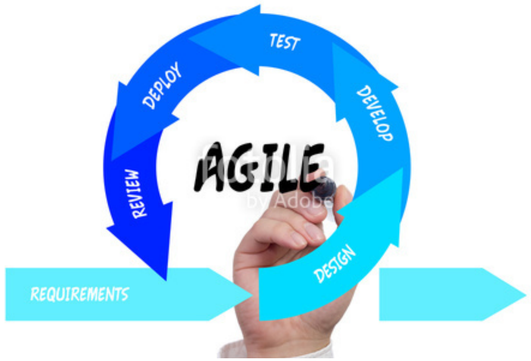


Classe Problemes Setmana 11: Disseny: Patrons de Disseny

Anna Puig

Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona
Curs 2021/22

UNIVERSITAT DE BARCELONA

Temari

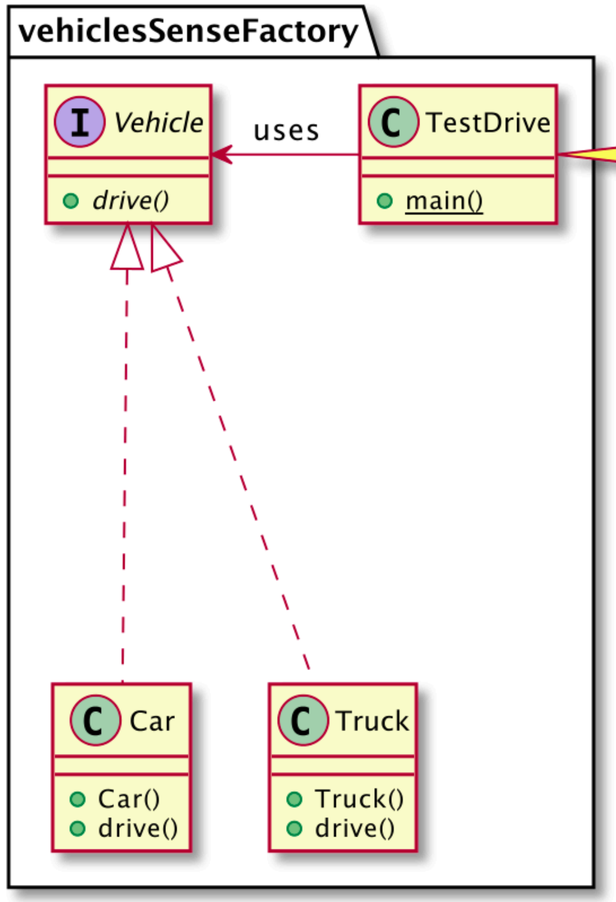
1. Introducció al procés de desenvolupament del software
2. Anàlisi de requisits i especificació
3. Disseny
   - 3.1 Introducció
   - 3.2 Patrons arquitectònics
   - 3.3 Criteris de Disseny: G.R.A.S.P.
   - 3.4 Principis de Disseny: S.O.L.I.D.
   - 3.5 **Patrons de Disseny**
4. Del disseny a la implementació
5. Ús de frameworks de testing

3.4. Patrons de disseny

Propòsit → **Àmbit ↓**	**CREACIÓ**	**ESTRUCTURA**	**COMPORTAMENT**
CLASSE	• **Factory method**	• class Adapter	• Interpreter • Template method
OBJECTE	• Abstract Factory • Builder • Prototype • Singleton • Object pool	• Object Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsability • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

Patró Simple Factory

El problema




```
public class TestDrive {
    public static void main(String[] args) {

        System.out.println("Quin tipus de vehicle vols?");

        String name;
        Scanner teclado = new Scanner(System.in);
        name = teclado.nextLine();
        Vehicle vehicle = null;

        switch (name) {
            case "CAR":
                vehicle = new Car();
                break;
            case "TRUCK":
                vehicle = new Truck();
                break;
        }
        vehicle.drive();

    }
}
```

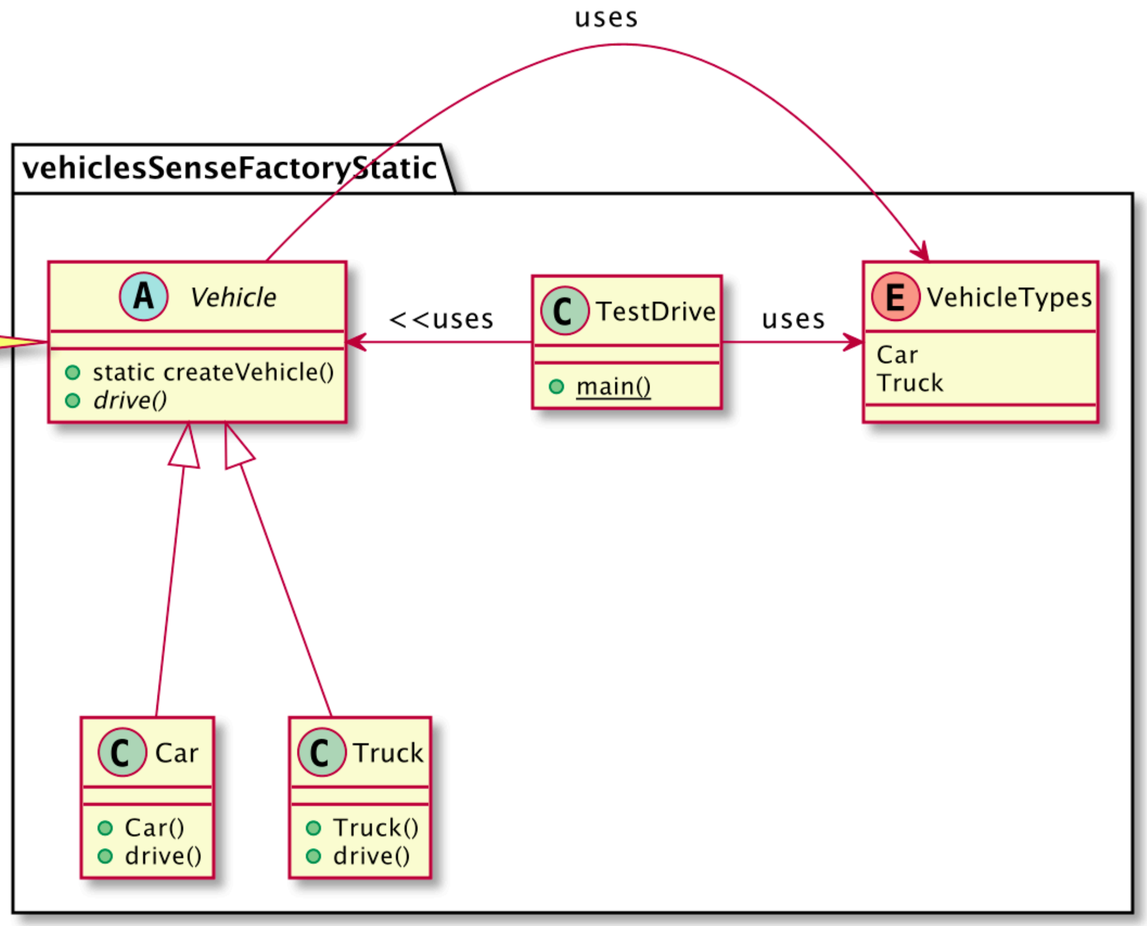
- Vulnera el principi **?** de S.O.L.I.D.

Patró Factory

Una primera solució?

```
public static Vehicle createVehicle (VehicleTypes name) {
        Vehicle vehicle = null;
        switch (name) {
            case Car:
                vehicle = new Car();
                break;
            case Truck:
                vehicle = new Truck();
                break;
        }
        return vehicle;

    }
    public abstract void drive();
```



- Vulnera el principi **?** de S.O.L.I.D.

Patrons Factory

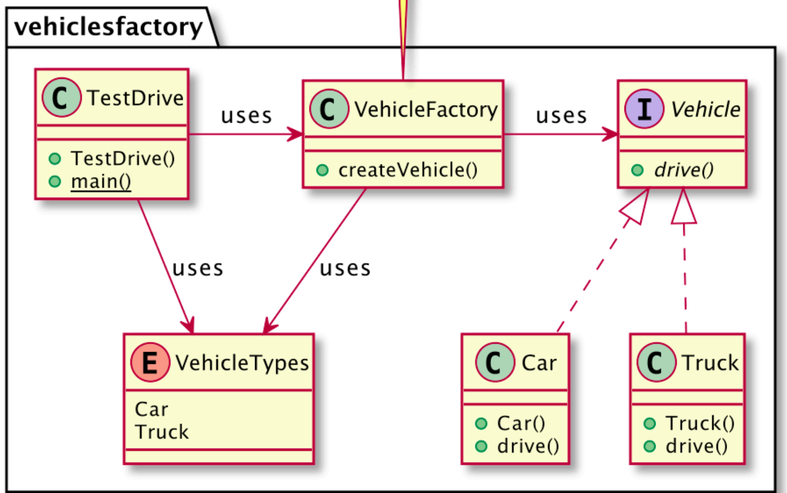
- **Simple Factory** – Defineix una classe per crear objectes i consulta el nou objecte creat a través d'una interfície comú dels objectes creats

Patró Simple Factory

Primera aproximació (versió simplificada Simple Factory)

```
public class VehicleFactory {
        /**
         * Method to create vehicle types
         * @param vehicleType
         * @return Vehicle
         */
        public Vehicle createVehicle(VehicleTypes vehicleType) {
                Vehicle vehicle = null;
                switch (vehicleType) {
                        case Car:
                                vehicle = new Car();
                                break;
                        case Truck:
                                vehicle = new Truck();
                                break;
                }
                return vehicle;

        }
}
```



SOLUCIÓ:

- Es separa el creador de les instàncies de la pròpia classe
- Les instàncies es creen en una classe Factoria, en aquest cas VehicleFactory

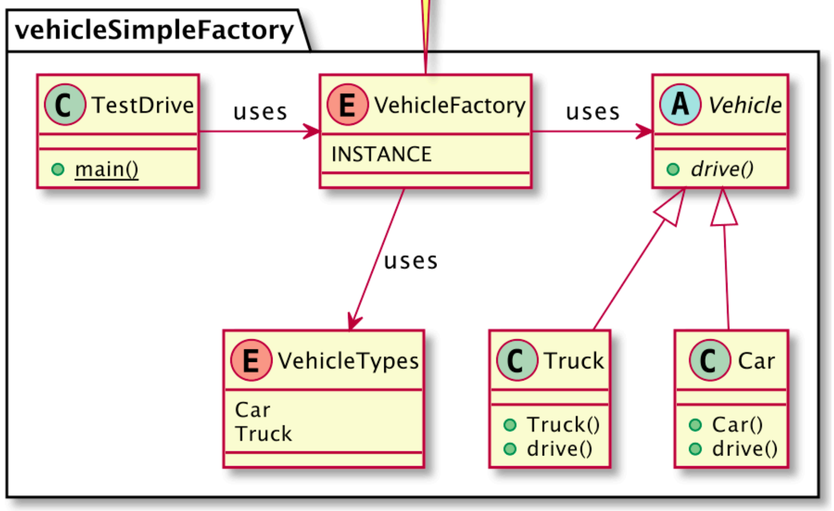
Patró Simple Factory

Primera aproximació (versió simplificada del Factory Method)

```
public enum VehicleFactory {
        INSTANCE;

        public Vehicle createVehicle (VehicleTypes name) {
                Vehicle vehicle = null;
                switch (name) {
                        case Car:
                                vehicle = new Car();
                                break;
                        case Truck:
                                vehicle = new Truck();
                                break;
                }
                return vehicle;

        }
}
```



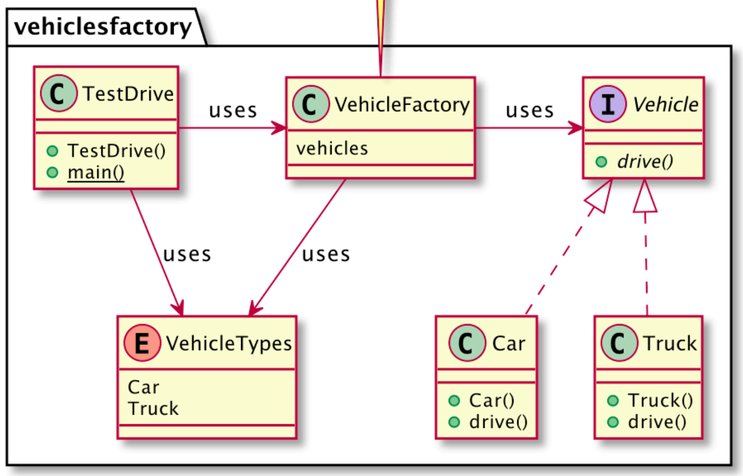
- Pot ser un Singleton?
- com solucionar el Open-Closed?

Patró Simple Factory

Primera aproximació (versió simplificada del Factory Method)

```
private Map<String, Vehicle> vehicles = new HashMap<String, Vehicle>();
    /**
     * Method to create vehicle types
     * @param vehicleType
     * @return Vehicle
     * @throws Exception
     */
    public Vehicle createVehicle(String vehicleType)
            throws Exception {
        Vehicle vehicle = vehicles.get(vehicleType);
        if (vehicle != null) {
            return vehicle;
        } else {
            try {
                String name = Vehicle.class.getPackage().getName();
                vehicle = (Vehicle) Class.forName(name+"."+vehicleType).newInstance();
                vehicles.put(vehicleType, vehicle);
                return vehicle;
            } catch (Exception e) {
                throw new Exception("The vehicle type is unknown!");
            }
        }

    }
```



- Com solucionar el Open Closed?

ús de reflexivitat

- com són les instàncies dels vehicles concrets?

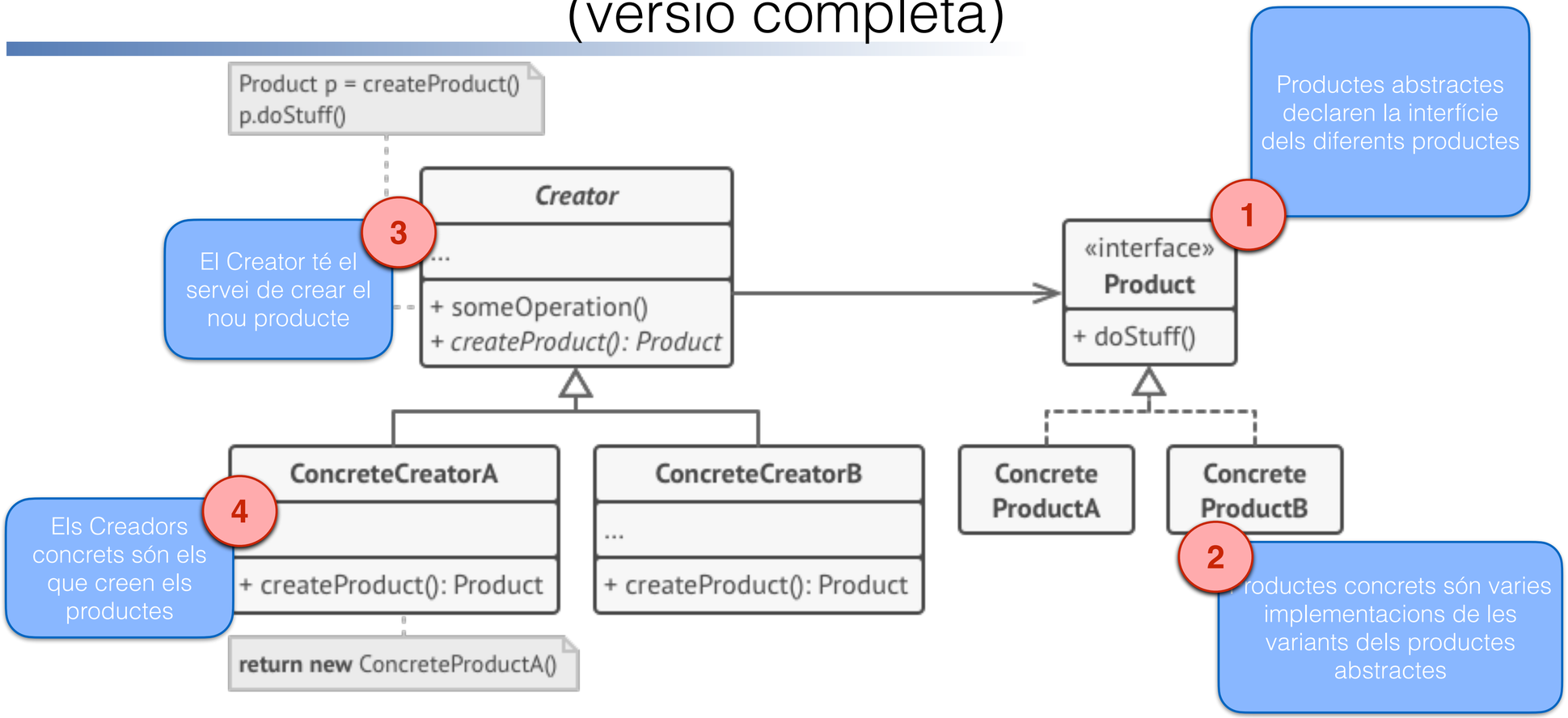
I si vull tenir diferents criteris per a construir les classes concretes?

Patrons Factory

- **Factory Method** – Defineix una classe abstracte per crear objectes, però deixa a les subclasses decidir quina classe ha d'instanciar i consulta el nou objecte creat a través d'una interfície comú dels objectes creats

Patró Factory Method

(versió completa)



Creator proporciona la signatura d'un mètode per crear els objectes.

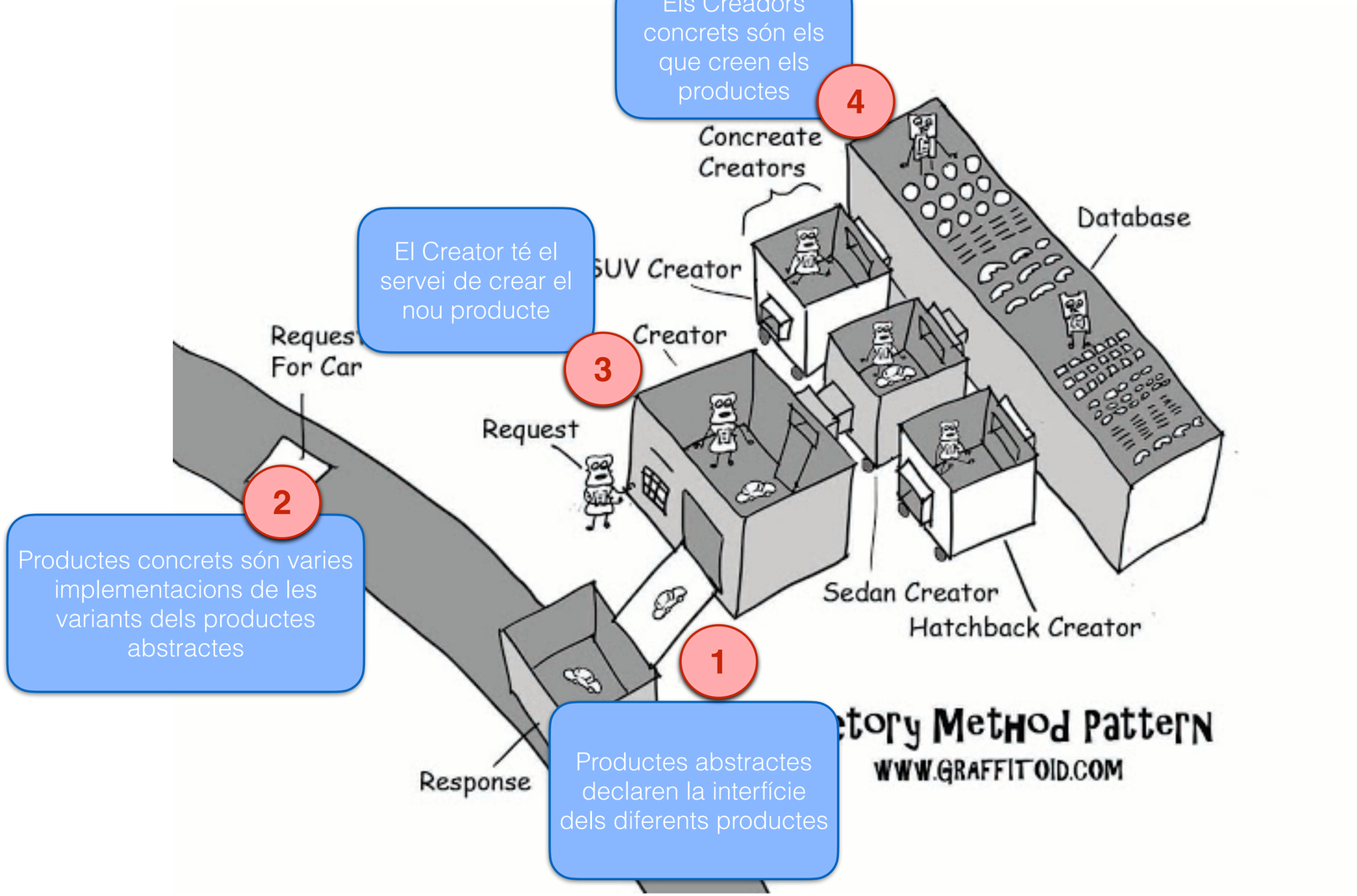
La resta de mètodes a la classe Creator són per operar amb els productes creats en el ConcreteCreator

Creator NO crea els objectes

ConcreteCreators creen els objectes de la jerarquia **Product**

Patró Factory Method

(versió completa)



Els Creadors concrets són els que creen els productes

4

El Creator té el servei de crear el nou producte

3

2

Productes concrets són varies implementacions de les variants dels productes abstractes

1

Productes abstractes declaren la interfície dels diferents productes

Patró Factory Method

Nom del patró: **Factory method**
Context: **Creació**

On s'usa en la realitat?

- A la JDK per exemple:
  - getInstance() de java.util.NumberFormat o ResourceBoundle
  - wrapper classes com Integer, Boolean, etc. per a retornar valors en usar el mètode valueOf()
  - java.nio.charset.Charset.forName(), java.sql.DriverManager#getConnection(), java.net.URL.openConnection(), java.lang.Class.newInstance(), java.lang.Class.forName()

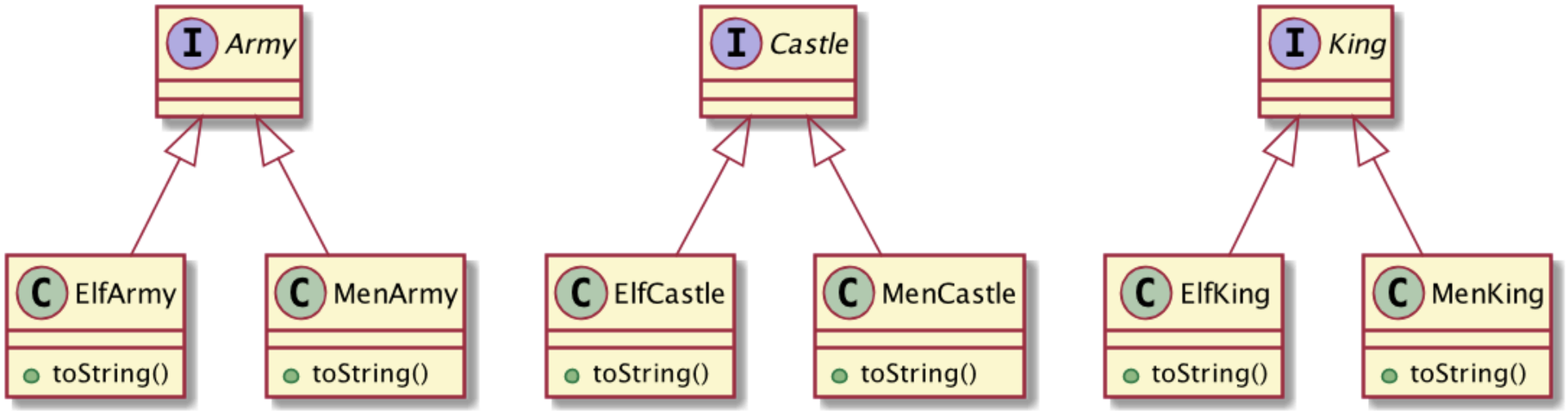
Patrons Factory

- **Factory Method** – Defineix una classe abstracte per crear objectes, però deixa a les subclasses decidir quina classe ha d'instanciar i consulta el nou objecte creat a través d'una interfície comú dels objectes creats

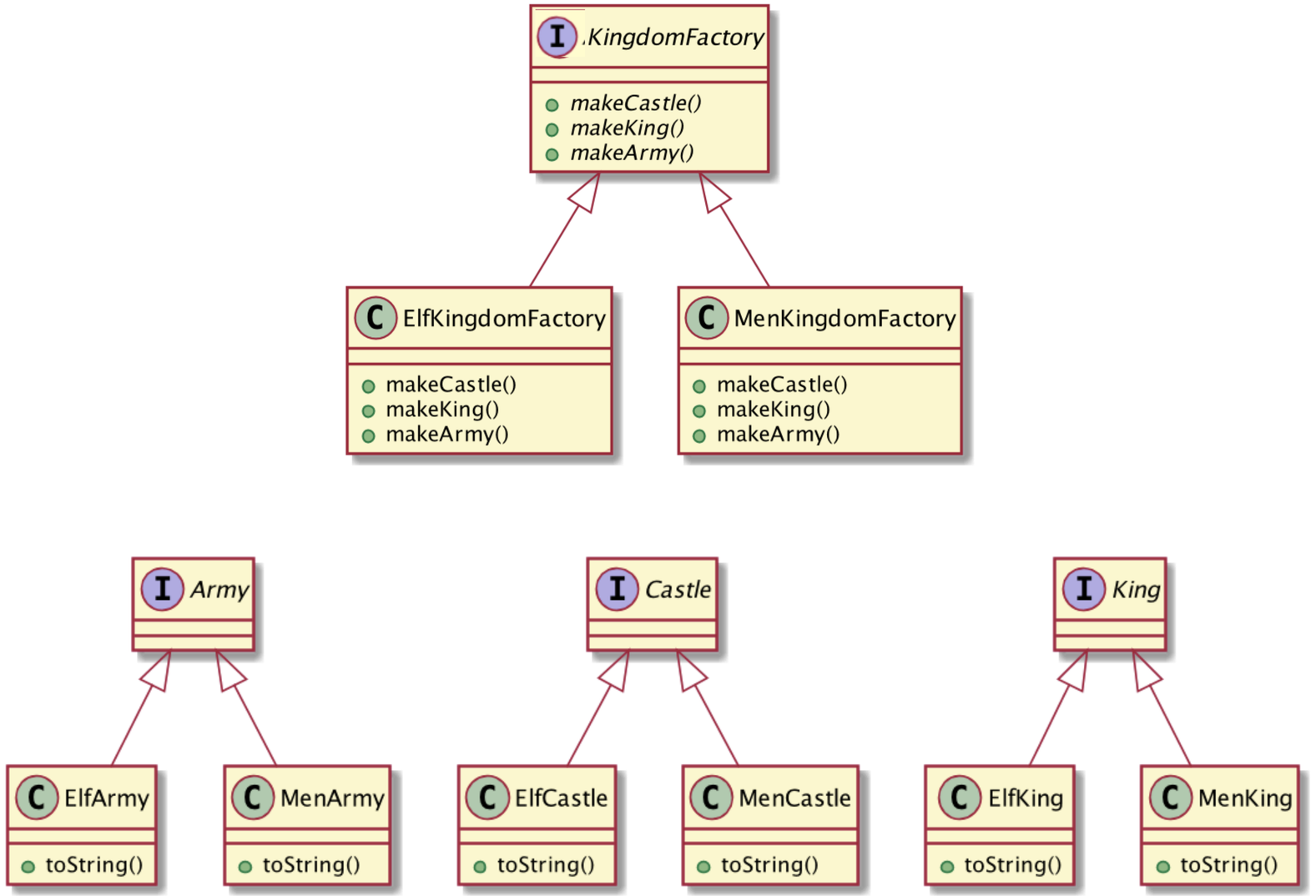
- **Abstract Factory** – Ofereix una interfície per crear una **família d'objectes** relacionats, sense explícitament especificar les seves classes

Exemple Patró Abstract Factory

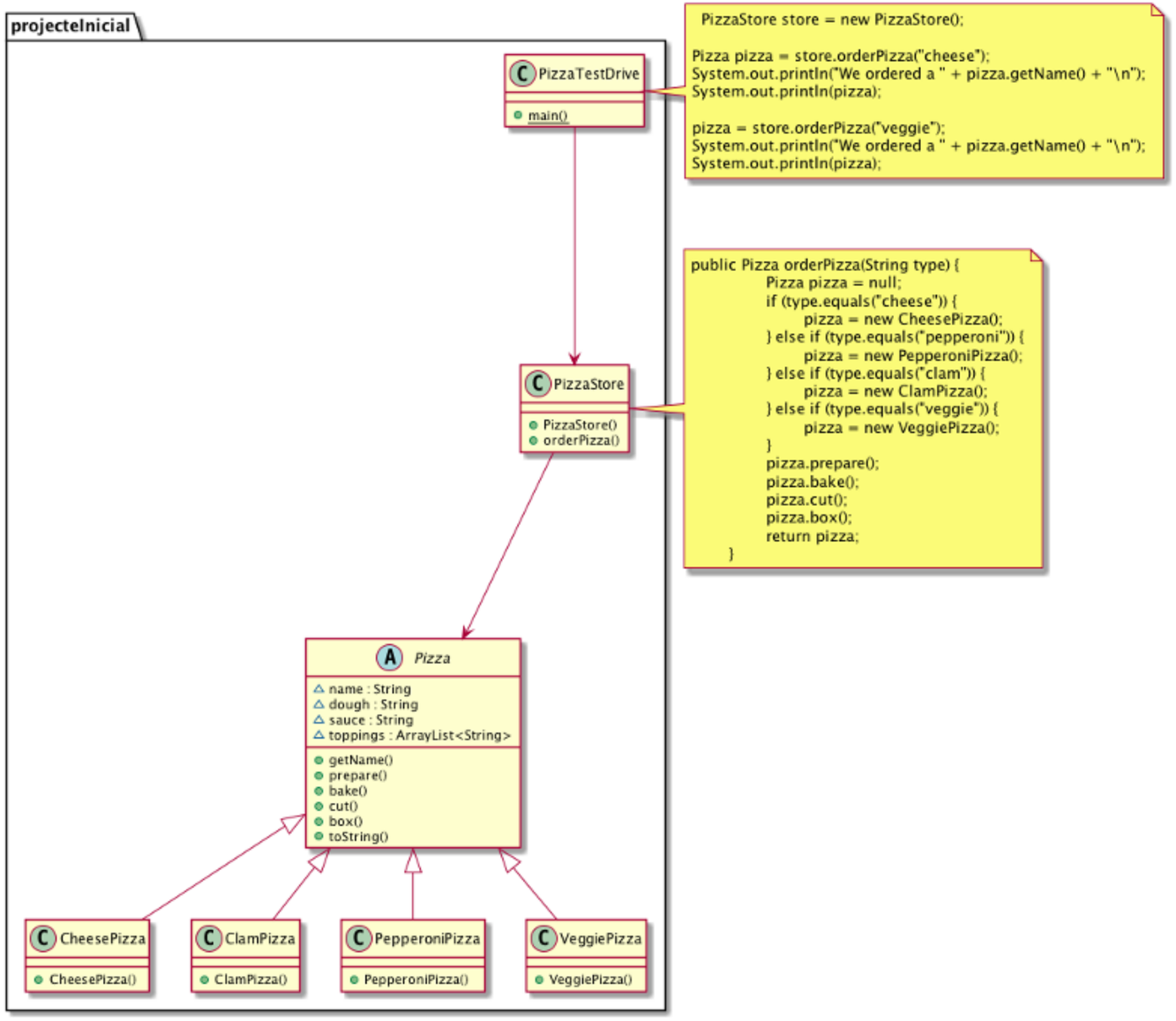
- Volem crear dos regnes (els dels elfs i els dels homes). Cada regne té un castell, un rei i una armada. Per a cada un dels elements d'un regne es dissenya una interfície
- Com solucionem la seva creació "coordinada"?



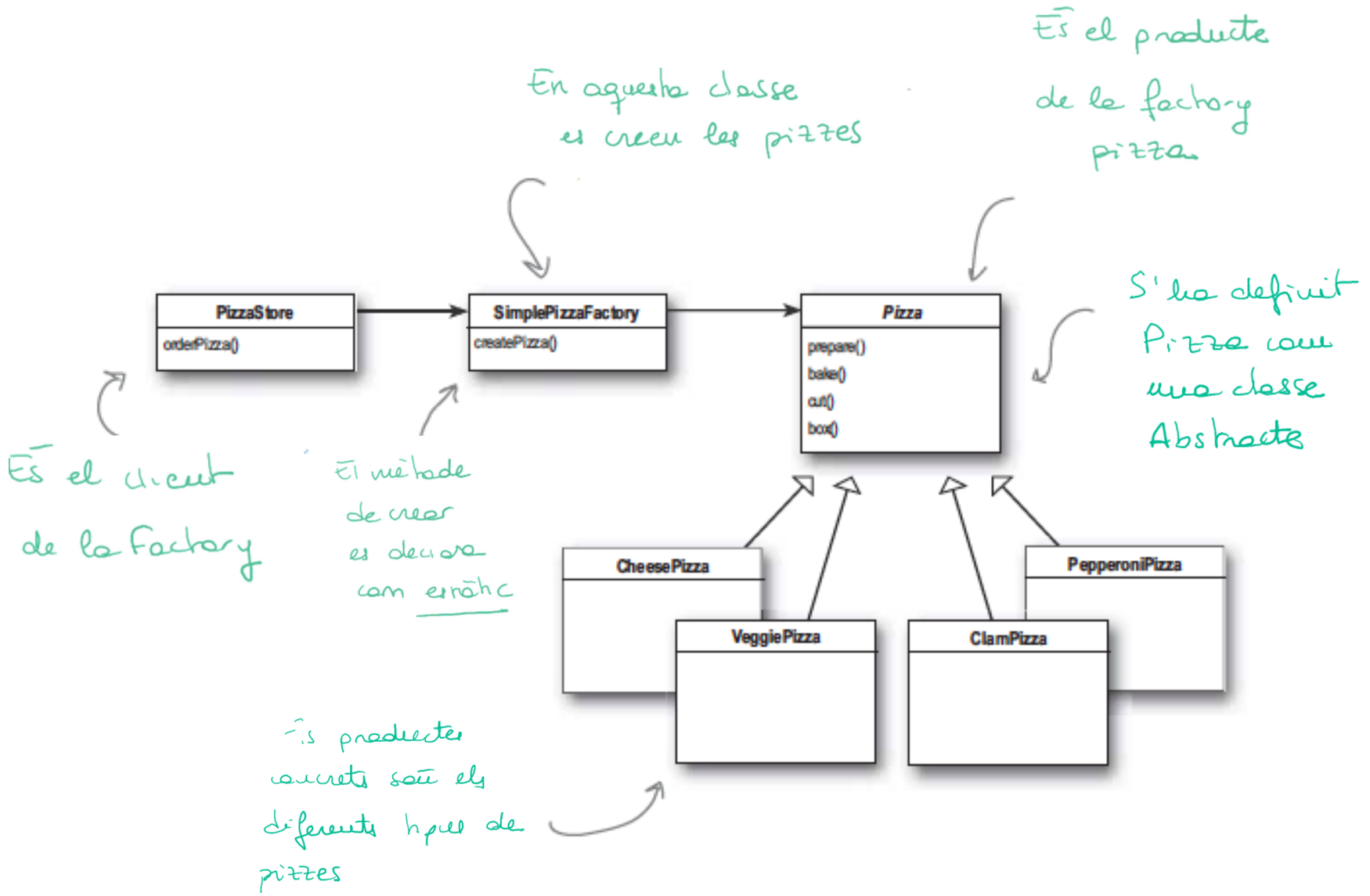
Exemple Patró Abstract Factory



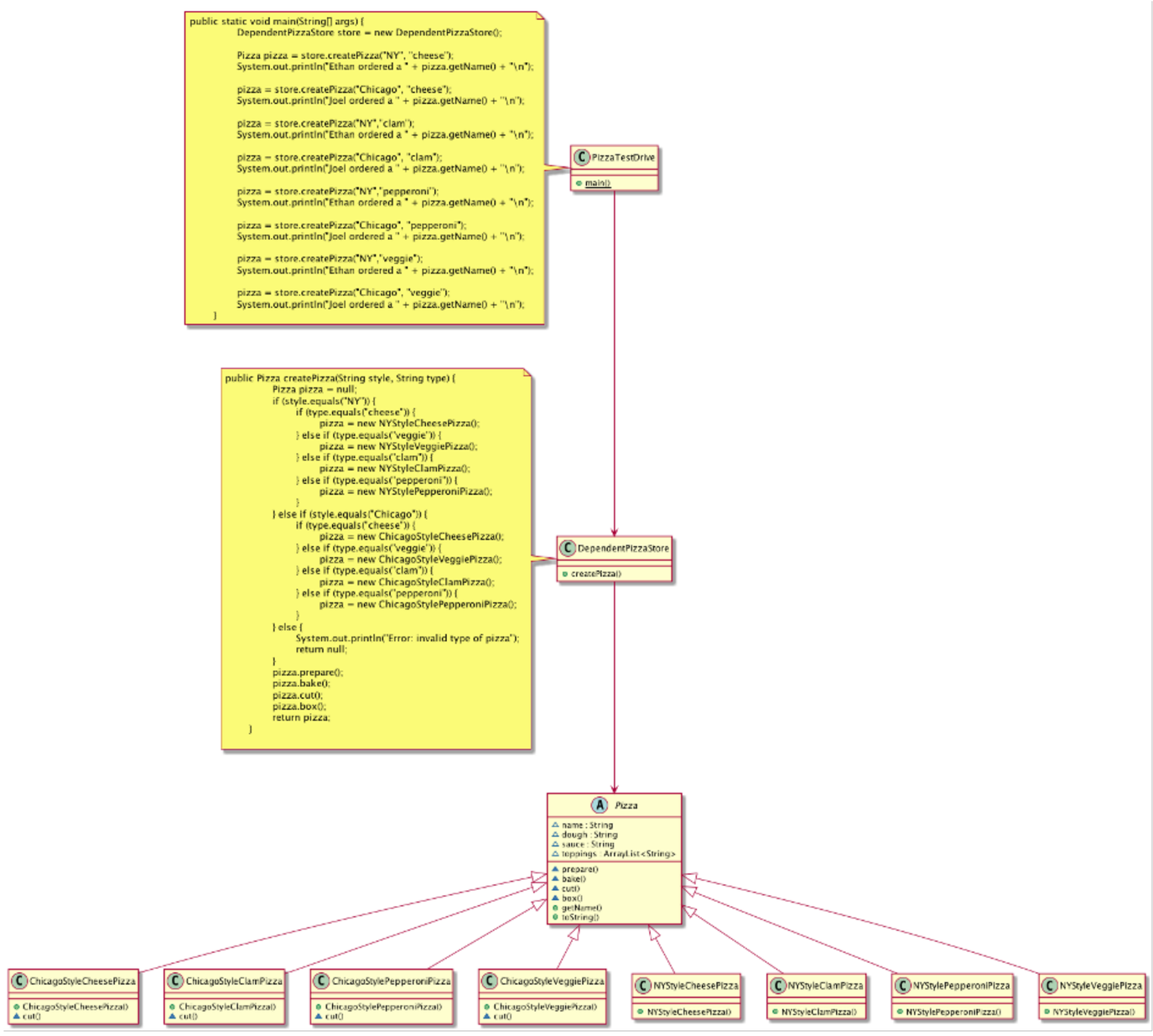
Exercici Pizza Store: Projecte inicial



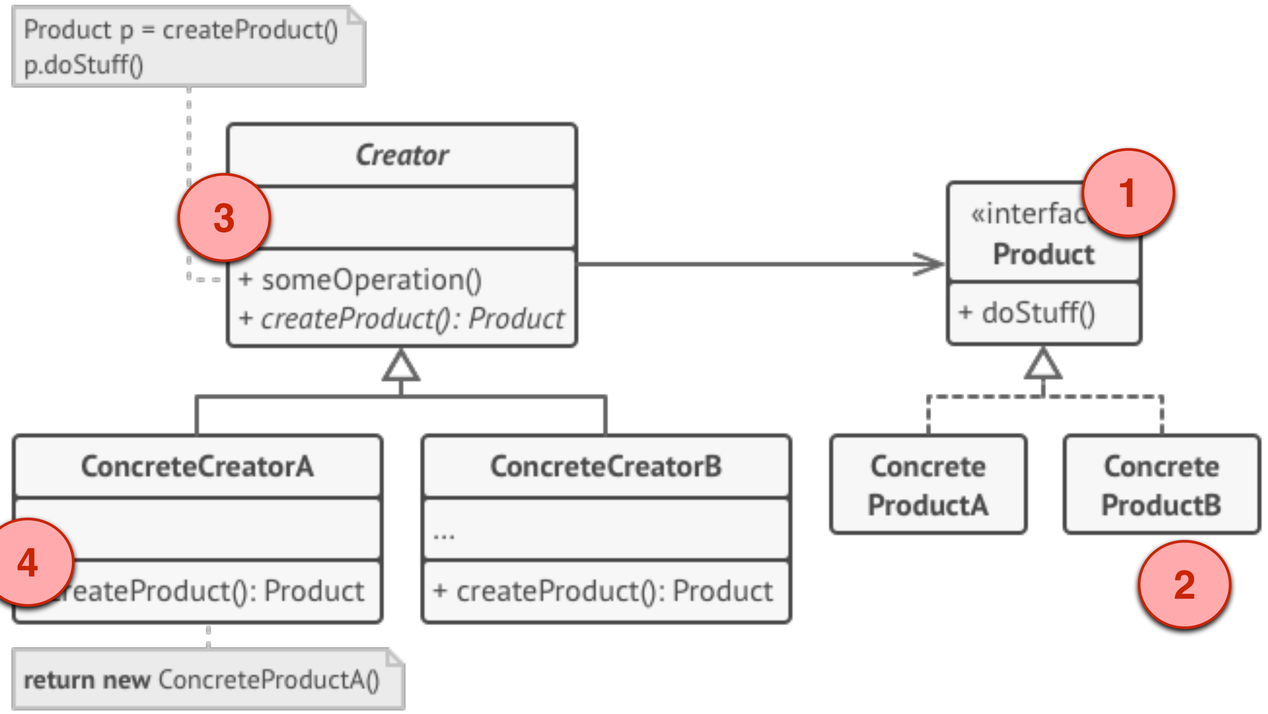
Exercici Pizza Store

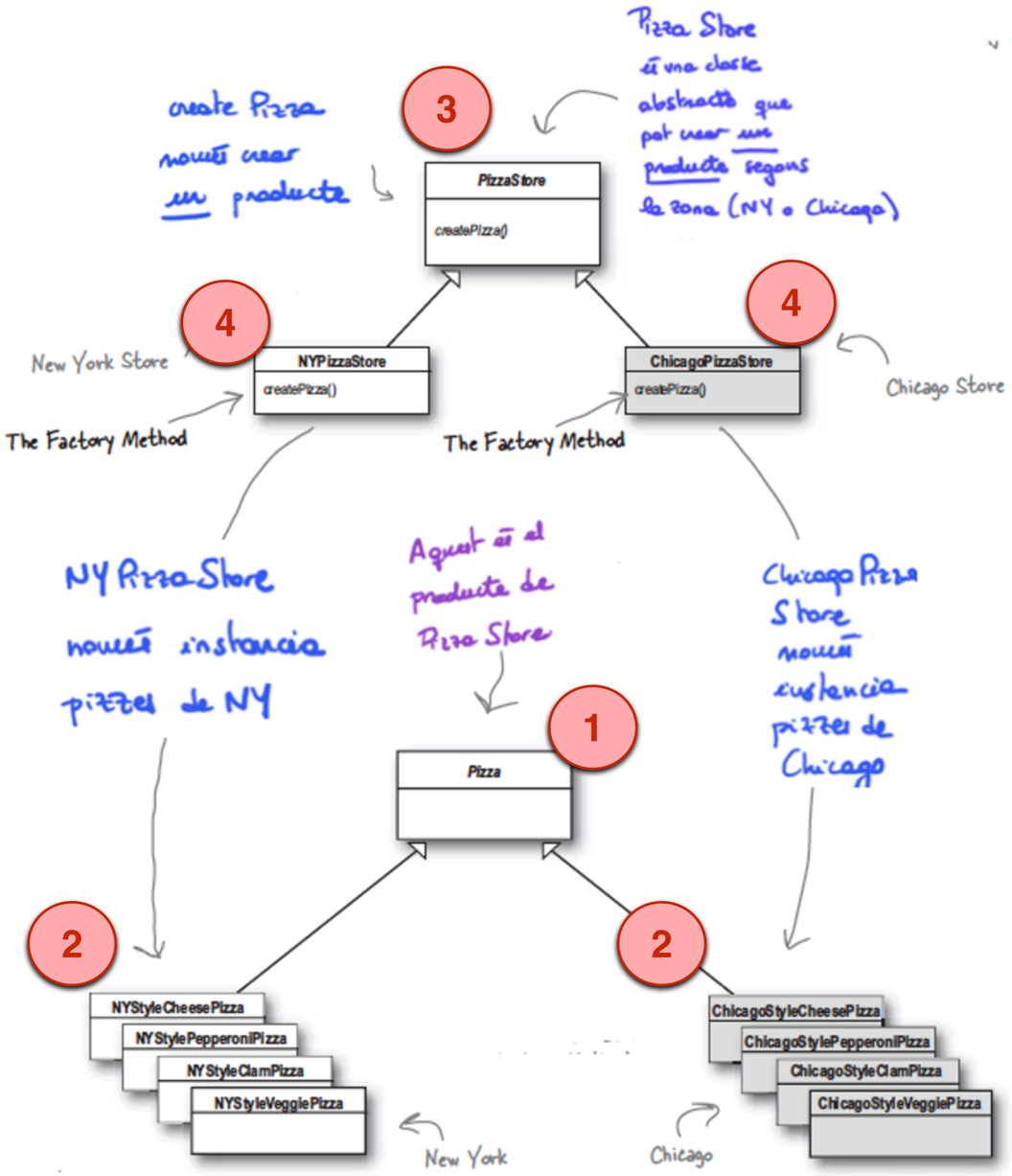


Exercici Pizza Store: Projecte franquícies



Exercici Pizza Store





Exercici Pizza Store: Projecte ingredients

Patró Abstract Factory