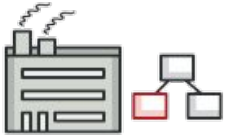
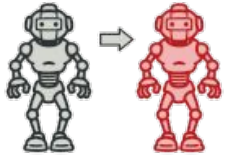
Patrones de Diseño

Capítulo II:
Template Method, Adapter y Factory Method

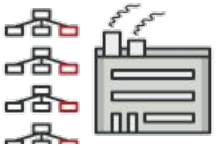
Patrones creacionales



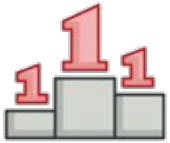
Factory Method
Popularidad: ⭐⭐⭐
Complejidad: 🧠



Prototype
Popularidad: ⭐⭐
Complejidad: 🧠



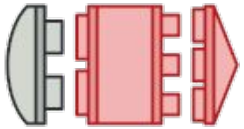
Abstract Factory
Popularidad: ⭐⭐⭐
Complejidad: 🧠🧠



Singleton
Popularidad: ⭐⭐
Complejidad: 🧠

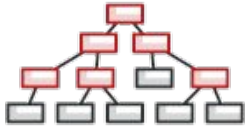
Builder
Popularidad: ⭐⭐⭐
Complejidad: 🧠🧠

Patrones estructurales

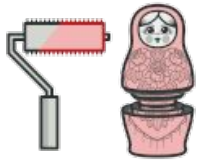


Adapter
Popularidad: ⭐⭐⭐
Complejidad: 🤯

Bridge
Popularidad: ⭐
Complejidad: 🤯🤯🤯

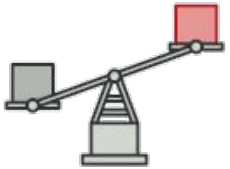


Composite
Popularidad: ⭐⭐⭐
Complejidad: 🤯🤯

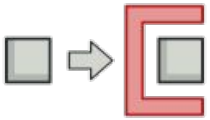


Decorator
Popularidad: ⭐⭐
Complejidad: 🤯🤯

Facade
Popularidad: ⭐⭐
Complejidad: 🤯

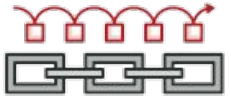


Flyweight
Popularidad:
Complejidad: 🤯🤯🤯



Proxy
Popularidad: ⭐
Complejidad: 🤯🤯

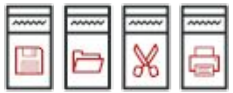
Patrones comportamiento



Chain of Responsibility

Popularidad: ⭐

Complejidad: 🤯 🤯



Command

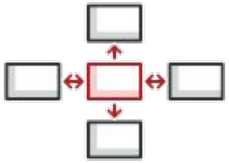
Popularidad: ⭐

Complejidad: 🤯 🤯 🤯

Iterator

Popularidad: ⭐ ⭐ ⭐

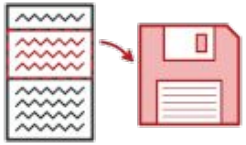
Complejidad: 🤯 🤯



Mediator

Popularidad: ⭐ ⭐

Complejidad: 🤯 🤯



Memento

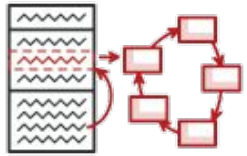
Popularidad: ⭐ ⭐

Complejidad: 🤯

Observer

Popularidad:

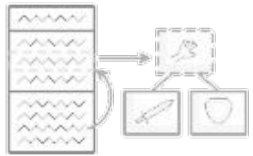
Complejidad: 🤯 🤯 🤯



State

Popularidad: ⭐

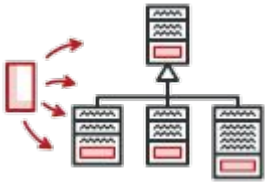
Complejidad: 🤯 🤯



Strategy

Popularidad: ⭐ ⭐

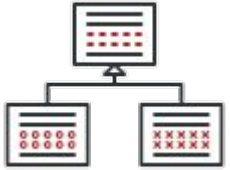
Complejidad: 🤯 🤯



Visitor

Popularidad: ⭐

Complejidad: 🤯 🤯



Template Method

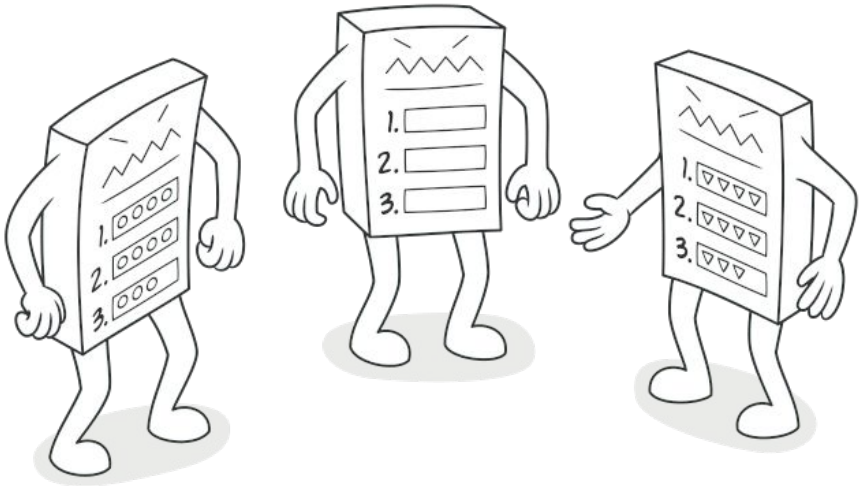
Popularidad: ⭐⭐

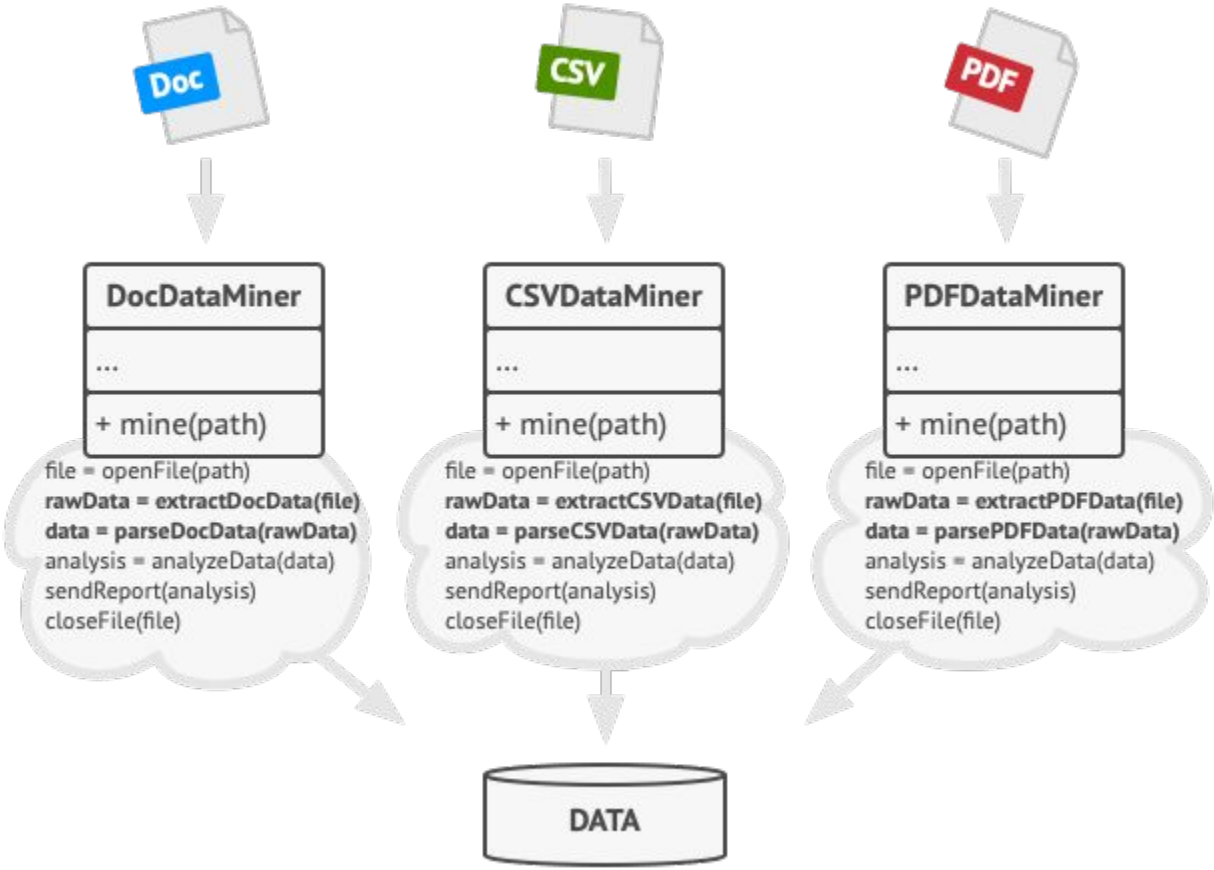
Complejidad: 🤯

Recordatorio de interfaces con Python

Template Method

“Método plantilla”

