

# Guida Introduttiva alla Programmazione Java con Spring Boot

## 1. Programmazione Object-Oriented (OOP)

La programmazione orientata agli oggetti (OOP) si basa su quattro concetti fondamentali:

### 1.1 Incapsulamento

L'incapsulamento consiste nel proteggere i dati all'interno di una classe, esponendo solo i metodi necessari per interagire con essi.

**Esempio:**

```
public class Persona {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

### 1.2 Ereditarietà

Permette a una classe di derivare da un'altra, ereditando metodi e attributi.

**Esempio:**

```
public class Animale {  
    public void faiVerso() {  
        System.out.println("L'animale fa un verso.");  
    }  
}  
  
public class Cane extends Animale {  
    @Override  
    public void faiVerso() {  
        System.out.println("Il cane abbaia.");  
    }  
}
```

```
}  
}
```

### 1.3 Polimorfismo

Il polimorfismo consente di trattare oggetti diversi in modo uniforme.

**Esempio:**

```
Animale mioAnimale = new Cane();  
mioAnimale.faiVerso(); // Output: Il cane abbaia.
```

### 1.4 Astrazione

L'astrazione permette di nascondere i dettagli complessi e mostrare solo le funzionalità essenziali.

**Esempio:**

```
public abstract class Veicolo {  
    public abstract void muovi();  
}  
  
public class Auto extends Veicolo {  
    @Override  
    public void muovi() {  
        System.out.println("L'auto si muove.");  
    }  
}
```

---

## 2. Pacchetti Java

### Struttura e Vantaggi

Un pacchetto è una cartella logica che contiene classi correlate. Aiuta a organizzare il progetto e a evitare conflitti di nome.

**Esempio:**

```
package com.miaazienda.modello;  
  
public class Prodotto {  
    // codice qui  
}
```

---

### 3. Pattern Model e DAO

#### Model

Rappresenta i dati.

#### Esempio:

```
public class Prodotto {  
    private int id;  
    private String nome;  
  
    // Costruttore, getter, setter  
}
```

#### DAO (Data Access Object)

Gestisce le operazioni sul database.

#### Esempio:

```
public class ProdottoDAO {  
    public void salva(Prodotto p) {  
        System.out.println("Prodotto salvato: " + p.getNome());  
    }  
}
```

#### Vantaggio

Separa la logica di accesso ai dati dalla logica di business.

---

### 4. Interfacce

Un'interfaccia definisce un contratto che una classe deve rispettare.

#### Esempio:

```
public interface Forma {  
    void disegna();  
}  
  
public class Cerchio implements Forma {  
    @Override  
    public void disegna() {  
        System.out.println("Disegna un cerchio.");  
    }  
}
```

```
}  
}
```

## 5. Annotazioni (@)

### Built-in

```
@Override // indica che si sta sovrascrivendo un metodo
```

### JPA

```
@Entity // rappresenta una tabella nel database  
@Id      // chiave primaria
```

### Spring

```
@RestController // gestisce richieste HTTP  
@Autowired      // inietta automaticamente le dipendenze
```

## 6. Override e Mapping

### Override

Permette di riscrivere un metodo ereditato.

#### Esempio:

```
@Override  
public void metodo() { }
```

### Mapping con JPA

Collega una classe a una tabella del database.

#### Esempio:

```
@Entity  
public class Prodotto {  
    @Id  
    private int id;
```

```
    private String nome;  
}
```

---

## 7. Collegamento MySQL

Configurazione base in `pom.xml`

```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>8.0.33</version>  
</dependency>
```

Configurazione in `application.properties`

```
spring.datasource.url=jdbc:mysql://localhost:3306/miodatabase  
spring.datasource.username=root  
spring.datasource.password=pass  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true
```

---

## 8. Spring Boot

Spring Boot semplifica la creazione di applicazioni web e la connessione ai database.

**Avvio applicazione**

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

---

## 9. Spring - Componenti Principali

### 9.1 Controller

Gestisce le richieste HTTP e restituisce risposte.

**Esempio:**

```

@RestController
@RequestMapping("/prodotti")
public class ProdottoController {

    @GetMapping("/{id}")
    public String getProdotto(@PathVariable int id) {
        return "Prodotto con id: " + id;
    }
}

```

## 9.2 Repository

Interfaccia che permette l'accesso al database.

**Esempio:**

```

@Repository
public interface ProdottoRepository extends JpaRepository<Prodotto, Integer>
{
}

```

## 9.3 Entity

Classe che rappresenta una tabella nel database.

**Esempio:**

```

@Entity
public class Prodotto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String nome;
}

```

## 9.4 Service

Classe che contiene la logica di business.

**Esempio:**

```

@Service
public class ProdottoService {

    @Autowired
}

```

```
private ProdottoRepository prodottoRepository;  
  
public List<Prodotto> listaProdotti() {  
    return prodottoRepository.findAll();  
}  
}
```

---

Questa guida fornisce una base teorica e pratica per iniziare con Java e Spring Boot, spiegando i concetti fondamentali e mostrando esempi semplici ma efficaci.