

# Spring

Riccardo Cattaneo



# Spring MVC

Spring MVC è un framework per realizzare applicazioni web basate sul modello MVC sfruttando i punti di forza offerti dal framework Spring come l'inversion of control (tramite dependency injection).

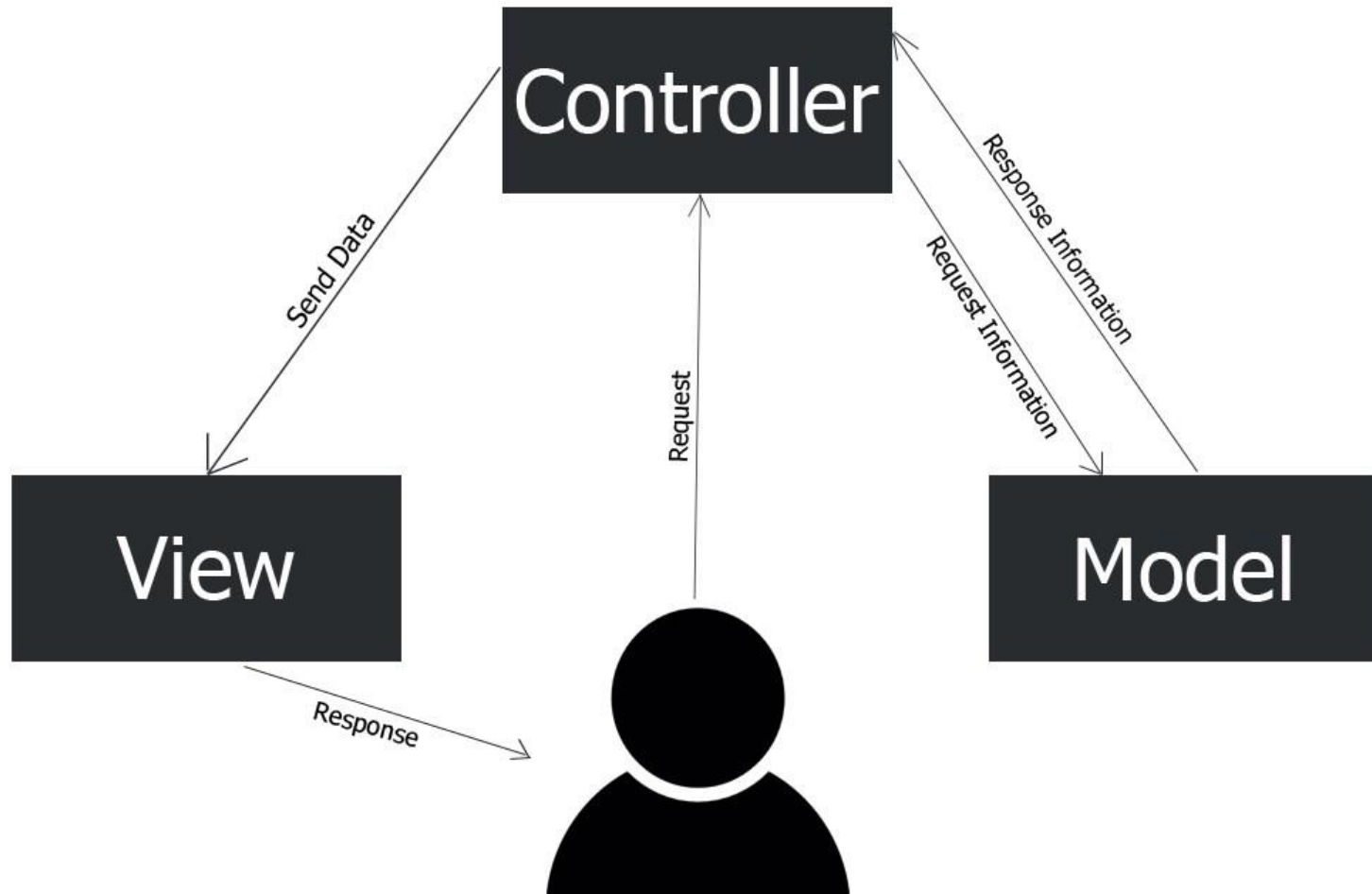
Esso si occupa di mappare metodi e classi Java con determinati url, di gestire differenti tipologie di “viste” restituite al client.