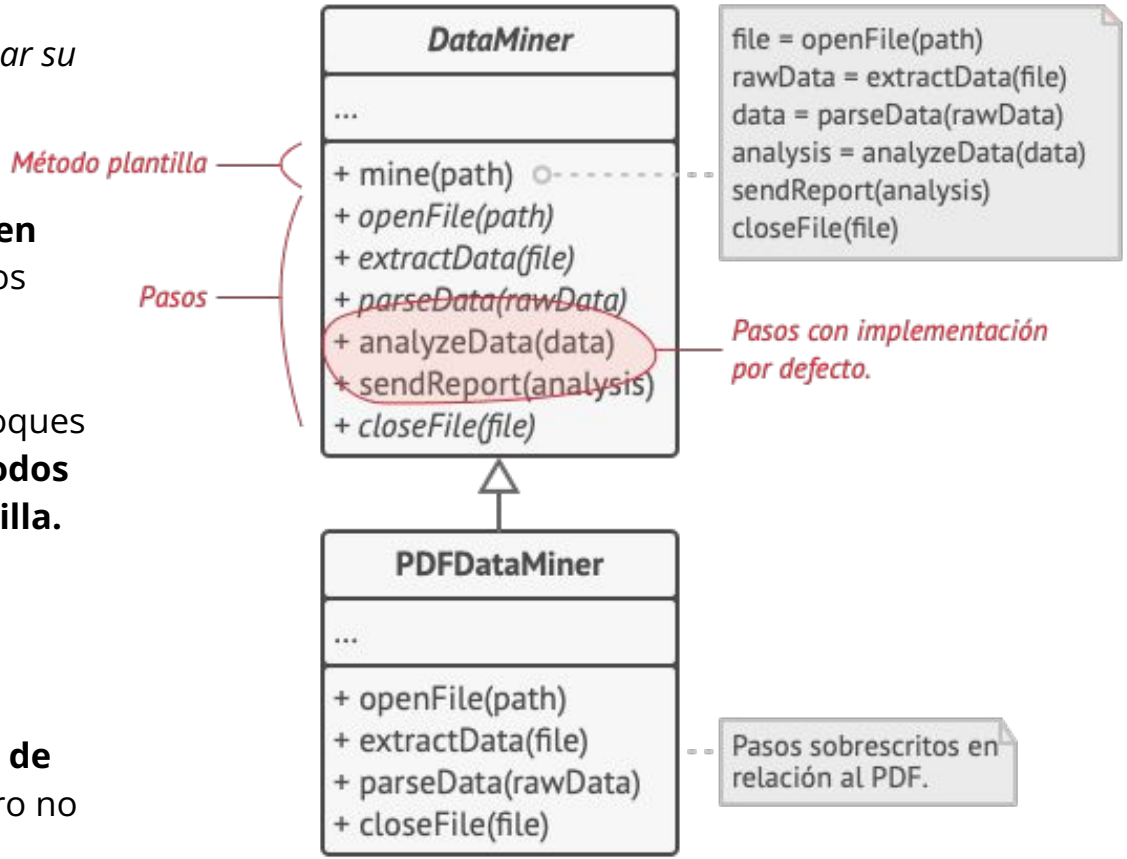



Doc
CSV
PDF
DocDataMiner
...
+ mine(path)
file = openFile(path)
rawData = extractDocData(file)
data = parseDocData(rawData)
analysis = analyzeData(data)
sendReport(analysis)
closeFile(file)
CSVDataMiner
...
+ mine(path)
file = openFile(path)
rawData = extractCSVData(file)
data = parseCSVData(rawData)
analysis = analyzeData(data)
sendReport(analysis)
closeFile(file)
PDFDataMiner
...
+ mine(path)
file = openFile(path)
rawData = extractPDFData(file)
data = parsePDFData(rawData)
analysis = analyzeData(data)
sendReport(analysis)
closeFile(file)
DATA

Define el esqueleto de un algoritmo *en la superclase* ***pero permite a las subclases sobrescribir pasos*** *del algoritmo sin cambiar su estructura.*

- Sugiere que **dividas un algoritmo en una serie de pasos**, conviertas estos pasos en métodos (pueden ser abstractos, o contar con una implementación por defecto) y coloques una serie de **llamadas a esos métodos dentro de un único método plantilla.**

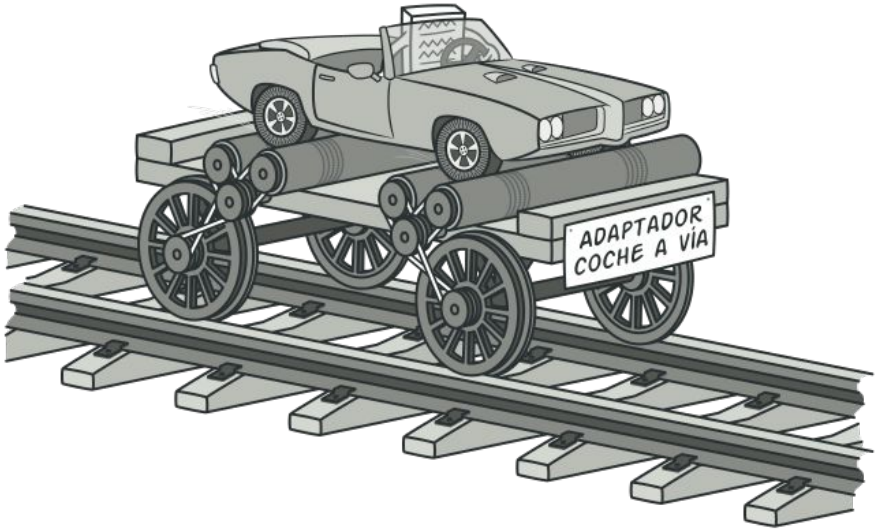
- Para utilizar el algoritmo, **el cliente debe aportar su propia subclase, implementar todos los pasos abstractos y sobrescribir algunos de los opcionales si es necesario** (pero no el propio método plantilla)



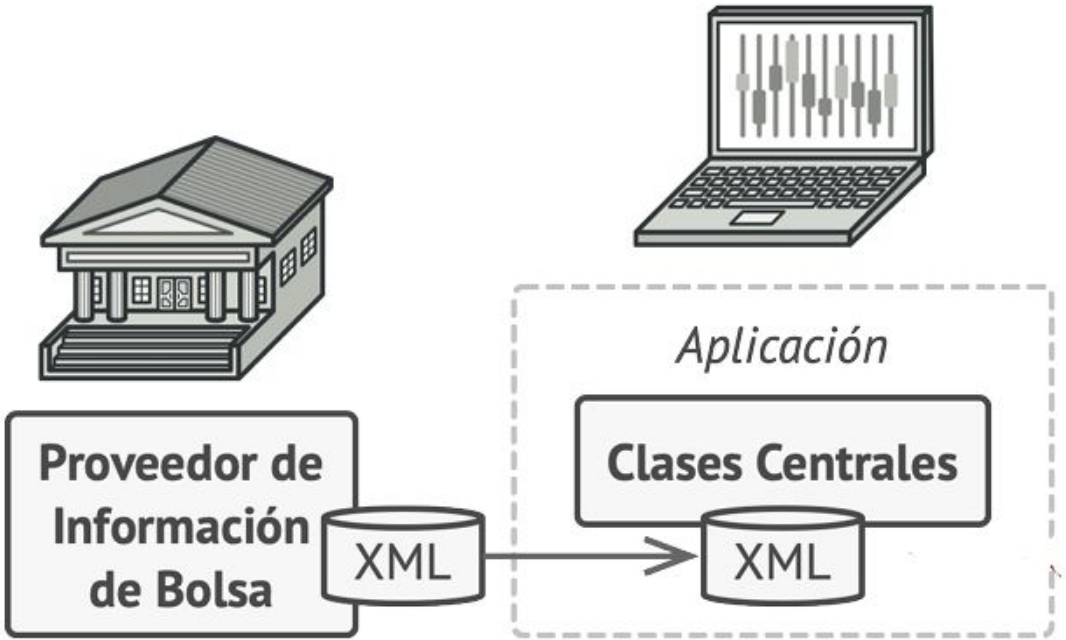
Popularidad: ⭐⭐⭐
Complejidad:

