UNIVERSIDAD
Popular del cesar

Ingeniería de Sistemas

ESPECIALIZACION EN INGENIERIA DE SOFTWARE
MODULO PATRONES DE DISEÑO DE SOFTWARE

EL DOCENTE

JAIRO FRANCISCO SEOANES LEON

jairoseoanes@unicesar.edu.co
(300) 600 06 70

Educación formal

- ✓ **Ingeniero de sistemas**, Universidad Popular del Cesar sede Valledupar, Feb 2002 – Jun 2009.
- ✓ **MsC en Ingeniería de Sistemas y Computación**, Universidad Nacional de Colombia, Bogotá, Feb 2011 – Mar 2015
- ✓ **PhD Ciencia, Tecnologia e innovación, Urbe, Venezuela, Mayo 2024**

Formación complementaria

- ✓ **AWS Academy Graduate -** AWS Academy Cloud Foundations, 2022
  https://www.credly.com/go/p3Uwht36
- ✓ **Associate Cloud Engineer Path -** Google Cloud Academy, 2022
  https://www.cloudskillsboost.google/public_profiles/c7e7936c-3e37-4bad-b822-74d40c49d0db
- ✓ **Fundamentos De Programación Con Énfasis En Cloud Computing –** AWS Academy y Misión Tic 2022
- ✓ **Google Cloud Computing Foundations –** Google Academy, 2022
- ✓ **Aplicación de cloud: retos y oportunidades de mejora para las empresas de software gestionando la computación en la nube –** Fedesoft, 2023
- ✓ **Desarrollo De Aplicaciones Web En Angular, Para El Nivel Frontend** – Universidad EAFIT, 2023
- ✓ **Microsoft Scrum Foundations –** Intelligent Training - MinTic , 2023

Experiencia profesional

- ✓ **Docente Universitario**, Universidad Popular del Cesar sede Valledupar, marzo del 2013.
- ✓ **Técnico de Sistemas Grado 11**, Rama judicial Seccional Cesar, SRPA Valledupar, Junio del 2009

UNIVERSIDAD Popular del cesar

MODULO DE PATRONES DE DISEÑO DE SOFTWARE

Unidad 1

Introducción a UML Y Diseño O.O

Unidad 2

Introducción A Los Patrones de Diseño

Unidad 3

Patrones de Diseño Creacionales

Unidad 4

Patrones de Diseño Estructurales

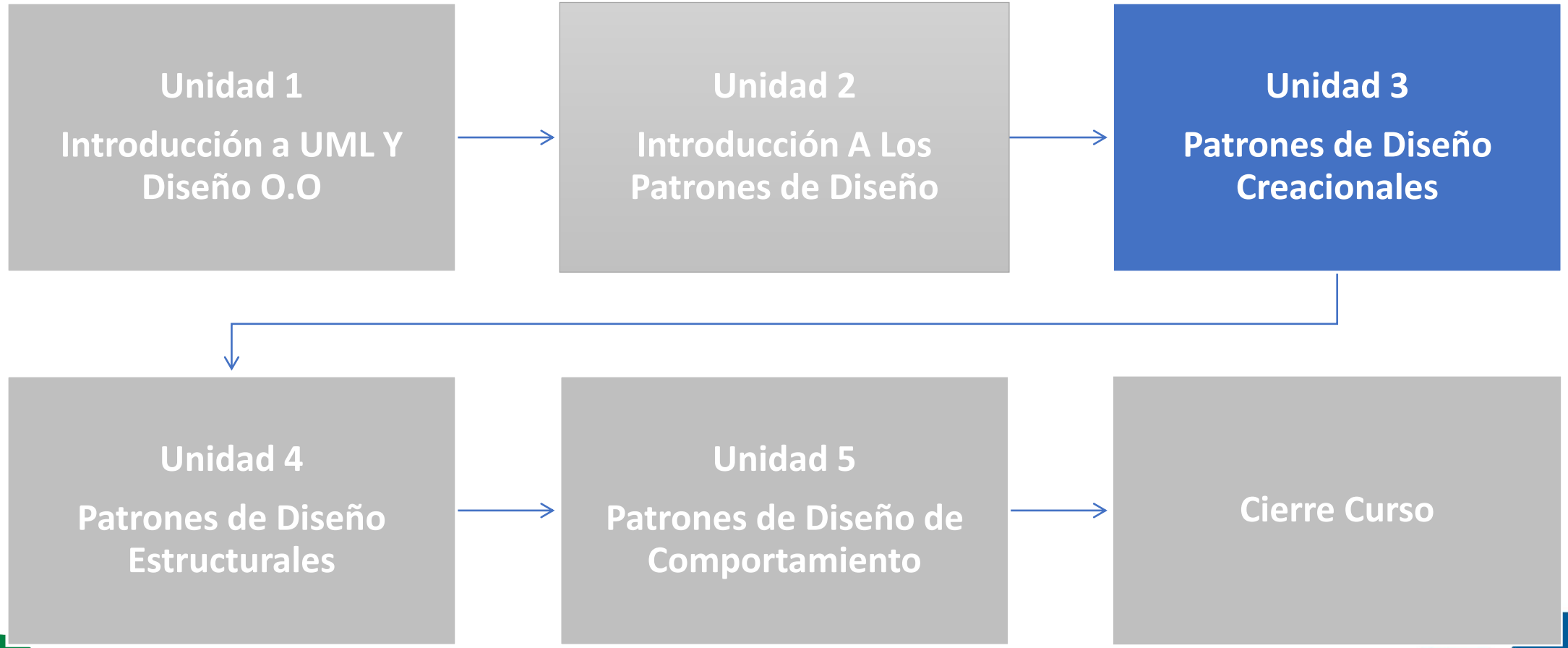
Unidad 5

Patrones de Diseño de Comportamiento

Cierre Curso

UNIVERSIDAD Popular del cesar

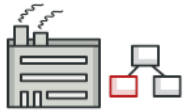
MODULO DE PATRONES DE DISEÑO DE SOFTWARE



UNIVERSIDAD Popular del cesar

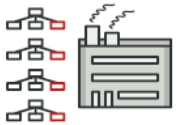
Unidad 3. Patrones creacionales

3.1 Singleton
3.1 Factory Method
3.1 Protoype
3.1 Abstract Factory
3.1 Builder



Factory Method

Proporciona una interfaz para la creación de objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

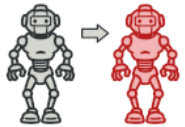


Abstract Factory

Permite producir familias de objetos relacionados sin especificar sus clases concretas.

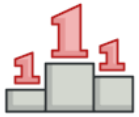
Builder

Permite construir objetos complejos paso a paso. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.



Prototype

Permite copiar objetos existentes sin que el código dependa de sus clases.



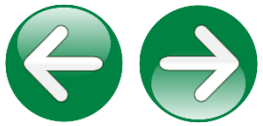
Singleton

Permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

UNIVERSIDAD Popular del cesar

Patrones creacionales - Generales

- ✓ Sirven para controlar la forma en que se crean los objetos
- ✓ La creación de objetos se realiza gracias al mecanismo de instanciación (operador new)
  Clase objeto = **new** Clase();
- ✓ Múltiples razones para que los objetos se creen de forma controlada
  - Necesidad de existencia de una sola instancia de un tipo de clase
  - No se sabe que tipo de objeto requiero utilizar hasta en tiempo de ejecución
  - La dificultad es todavía mayor cuando hay que construir objetos compuestos cuyos componentes pueden instanciarse mediante clases diferentes.



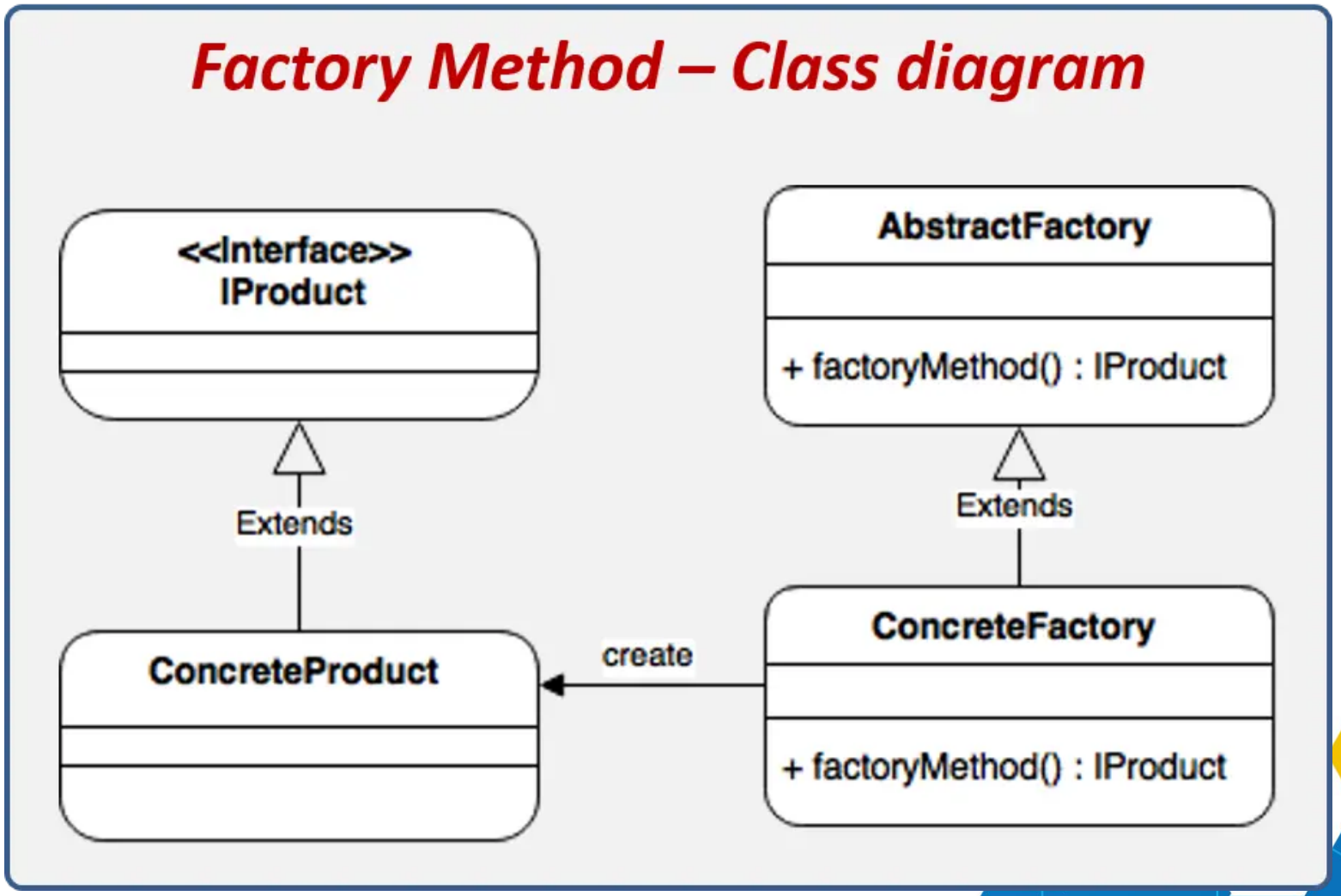
UNIVERSIDAD Popular del cesar

Patrones creacionales – Factory Method

Permite la creación de objetos de un subtipo determinado a través de una clase Factory

Útil cuando no sabemos en tiempo de diseño el subtipo que se va a utilizar

Permite crear instancias de manera dinámica, a través de archivos xml, texto, properties, o cualquier otra estrategia



UNIVERSIDAD Popular del cesar

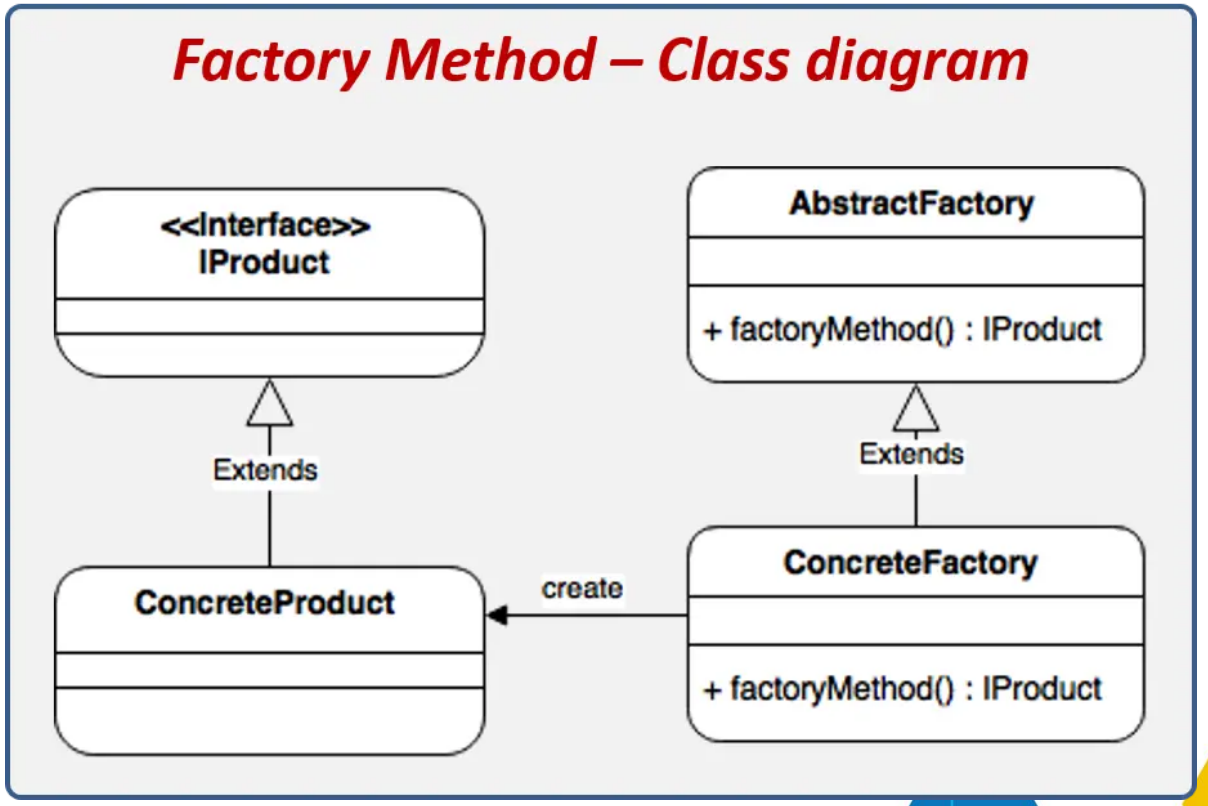
Patrones creacionales – Factory Method

IProduct: forma abstracta del objeto que de desea crear, define la estructura que tendrá el objeto creado

