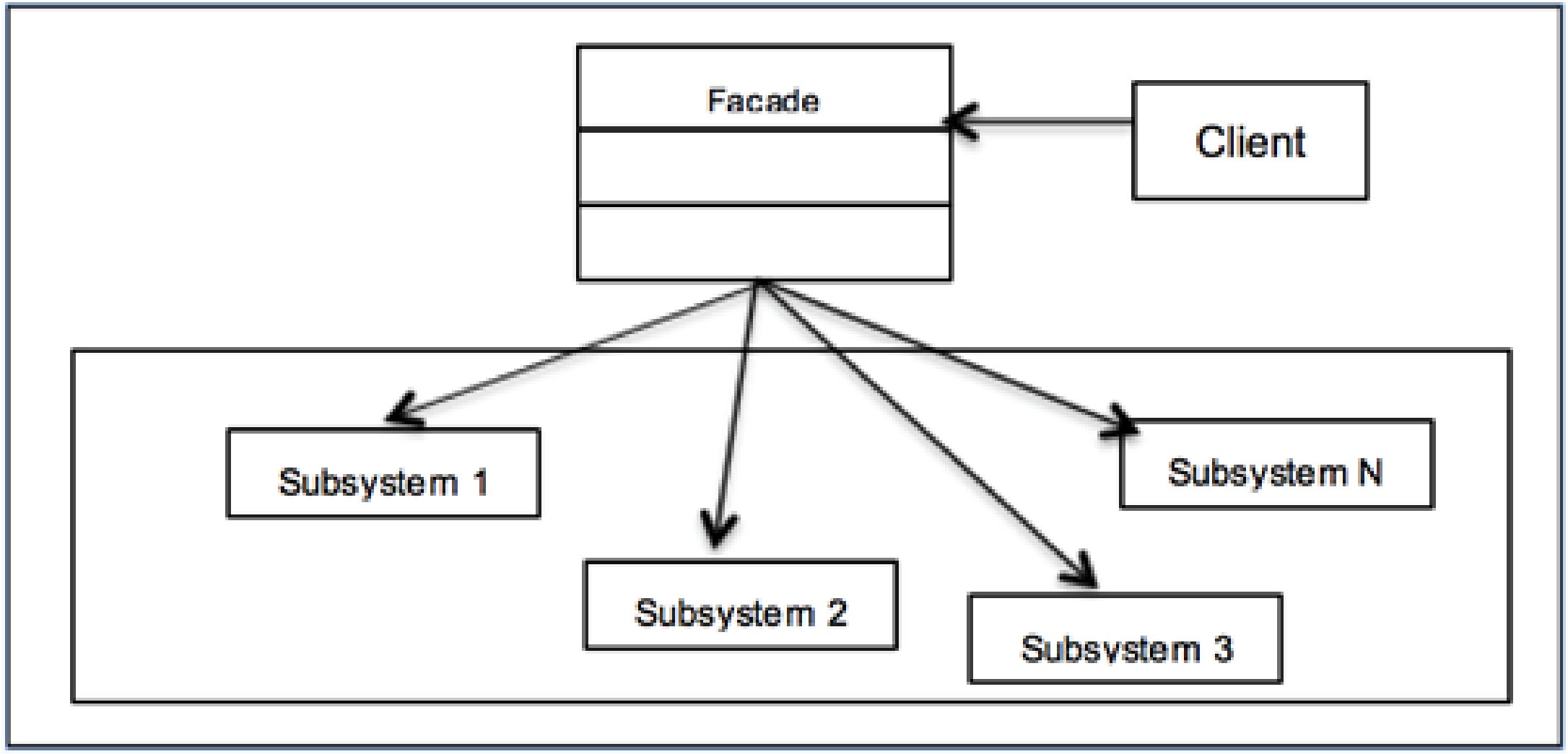


- Funcionamiento correcto, interfaz incorrecta.
- Implementación de un envoltorio.

