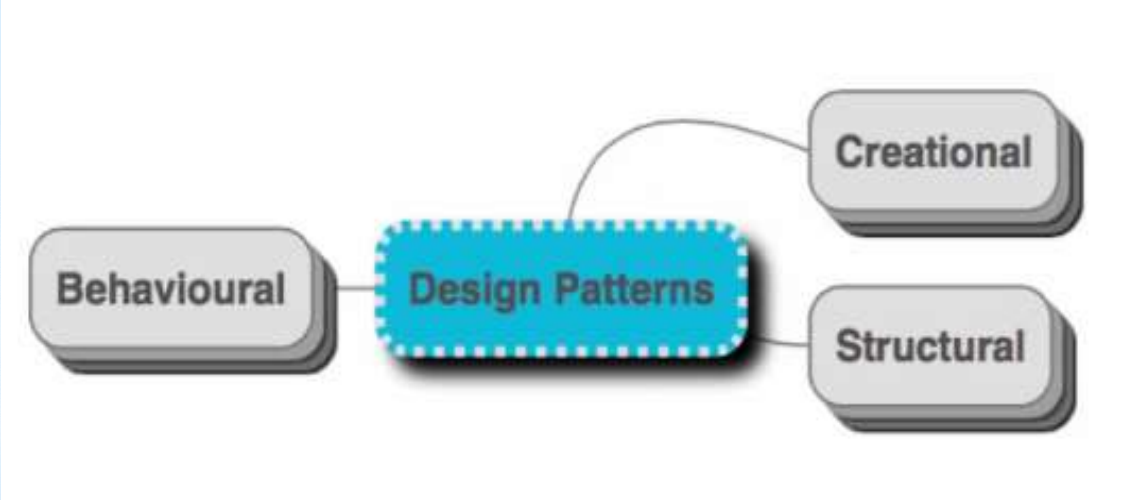
Patrones de Diseño

Diseño Orientado a Objetos

Patrón de Diseño

- Solución de diseño a un problema recurrente en un contexto particular.
- Problemas comunes y sus soluciones (GoF).



Design Patterns

Aplicaciones

- Frameworks
- Angular (JavaScript)
- Spring (ORM, servicios REST, Swing,…)
- Spring MVC (capa presentación)
- Spring Core (inyección dependencias/IoC)
- Configuración Beans Spring (singleton)

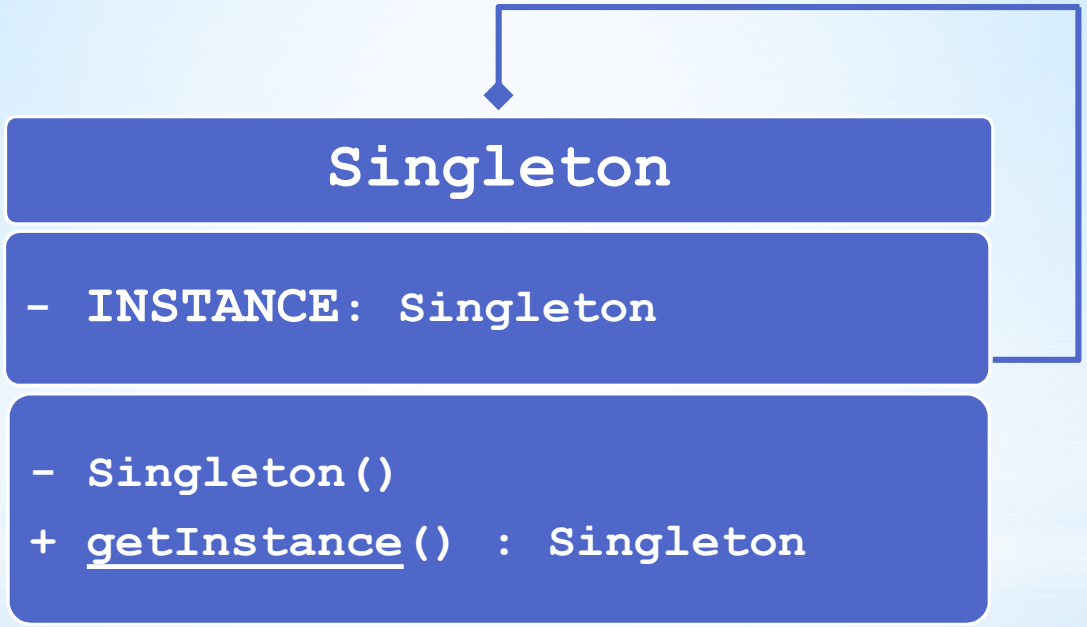
Design Patterns

Creacionales

- ✓ Singleton
- Prototype
- Builder
- ✓ Factory Method

Singleton

Asegurar que una clase sólo tiene **una instancia** y proporciona un **punto global de acceso** a este.



