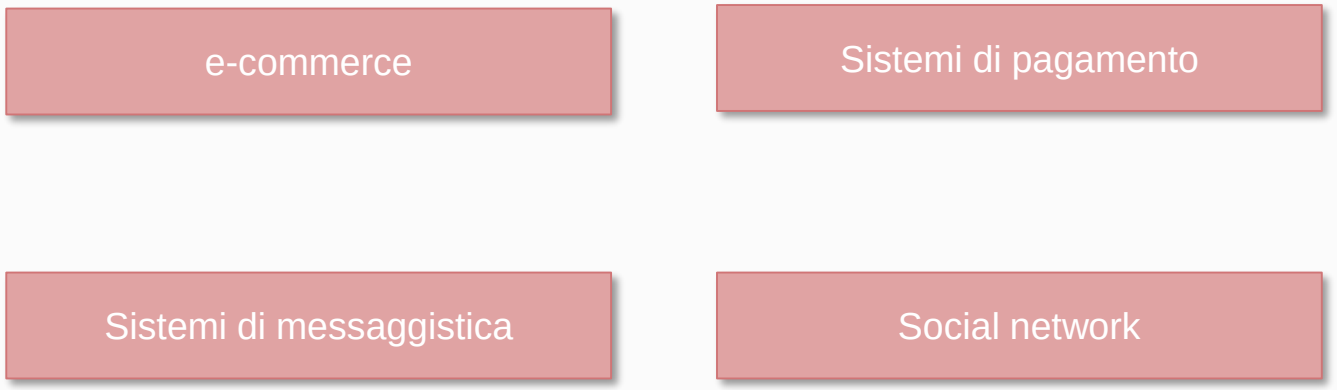
Gestione delle dipendenze tra bean – Parte I

Cosa imparerai

- Come si definiscono e gestiscono le dipendenze tra oggetti in Spring tramite la Dependency Injection

Gestione delle dipendenze tra bean

Le applicazioni (da quelle più piccole a quelle di tipo enterprise) generalmente sono costituite da più entità che in qualche modo sono collegate tra loro.



In Spring la gestione delle relazioni tra i bean viene effettuata tramite la Dependency Injection.

Ricordiamo cos'è la Dependency Injection

La **Dependency Injection** (**DI**) è un design pattern utilizzato per implementare l'IoC.

È un processo in base al quale gli oggetti definiscono le loro dipendenze da altri oggetti.

Il container inietta le dipendenze quando crea il bean.

Il bean non si occupa di nulla, c'è il container che lavora per lui (da cui il nome di *Inversion of Control*).

Il pattern DI può essere implementato in tre modi, ma in Spring ne sono utilizzati due:

- **Constructor Injection**
- **Setter Injection**

Dependency Injection

Constructor Injection

La DI basata sul costruttore viene realizzata invocando il costruttore della classe e passando in input l'istanza della classe da creare.

```java
public class Biblioteca {
    private BibliotecaService service;

    public Biblioteca(BibliotecaService serv) {
        super();

        service = serv;
    }

    public List<Libro> getLibriDisponibili() {
        return service.getLibriDisponibili();
    }
}
```

```java
public class BibliotecaFactory {
    public static Biblioteca getBiblioteca() {
        Biblioteca b = new Biblioteca(new BibliotecaServiceImpl());

        return b;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        BibliotecaFactory.getBiblioteca().getLibriDisponibili();
    }
}
```

Constructor Injection in Spring

In caso di DI basata sul costruttore, il container richiama un costruttore che richiede in ingresso un certo numero di argomenti, ciascuno dei quali rappresenta una dipendenza.

Facciamo un esempio…

Il service layer dell'entità *Fattura* accede al service layer dell'entità *Cliente* per recuperare le informazioni di un cliente.

```
public class FatturaService {
    private ClienteService cliente;

    public FatturaService(ClienteService cliente) {
        super();
        this.cliente = cliente;
    }

    public String getInfoFattura(Long id) {
        return "Fattura n. 123/21 - Cliente: " + cliente.getCliente(22l);
    }
}
```

Ecco la DI basata sul costruttore

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

Se usiamo XML per i metadati, dobbiamo utilizzare l'elemento **<constructor-arg … />** utilizzando, a seconda dei casi, una delle seguenti combinazioni di parametri

- **ref**
- **index + value**
- **type + value**
- **name + value**

```
<bean id="myBean" class="it.test.MyBean">
      <constructor-arg ref="..."/>
      <constructor-arg ref="..."/>
</bean>
```

```
<bean id="myBean" class="it.test.MyBean">
      <constructor-arg type="..." value="…"/>
      <constructor-arg type="..." value="…"/>
</bean>
```

```
<bean id="myBean" class="it.test.MyBean">
     <constructor-arg index="..." value="…"/>
     <constructor-arg index="..." value="…"/>
</bean>
```

```
<bean id="myBean" class="it.test.MyBean">
     <constructor-arg name="..." value="…"/>
     <constructor-arg name="..." value="…"/>
</bean>
```