Adapter

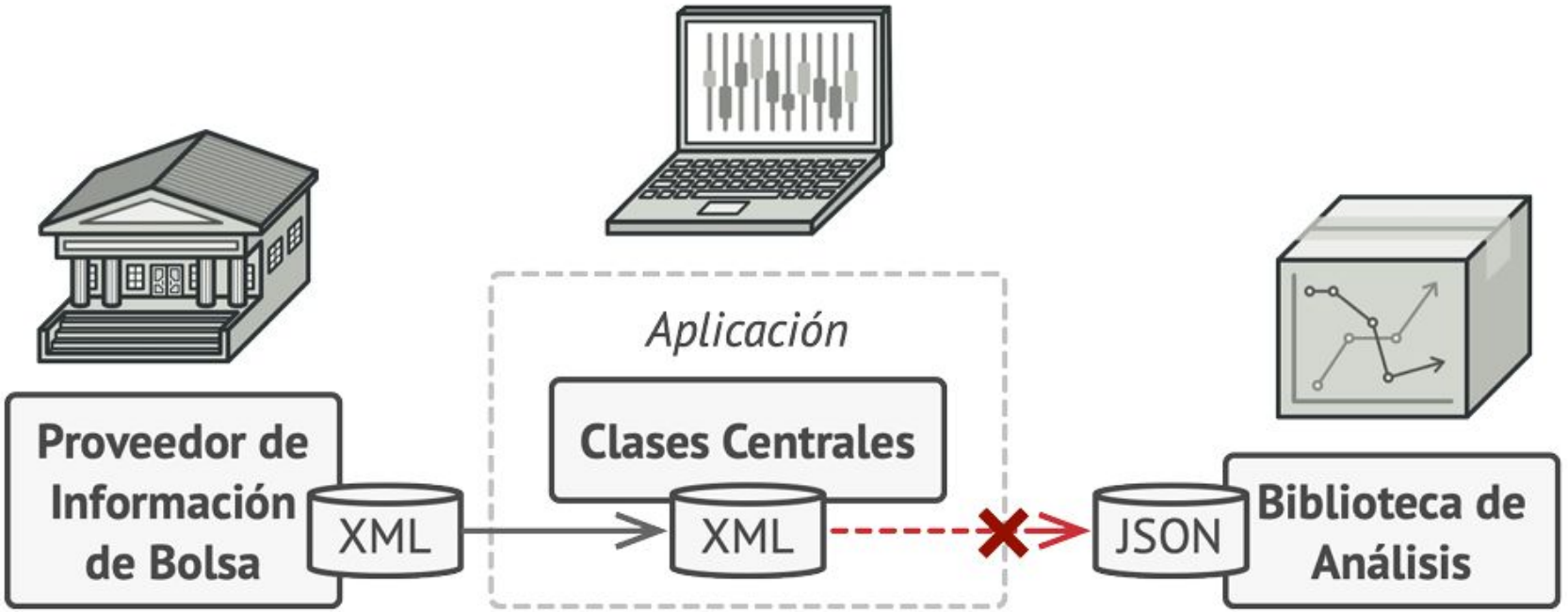
“Adaptador”, “Envoltorio”, “Wrapper”



Escenario actual

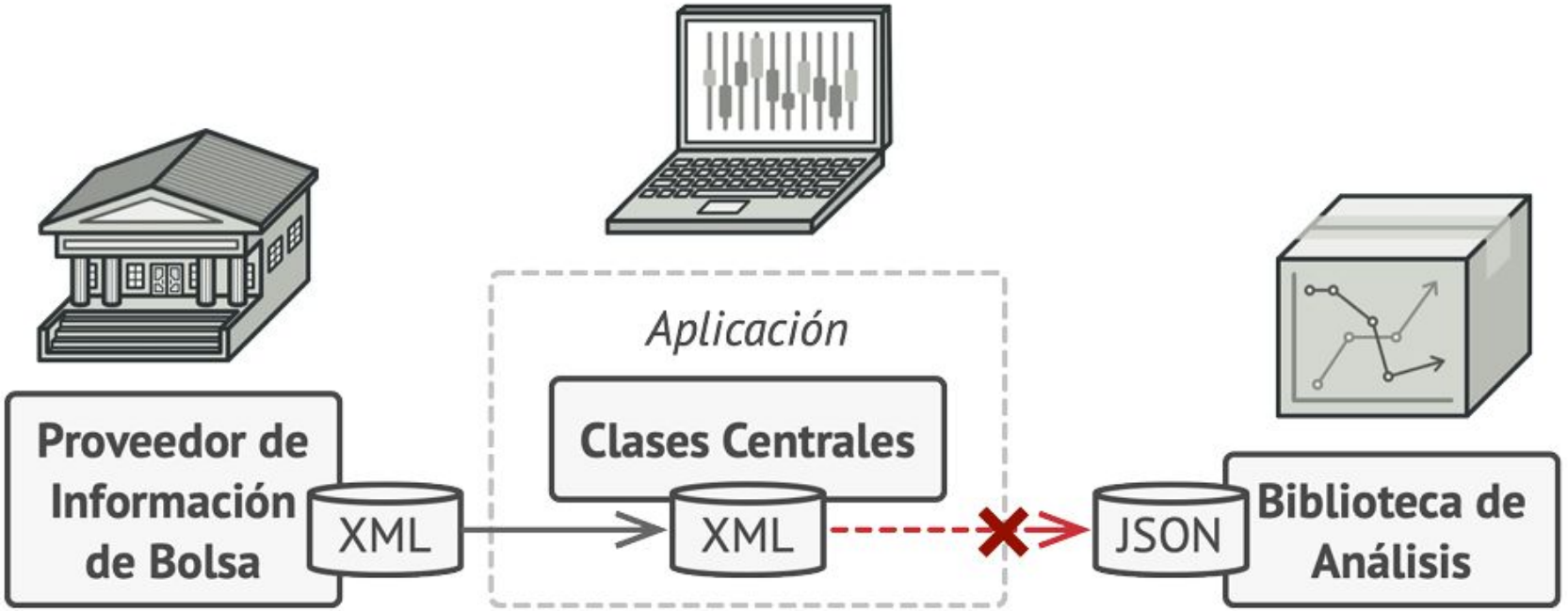


Problema

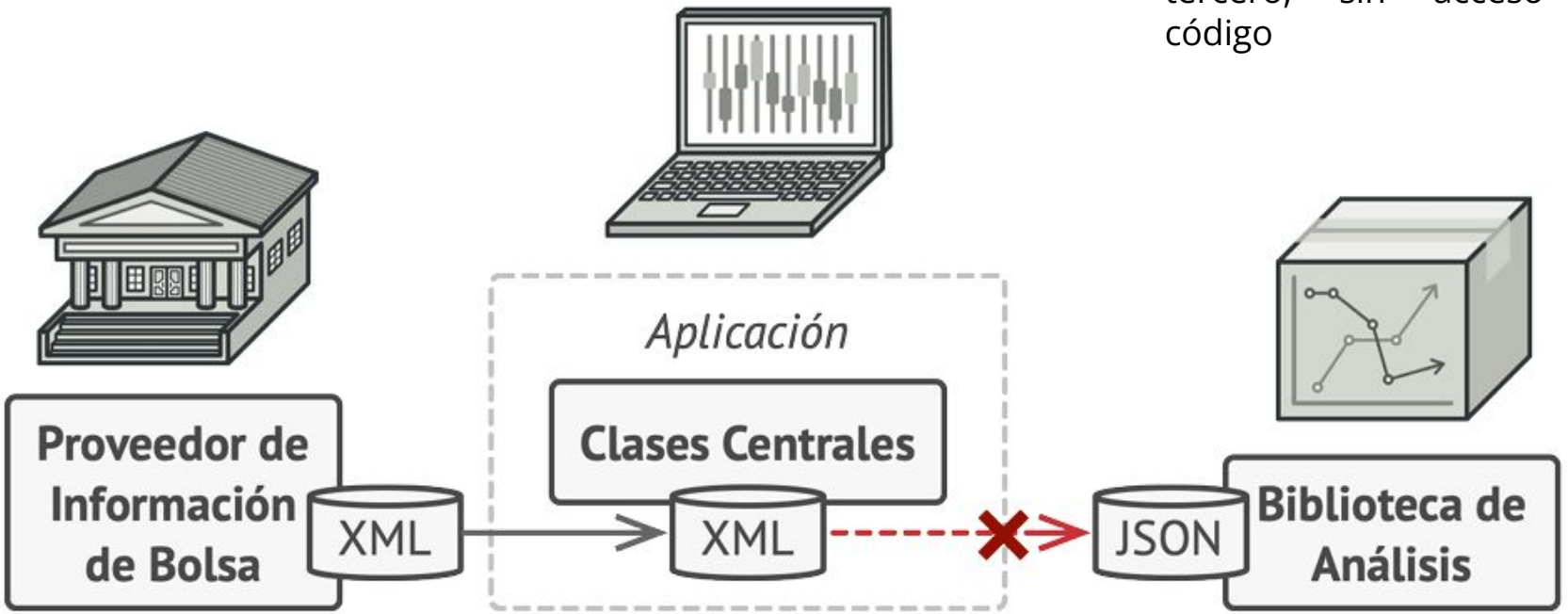

Aplicación
Proveedor de Información de Bolsa
XML
Clases Centrales
XML
JSON
Biblioteca de Análisis

Posible solución

Cambiar métodos para que funcione con XML.