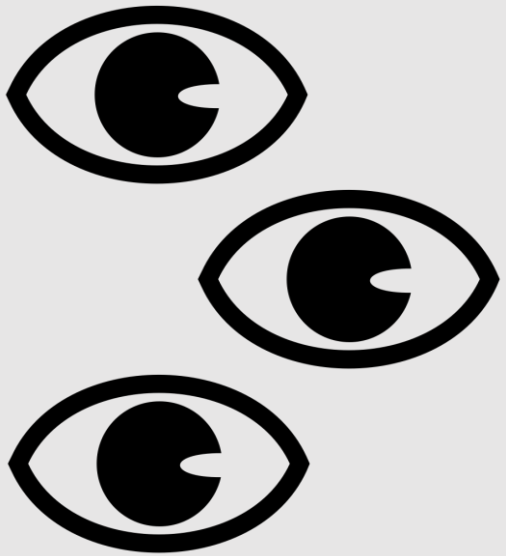
Fachada

- Sistema más complejo de lo necesario.
- Simplificación de uso mediante una nueva interfaz.

Fachada

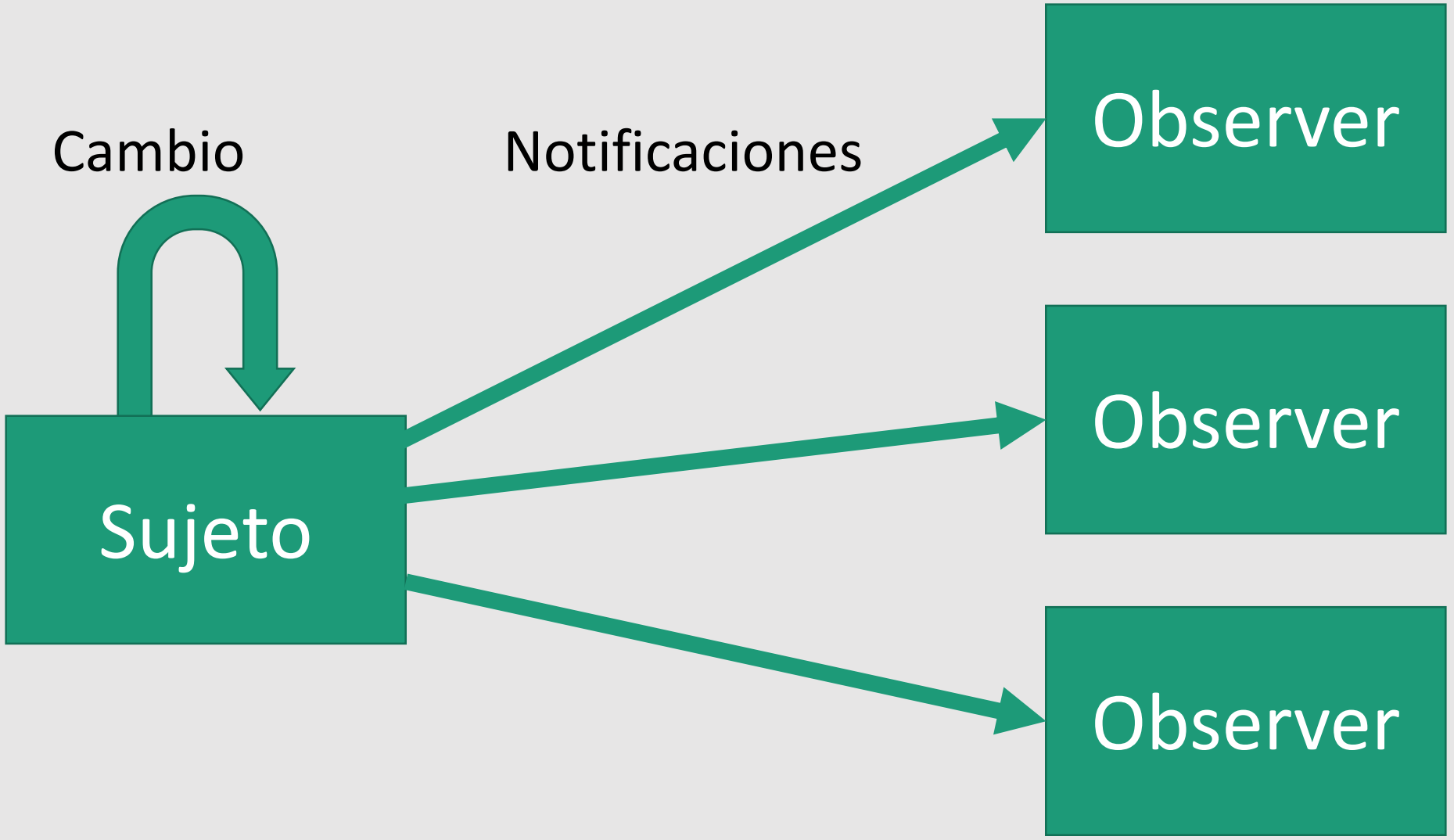

Facade
Client
Subsystem 1
Subsystem 2
Subsystem 3
Subsystem N

Observador



- Necesidad de monitorear un objeto por parte de múltiples otros (1-N).