# Il Pattern MVC

MVC rappresenta (**M**odel **V**iew **C**ontroller) le tre componenti principali di un'applicazione web. Grazie a questo pattern i compiti vengono separati in 3 componenti:

- **Model** si occupano di accedere ai dati necessari alla logica di business implementata nell'applicazione (nel caso di un'applicazione per una biblioteca potrebbero essere le classi Libro, Autore, Scaffale);
- **View** si occupano di creare l'interfaccia utilizzabile dall'utente e che espone i dati da esso richiesti (nel caso bibliotecario potrebbero essere le pagine HTML del catalogo);
- **Controller** si occupano di implementare la vera logica di business dell'applicazione integrando le due componenti precedenti, ricevendo gli input dell'utente, gestendo i modelli per la ricerca dei dati e la creazione di viste da restituire all'utente.

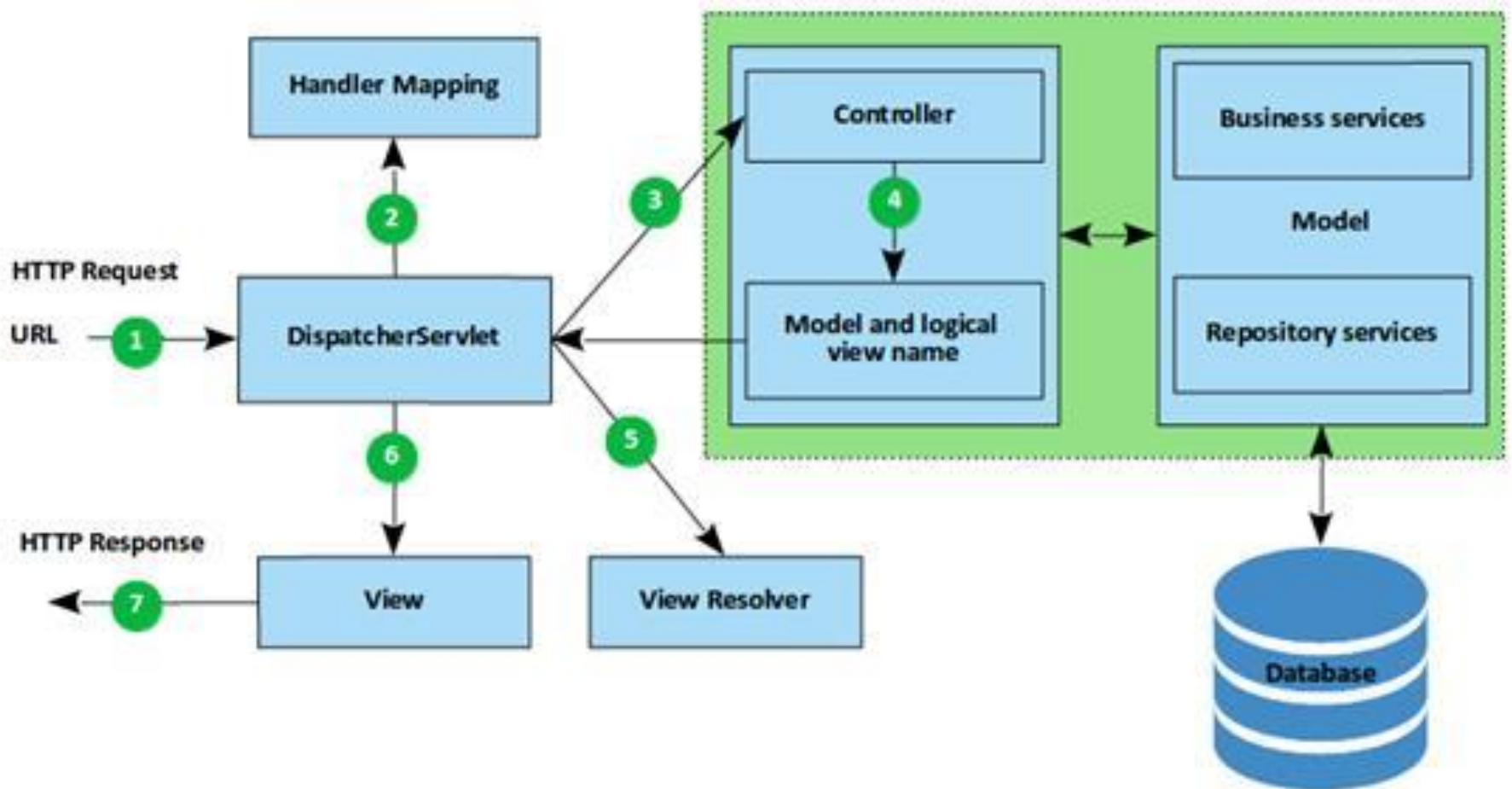
# Model-View-Controller



# MVC in Spring

Spring MVC implementa perfettamente questo approccio mantenendo sia i concetti che la nomenclatura del pattern. All'interno di una applicazione Spring MVC avremo quindi:

- i Model sono rappresentati dalle classi che a loro volta rappresentano gli oggetti gestiti e le classi di accesso al database;
- le View sono rappresentate dai vari file JSP (che vengono compilati in HTML) ;
- i Controller sono rappresentati da classi che rimangono “in ascolto” su un determinato URL e, grazie ai Model e alle View, si occupano di gestire la richiesta dell'utente.



# Dispatcher Servlet

Il funzionamento di Spring MVC è abbastanza semplice: tutto ruota intorno ad una servlet, la **DispatcherServlet**, che permette di redirigere tutte le chiamate della nostra applicazione verso il mondo di Spring MVC.

Per ciascuna servlet utilizzata viene istanziato un **WebApplicationContext** all'interno del quale vivono tutti i bean definiti. Questo contesto viene agganciato allo standard ServletContext in modo da essere recuperato facilmente.

All'interno di questo speciale contesto sono presenti alcuni **bean** che configurano in maniera particolare Spring MVC. Alcuni bean sono già presenti di default ma ovviamente si possono personalizzare.

Tra i più usati troviamo **HandlerMapping bean** che si occupa di mappare le url invocate dal client verso un particolare metodo o classe; il bean di default utilizza le java annotations.



# @Controller e @RequestMapping

Il controller rappresenta l'anima del paradigma MVC: è il componente che viene invocato direttamente dai client che si occupano delle principali logiche di business.

Il tipo di bean che si occupa del mapping tra url invocato dall'utente e metodo Java da invocare è l'handlerMapping.

L'implementazione di default presente in Spring MVC è la classe **RequestMappingHandlerMapping** che permette di sfruttare le java annotations per identificare i metodi da mappare. Le annotation utili a questo sono :

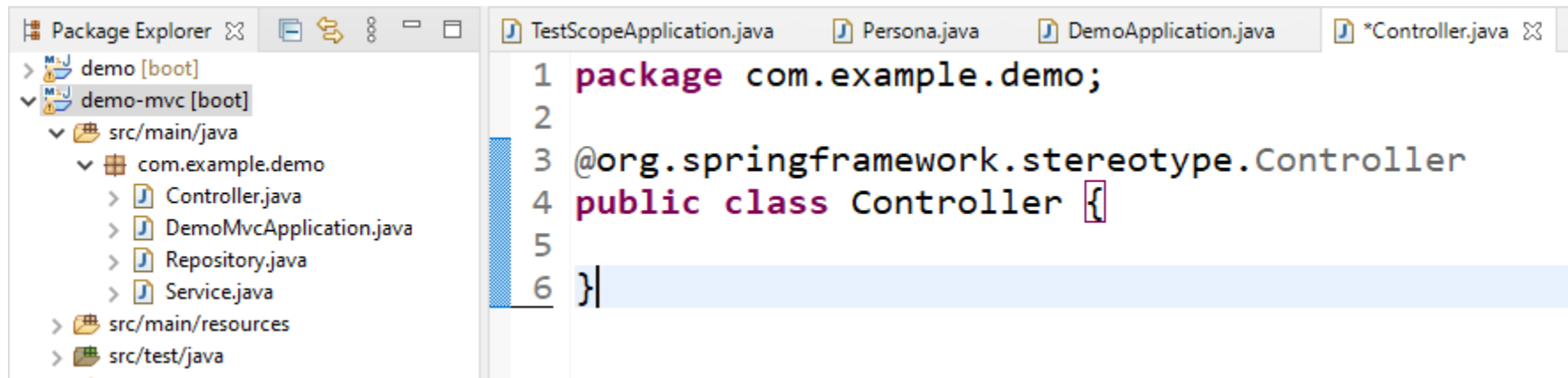
- **@controller** Da utilizzare a livello di classe per identificarla come controller
- **@RequestMapping** Da utilizzare per evidenziare il metodo e l'url da mappare
- **@Service** è una specializzazione dell'annotazione @Component
- **@Repository** è una specializzazione dell'annotazione @Component

Queste annotation sono il cuore del pattern MVC, ed in particolare del modulo MVC di Spring. Non fanno altro che separare il controller dal service e dal repository.