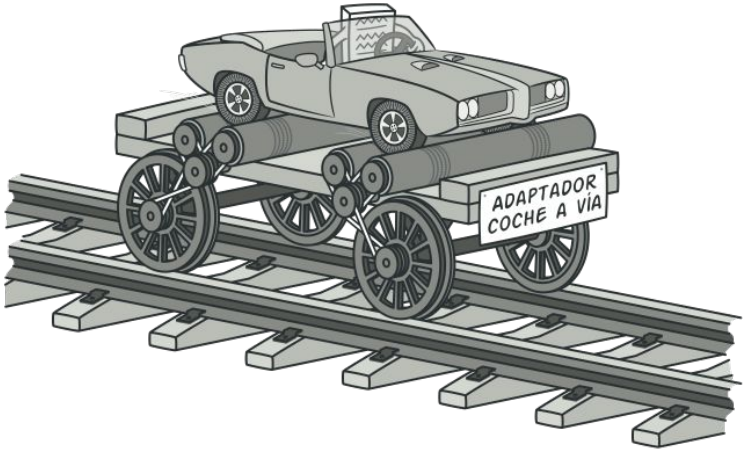


- Descomponer partes con dependencia de esta clase
- En ocasiones es de un tercero, sin acceso al código



Solución: Crear un adaptador

- Colaboración entre objetos existentes con interfaces incompatibles
  - Envuelve uno de los objetos incompatibles para que otro pueda “comprenderla”,



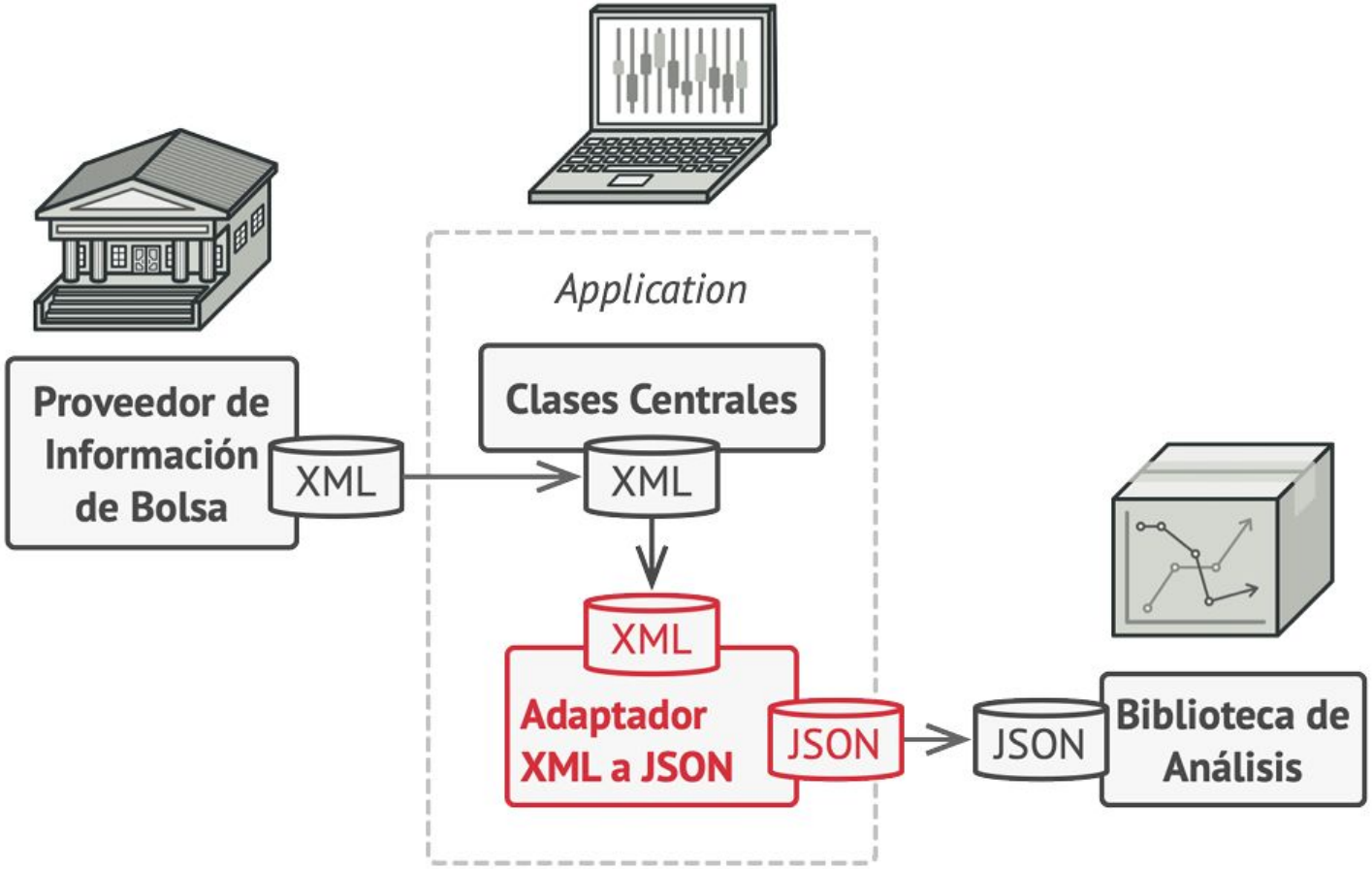

Application
Proveedor de Información de Bolsa
XML
Clases Centrales
XML
XML
Adaptador XML a JSON
JSON
JSON
Biblioteca de Análisis

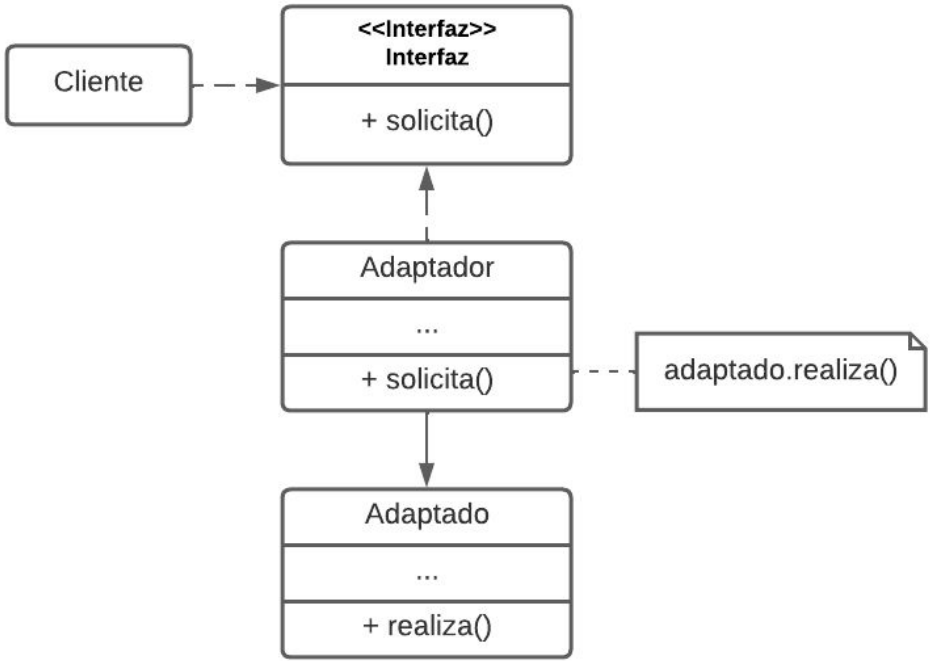
Diagrama UML: Adaptador

Interfaz – Incluye la firma de los métodos del objeto

Cliente – Interactúa con los objetos respondiendo a la interfaz. *(Programa a un interfaz, no a una implementación*).

Adaptador – Implementa los métodos de la interfaz y dentro llama a los métodos del objeto adaptado.