Singleton

Implementación

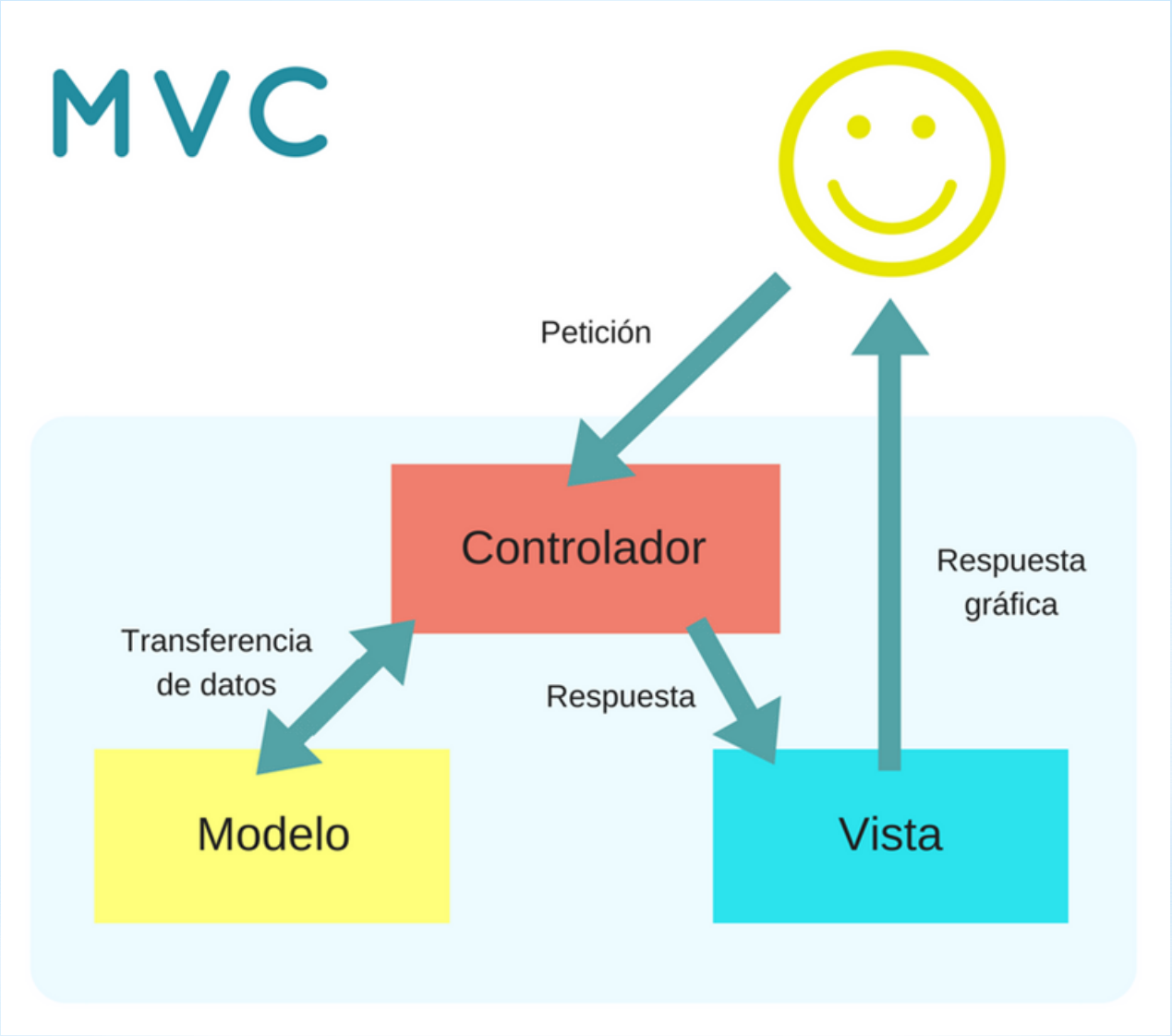
- Sólo una instancia: Constructor privado y hacer que la clase gestione su instancia.
- Punto global de acceso: Método estático para obtener la única instancia.