Passiamo subito al codice per capire a cosa ci stiamo riferendo. Creare un nuovo progetto con il nome «demo-mvc» con le stesse impostazioni iniziali del progetto precedente (Ricordarsi di importare il modulo Spring Web).

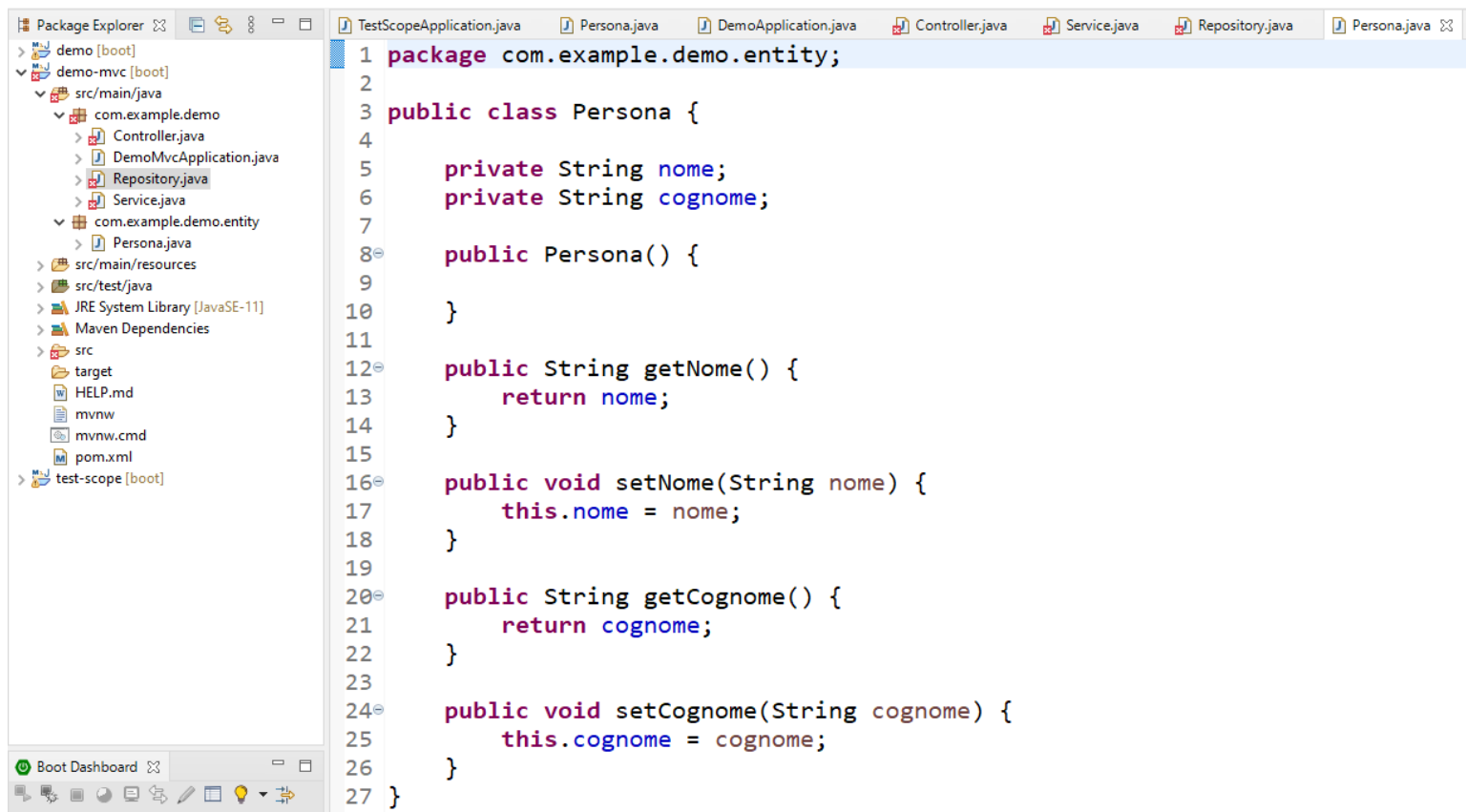
Andiamo adesso a creare un semplice esempio strutturato con il pattern MVC. Per prima cosa andiamo a creare una classe (per comodità la chiamiamo Controller) una classe Service ed una classe Repository.