ConcreteProduct: representa una implementación concreta de **IProduct**

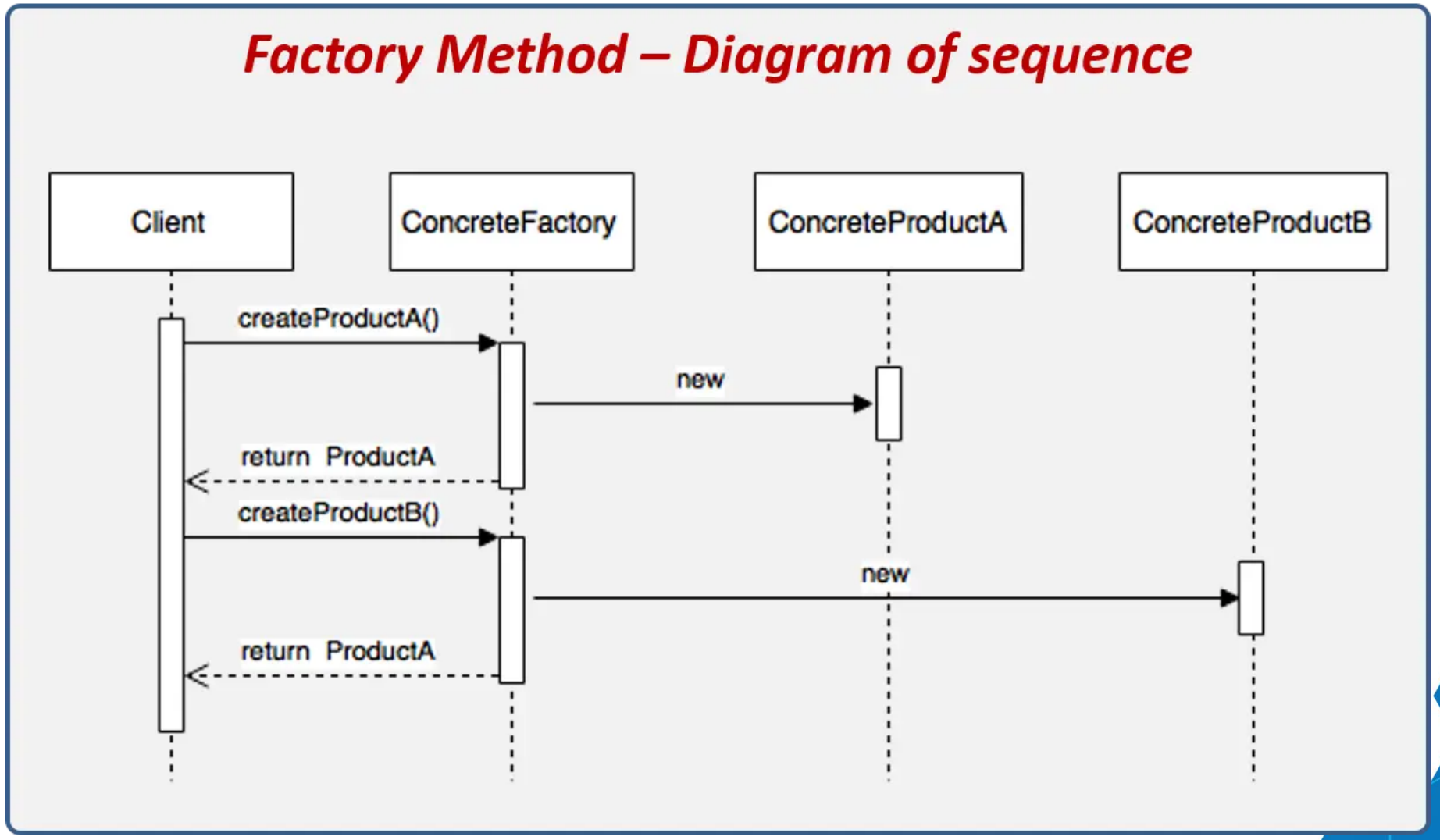
AbstractFactory: puede ser opcional, define el comportamiento por default de los **ConcreteFactory**

ConcreteFactory: representa una fabrica concreta, utilizada para la creación de los ConcreteProduct. Implementa el comportamiento básico del **AbstractFactory.**



UNIVERSIDAD Popular del cesar

Patrones creacionales – Factory Method

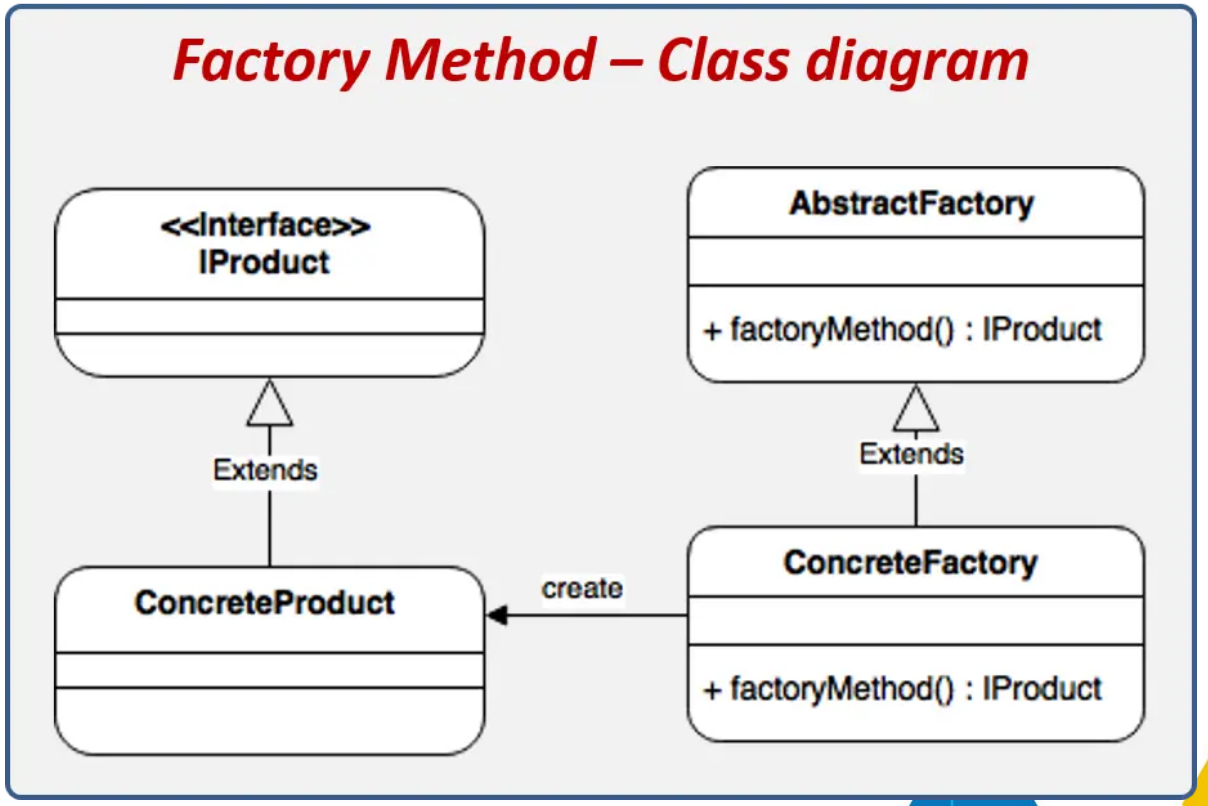


UNIVERSIDAD Popular del cesar

Patrones creacionales – Factory Method

Cuando implementarlo:

- Utiliza el Método Fábrica cuando no conozcas de antemano las dependencias y los tipos exactos de los objetos con los que deba funcionar tu código.
- Utiliza el Factory Method cuando quieras ofrecer a los usuarios de tu biblioteca o framework, una forma de extender sus componentes internos.
- Cuando es necesario crear objetos por medio de una interface común
- Cuando construimos objetos a partir de condiciones



UNIVERSIDAD Popular del cesar

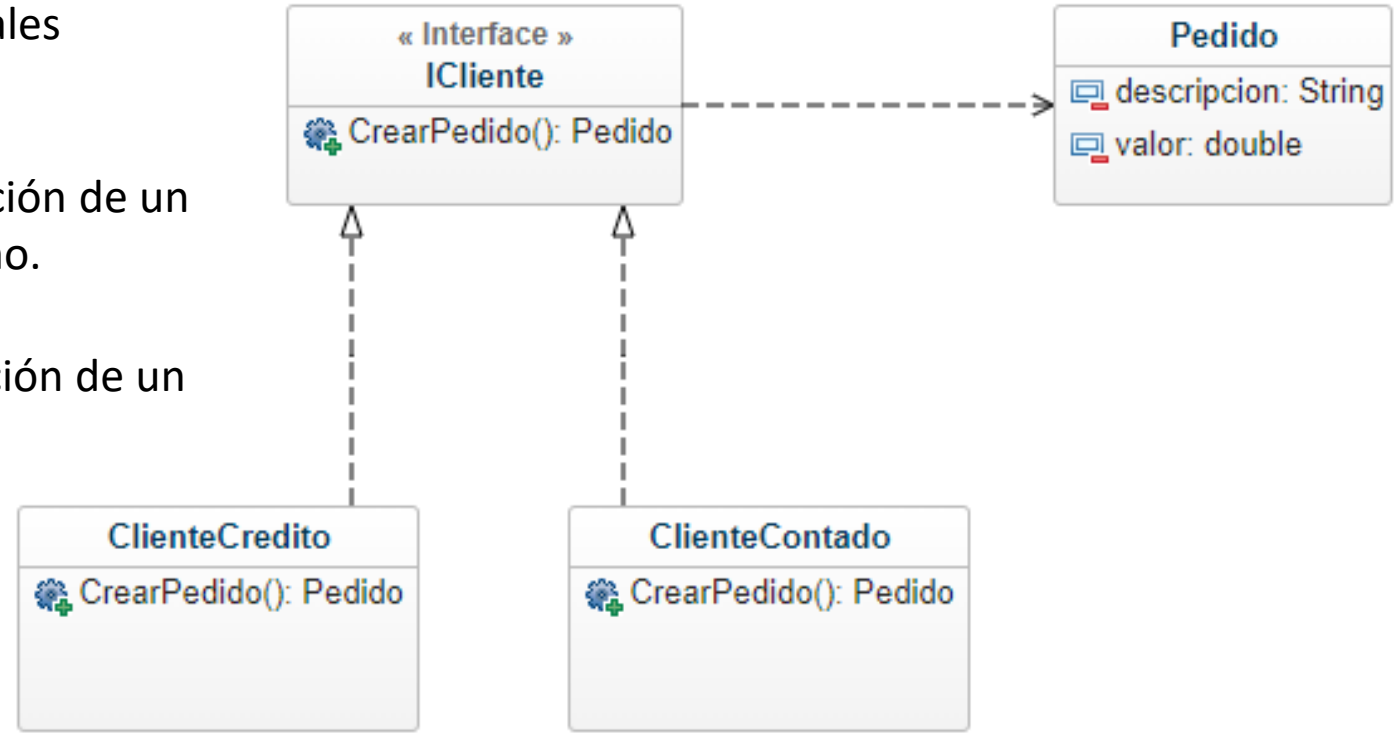
Patrones creacionales – Factory Method

Ejemplo:

Consideremos una aplicación que maneja dos tipos de clientes: ClienteContado y ClienteCredito, los cuales pueden crear un Pedido.

A partir de una clase principal, se solicita la creación de un cliente, en función de la forma de pago del mismo.

Así mismo, s ele solicita al cliente creado la creación de un pedido.



UNIVERSIDAD Popular del cesar

Patrones creacionales – Factory Method

Ejemplo:

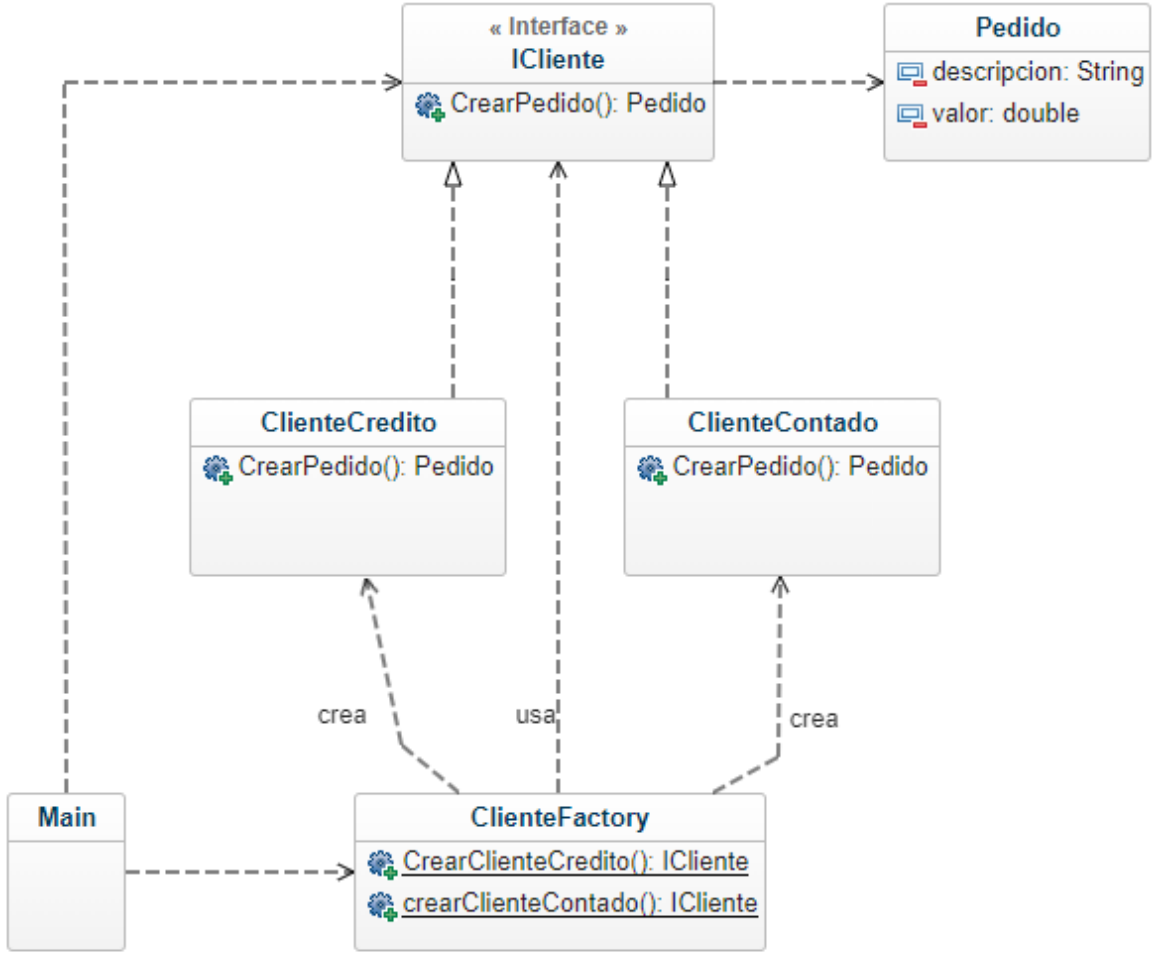
Consideremos una aplicación que maneja dos tipos de clientes: ClienteContado y ClienteCredito, los cuales pueden crear un Pedido.

