**SETTIMANA 04**

**SPRING BOOT I**

**GIORNO 01**

Introduzione a Spring Boot

L’obiettivo finale sarà la realizzazione di web server tramite il framework Spring. Senza di esso siamo costretti a gestire tutto manualmente. Lo svantaggio del framework è una perdita di prestazioni.

Spring Boot ci fornisce una struttura iniziale nelle applicazioni Java e si occuperà di risolvere alcuni problemi che sorgono nello sviluppo con Java (come, ad esempio, istanziare nuovi oggetti partendo da una Classe).

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamenteSpring è MODULARE nel senso che installeremo solo i plug-in che ci servono in base a quello che dobbiamo fare (che si tratti della creazione di un web server, o sulla sicurezza, ecc…): sapendo ciò che dobbiamo assemblare, possiamo assemblarlo come vogliamo e con le funzionalità che ci servono maggiormente.

Spring crea un ambiente di lavoro in cui gestisce gli oggetti che noi decidiamo (bean). In Spring, si utilizzano comunemente i POJO, che sono semplici oggetti Java senza alcuna dipendenza da framework o librerie specifiche. I POJO sono utilizzati per rappresentare i dati dell'applicazione e sono gestiti da Spring come bean. Sono fondamentali per la creazione di oggetti gestiti all'interno del contesto di Spring. Lo Spring Bean è l’oggetto che viene gestito da Spring stesso.

Spring ci permette, tramite @annotazioni dichiarative specifiche, di abilitare particolari comportamenti per specifiche Classi che creiamo. Questo semplifica la scrittura del codice, evitando anche ripetizioni di pezzi di codici (come, ad esempio, la creazione degli oggetti DAO per richiamare i metodi definiti per interagire col Persistence Context e quando avrò bisogno di quell’oggetto, lo richiamerò con una annotazione in qualsiasi punto del progetto, semplificandomi il codice). In generale Spring mi riduce il boilerplate.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Noi definiamo le classi (Student, Animal ecc…) e le annotiamo con specifiche annotazioni per farle entrare nell’ottica di gestione di Spring. La Dependency Injection è uno dei principali concetti di Spring. Con la DI, Spring si occupa di fornire automaticamente le dipendenze (come gli oggetti necessari) a una classe quando ne ha bisogno. Invece di creare manualmente le istanze delle dipendenze, Spring le fornisce tramite costruttori o metodi setter. Questo rende il codice più modulare e facilita il testing e la manutenzione.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Spesso esistono delle dipendenze tra le Classi (una Classe A ha un attributo che deriva da una Classe B: un User che ha tra i suoi attributi un Address). Spring sfrutta l’iniezione delle dipendenze (DI), fornendo le dipendenze necessarie per il nostro progetto. Non saremo noi a creare un new Address e passarlo all’User, ma lo farà direttamente Spring che si occuperà di tutte le dipendenze tra le Classi.

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

La prima cosa da fare sarà configurare il nostro progetto e le nostre Classi nell’ottica Spring: posso crearmi un elenco di “bean”, cioè di oggetti che verranno gestiti da Spring, ad esempio. In generale tali configurazioni avvengono o su un file.xml o mediante annotazioni nel progetto Java stesso.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Spring semplifica l'interazione con il database tramite Spring Data JPA. Questo modulo permette di definire interfacce per i repository dei dati senza la necessità di implementarle manualmente. Spring Data JPA fornisce implementazioni di base per i metodi CRUD (Create, Read, Update, Delete) e altre operazioni di query, riducendo notevolmente il codice boilerplate necessario per interagire con il database.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

IoC (Inversion of Control) è un concetto alla base di Spring. La IoC è strettamente correlata alla DI. Invece di avere il controllo diretto sull'istanziazione e sulla gestione delle dipendenze, in Spring, è il framework stesso a gestire il ciclo di vita degli oggetti e le dipendenze. Questo concetto consente una maggiore flessibilità e scalabilità dell'applicazione.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamenteSe prima eravamo noi, manualmente, a creare oggetti, definirne il ciclo di vita e le dipendenze, ora sarà Spring a farlo (inverto quindi il controllo) e sarà Spring a creare, controllare e definire il ciclo di vita degli oggetti. (ad esempio, possiamo chiedergli di creare un oggetto singleton che sarà unico).

Gli oggetti che rappresentano delle funzionalità verranno create solo una volta da Spring e gestite solamente quando ne avrò bisogno mediante opportune annotazioni (vedi gli oggetti Dao). Immaginiamo Spring come una specie di scatolone, con al suo interno tutta una serie di oggetti gestiti (bean). Questi oggetti ci verranno forniti all’occorrenza in qualsiasi parte del progetto Java grazie ad annotazioni specifiche. I Dao vengono creati da Spring e passarli nei punti in cui ne ho bisogno. Quindi in questo scatolone troviamo gli oggetti che Spring crea, e li mette in relazione tra loro (tramite la DI), e gestisce il loro ciclo di vita.

Per creare i bean all’interno dello scatolone, posso usare un file xml, oppure un file di Java di configurazione, oppure attraverso delle @annotazioni di Spring (questa strada si segue soprattutto per i Dao).

Esistono diverse tipologie di “scatoloni”, quella che useremo noi è la Spring Application Context (si tratta di una ben precisa Interfaccia).