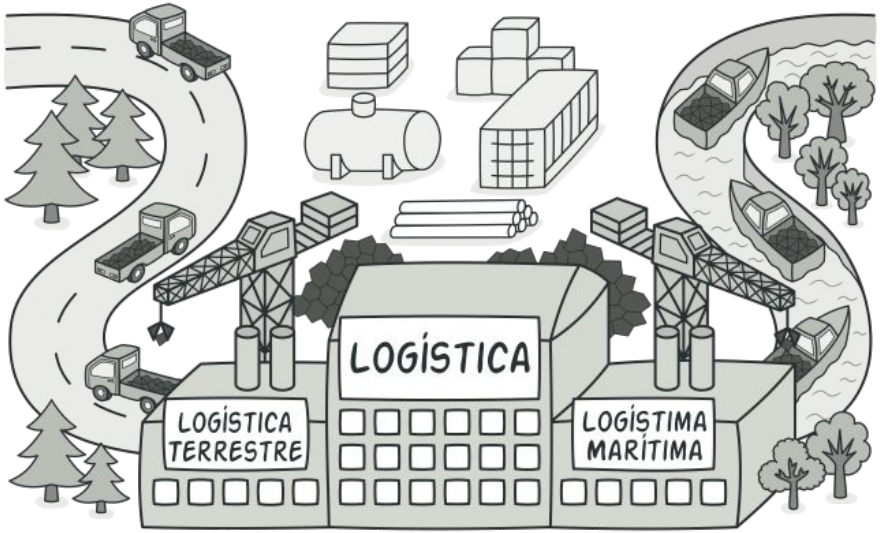
Adaptado – Objeto cuya interfaz ha sido adaptada para corresponder a la interfaz.



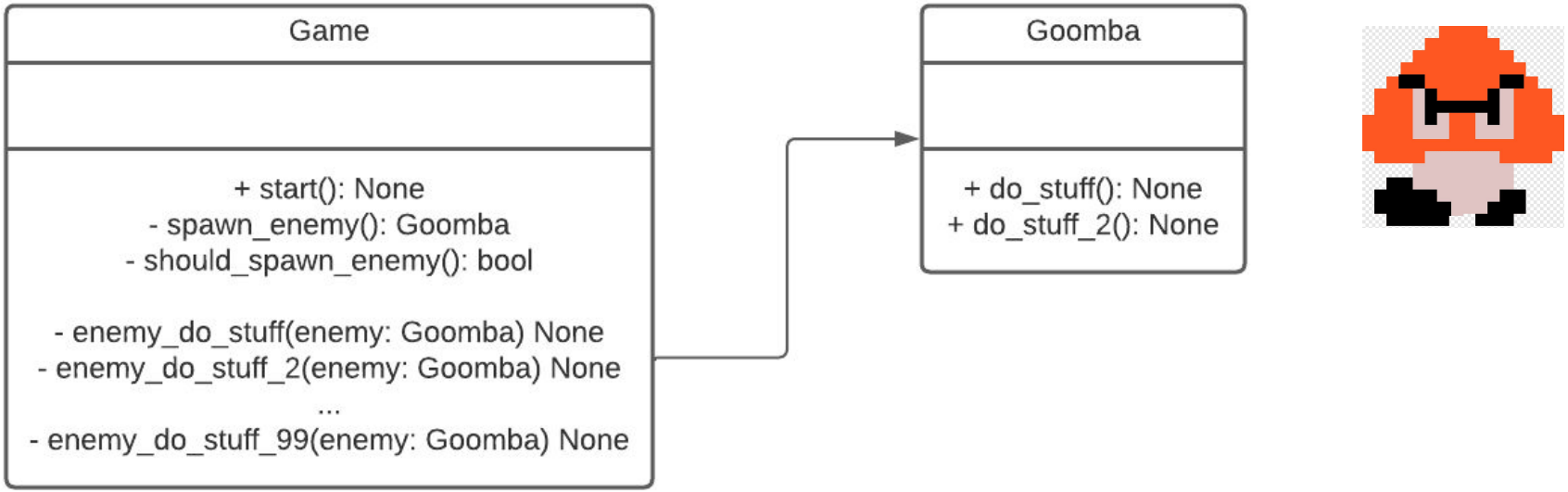
Popularidad: ⭐⭐⭐
Complejidad:

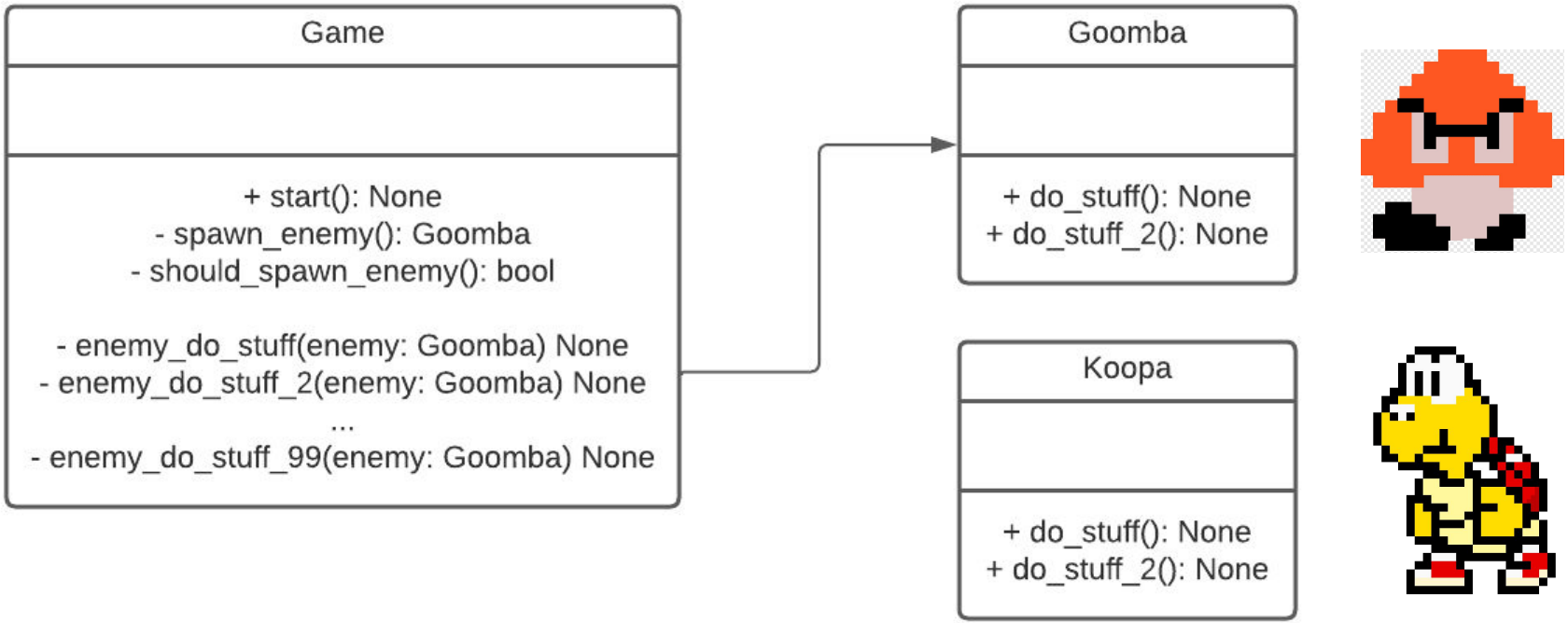
Factory Method

“ Método fábrica”, “Constructor virtual”