A partir de una clase principal, se solicita la creación de un cliente, en función de la forma de pago del mismo.

Así mismo, s ele solicita al cliente creado la creación de un pedido

La compañía manera un nuevo tipo de cliente denominado ClienteCrediContado.

La compañía maneja pedidos a crédito y pedidos de contado



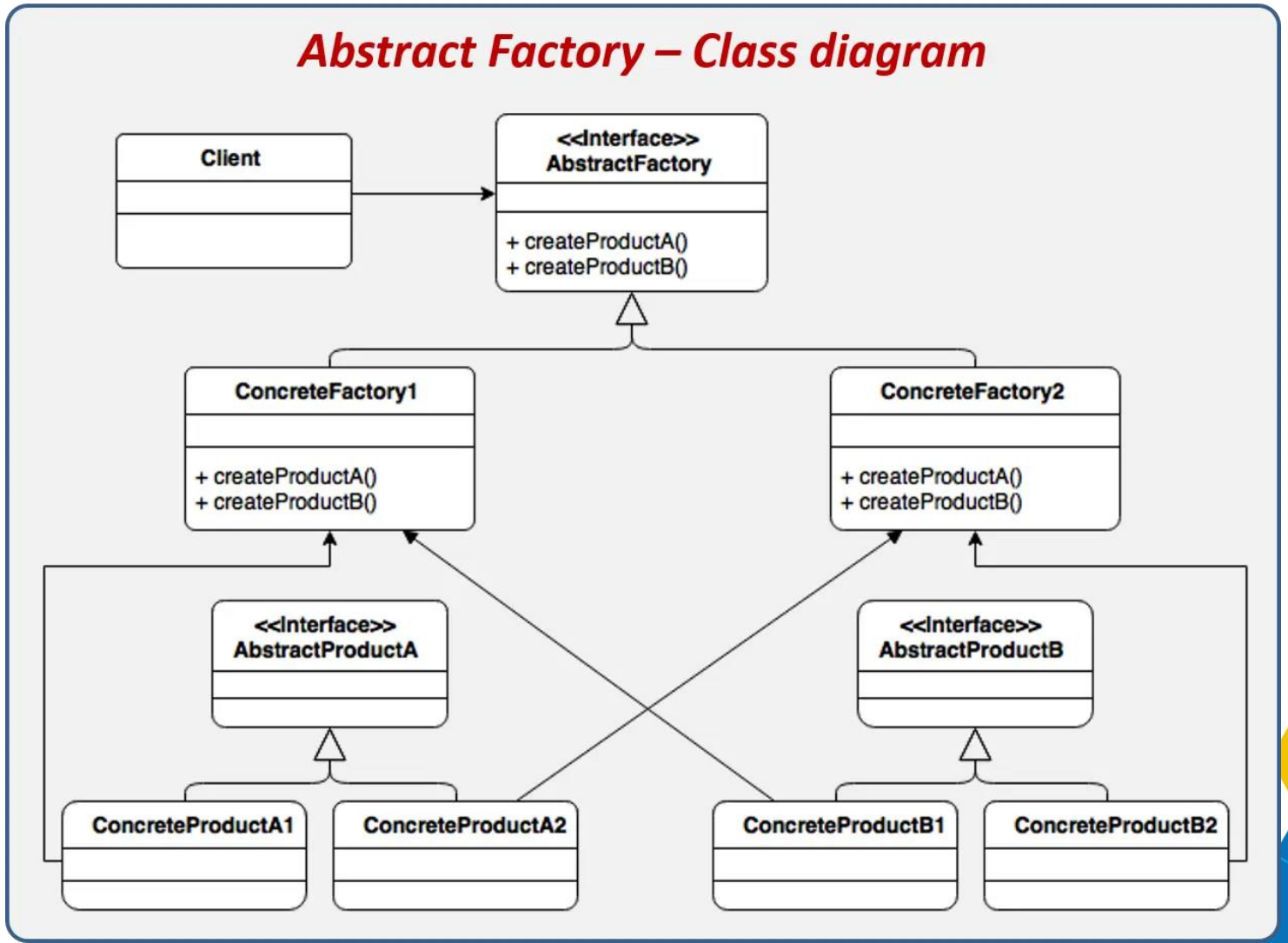
UNIVERSIDAD Popular del cesar

Patrones creacionales – Abstract Factory

Permite producir familias de objetos relacionados sin especificar sus clases concretas, a partir de un **Factory**

Útil cuando se requiere tener un conjunto de familias de clases para resolver un problema

Sin embargo se requieren implementaciones paralelas para resolver el problema de una forma distinta



UNIVERSIDAD Popular del cesar

Patrones creacionales – Abstract Factory

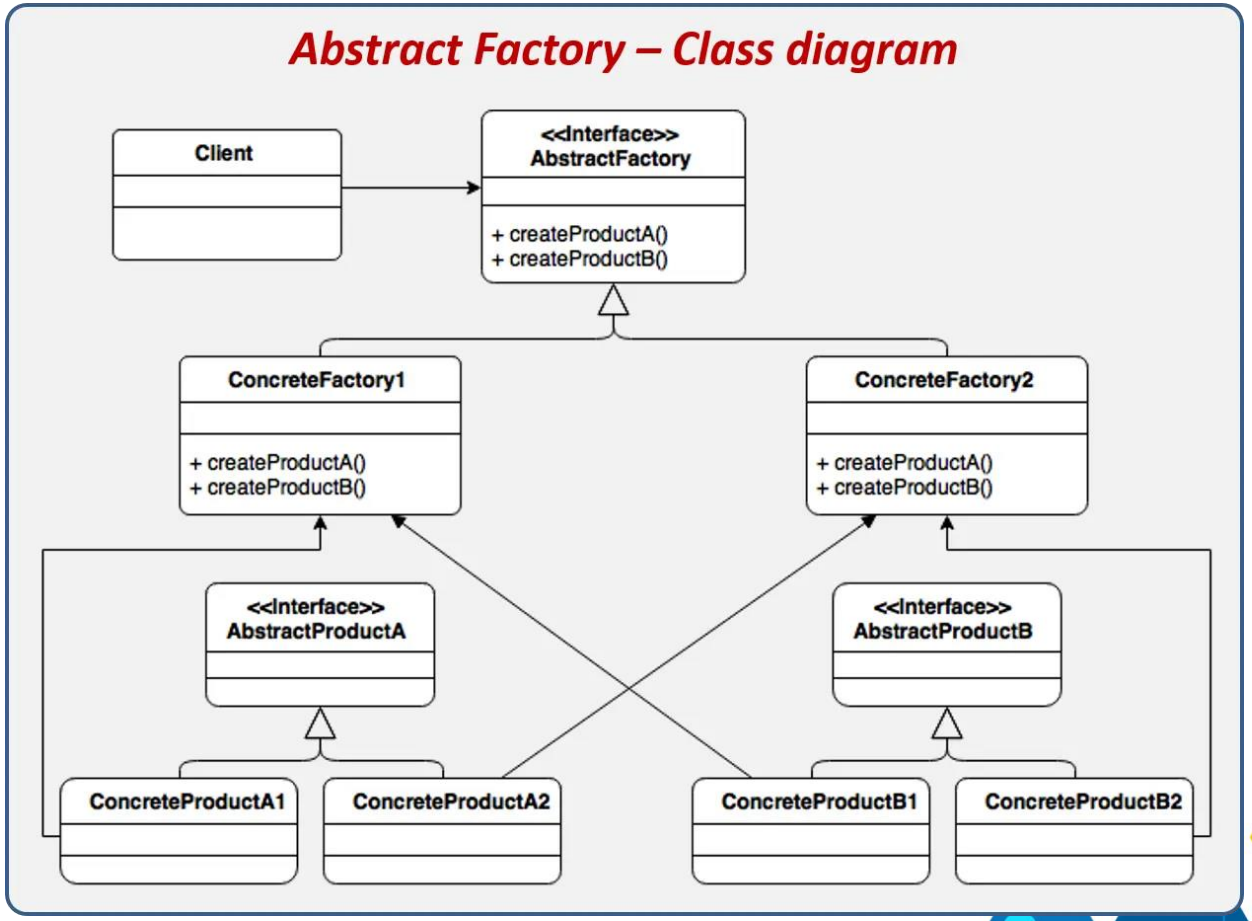
Client: Quien dispara la ejecución del patrón.

AbstractProduct (A, B): Interfaces que definen la estructura de los objetos para crear familias.

ConcreteProduct (A, B): Clases que heredan de **AbstractProduct** con el fin de implementar familias de objetos concretos.

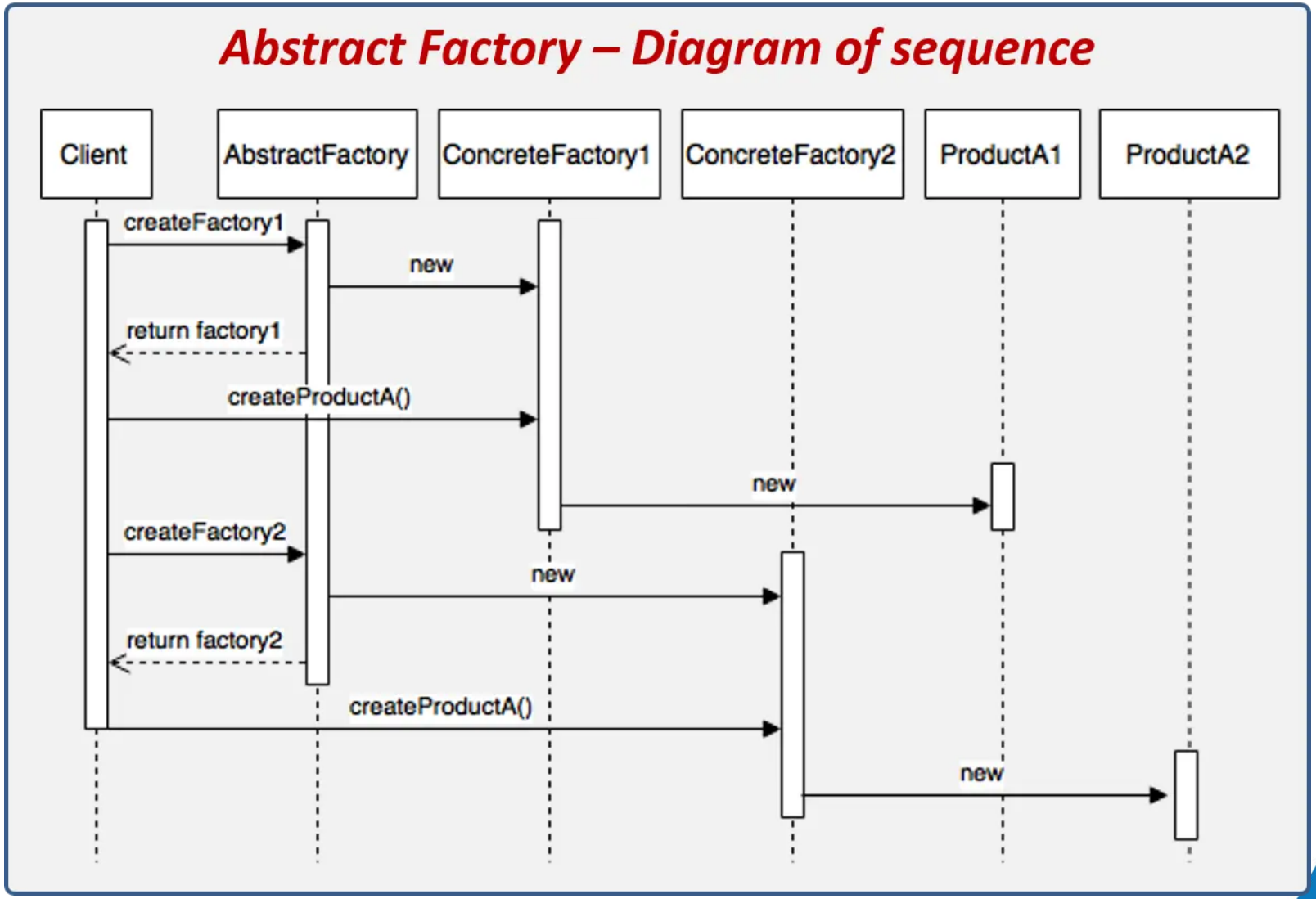
AbstractFactory: Define la estructura de las fábricas y deben proporcionar un método para cada clase de la familia

ConcreteFactory: Representan las fábricas concretas que servirán para crear las instancias de todas las clases de la familia. En esta clase debe existir un método para crear cada una de las clases de la familia.



