# Spring Boot Test (Gelateria)

## 🏗️ 1. Creazione progetto con Spring Initializr

Vai su 👉 https://start.spring.io e configura così:

- **Project**: Maven
- **Language**: Java
- **Spring Boot**: 3.x
- **Group**: `it.epicode`
- **Artifact**: `gelateria`
- **Name**: `gelateria`
- **Package name**: `it.epicode.gelateria`
- **Packaging**: Jar
- **Java**: 17 o superiore

### 📦 Dipendenze da selezionare:

- Spring Boot DevTools
- Lombok

### ➕ Da aggiungere a mano nel `pom.xml`:

```
<!-- Faker -->
<dependency>
```

```
    <groupId>com.github.javafaker</groupId>
    <artifactId>javafaker</artifactId>
    <version>1.0.2</version>
</dependency>

<!-- JUnit 5 e parametrici -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
</dependency>
```

## 🧬 2. Struttura del progetto

```
src/
├── main/
│   └── java/it/epicode/gelateria/
│       ├── GelateriaApplication.java
│       ├── config/GelateriaConfig.java
│       ├── model/
│       │   ├── Prodotto.java
│       │   ├── InformazioniNutrizionali.java
│       │   ├── Topping.java
│       │   ├── Gelato.java
│       │   ├── Bevanda.java
│       │   └── Menu.java
│       └── runner/MenuRunner.java
└── test/
    └── java/it/epicode/gelateria/
        ├── GelateriaApplicationTests.java
        └── model/
```

```
├── GelatoParametricTest.java
├── BevandaParametricTest.java
└── ToppingCsvTest.java
```

## 🔧 3. Codice principale

### ✅ `GelateriaApplication.java`

```java
@SpringBootApplication
public class GelateriaApplication {
    public static void main(String[] args) {
        SpringApplication.run(GelateriaApplication.class, args);
    }
}
```

### ✅ `GelateriaConfig.java`

```java
@Configuration
public class GelateriaConfig {

    Faker faker = new Faker();

    @Bean
    public Topping cioccolato() {
        return new Topping("Cioccolato", 0.5, new InformazioniNutrizionali(50, 10, 5));
    }

    @Bean
```

```
    public Topping granella() {
        return new Topping("Granella", 0.4, new InformazioniNutrizionali(30, 5, 2));
    }

    @Bean
    public Gelato coppaGolosa() {
        return new Gelato("Coppa Golosa", List.of(cioccolato(), granella()), false);
    }

    @Bean
    public Gelato maxiDelizia() {
        return new Gelato("Maxi Delizia", List.of(cioccolato(), cioccolato(), granella()), true);
    }

    @Bean
    public Bevanda bibitaFresca() {
        return new Bevanda(faker.beer().name(), 1.5, new InformazioniNutrizionali(80, 20, 0));
    }

    @Bean
    public Menu menu(List<Prodotto> prodotti) {
        return new Menu(prodotti);
    }
}
```

## ✅ MenuRunner.java

```
@Component
public class MenuRunner implements CommandLineRunner {

    private final Menu menu;

    public MenuRunner(Menu menu) {
```

```
        this.menu = menu;
    }

    @Override
    public void run(String... args) {
        menu.stampaMenu();
    }
}
```

## 📚 4. Modelli

### `Prodotto.java` (interfaccia base)

```
public interface Prodotto {
    String getNome();
    double getPrezzo();
    InformazioniNutrizionali getInfoNutrizionali();
}
```

### `InformazioniNutrizionali.java`

```
@Data
@AllArgsConstructor
public class InformazioniNutrizionali {
    private int calorie;
    private int zuccheri;
```

```
    private int grassi;
}
```

## Topping.java

```
@Data
@AllArgsConstructor
public class Topping {
    private String nome;
    private double prezzo;
    private InformazioniNutrizionali infoNutrizionali;
}
```

## Gelato.java

```
@Data
public class Gelato implements Prodotto {

    private String nome;
    private List<Topping> toppings;
    private boolean maxi;

    public Gelato(String nome, List<Topping> toppings, boolean maxi) {
        this.nome = nome;
        this.toppings = toppings;
        this.maxi = maxi;
    }

    @Override
```

```
    public double getPrezzo() {
        double base = 2.0;
        double extra = toppings.stream().mapToDouble(Topping::getPrezzo).sum();
        double totale = base + extra;
        return maxi ? totale * 1.5 : totale;
    }

    @Override
    public InformazioniNutrizionali getInfoNutrizionali() {
        int calorie = 100;
        int zuccheri = 15;
        int grassi = 5;
        for (Topping t : toppings) {
            var info = t.getInfoNutrizionali();
            calorie += info.getCalorie();
            zuccheri += info.getZuccheri();
            grassi += info.getGrassi();
        }
        if (maxi) {
            calorie *= 1.5;
            zuccheri *= 1.5;
            grassi *= 1.5;
        }
        return new InformazioniNutrizionali(calorie, zuccheri, grassi);
    }
}
```

## Bevanda.java

```
@Data
@AllArgsConstructor
public class Bevanda implements Prodotto {
    private String nome;
```

```
    private double prezzo;
    private InformazioniNutrizionali infoNutrizionali;
}
```

### Menu.java

```
@Data
@AllArgsConstructor
public class Menu {
    private List<Prodotto> prodotti;

    public void stampaMenu() {
        System.out.println("=== MENU GELATERIA ===");
        prodotti.forEach(p -> {
            var info = p.getInfoNutrizionali();
            System.out.printf("- %s: %.2f€ | Cal: %d, Zuccheri: %dg, Grassi: %dg%n",
                p.getNome(), p.getPrezzo(),
                info.getCalorie(), info.getZuccheri(), info.getGrassi());
        });
    }
}
```