Vediamo caso per caso…

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

- L'attributo **ref** va utilizzato quando **il costruttore ha come argomenti dei bean**

```
<bean id="fatturaService" class="it.test.service.FatturaService">
    <constructor-arg ref="clienteService" />
    <constructor-arg ref="prodottoService" />
</bean>

<bean id="clienteService" class="it.test.service.ClienteService">
</bean>

<bean id="prodottoService" class="it.test.service.ProdottoService">
</bean>
```

```
public class FatturaService {
    private ClienteService cliente;
    private ProdottoService prodotto;

    public FatturaService(ClienteService cliente, ProdottoService prodotto) {
        super();
        this.cliente = cliente;
        this.prodotto = prodotto;
    }

    public String getInfoFattura(Long id) {
        return "Fattura n. 123/21 - Cliente: " + cliente.getCliente(22l) + " - " + prodotto.sayHello();
    }
}
```

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

- La coppia di attributi **type + value** va utilizzata quando **il costruttore ha come argomenti dei tipi semplici (long, String, …)** ed il container non può determinare il tipo dell'argomento

```
<bean id="clienteService" class="it.test.service.ClienteService">
    <constructor-arg type="String" value="PRIVATO" />
    <constructor-arg type="int" value="1" />
</bean>
```

```
public class ClienteService {
    private String tipoPredefinito;
    private int cercaCancellati;

    public ClienteService(String tipoPredefinito, int cercaCancellati) {
        super();
        this.tipoPredefinito = tipoPredefinito;
        this.cercaCancellati = cercaCancellati;
    }

    public String getCliente(Long id) {
}
```

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

- La coppia di attributi **index + value** va utilizzata per specificare esplicitamente il tipo dell'i-esimo argomento. **L'index è 0-based**. È utile in caso di ambiguità tra gli argomenti, cioè se un costruttore ha due argomenti dello stesso tipo.

```xml
<bean id="prodottoService" class="it.test.service.ProdottoService">
    <constructor-arg index="0" value="TELEVISIONI" />
    <constructor-arg index="1" value="Descrizione non presente..." />
</bean>
```

```java
public class ProdottoService {
    private String categoriaPredefinita;
    private String descrizionePredefinita;

    public ProdottoService(String categoriaPredefinita, String descrizionePredefinita) {
        super();
        this.categoriaPredefinita = categoriaPredefinita;
        this.descrizionePredefinita = descrizionePredefinita;
    }

    public String sayHello() {
}
```

NOTA: in questo caso, possiamo sempre utilizzare l'attributo **type** in aggiunta a *index* e *value* per specificare il tipo dell'i-esimo elemento.

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

- La coppia di attributi **name + value** va utilizzata per specificare esplicitamente il valore da assegnare alla variabile che ha il nome specificato nell'attributo *name*.

```
<bean id="prodottoService" class="it.test.service.ProdottoService">
    <constructor-arg name="categoriaPredefinita" value="TELEVISIONI" />
    <constructor-arg name="descrizionePredefinita" value="Descrizione non presente..." />
</bean>
```

```
public class ProdottoService {
    private String categoriaPredefinita;
    private String descrizionePredefinita;

    public ProdottoService(String categoriaPredefinita, String descrizionePredefinita) {
        super();
        this.categoriaPredefinita = categoriaPredefinita;
        this.descrizionePredefinita = descrizionePredefinita;
    }

    public String sayHello() {
}
```

NOTA: in questo caso, possiamo sempre utilizzare l'attributo **type** in aggiunta a *name* e *value* per specificare il tipo dell'i-esimo elemento.

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

NOTA: per fare in modo che la coppia **name+value** funzioni, il codice deve essere compilato con il flag di debug abilitato in modo che Spring possa cercare il nome del parametro dal costruttore.
Se non è possibile o non si vuole compilare il codice con il flag di debug, è possibile utilizzare l'annotation **@ConstructorProperties** per denominare esplicitamente gli argomenti del costruttore.

```
<bean id="prodottoService" class="it.test.service.ProdottoService">
    <constructor-arg name="categoriaPredefinita" value="TELEVISIONI" />
    <constructor-arg name="descrizionePredefinita" value="Descrizione non presente..." />
</bean>
```

```
public class ProdottoService {
    private String categoria;
    private String descrizione;

    @ConstructorProperties({"categoriaPredefinita", "descrizionePredefinita"})
    public ProdottoService(String val1, String val2) {
        super();
        this.categoria = val1;
        this.descrizione = val2;
    }

    public String sayHello() {
}
```

Constructor Injection in Spring

Come fa Spring a risolvere gli argomenti del costruttore?

È possibile anche avere un mix tra i vari attributi, a seconda delle esigenze e necessità.

```xml
<bean id="fatturaService" class="it.test.service.FatturaService">
    <constructor-arg ref="clienteService" />
    <constructor-arg ref="prodottoService" />
    <constructor-arg type="String" value="ABCD" />
</bean>
```

```java
public class FatturaService {
    private ClienteService cliente;
    private ProdottoService prodotto;
    private String numerazionePredefinita;


    public FatturaService(ClienteService cliente, ProdottoService prodotto, String numerazionePredefinita) {
        super();
        this.cliente = cliente;
        this.prodotto = prodotto;
        this.numerazionePredefinita = numerazionePredefinita;
    }

    public String getInfoFattura(Long id) {
        return "Fattura n. 123/21 - Cliente: " + cliente.getCliente(22l) + " - " + prodotto.sayHello() + " -
    }
}
```

Di cosa abbiamo parlato in questa lezione

- Come si definiscono e gestiscono le dipendenze tra oggetti in Spring tramite la Dependency Injection