Singleton

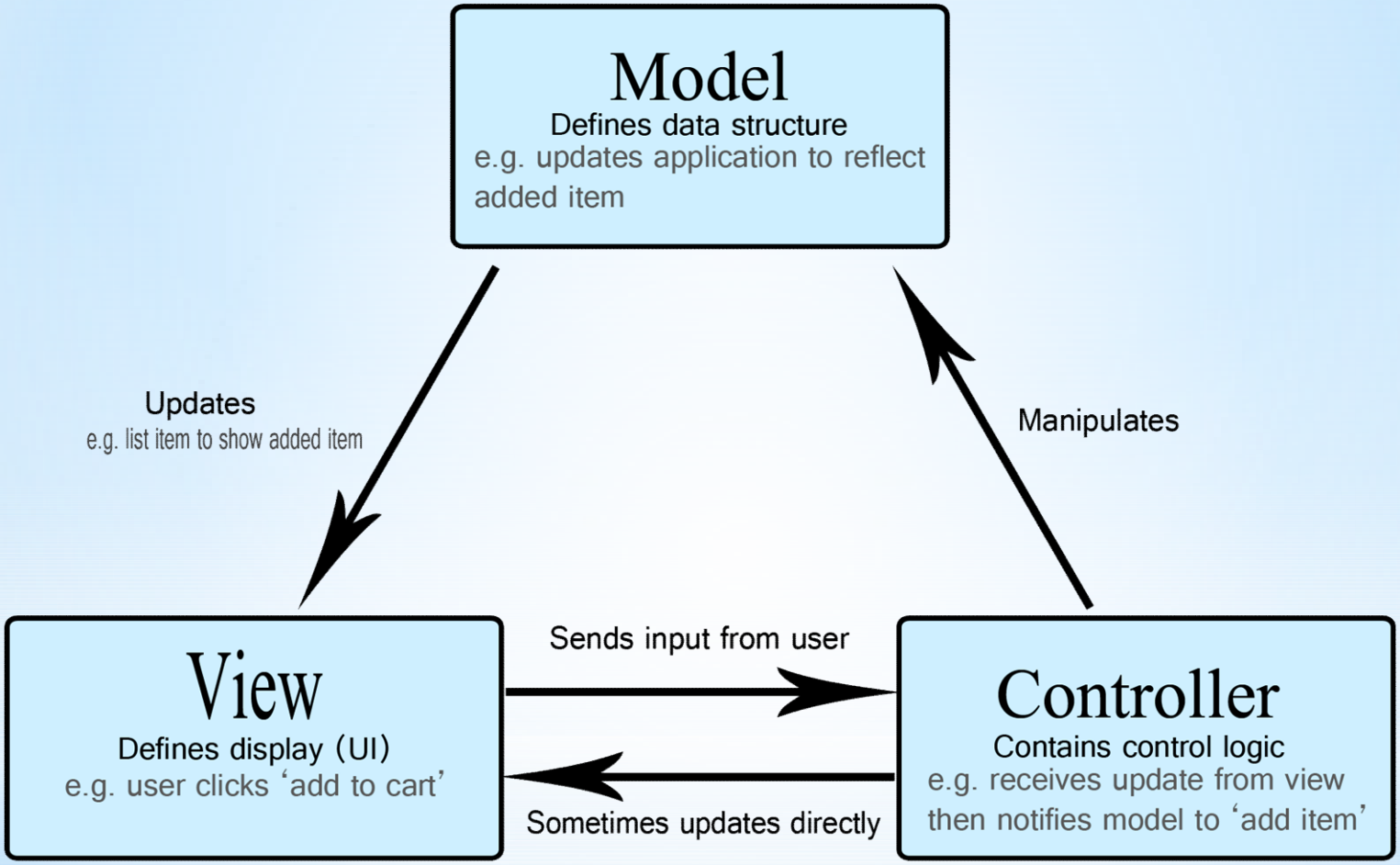
A tener en cuenta…

- Reflection
- Serialization / Deserialization
- Clone
- Multi-threaded access
- Multiple class loaders**