1 - Occupiamoci per prima cosa del controller. Andiamo a mettere l'annotazione `@Controller` sulla classe Controller appena creata. Ricordiamo che il controller intercetterà la chiamata del Client ed eseguirà «qualcosa».



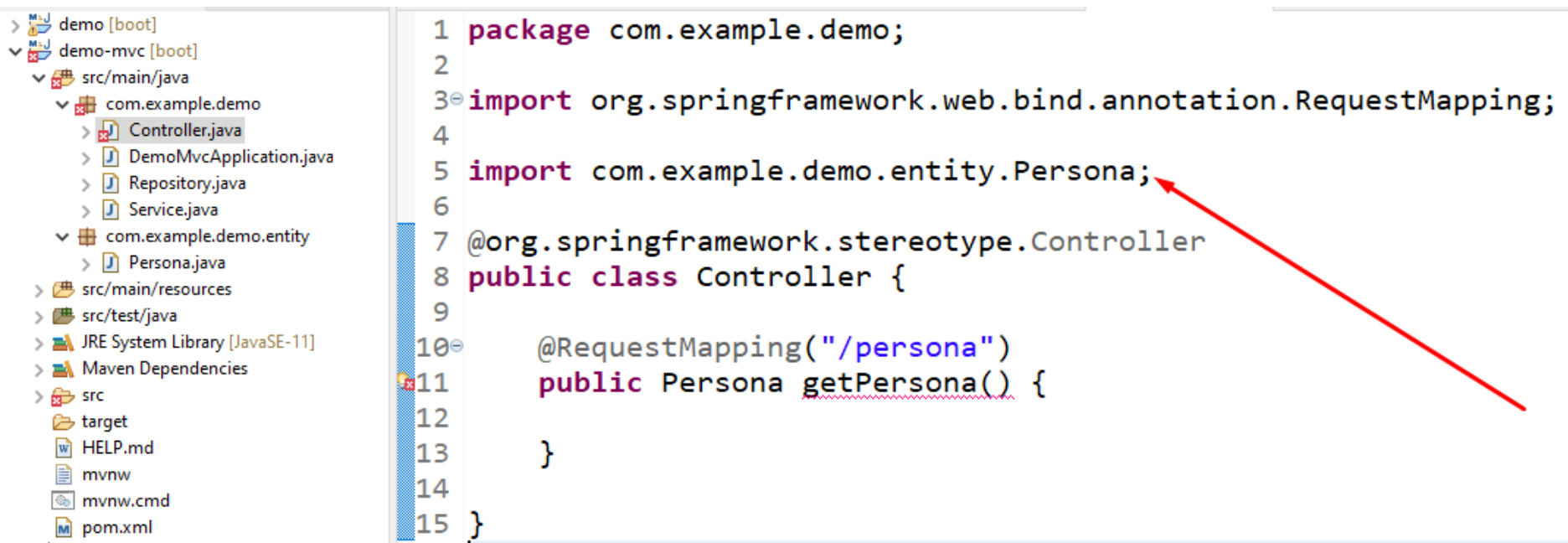
2 - Creiamo adesso un metodo all'interno della classe `Controller` e lo annotiamo con il `RequestMapping`. Il metodo lo chiameremo `getPersona` e tornerà un oggetto di tipo `Persona` (per riprendere l'esempio della lezione precedente)

3 – Andiamo adesso a creare un nuovo package e lo chiamiamo entity ed all'interno creeremo la nostra classe Persona (che avrà un nome, un cognome e un costruttore) con i relativi metodi get e set.



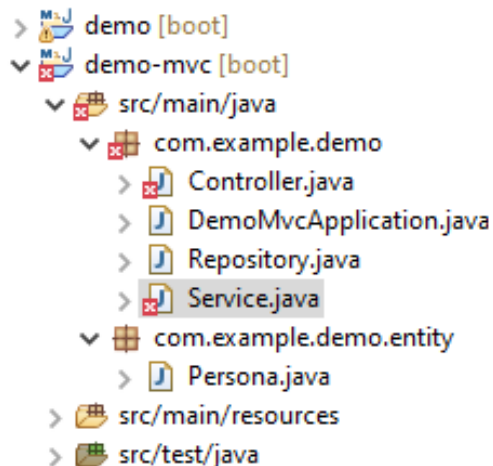
```
1 package com.example.demo.entity;
2
3 public class Persona {
4
5     private String nome;
6     private String cognome;
7
8     public Persona() {
9
10    }
11
12    public String getNome() {
13        return nome;
14    }
15
16    public void setNome(String nome) {
17        this.nome = nome;
18    }
19
20    public String getCognome() {
21        return cognome;
22    }
23
24    public void setCognome(String cognome) {
25        this.cognome = cognome;
26    }
27 }
```