## 🧪 5. Test Spring Boot e parametrici

### 🔷 GelatoParametricTest.java

```
@SpringBootTest
@DisplayName("Test parametrici con @CsvSource sulla classe Gelato")
```

```java
public class GelatoParametricTest {

    @ParameterizedTest
    @CsvSource({
        "Vaniglia,false,2.0",
        "Cioccolato,true,3.0"
    })
    @DisplayName("Verifica prezzo corretto per gelati con o senza maxi")
    void testPrezzoGelatoConCsv(String nome, boolean maxi, double expected) {
        Gelato g = new Gelato(nome, List.of(), maxi);
        assertAll(
            () -> assertEquals(nome, g.getNome()),
            () -> assertEquals(expected, g.getPrezzo(), 0.01),
            () -> assertEquals(maxi, g.isMaxi())
        );
    }
}
```

### 🔷 BevandaParametricTest.java

```java
@SpringBootTest
@DisplayName("Test parametrici su Bevanda con @CsvSource")
public class BevandaParametricTest {

    @ParameterizedTest
    @CsvSource({
        "Cola,1.5,90",
        "Fanta,1.2,85"
    })
    @DisplayName("Verifica attributi bevande")
    void testBevandaCsv(String nome, double prezzo, int calorie) {
        InformazioniNutrizionali info = new InformazioniNutrizionali(calorie, 20, 0);
        Bevanda b = new Bevanda(nome, prezzo, info);
```

```
        assertAll(
            () -> assertEquals(nome, b.getNome()),
            () -> assertEquals(prezzo, b.getPrezzo(), 0.01),
            () -> assertEquals(calorie, b.getInfoNutrizionali().getCalorie())
        );
    }
}
```

## 🧪 GelatoSimpleTest.java

```
package it.epicode.gelateria.model;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
@DisplayName("Test semplici su Gelato")
public class GelatoSimpleTest {

    @Test
    @DisplayName("Calcolo corretto del prezzo con topping")
    public void testPrezzoGelatoConTopping() {
        Topping nutella = new Topping("Nutella", 0.7, new InformazioniNutrizionali(80, 20, 10));
        Gelato g = new Gelato("Nutelloso", List.of(nutella), false);

        double expected = 2.0 + 0.7;
        assertEquals(expected, g.getPrezzo(), 0.01);
    }
```

```
    @Test
    @DisplayName("Informazioni nutrizionali di un gelato senza topping")
    public void testInfoGelatoBase() {
        Gelato g = new Gelato("Semplice", List.of(), false);
        InformazioniNutrizionali info = g.getInfoNutrizionali();

        assertAll(
            () -> assertEquals(100, info.getCalorie()),
            () -> assertEquals(15, info.getZuccheri()),
            () -> assertEquals(5, info.getGrassi())
        );
    }
}
```

## BevandaSimpleTest.java

```
package it.epicode.gelateria.model;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
@DisplayName("Test semplici su Bevanda")
public class BevandaSimpleTest {

    @Test
    @DisplayName("Costruzione corretta di una bevanda")
    public void testCostruttoreBevanda() {
```

```
        InformazioniNutrizionali info = new InformazioniNutrizionali(70, 18, 0);
        Bevanda bevanda = new Bevanda("Tè Verde", 1.0, info);

        assertAll(
            () -> assertEquals("Tè Verde", bevanda.getNome()),
            () -> assertEquals(1.0, bevanda.getPrezzo(), 0.01),
            () -> assertEquals(70, bevanda.getInfoNutrizionali().getCalorie())
        );
    }
}
```

## 🍫 ToppingSimpleTest.java

```
package it.epicode.gelateria.model;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
@DisplayName("Test semplici su Topping")
public class ToppingSimpleTest {

    @Test
    @DisplayName("Verifica valori del topping Caramello")
    public void testToppingCaramello() {
        InformazioniNutrizionali info = new InformazioniNutrizionali(45, 12, 3);
        Topping t = new Topping("Caramello", 0.5, info);

        assertAll(
```

```
            () -> assertEquals("Caramello", t.getNome()),
            () -> assertEquals(0.5, t.getPrezzo(), 0.01),
            () -> assertEquals(12, t.getInfoNutrizionali().getZuccheri())
        );
    }
}
```

## 📋 MenuSimpleTest.java

```java
package it.epicode.gelateria.model;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
@DisplayName("Test semplici su Menu")
public class MenuSimpleTest {

    @Test
    @DisplayName("Controlla che il menu abbia prodotti corretti")
    public void testMenuConProdotti() {
        Prodotto g = new Gelato("Base", List.of(), false);
        Prodotto b = new Bevanda("Acqua", 1.0, new InformazioniNutrizionali(0, 0, 0));
        Menu menu = new Menu(List.of(g, b));

        assertEquals(2, menu.getProdotti().size());
        assertEquals("Base", menu.getProdotti().get(0).getNome());
```

```
        assertEquals("Acqua", menu.getProdotti().get(1).getNome());
    }

    @Test
    @DisplayName("Test della stampa del menu (verifica che non lanci eccezioni)")
    public void testStampaMenu() {
        Menu menu = new Menu(List.of(
            new Gelato("Test", List.of(), false),
            new Bevanda("TestDrink", 1.2, new InformazioniNutrizionali(90, 20, 0))
        ));

        // Se non lancia eccezioni, il test passa
        assertDoesNotThrow(menu::stampaMenu);
    }
}
```