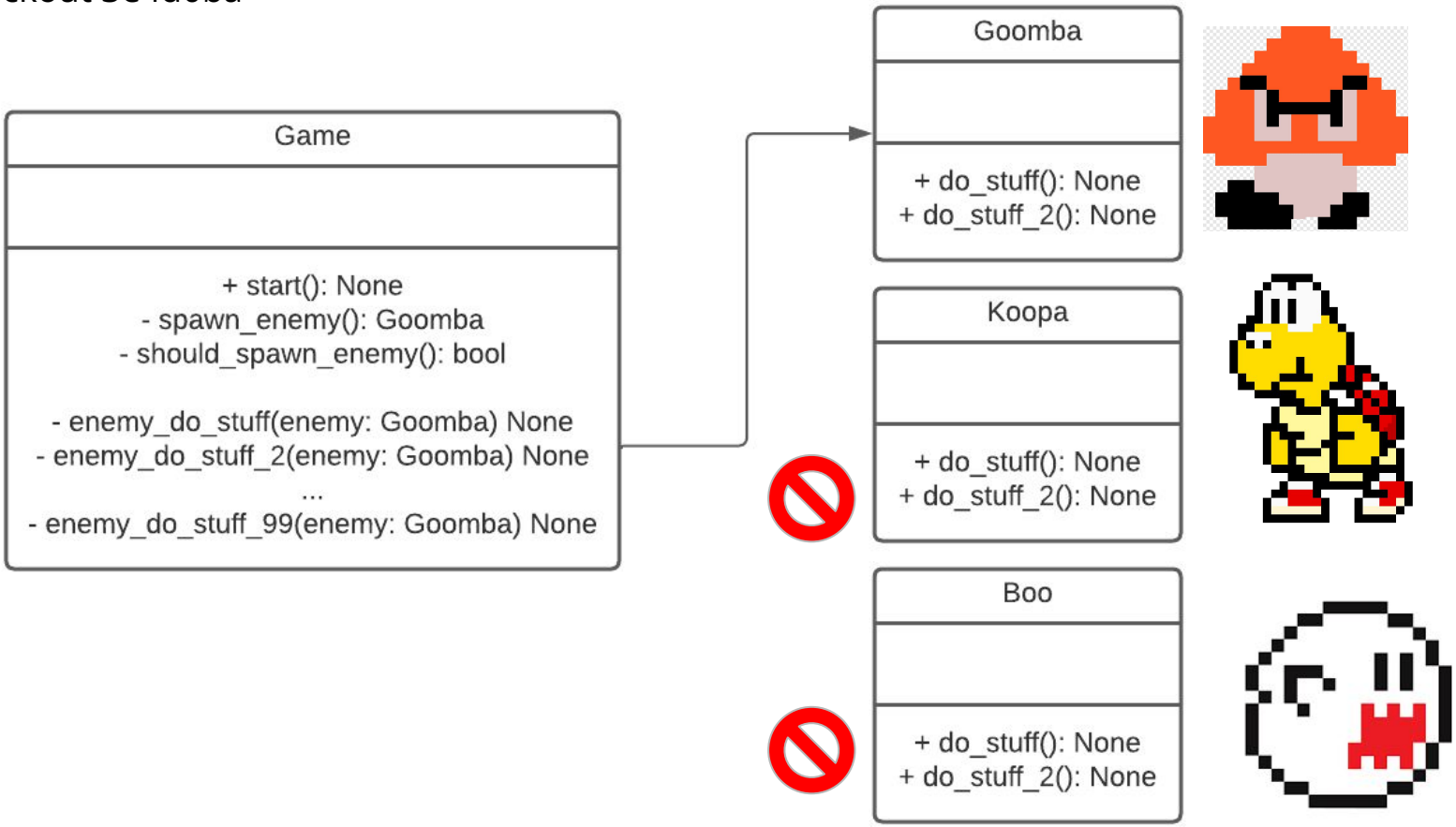


Ver commit:
git checkout 8be34c4

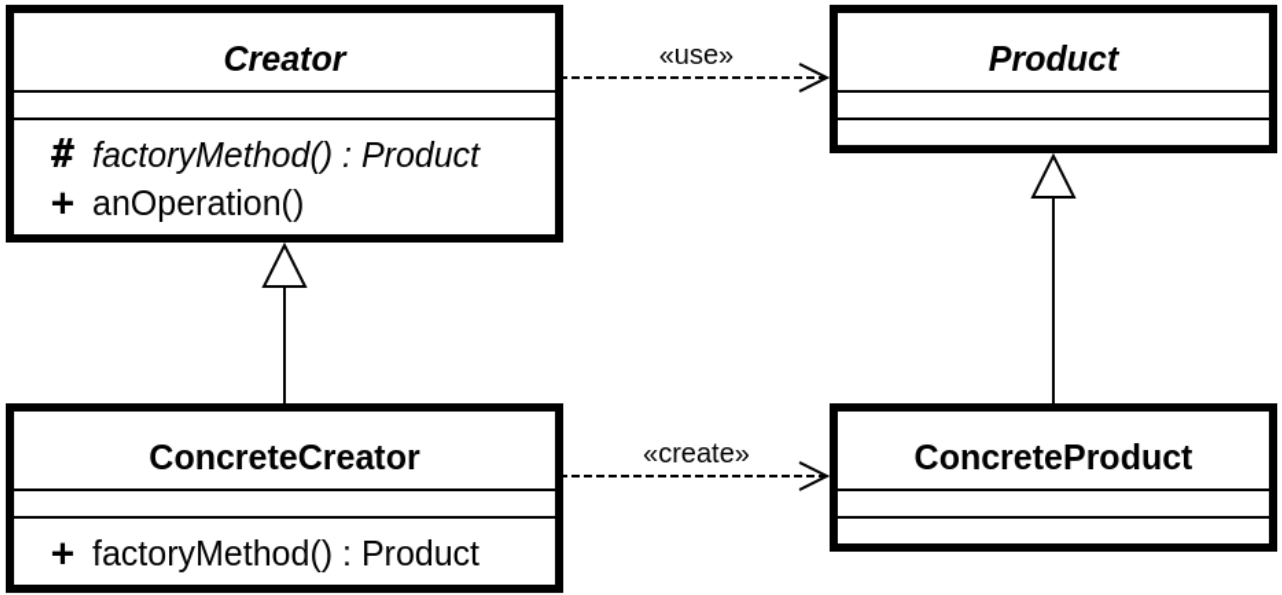


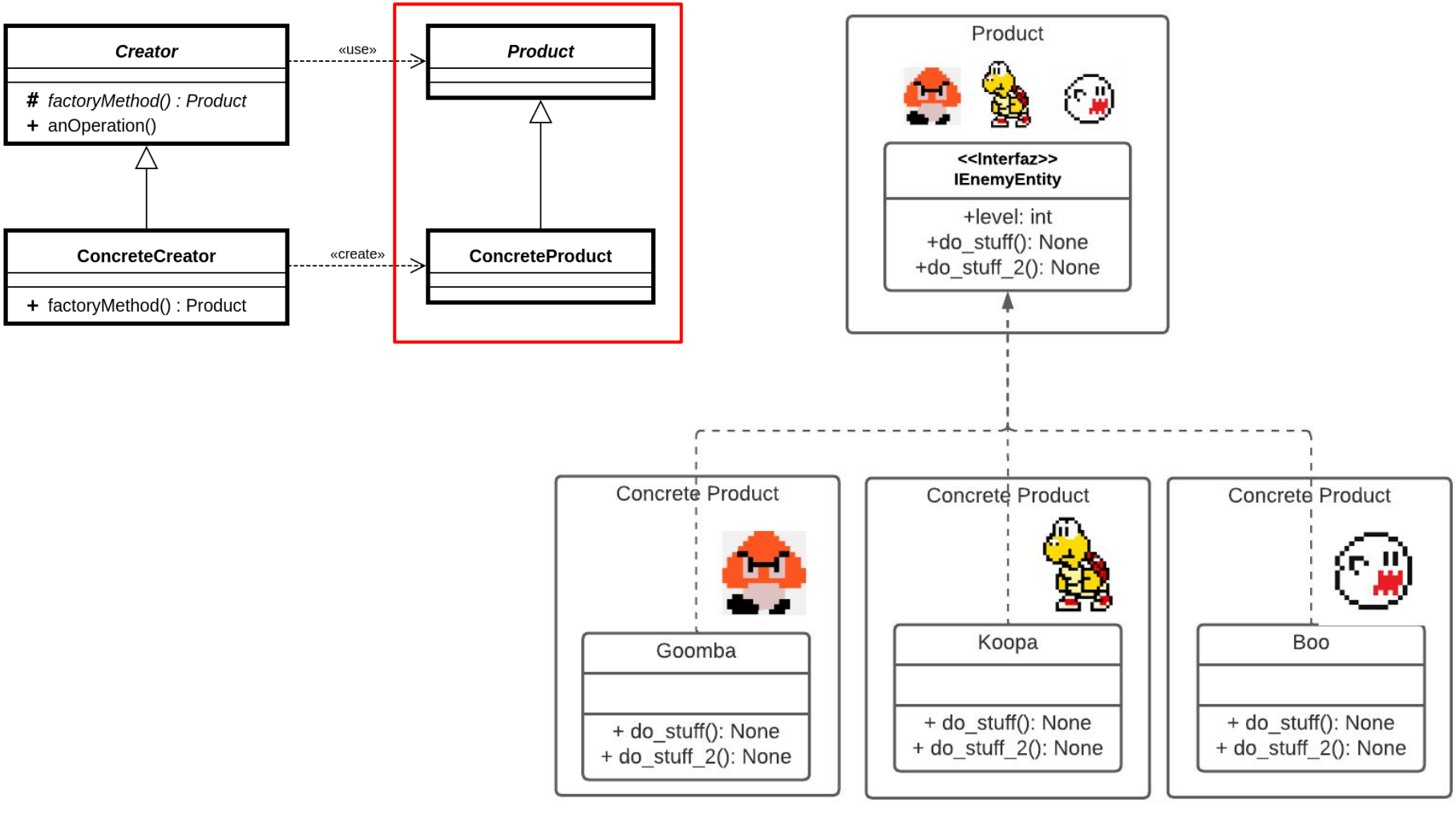

Game
+ start(): None
- spawn_enemy(): Goomba
- should_spawn_enemy(): bool
- enemy_do_stuff(enemy: Goomba) None
- enemy_do_stuff_2(enemy: Goomba) None
...
- enemy_do_stuff_99(enemy: Goomba) None
Goomba
+ do_stuff(): None
+ do_stuff_2(): None
Koopa
+ do_stuff(): None
+ do_stuff_2(): None