- Sujeto notifica a observadores.

Observador


Cambio
Notificaciones
Observer
Observer
Observer
Sujeto

Template Method

- Proceso con múltiples pasos y posibles variaciones.
- Uso de subclases para cada variación.

Strategy

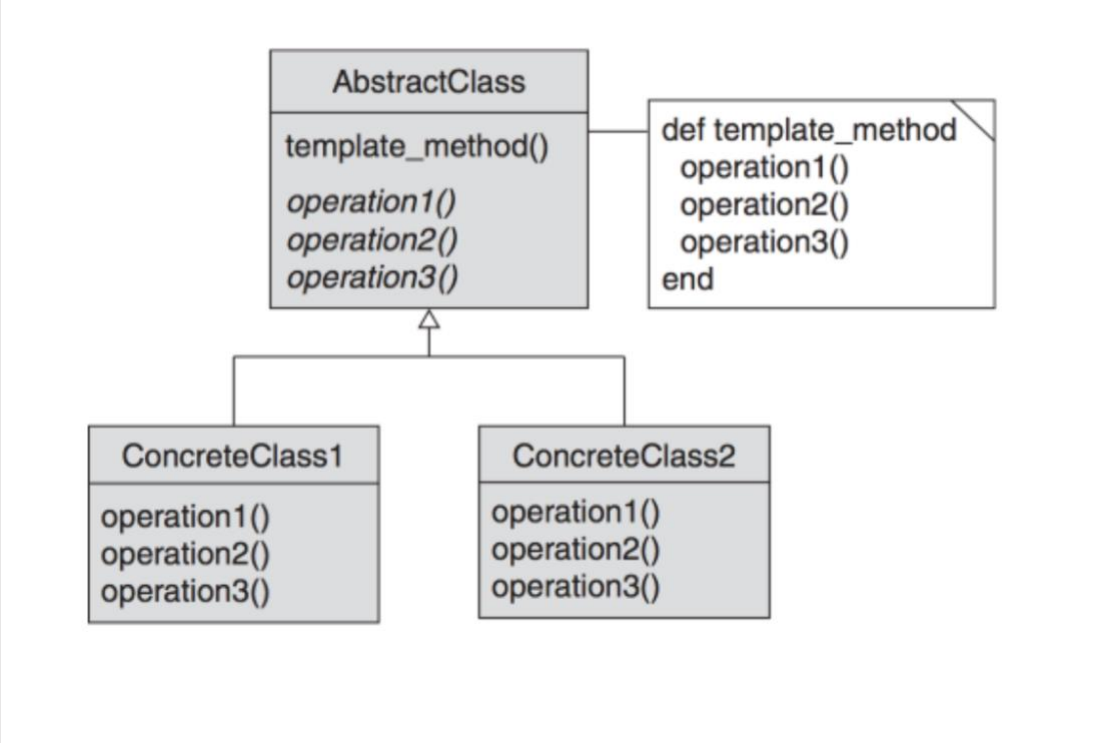
- Proceso con múltiples pasos y posibles variaciones.
- Uso de subclases para cada variación.
- Delegación