4 – Una volta creata la classe Persona possiamo tornare nel nostro controller ed importare la classe appena creata.



```
1 package com.example.demo;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4
5 import com.example.demo.entity.Persona;
6
7 @org.springframework.stereotype.Controller
8 public class Controller {
9
10     @RequestMapping("/persona")
11     public Persona getPersona() {
12
13     }
14
15 }
```

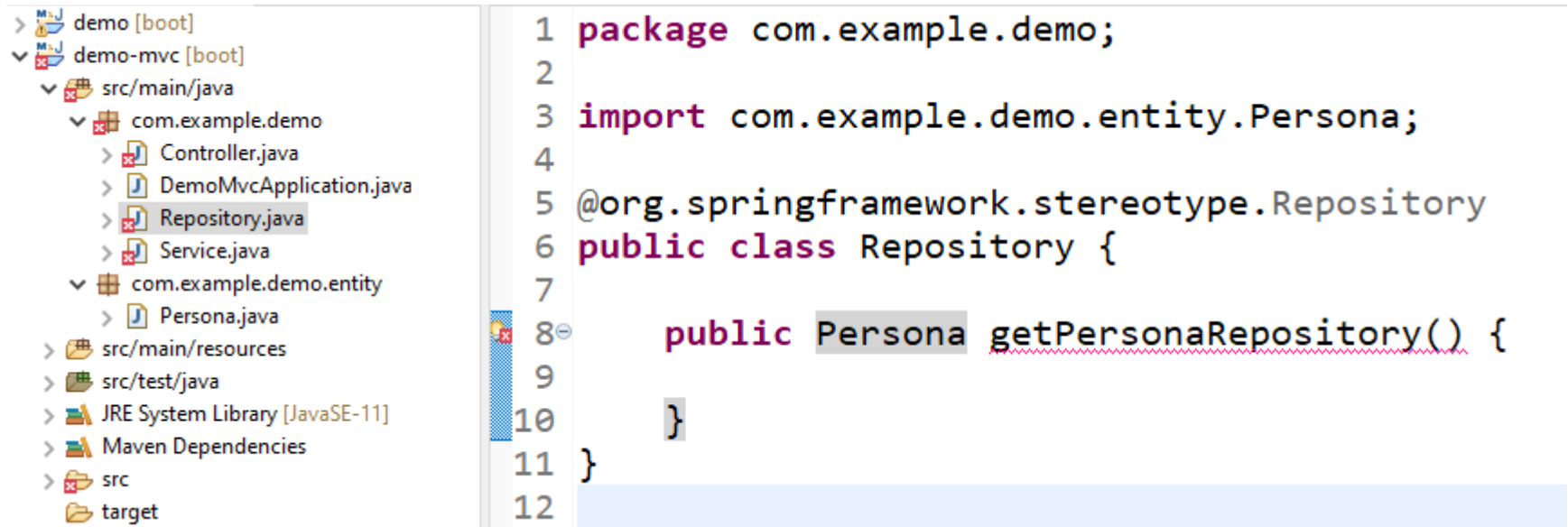
5 – Andiamo adesso nel nostro service (all'interno della classe service aggiungere l'annotazione Service) ed aggiungere un metodo che chiameremo getPersonaService() che tornerà sempre un oggetto di tipo persona :



```
1 package com.example.demo;
2
3 @org.springframework.stereotype.Service
4 public class Service {
5
6     public Persona getPersonaService() {
7
8     }
9 }
```