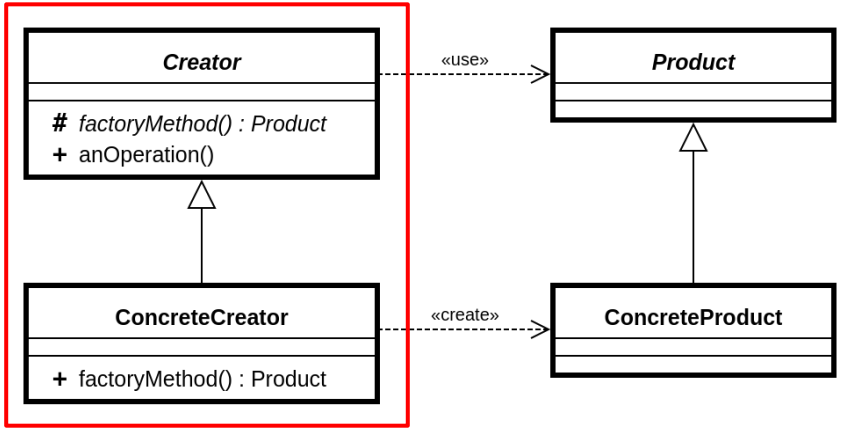
Ver commit:
git checkout 3e4d0ba

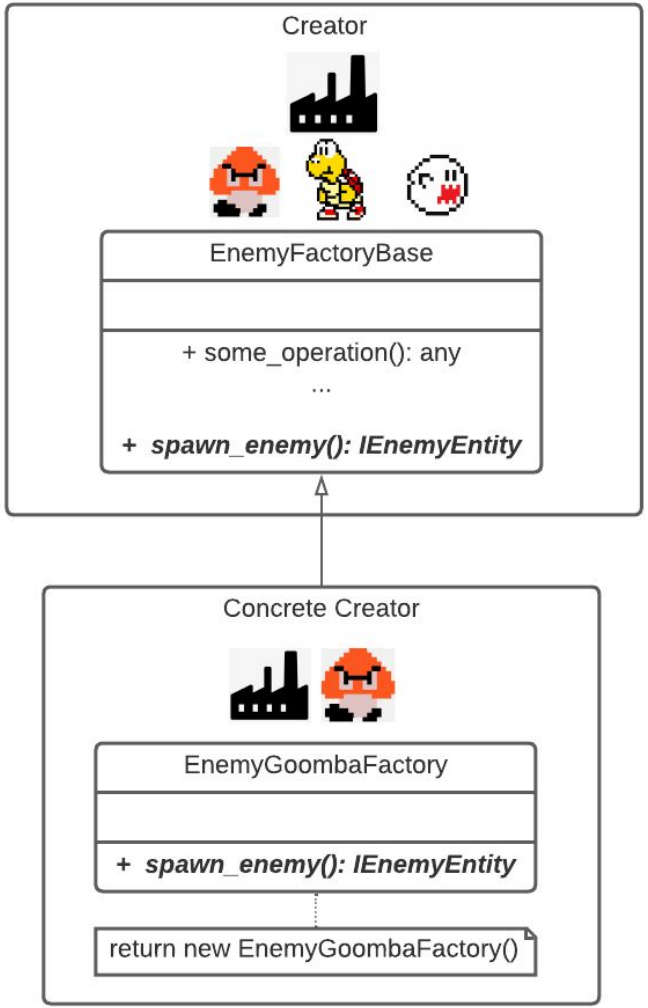


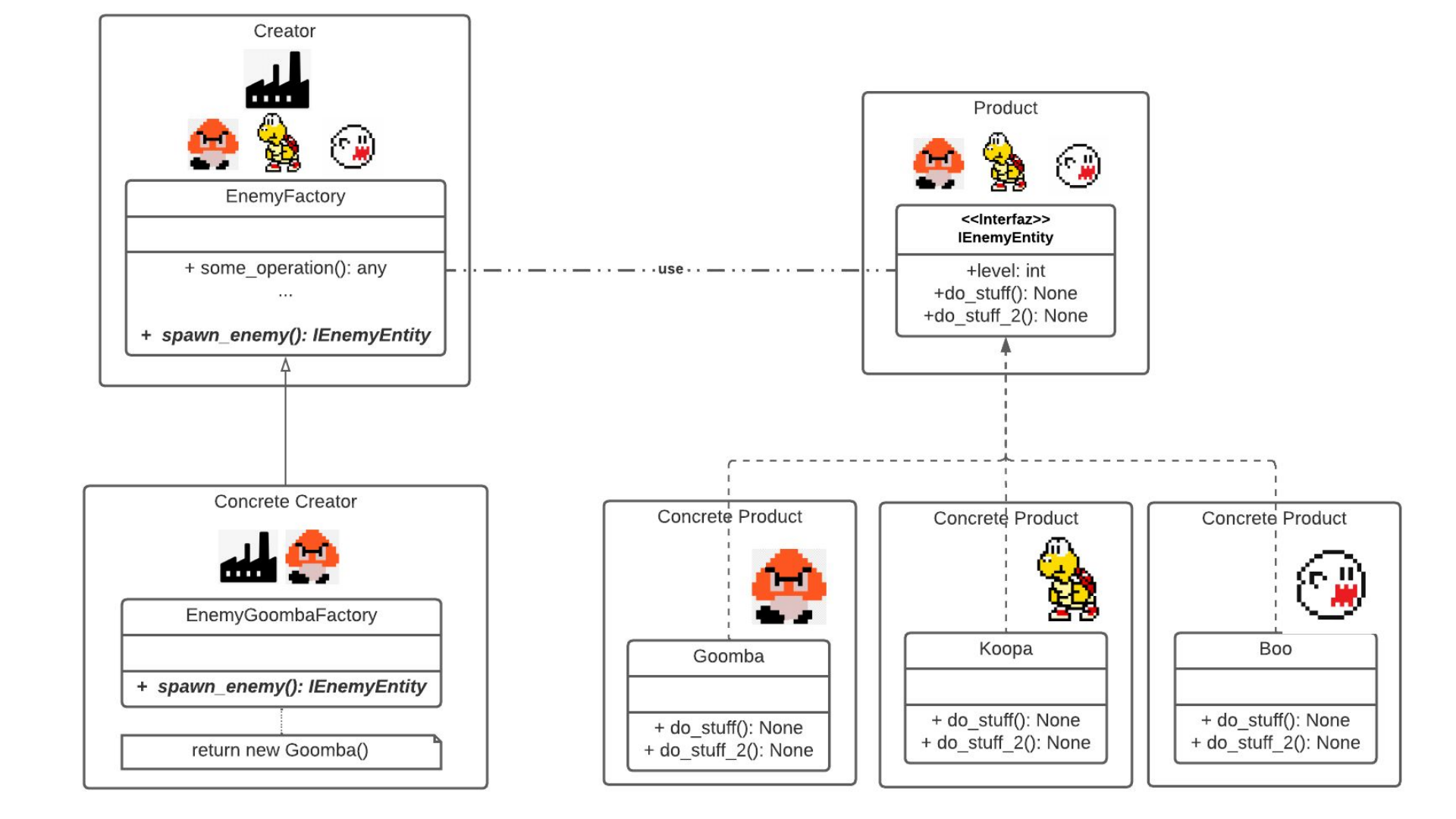
Sugiere que, en lugar de llamar al operador *new,* **se llame a un método abstracto de creación** para regresar los objetos / ”productos”.