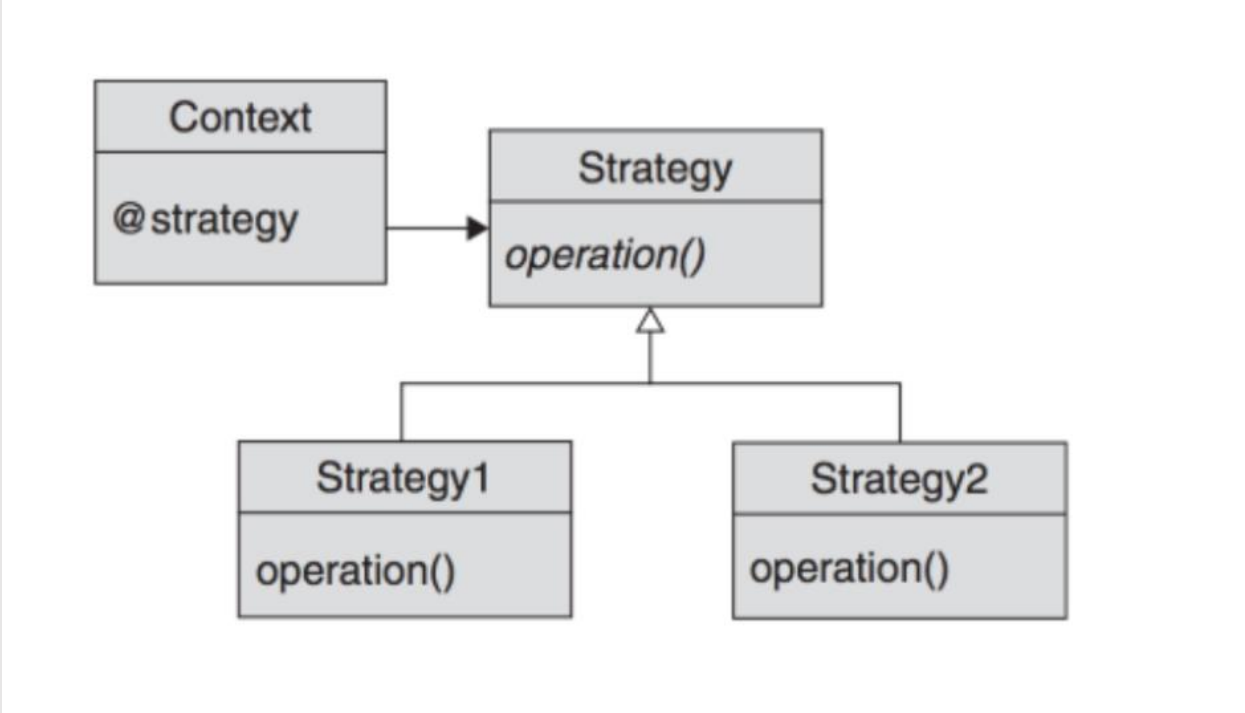
Template vs Strategy

```
report = HTMLReport.new
report.output_report

report = PlainTextReport.new
report.output_report
```

```
report = Report.new(HTMLFormatter.new)
report.output_report

report = Report.new(PlainTextFormatter.new)
report.output_report
```

Template vs Strategy

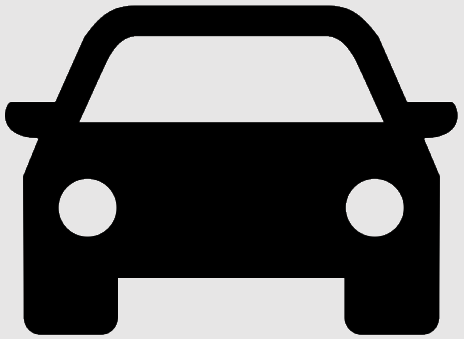




Decorador

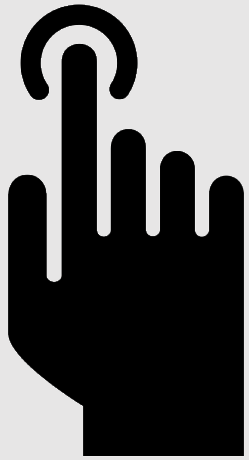
- Anexar funcionalidades.
- Uso de envoltorios para agregarlas.
- NO usa herencia.