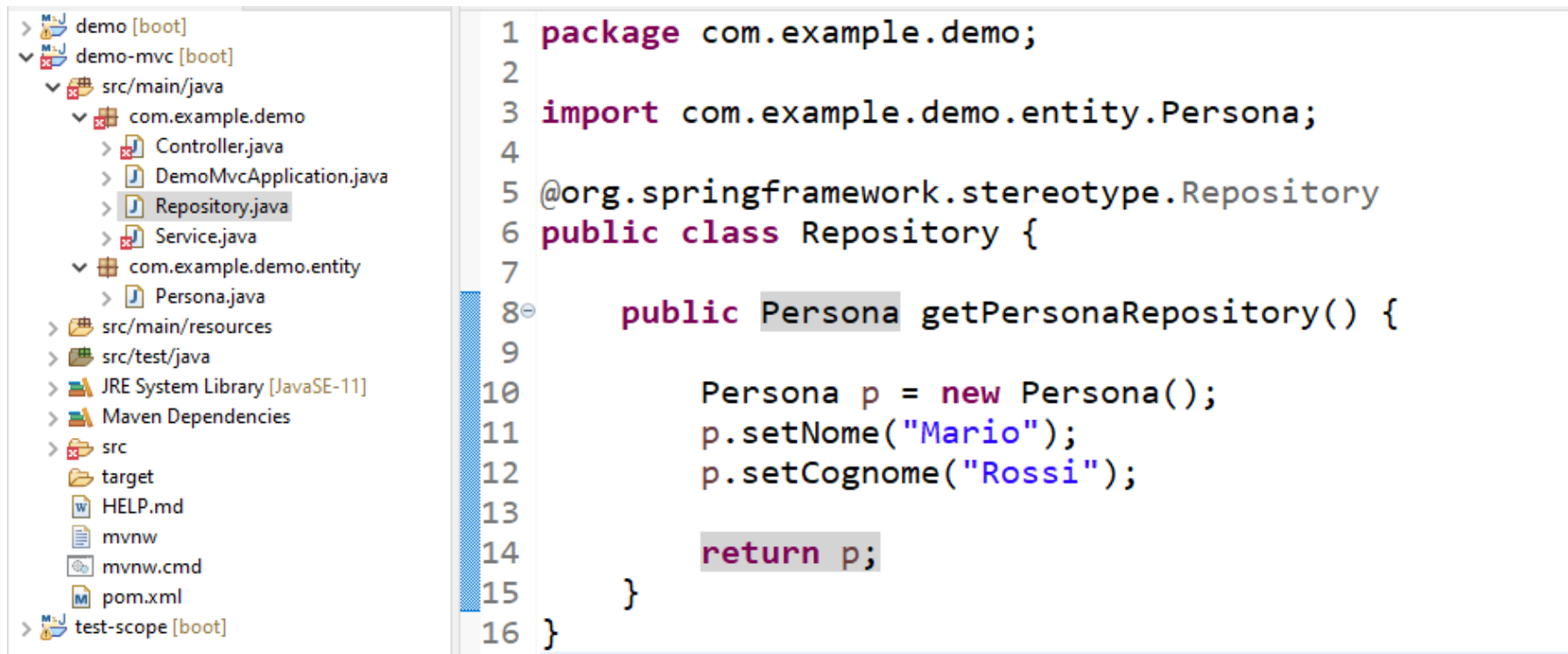


6 – Andiamo adesso nel nostro Repository (all'interno della classe aggiungere l'annotazione Repository) e all'interno scrivere un metodo chiamato per convenzione getPersonaRepository()