Modelo-Vista-Controlador

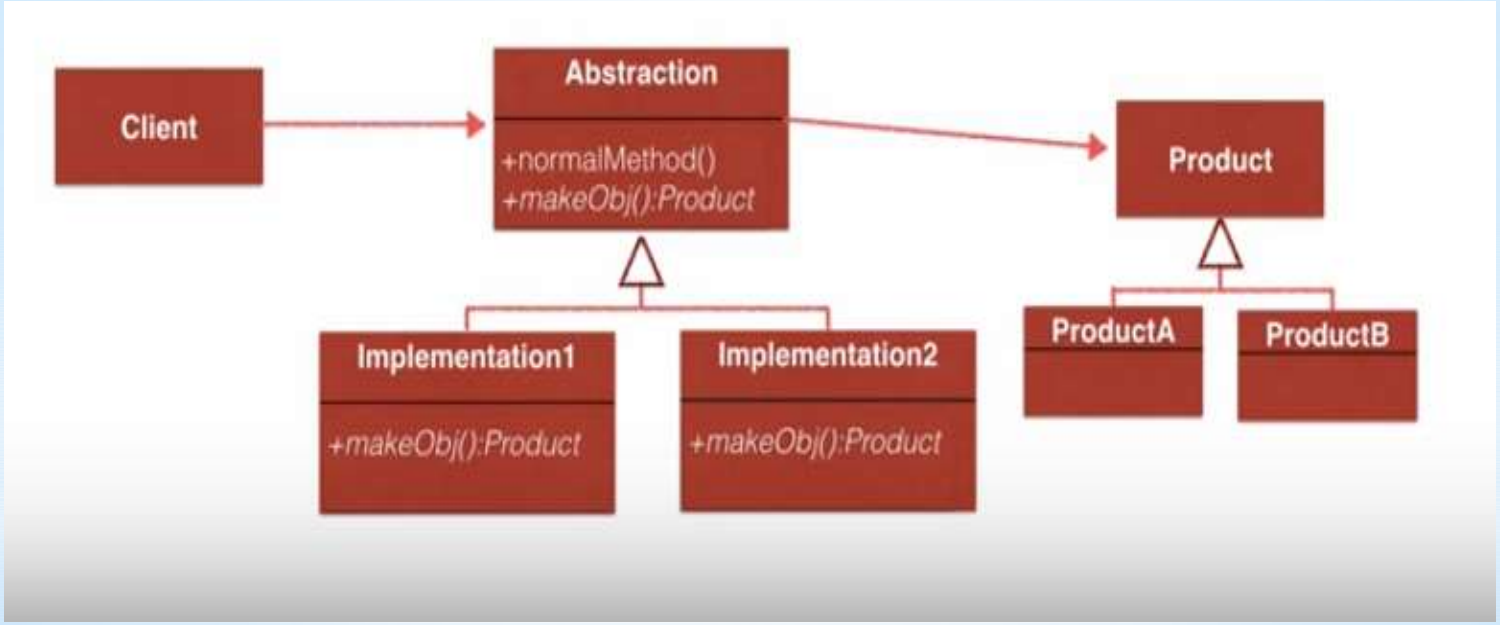


Modelo-Vista-Controlador



Factory Method

Define una **interfaz*** para crear un objeto, pero deja que las **subclases** decidan qué objeto **instanciar**.



Factory Method

Implementación

- Proveer un método para crear objetos de una subclase con una superclase en común.
- Sólo las subclases deciden qué objeto exacto instanciar → este patrón implica Herencia.

Factory Method

Justificación

- Cuando no sabemos con antelación qué objeto necesitamos crear.
- Para centralizar el código que selecciona las clases a usar.
- Para ocultar al usuario todas las subclases.
- Para encapsular la creación de objetos.

Design Patterns

Estructurales

- ✓ Adapter
- Composite
- Decorator
- ✓ Facade
- Proxy

Proxy

Un objeto que representa a otro objeto.

- Credit card is a proxy for what is in our bank account.
- Objetos remotos (sistemas distribuidos)
- EJB – Remote and Home: Proxy oculta la complejidad en la comunicación con el objeto real

Decorator

Añade responsabilidades a los objetos dinámicamente.