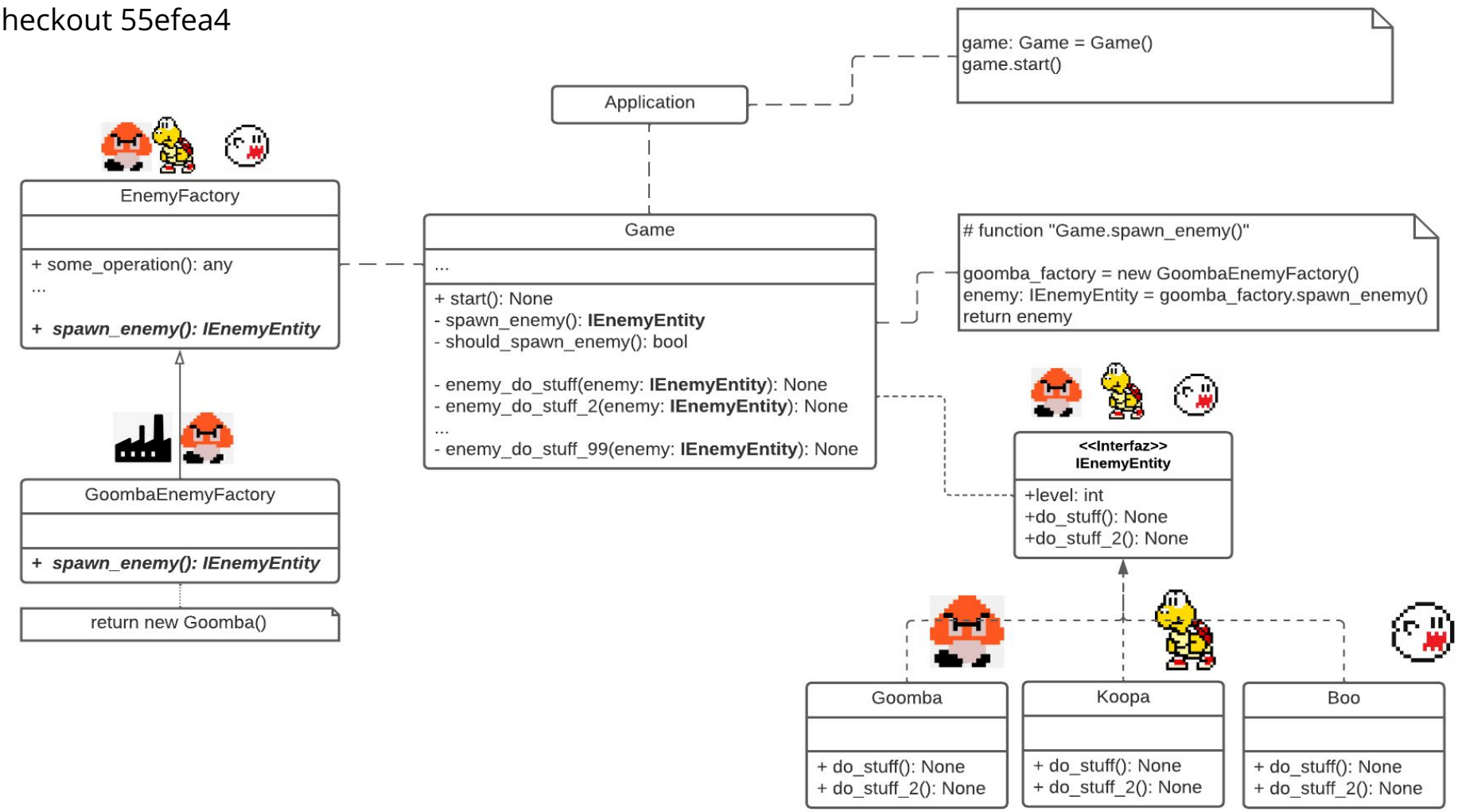

Creator
factoryMethod() : Product
+ anOperation()
«use»
Product
ConcreteCreator
+ factoryMethod() : Product
«create»
ConcreteProduct
Product
<<Interfaz>>
IEnemyEntity
+level: int
+do_stuff(): None
+do_stuff_2(): None
Concrete Product
Goomba
+ do_stuff(): None
+ do_stuff_2(): None
Concrete Product
Koopa
+ do_stuff(): None
+ do_stuff_2(): None
Concrete Product
Boo
+ do_stuff(): None
+ do_stuff_2(): None


Creator
factoryMethod() : Product
+ anOperation()
«use»
Product
ConcreteCreator
+ factoryMethod() : Product
«create»
ConcreteProduct

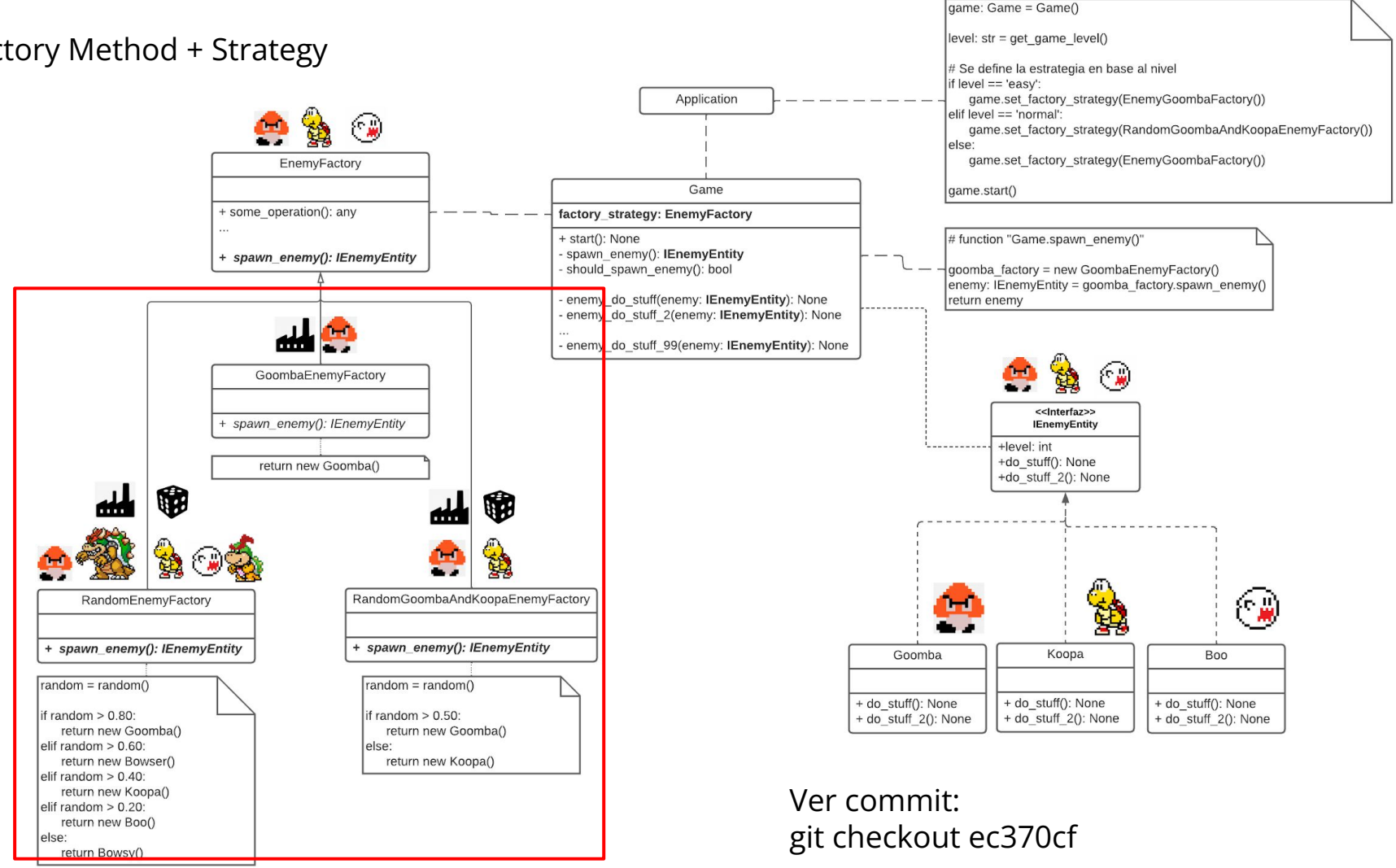

Creator
EnemyFactoryBase
+ some_operation(): any
...
+ spawn_enemy(): IEnemyEntity
Concrete Creator
EnemyGoombaFactory
+ spawn_enemy(): IEnemyEntity
return new EnemyGoombaFactory()


Creator
EnemyFactory
+ some_operation(): any
...
+ spawn_enemy(): IEnemyEntity
use
Product
<<Interfaz>>
IEnemyEntity
+level: int
+do_stuff(): None
+do_stuff_2(): None
Concrete Creator
EnemyGoombaFactory
+ spawn_enemy(): IEnemyEntity
return new Goomba()
Concrete Product
Goomba
+ do_stuff(): None
+ do_stuff_2(): None
Concrete Product
Koopa
+ do_stuff(): None
+ do_stuff_2(): None
Concrete Product
Boo
+ do_stuff(): None
+ do_stuff_2(): None

t checkout 55efea4



Factory Method + Strategy



Ver commit:
git checkout ec370cf

Gracias