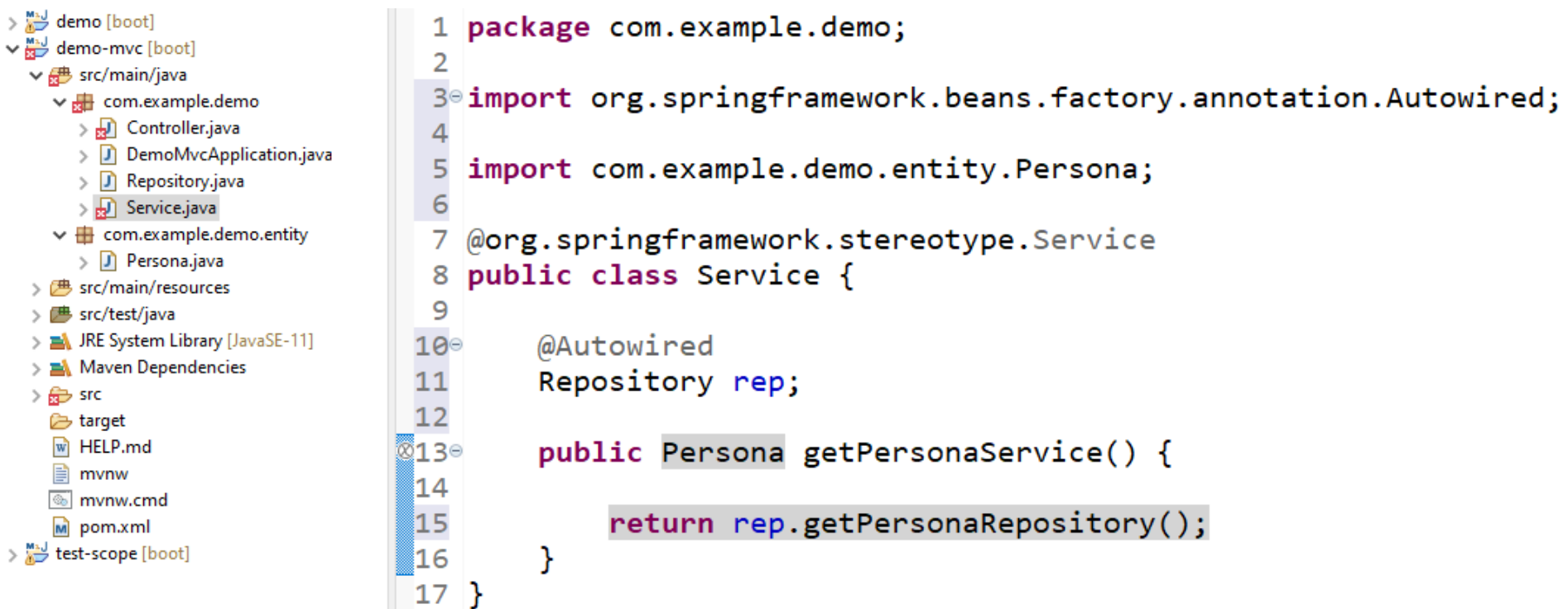
```
1 package com.example.demo;
2
3 import com.example.demo.entity.Persona;
4
5 @org.springframework.stereotype.Repository
6 public class Repository {
7
8     public Persona getPersonaRepository() {
9
10    }
11 }
12
```

7 - Nella realtà ora dovremmo creare all'interno del Repository una chiamata ad un database, ma per semplificare l'esempio e concentrarci su Spring, «simuliamo» una finta chiamata in questo modo : nel Repository andremo a creare una nuova Persona ed impostiamo un nome ed un cognome :



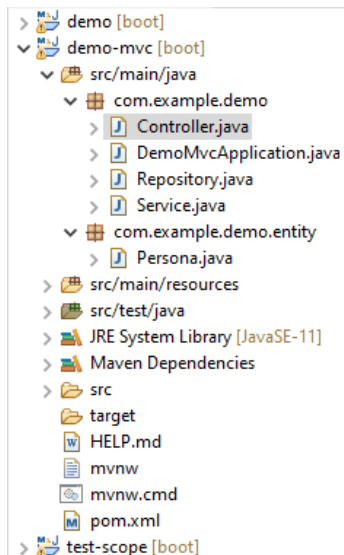
```
1 package com.example.demo;
2
3 import com.example.demo.entity.Persona;
4
5 @org.springframework.stereotype.Repository
6 public class Repository {
7
8     public Persona getPersonaRepository() {
9
10         Persona p = new Persona();
11         p.setNome("Mario");
12         p.setCognome("Rossi");
13
14         return p;
15     }
16 }
```

8 – Ora andiamo nel nostro Service e andiamo a chiamare il metodo che abbiamo appena scritto nel Repository. Come facciamo ? Ci servirà un'istanza del Repository per chiamare quel metodo, ed ecco che ci viene in aiuto Spring con la Dependency Injection.



```
1 package com.example.demo;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 import com.example.demo.entity.Persona;
6
7 @org.springframework.stereotype.Service
8 public class Service {
9
10     @Autowired
11     Repository rep;
12
13     public Persona getPersonaService() {
14
15         return rep.getPersonaRepository();
16     }
17 }
```

9 – Andiamo ora nel nostro controller andremo a chiamare il metodo del Service, quindi un altro Autowired per importare il Service e nel metodo getPersona andiamo a chiamare il metodo del Service :



```
1 package com.example.demo;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 import com.example.demo.entity.Persona;
7
8 @org.springframework.stereotype.Controller
9 public class Controller {
10
11     @Autowired
12     Service ser;
13
14     @RequestMapping("/persona")
15     public Persona getPersona() {
16
17         Persona p = ser.getPersonaService();
18
19         System.out.println(p.getCognome());
20         System.out.println(p.getNome());
21
22         return p;
23     }
24
25 }
```

10 – Proviamo ora la nostra applicazione avviando il server per prima cosa, poi tramite il browser chiamare l'indirizzo del controller /persona :

`http://localhost:8080/persona`