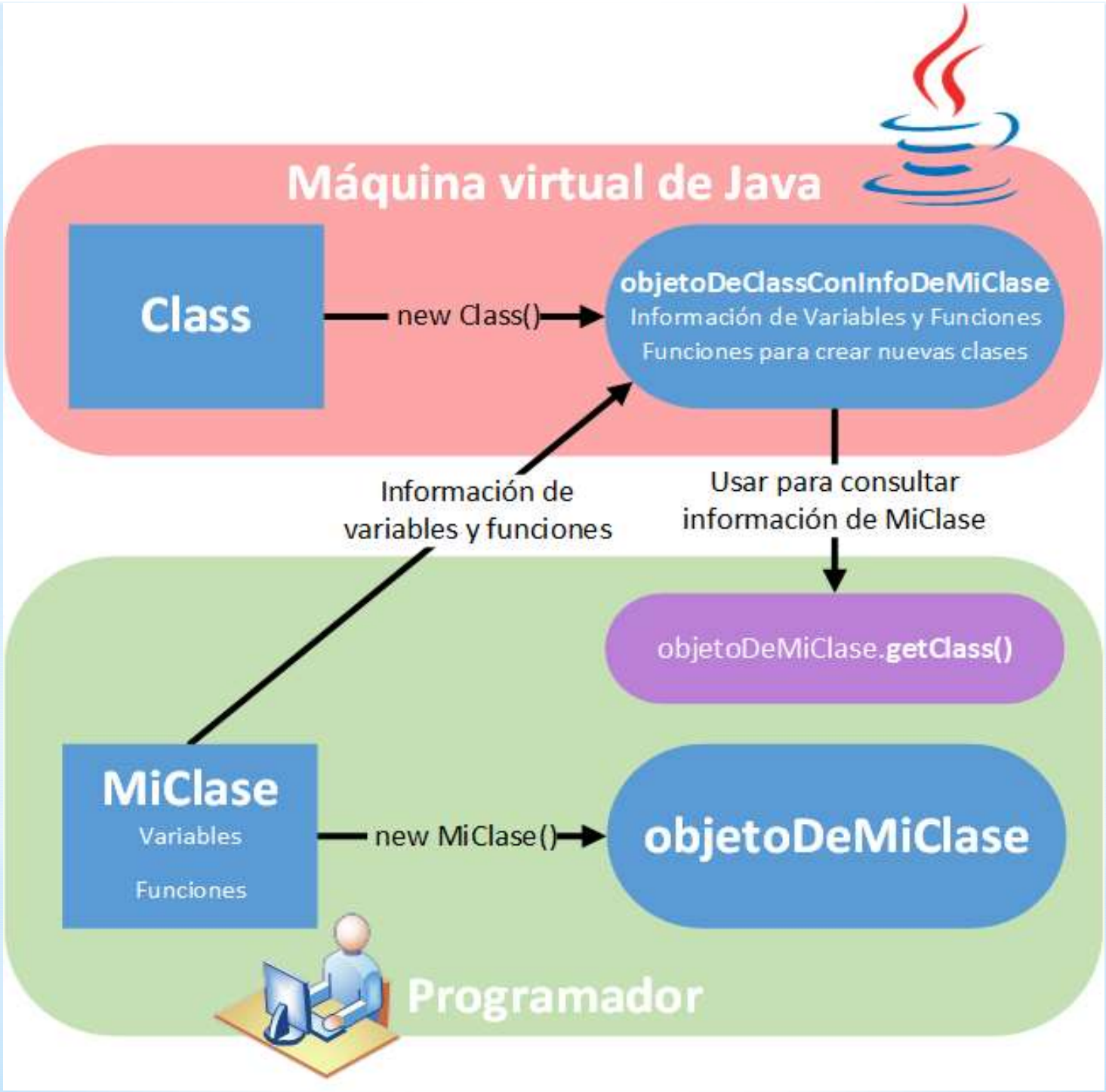
- Permite añadir comportamiento en tiempo de ejecución de forma sencilla.

- Java I/O: Complejidad al crear objetos.

Reflection

Java Reflection API

- “Manipulador” de clases.
- Librería para manipular clases (atributos, métodos, constructores, atributos y métodos privados…).
- Puede ralentizar la ejecución del código (JVM no puede optimizar el código)
- Debería usarse de forma controlada.
- No es un patrón de diseño. A menudo se usa junto con patrones de diseño.


Máquina virtual de Java
Class
new Class()
objetoDeClassConInfoDeMiClase
Información de Variables y Funciones
Funciones para crear nuevas clases
Información de
variables y funciones
Usar para consultar
información de MiClase
objetoDeMiClase.getClass()
MiClase
Variables
Funciones
new MiClase()
objetoDeMiClase
Programador

Reflection

Java Reflection API

- “methodToExecute” pasado como un String
- “methodToExecute”(int var1, String var2);