Composite



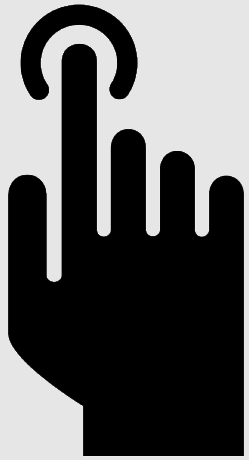
- Componente complejo y con muchos subcomponentes.
- Uso de “nodos y hojas”.
- Da libertad para *undo.*

Comando



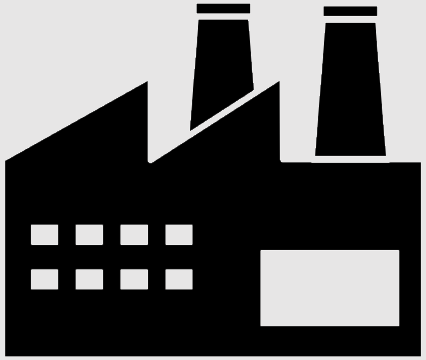
- Representar acciones en objeto.
- Permite agregar extras como *logs*, colas, *undo*.

Comando



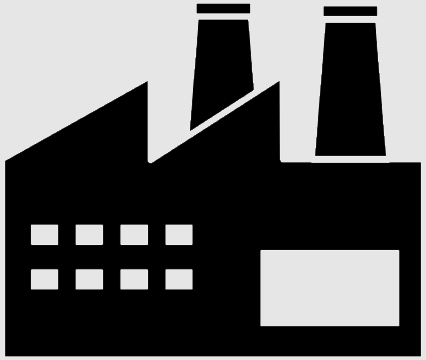
- Representar acciones en objeto.
- Permite agregar extras como *logs*, colas, *undo*.

Factory Method



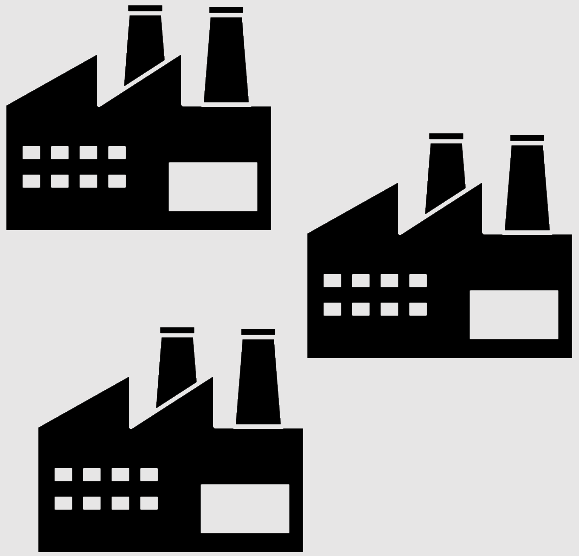
- Template para creación de objetos (variaciones).
- Cada subclase es una fábrica.

Factory Method



- Template para creación de objetos (variaciones).
- Cada subclase es una fábrica.

Abstract Factory



- Strategy para creación de objetos (variaciones).

Factory vs Abstract Factory

```
pond = FrogPond.new(3)
pond.simulate_one_day
```

```
jungle = Habitat.new(1, 4, JungleOrganismFactory.new)
jungle.simulate_one_day
pond = Habitat.new( 2, 4, PondOrganismFactory.new)
pond.simulate_one_day
```

Patrones de Diseño

Benjamín Earle

Jerónimo Salazar