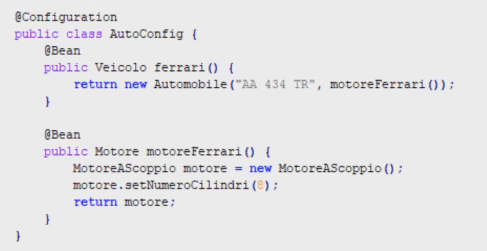
Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamenteNoi ci creiamo la nostra Classe che possiamo decidere far diventare un POJO. Da questa chiedo a Spring di gestire il ciclo di vita dei bean, cioè gli oggetti di quella Classe POJO.

Spring si occupa di iniettare le dipendenze necessarie all’interno dei bean attraverso tre modalità che saremo noi a scegliere.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Configurolo scatolone con due oggetti bean (Veicolo e Motore) definendone anche la dipendenza che esiste tra i due bean.

L’oggetto Veicolo definisce che ha bisogno di una dipendenza Motore e delego la creazione di queste dipendenze a Spring stesso. La dipendenza viene così “passata” invece che “creata”. Questo previene tutta una serie di problemi in fase di sviluppo e testing dell’applicazione. In genere è sempre meglio creare Classi separate e scollegate il più possibile, e passare le dipendenze tra esse solo in un secondo momento e tramite Spring, favorendo così un accoppiamento debole. In generale è sempre opportuno passare nel costruttore una interfaccia in modo da rendere le classi più adattabili per possibili cambi futuri, soprattutto per applicazioni grosse.

Immagine che contiene testo, linea, Carattere, schermata

Descrizione generata automaticamente

Una volta che creo la configurazione delle classi (@Configuration), nel main richiamo l’applicatiocontext che gestisce i bean che abbiamo definito nella configurazione iniziale. In questo modo è Spring che crea gli oggetti.

Immagine che contiene testo, Carattere, schermata, documento

Descrizione generata automaticamente

Spring ha molti automatismi che dovremo imparare. Ad esmepio posso creare delle Classi Componenti e Spring le scansioni, rilevando quali Classi debbano essere trattati come bean. Successivamente Spring li gestirà nello Spring Application Context (lo “scatolone”). Spring ha l’annotazione @Autowired che permette in maniera automatizzata di risolvere le dipendenze tra i bean.

Il vantaggio è che tutto ci rende più semplice rispetto al dover farlo manualmente.

Immagine che contiene testo, Carattere, algebra

Descrizione generata automaticamente

L’annotazione @Component dice che una particolare Classe farà parte dello “scatolone”.

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Immagine che contiene testo, software, Pagina Web, Sito Web

Descrizione generata automaticamente

Per utilizzare Spring è possibile installare una estensione di Eclipse.

Immagine che contiene testo, numero, software, schermata

Descrizione generata automaticamente

Immagine che contiene testo, software, schermata

Descrizione generata automaticamente

In alternativa (metodo usato da Riccardo): <https://start.spring.io/> che non dipende dal particolare IDE che uso. In questo caso scegliere MAVEN di default.

Bisogna dare un nome al Group (di solito nomeCognome), al Artifact e al Name (di solito uguali).

Posso inserire delle dipendenze (ADD DEPENDENCIES) ad esempio lombock.

Appena configurato -> GENERATE che crea un file .zip che contiene il progetto base pronto. Spacchetto .zip e lo importo su Eclipse(file->import->maven). Nel progetto che apro ci sono già una serie di cose: un main, un pom.xml ecc…

Mani sul codice:

Immagine che contiene testo, Software multimediale, software, Software per la grafica

Descrizione generata automaticamenteImmagine che contiene testo, software, Software multimediale, schermata

Descrizione generata automaticamenteCrea la Classe BackendStudent, la Classe FrontendStudent e la Classe FullStackStudent che implementano l’interfaccia IStudent,.

Crea Interviewer che ha come dipendenza IStudent.

Immagine che contiene testo, schermata

Descrizione generata automaticamenteTutti gli studenti possono rispondere (a prescindere dalla Classe (FS,BE, FE)). Questo spiega perché è meglio gestire le dipendenze con una interfaccia, proprio perché io posso dichiarare un metodo, ma non definirlo. La definizione sarà diversa a seconda della Classe ----Student che implemente IStudent.

Nel main:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Questo approccio è TRADIZIONALE.

Ovviamente ogni classe di studente andrà a fare @Override specifico del metodo dell’interfaccia.

Nel main dichiara 3 metodi per le 3 diverse configurazioni che possiamo creare (con @Configuration, con file.xml e con @Component)

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamenteImmagine che contiene testo, software, schermata

Descrizione generata automaticamenteA livello di bean posso farmi restituire delle stringhe, int, ecc…oltre che a bran di tipo IStudent o FSStudent.

Posso definire anche un nome specifico per ogni bean.

Immagine che contiene testo, Software multimediale, Software per la grafica, software

Descrizione generata automaticamente

Con questo metodo creo un riferimento allo “scatolone” Application Context grazie a ctx.

In questo modo recupero i bean che ho definito tali grazie a @Bean nella classe BeansConfiguration.

Ovviamente posso recuperare tutti gli altri bean che ho creato:

La configurazione mediante file.xml è molto obsoleta e superata.

Immagine che contiene testo, Software multimediale, software, Software per la grafica

Descrizione generata automaticamenteInfine possiamo utilizzare la metodologia del @Component: contrassegno con @Component tutte le classi le cui istanze voglio che vengano gestite da Spring. Nel metodo configWithCOmponent, mi creo il riferimento all’ application context tramite ctx, e: