# Builder

Patrones de Diseño

# Problema

Se desea permitir a los clientes de una pizzería:

- Ordenar pizzas de diferentes tamaños.
- Seleccionar diferentes combinaciones de ingredientes, pero no necesariamente todos.

# Problema

problem.java

```
Pizza(int size, boolean cheese, boolean pepperoni, boolean bacon, ...) { ... }
```

# Problema

problem.java

```
Pizza(int size, boolean cheese, boolean pepperoni, boolean bacon, ...) { ... }
```

problem.java

```
Pizza(int size) { ... }
Pizza(int size, boolean cheese) { ... }
Pizza(int size, boolean cheese, boolean pepperoni) { ... }
Pizza(int size, boolean cheese, boolean pepperoni, boolean bacon) { ... }
```

# Solución: Builder

# vs Setters

vsSetters.java

```java
Pizza pizza = new Pizza(12);
pizza.setCheese(true);
pizza.setPepperoni(true);
pizza.setBacon(true);
```

Hasta no asignar todos los parámetros el objeto queda en un estado **inconsistente**.

Genera problemas en ambientes **multithread.**

# vs Factory

## Factory

Busca abstraer el proceso de creación de diferentes tipos de objeto.

El foco es ***que objeto crear***.

## Builder

Busca resolver múltiples opciones y parámetros opcionales.

El foco es ***como crear el objeto de forma flexible y paso a paso***.

# What is `QueryBuilder`

`QueryBuilder` is one of the most powerful features of TypeORM - it allows you to build SQL queries using elegant and convenient syntax, execute them and get automatically transformed entities.

Simple example of `QueryBuilder`:

```
const firstUser = await dataSource
    .getRepository(User)
    .createQueryBuilder("user")
    .where("user.id = :id", { id: 1 })
    .getOne()
```

It builds the following SQL query:

```
SELECT
    user.id as userId,
    user.firstName as userFirstName,
    user.lastName as userLastName
FROM users user
WHERE user.id = 1
```