UNIVERSIDAD Popular del cesar

Patrones creacionales – Abstract Factory

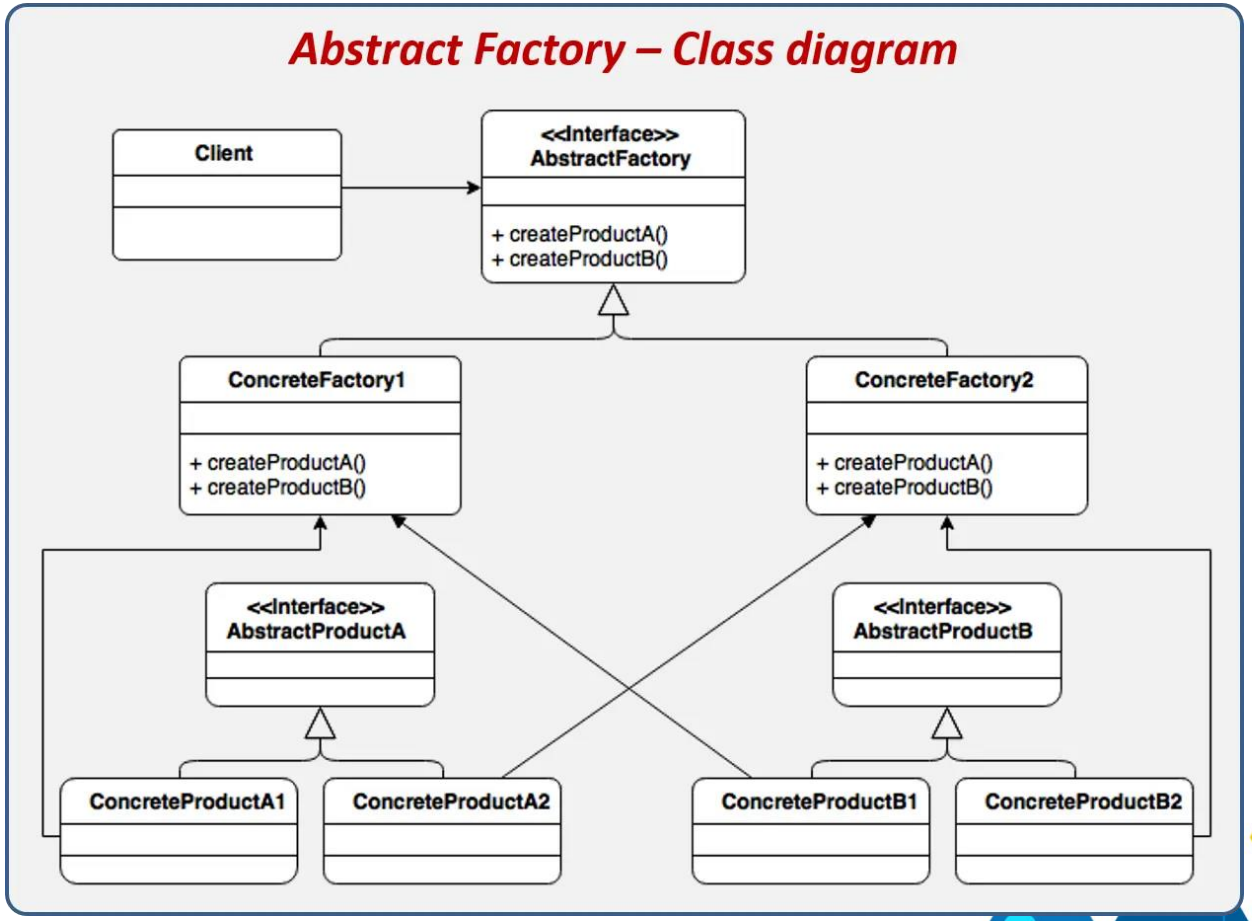


UNIVERSIDAD Popular del cesar

Patrones creacionales – Abstract Factory

Cuando implementarlo:

- Cuando no se conoce en tiempo de diseño la familia de clase que se utilizara en la ejecución
- Cuando es necesario crear objetos a partir de una interface común
- Cuando es necesario trabajar con distintas plataformas, manteniendo la misma estructura y compatibilidad con ella



UNIVERSIDAD Popular del cesar

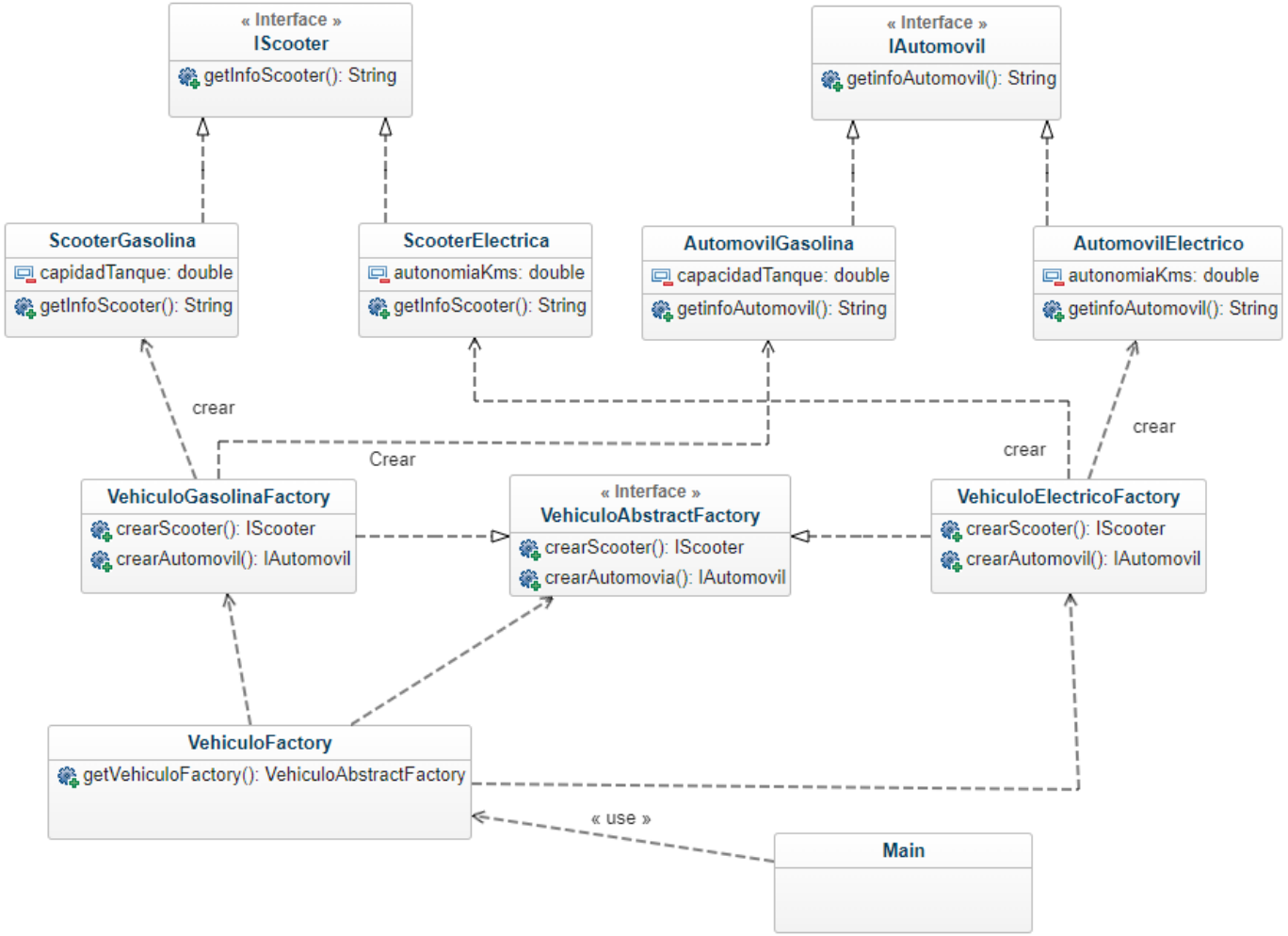
Patrones creacionales – Abstract Factory

Ejemplo:

Suponga un sistema de venta de vehículos que gestiona vehículos que funcionan con gasolina y vehículos eléctricos.

Actualmente maneja dos tipos de producto: Scotter y Automovil, de los cuales, encontramos en sus dos variedades (gasolina y eléctricos)

Desde una clase principal se requiere crear instancias de objetos concretos de cualquiera de estas dos familias de producto.



UNIVERSIDAD Popular del cesar

Patrones creacionales – Abstract Factory

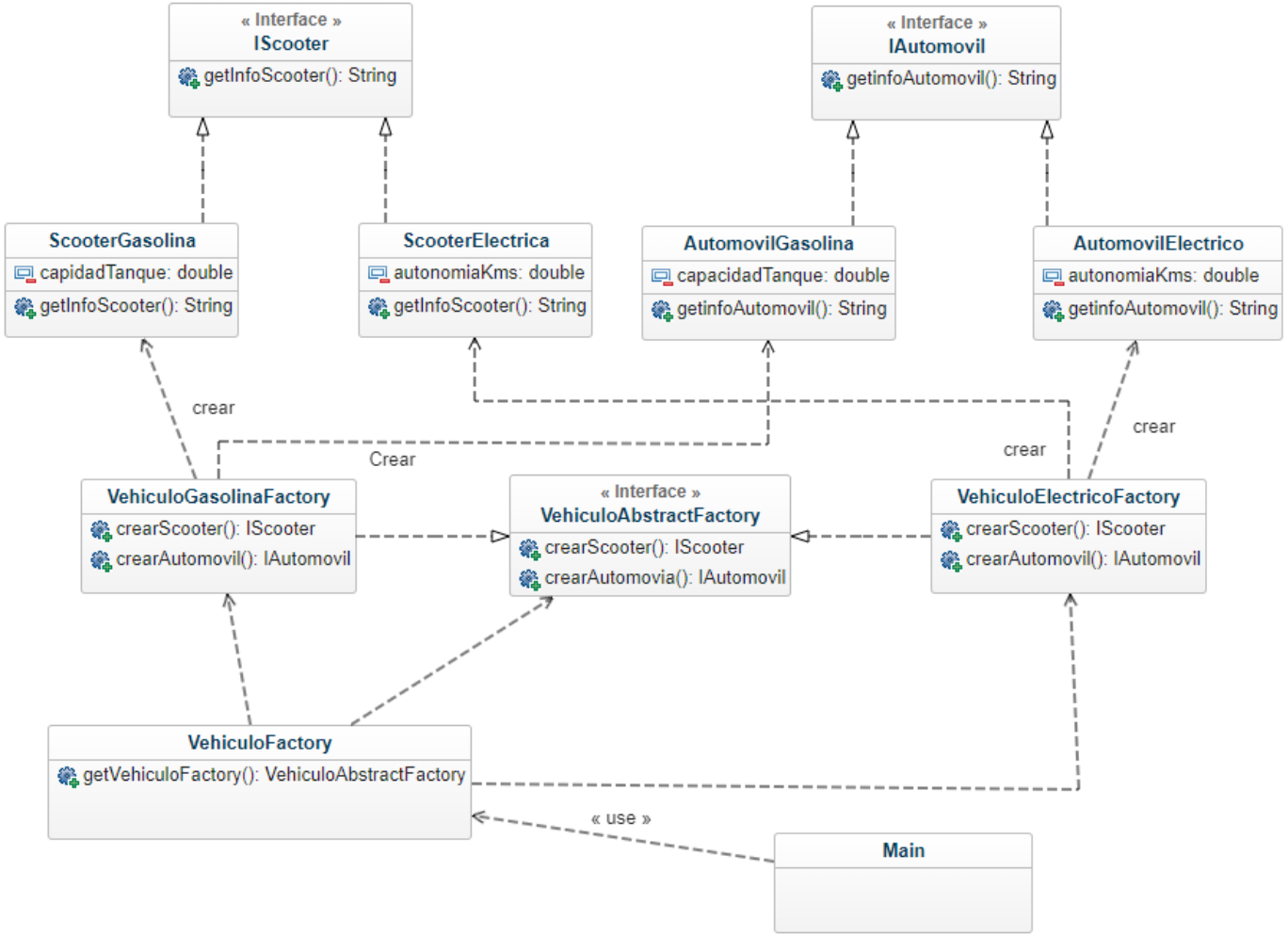
Ejemplo:

Suponga un sistema de venta de vehículos que gestiona vehículos que funcionan con gasolina y vehículos eléctricos.

Actualmente maneja dos tipos de producto: Scotter y Automovil, de los cuales, encontramos en sus dos variedades (gasolina y eléctricos)

Desde una clase principal se requiere crear instancias de objetos concretos de cualquiera de estas dos familias de producto.

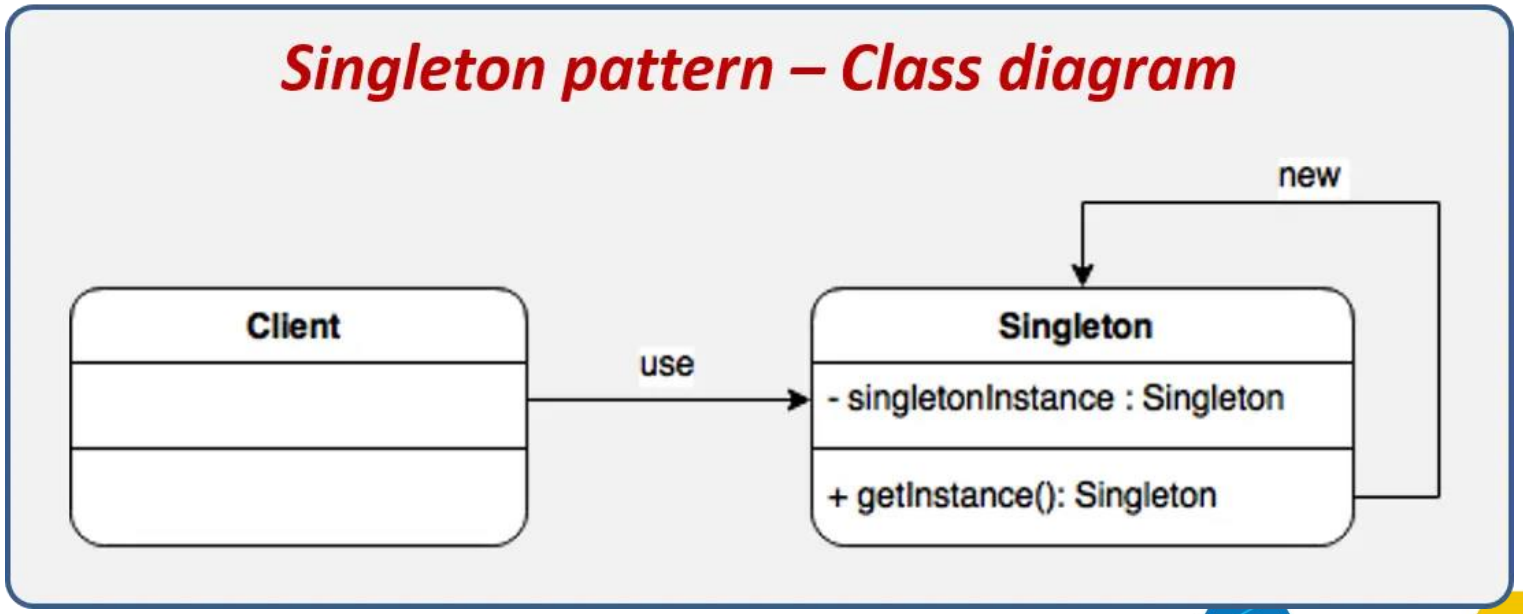
Se quiere permitir la creación de vehículos a Diesel



UNIVERSIDAD Popular del cesar

Patrones creacionales – Singleton

Permite
asegurarnos de que una clase tenga una única instancia,
a la vez que proporciona un punto de acceso global a dicha instancia.



Es el patrón mas pequeño en cuanto al numero de clases requeridas

Debe tener un solo constructor privado

Debe tener una referencia static a si mismo, para el almacenamiento global de la única instancia

UNIVERSIDAD Popular del cesar

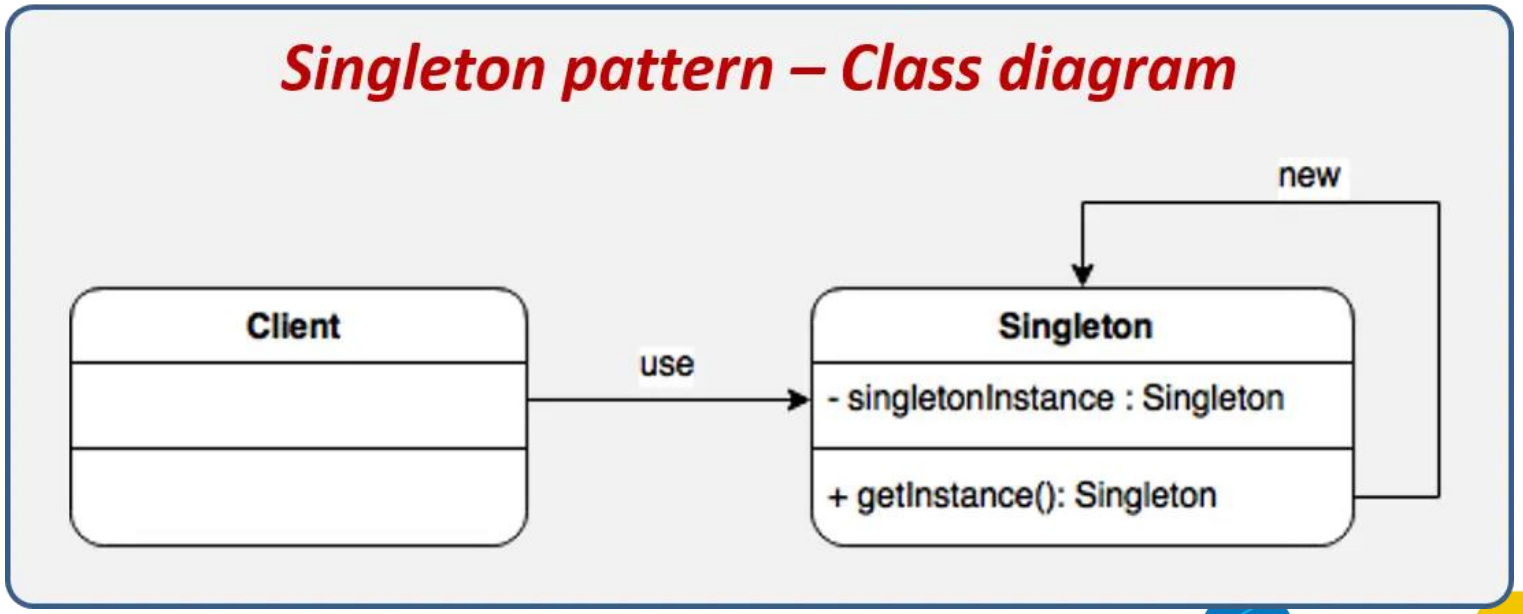
Patrones creacionales – Singleton

Client: Componente en el cual se desea obtener la instancia de la clase Singleton

Singleton: Clase que implementa el patrón singleton. De ella solo se permitirá la creación de una única instancia.

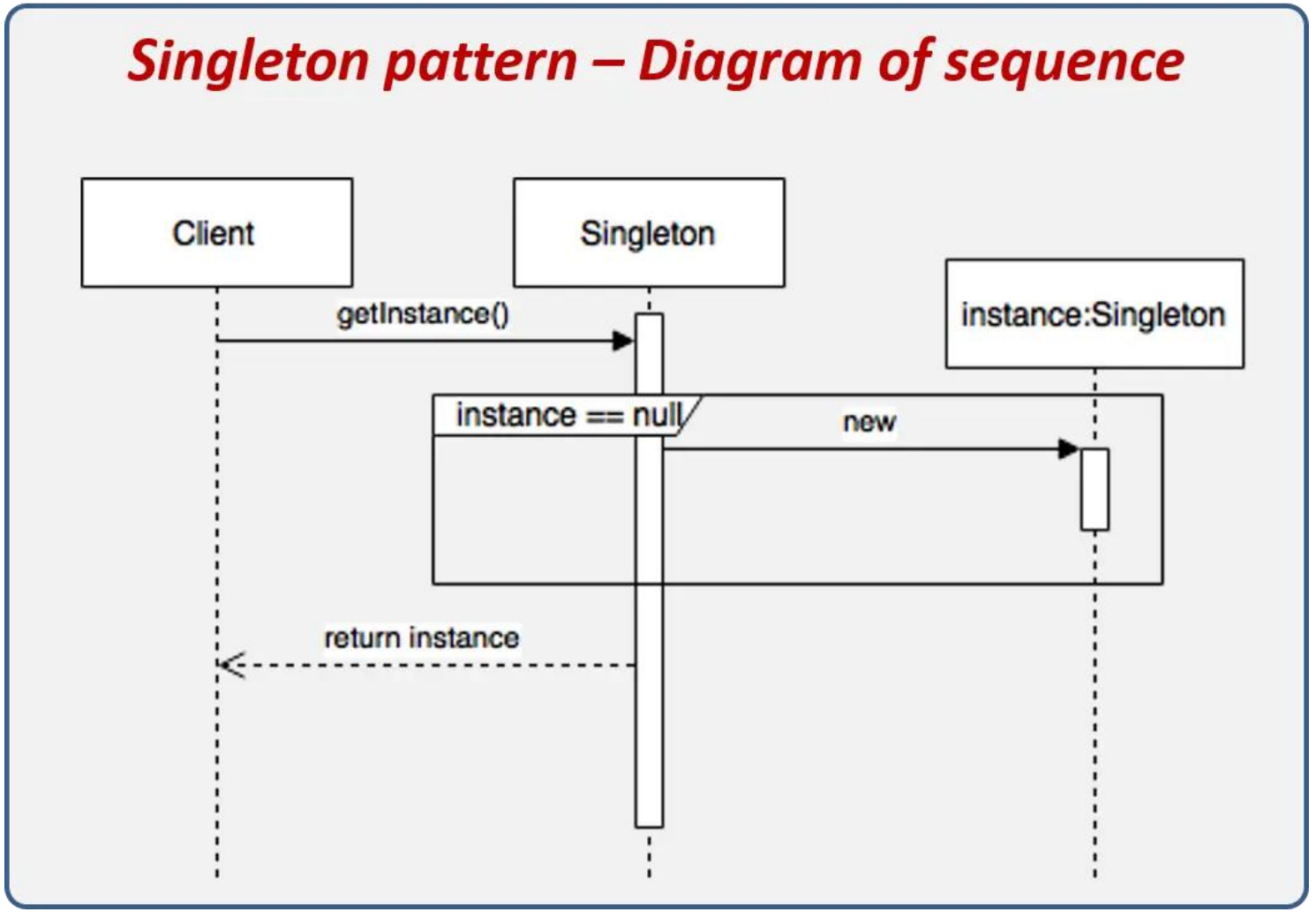
Cuando implementarlo:

Cuando se requiere tener solo una instancia en toda la aplicación y se requiere un acceso global a la misma



UNIVERSIDAD Popular del cesar

Patrones creacionales – Singleton

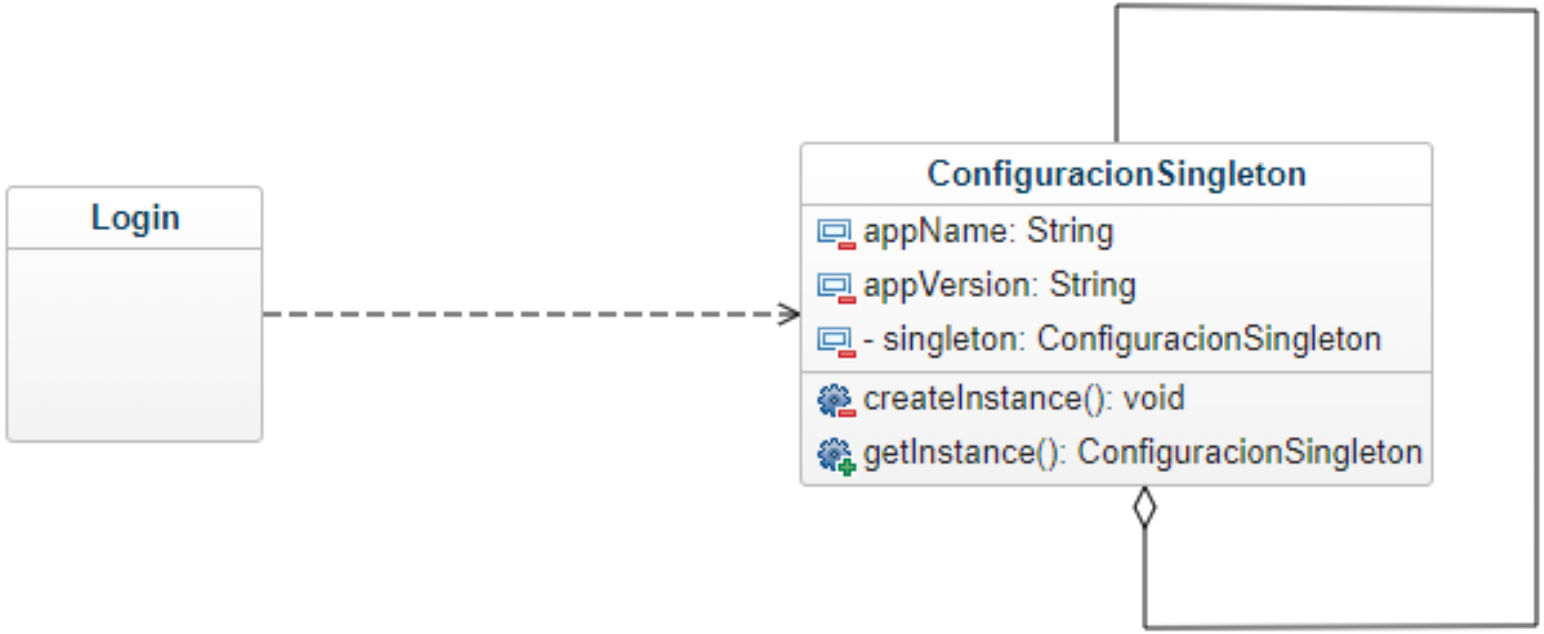


UNIVERSIDAD Popular del cesar

Patrones creacionales – Singleton

Ejemplo:

Supóngase que usted mantiene una aplicación, la cual, luego del proceso de validación de usuario, deberá cargar información de configuración, esta información deberá estar disponible para toda la aplicación, inclusive para cada uno de los módulos que esta pueda tener.

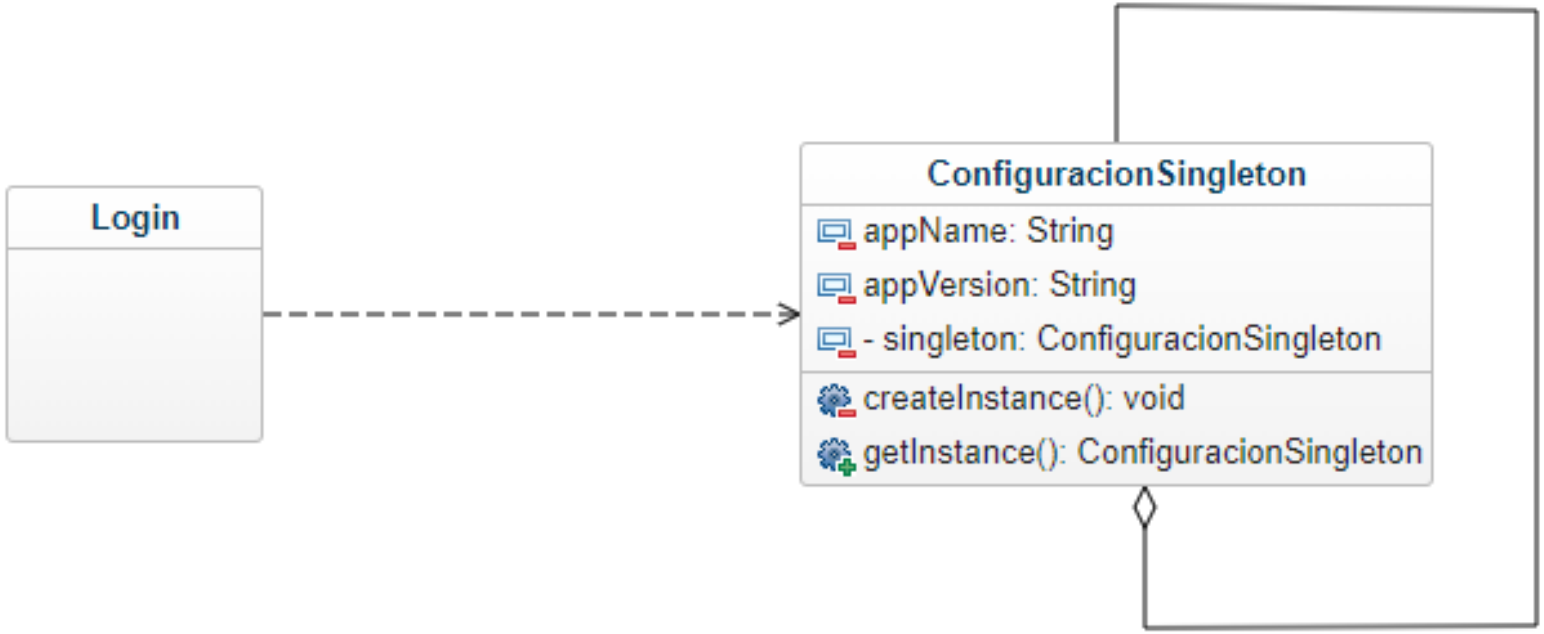


UNIVERSIDAD Popular del cesar

Patrones creacionales – Singleton

Ejemplo:

Supóngase que usted mantiene una aplicación, la cual, luego del proceso de validación de usuario, deberá cargar información de configuración, esta información deberá estar disponible para toda la aplicación, inclusive para cada uno de los módulos que esta pueda tener.



Crear un Singleton con la información de la sesión iniciada

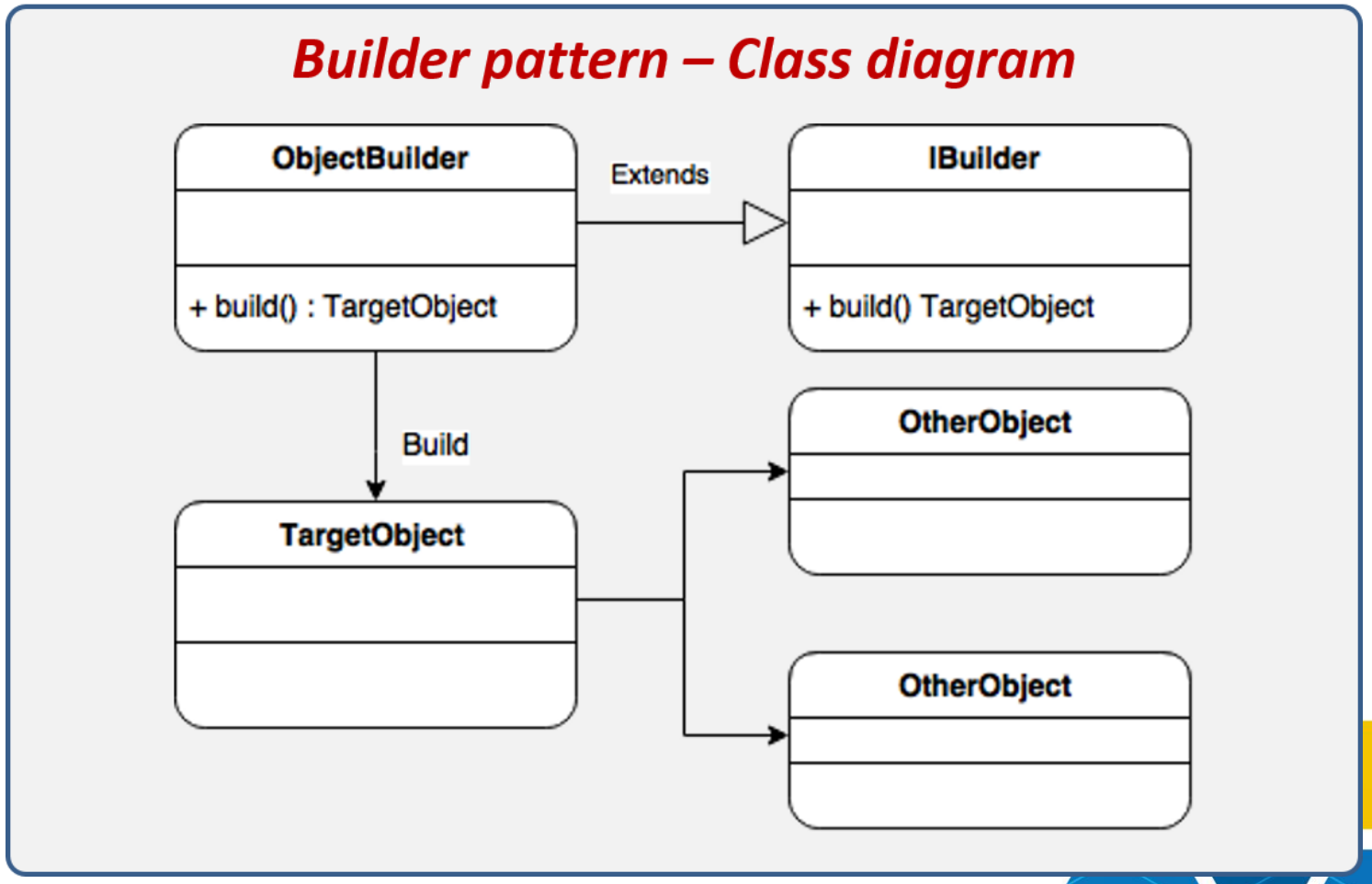
UNIVERSIDAD Popular del cesar

Patrones creacionales – Builder

Permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

Muy útil cuando se requiere crear objetos compuestos de forma manual y repetida.

Se debe establecer cada propiedad y si esta compuesto por otros objetos, primero hay que crear esos objetos y luego ser asignados al objeto que se esta construyendo



UNIVERSIDAD Popular del cesar

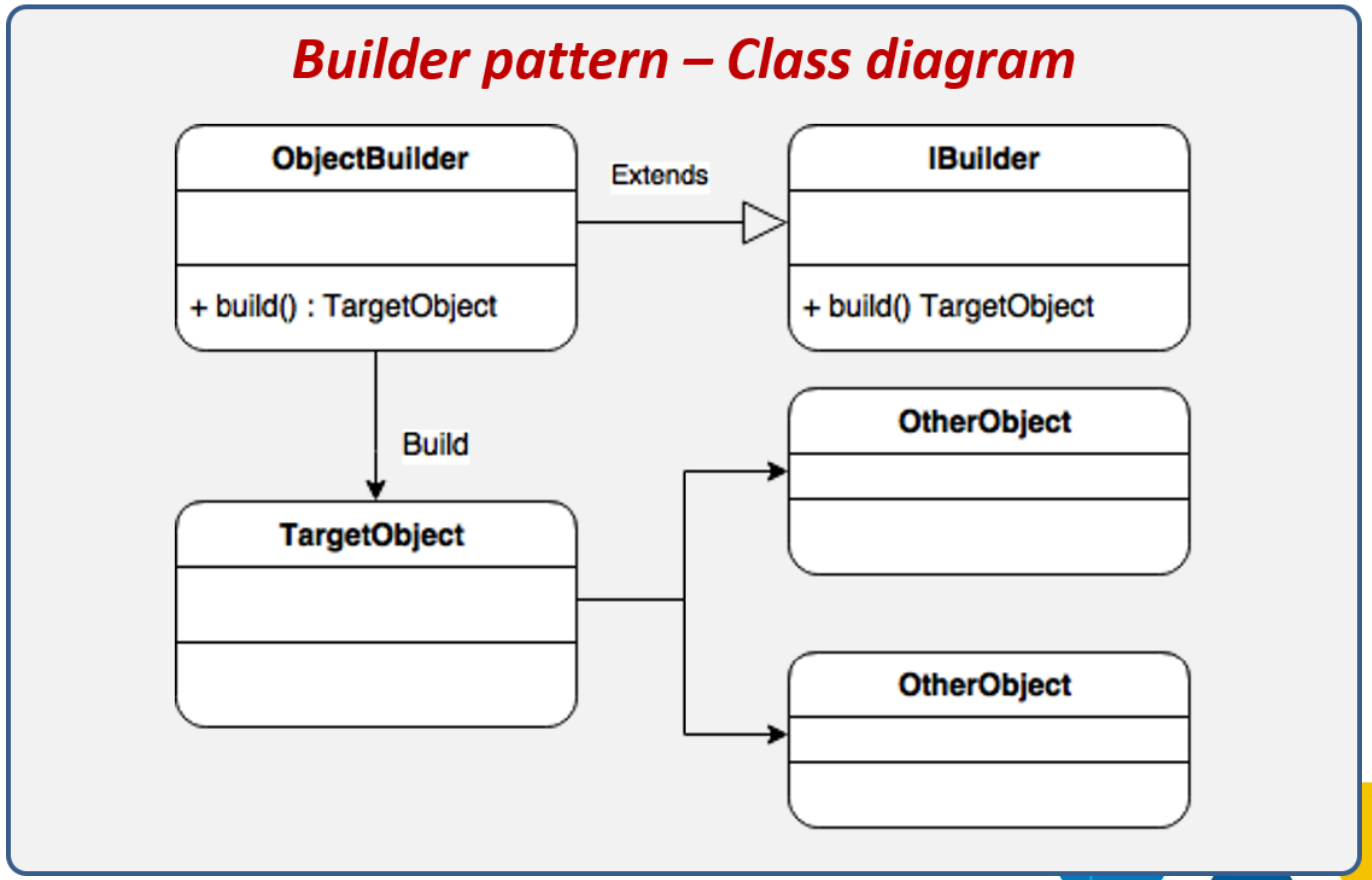
Patrones creacionales – Builder

IBuilder: Componente opcional, especifica una interface común que tendrán todos los Builder, puede ser una interface que defina únicamente el método build.

ObjectBuilder: Implementación de **IBuilder**, es la clase que utilizaremos para crear los **TarjetObjet**. Como regla general todos los métodos de esta clase retornan a si mismo con la finalidad de agilizar la creación, esta clase por lo general es creada como una clase interna del **TargetObject**.

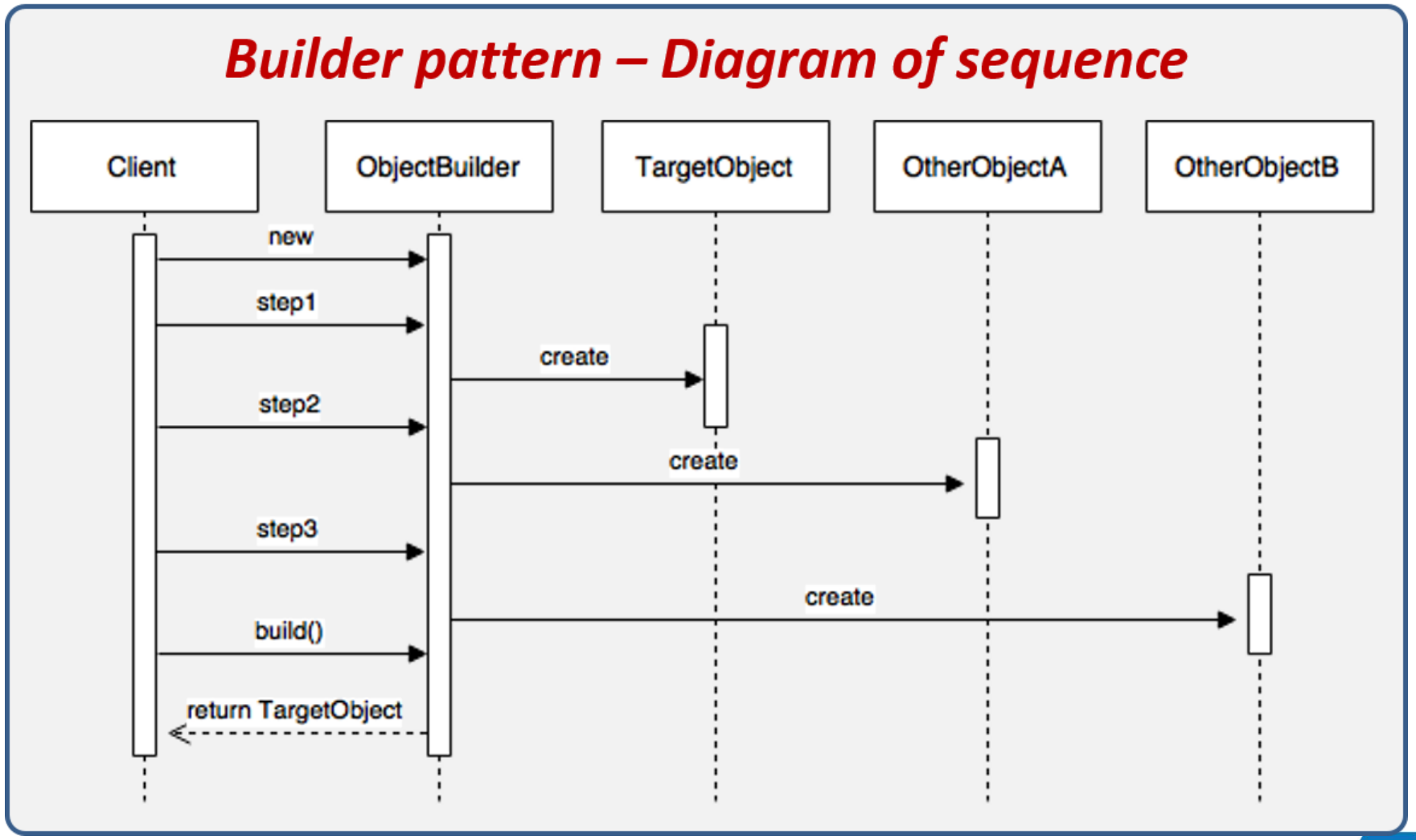
TarjetObjet: Representa el objeto que deseamos crear mediante el **ObjectBuilder**, ésta puede ser una clase simple o puede ser una clase muy compleja que tenga dentro más objetos.

OtherObjets: Representa los posibles objetos que deberán ser creados cuando el **TarjetObject** sea construido por el **ObjectBuilder**. Son objetos que hacen parte del **TarjetObject**



UNIVERSIDAD Popular del cesar

Patrones creacionales – Builder

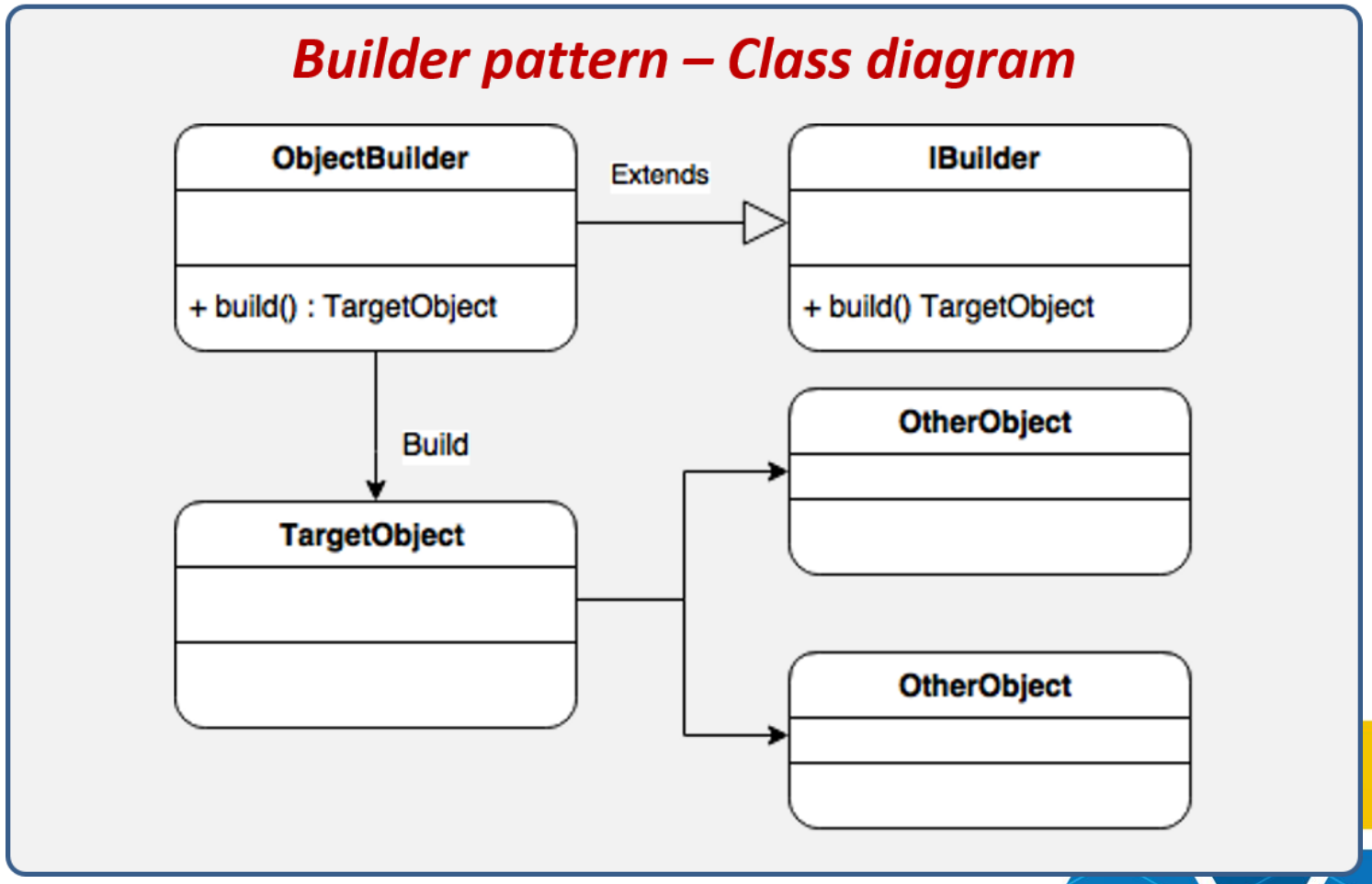


UNIVERSIDAD Popular del cesar

Patrones creacionales – Builder

Cuando utilizar Builder:

- Cuando necesitamos un mecanismo simple para la creación de objetos complejos
- Cuando necesitamos crear repetidamente objetos complejos
- Cuando necesitamos ocultar la complejidad de la creación de un objeto determinado



UNIVERSIDAD Popular del cesar

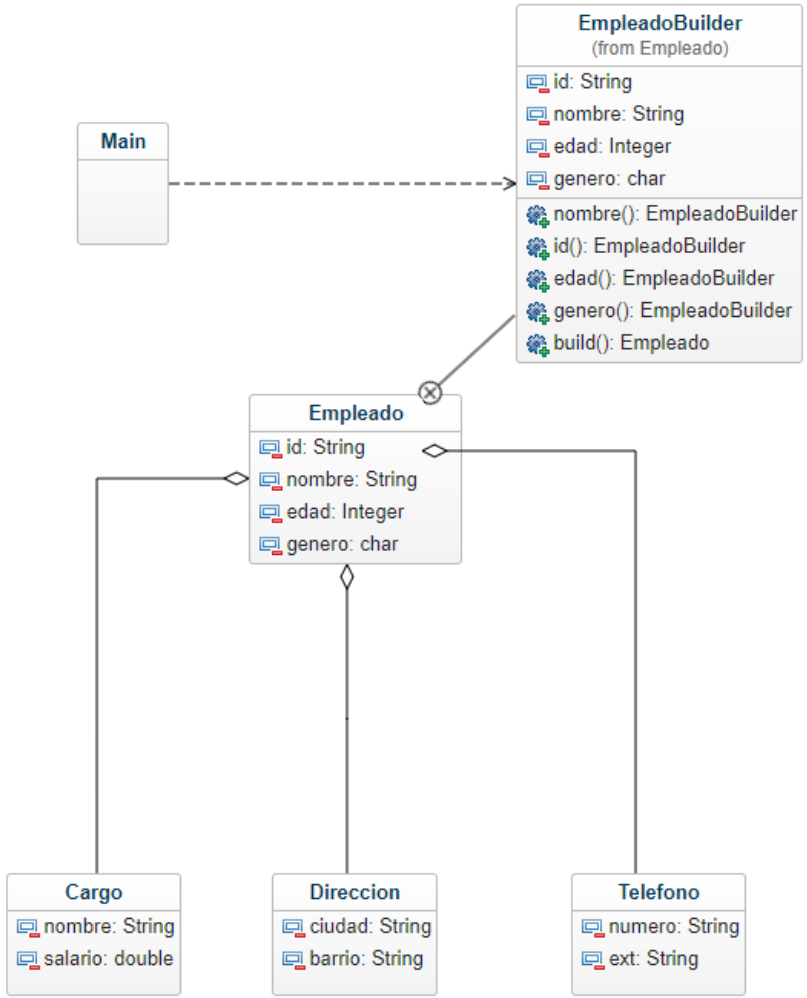
Patrones creacionales – Builder

Ejemplo:

Supóngase que en una aplicación de empleados, se requiere frecuentemente la creación de empleados.

Teniendo en cuenta que un empleado además de sus datos básico (id, nombre, edad y genero), también mantiene información sobre su Dirección(que tiene ciudad y barrio), Teléfono (que tienen numero y extensión), y Cargo(que tiene nombre y salario).

Se requiere una forma simple de crear objetos de tipo empleado, ocultando la de su creación.



UNIVERSIDAD Popular del cesar

Patrones creacionales – Builder

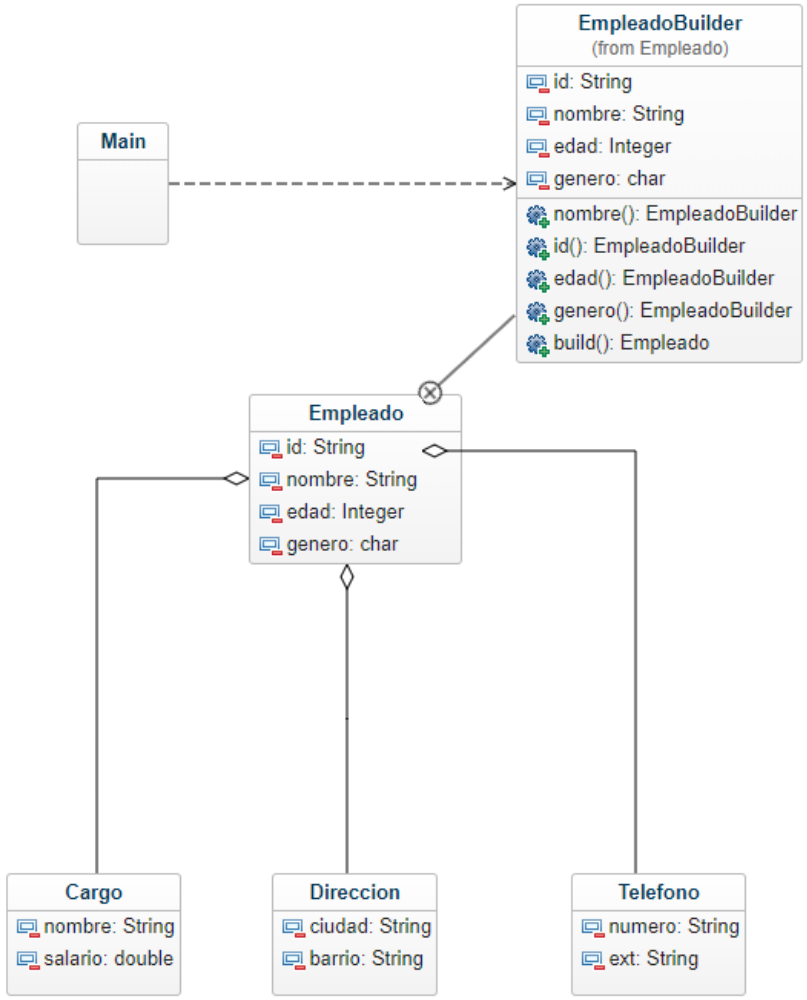
Ejemplo:

Supóngase que en una aplicación de empleados, se requiere frecuentemente la creación de empleados.

Teniendo en cuenta que un empleado además de sus datos básico (id, nombre, edad y genero), también mantiene información sobre su Dirección(que tiene ciudad y barrio), Teléfono (que tienen numero y extensión), y Cargo(que tiene nombre y salario).

Se requiere una forma simple de crear objetos de tipo empleado, ocultando la de su creación.

El empleado además debe incluir la información de la empresa (nit y razón social)



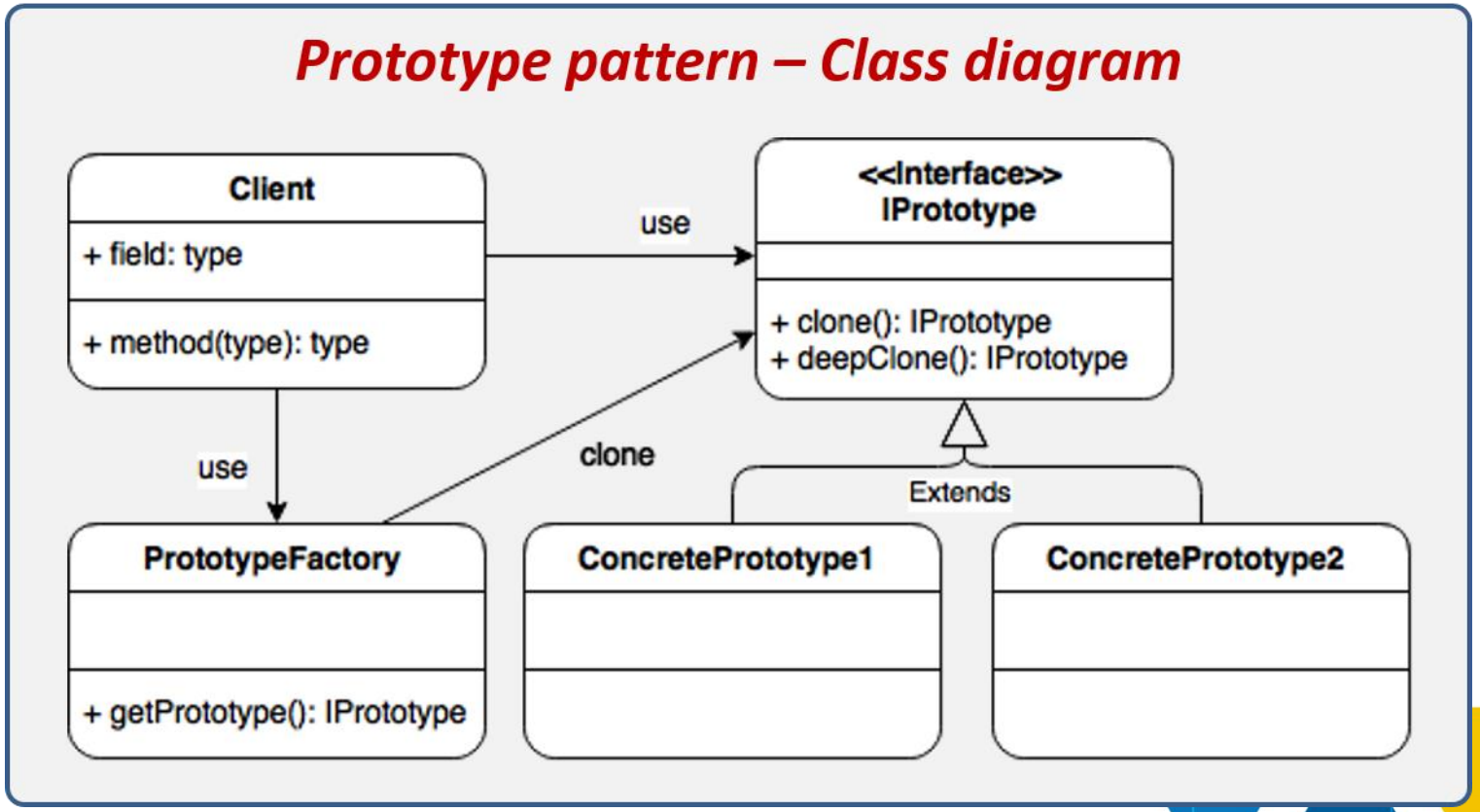
UNIVERSIDAD Popular del cesar

Patrones creacionales – Prototype

El objetivo de este patrón es la creación de nuevos objetos mediante duplicación de objetos existentes llamados prototipos que disponen de la capacidad de clonación.

Muy útil cuando se necesita crear objetos basados en otros ya existentes

También ayuda a ocultar la estrategia utilizada para clonar un objeto



UNIVERSIDAD Popular del cesar

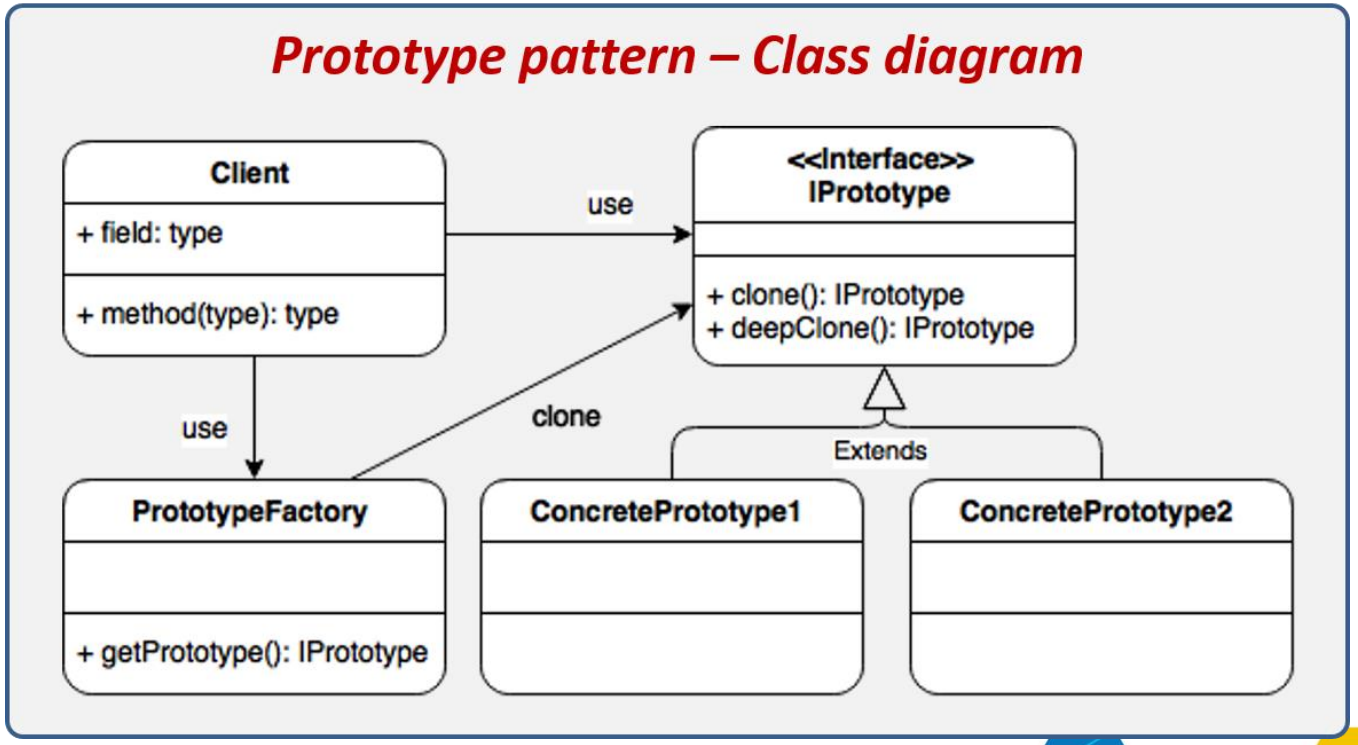
Patrones creacionales – Prototype

Client: Componente que interactúa con los prototipos.

IPrototype: Interface que define los atributos mínimos de un prototipo, esta interface debe contar por lo menos con alguno de los dos tipos de clonación. Clonación superficial (clone) o clonación en profundidad (deepClone) los cuales explicaremos más adelante.

ConcretePrototype: Implementaciones concretas de IPrototype los cuales podrán ser clonados.

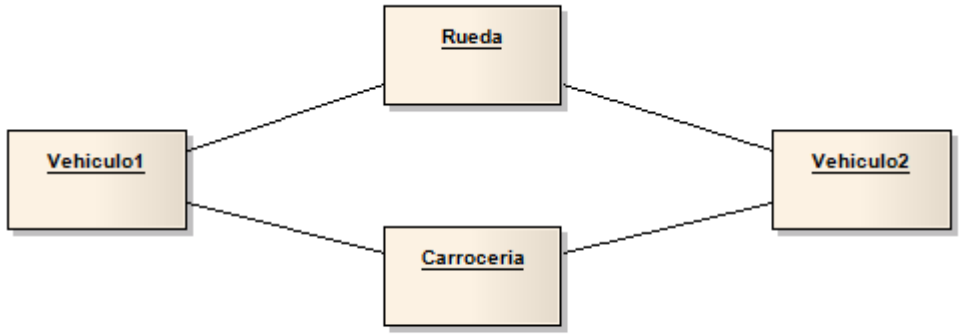
PrototypeFactory: Componente que utilizaremos para mantener el cache de los prototipos existentes, así como para crear clonaciones de los mismos.



UNIVERSIDAD Popular del cesar

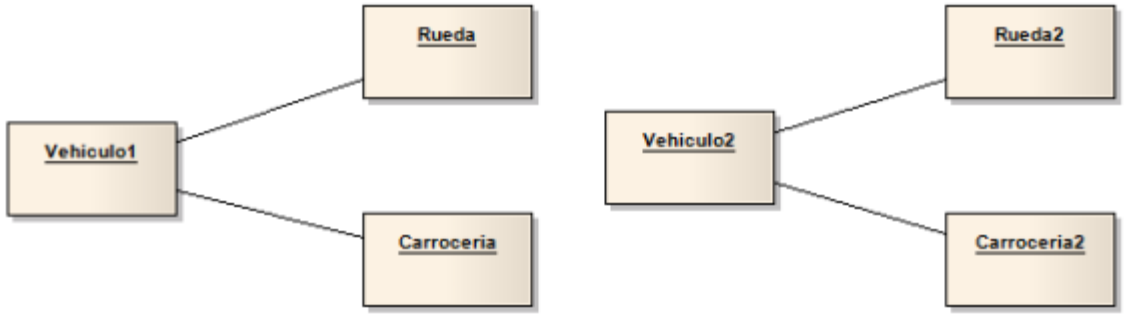
Patrones creacionales – Prototype

Clonación superficial:
Se crea una copia del objeto principal, pero todos los objetos internos no se clonan. Si no que son compartidos.



Clonación superficial

Clonación profunda:
Se realiza una copia idéntica del prototipo, incluyendo todos los objetos que este contenga



Clonación profunda

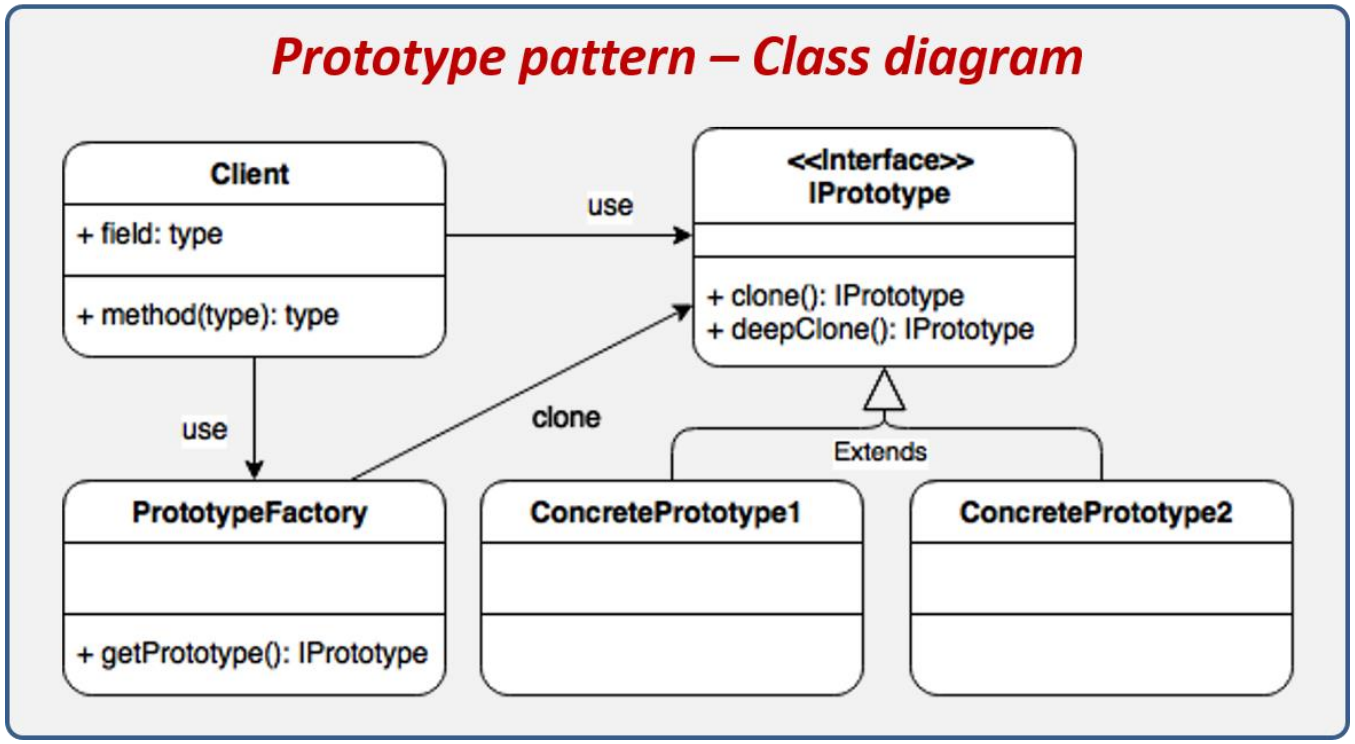
UNIVERSIDAD Popular del cesar

Patrones creacionales – Prototype

Cuando utilizarlo:

Cuando se necesitan crear nuevos objetos a partir de objetos existentes

Cuando se tiene gran cantidad de objetos con atributos repetidos, siempre es mucho mas rápido clonar que crear nuevos objetos y setear cada valor.



UNIVERSIDAD Popular del cesar

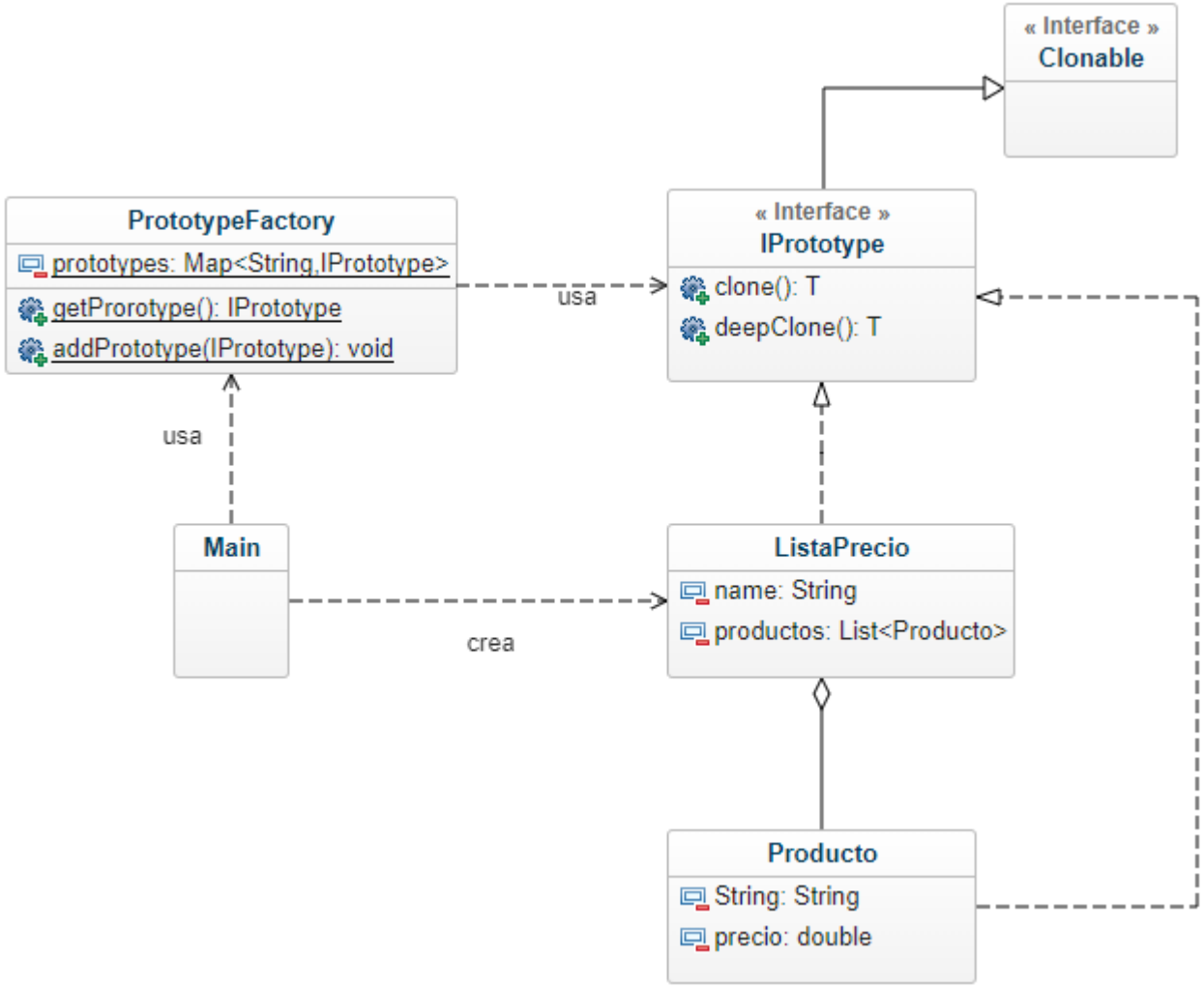
Patrones creacionales – Prototype

Ejemplo:

Supóngase que en una empresa de ventas de producto tiene un catalogo de los productos que ofrece, cada producto tiene un nombre y un precio. La lista de productos es extensa.

Sin embargo, a los **clientes mayoristas** se les hace un 10% de descuento sobre el listado original y los **clientes VIP** a los que se les hace un 10% de descuento sobre la lista Mayorista.

Se sugiere crear tres listas de precios con los mismos productos, para sus diferentes Clientes.



UNIVERSIDAD Popular del cesar

Patrones creacionales – Prototype

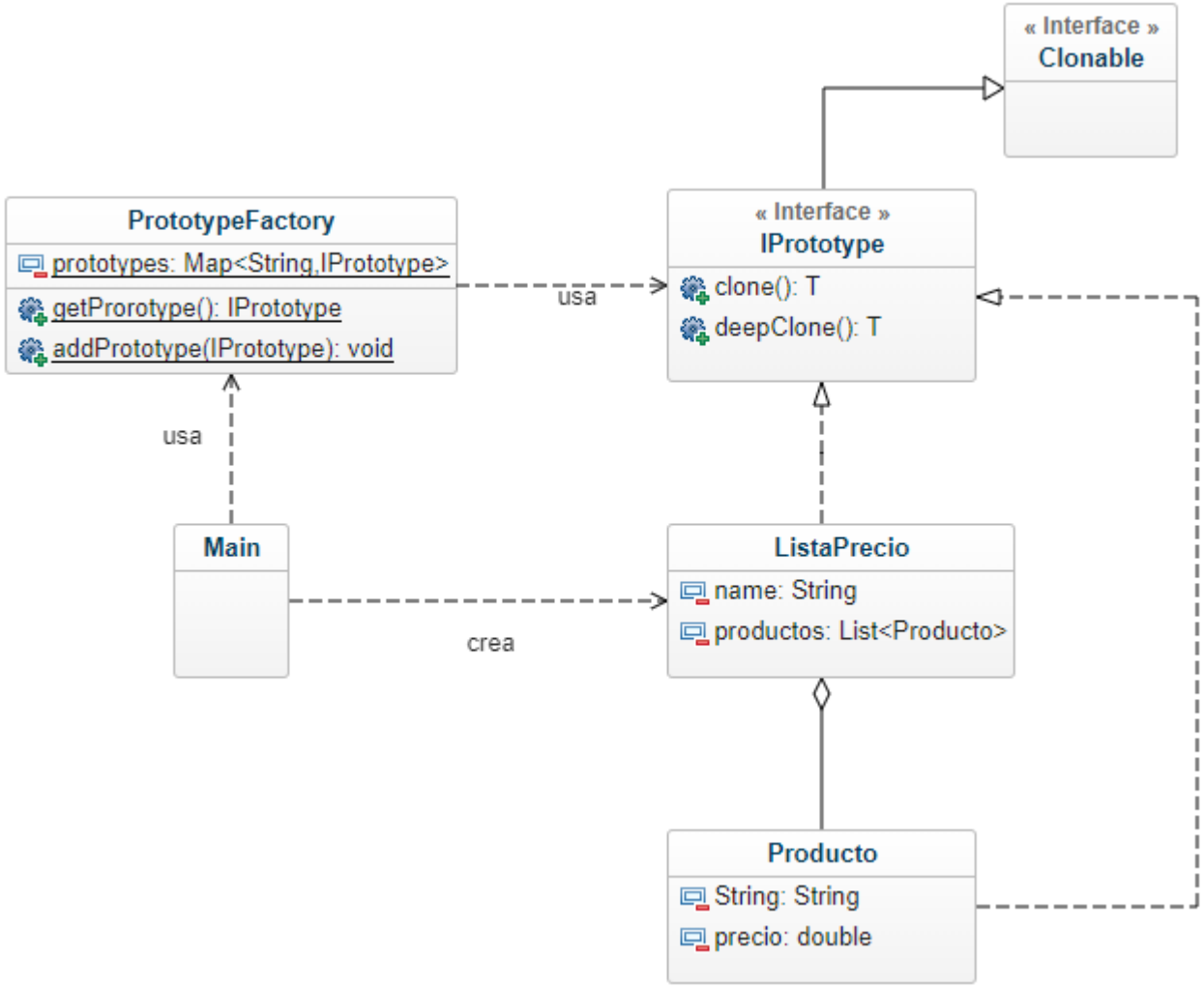
Ejemplo:

Supóngase que en una empresa de ventas de producto tiene un catalogo de los productos que ofrece, cada producto tiene un nombre y un precio. La lista de productos es extensa.

Sin embargo, a los **clientes mayoristas** se les hace un 10% de descuento sobre el listado original y los **clientes VIP** a los que se les hace un 10% de descuento sobre la lista Mayorista.

Se sugiere crear tres listas de precios con los mismos productos, para sus diferentes Clientes.

Cada producto mantiene un proveedor (id, nombre) que también se desea manejar como prototype



UNIVERSIDAD Popular del cesar

Patrón de diseño – Recursos

Libros:

- ***"Patrones de diseño. Elementos de software orientado a objetos reutilizables"***
  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

- ***"Sumérgete en los patrones de diseño"***
  Alexander Shvets

- ***"Introducción a los patrones de diseño. Un enfoque practico"***
  Oscar Blancharte

- ***"Patrones de diseño en java. Los 23 modelos de diseño: descripción y soluciones ilustradas "***
  Lauren Debrauwer

UNIVERSIDAD
Popular del cesar