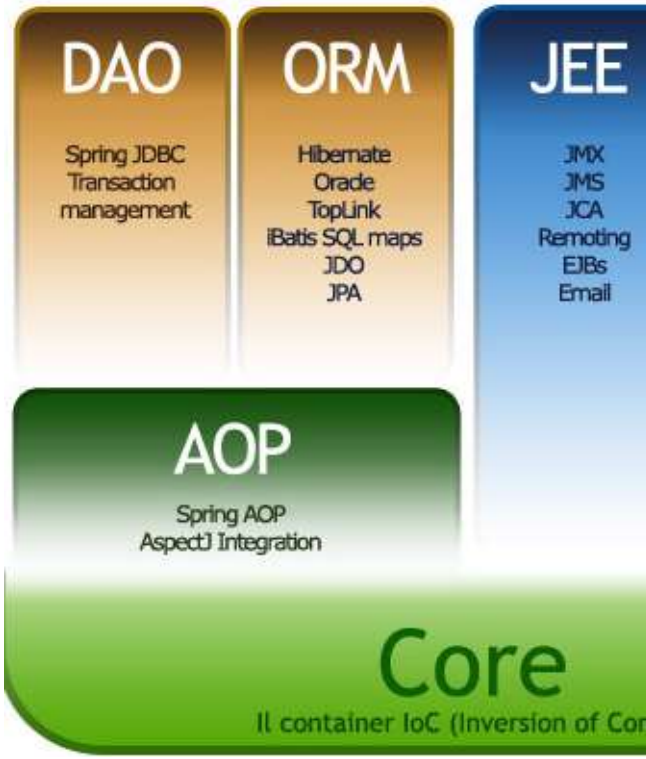
Java Spring

Spring

- Spring è un framework “**leggero**” e grazie alla sua architettura estremamente modulare é possibile utilizzarlo nella sua interezza o solo parte. L’adozione di Spring in un progetto è molto semplice, può avveni in maniera incrementale e non ne sconvolge l’architettura esistente. Questa sua peculiarità ne permette anche una facile integrazione con a framework esistenti, come ad esempio Struts.
- Spring è un lightweight container e si propone come alternativa/complemento a J2EE. A differenza di quest’ultimo, Spring propone un modello più semplice e leggero (soprattutto rispetto ad EJB per lo sviluppo di entità di business. Tale semplicità è rafforzata dall’util di tecnologie come l’Inversion of Control e l’Aspect Oriented che danno maggiore spessore al framework e favoriscono la focalizzazione dello sviluppatore sulle logica applicativa essenziale.
- A differenza di molti framework che si concentrano maggiormente nel

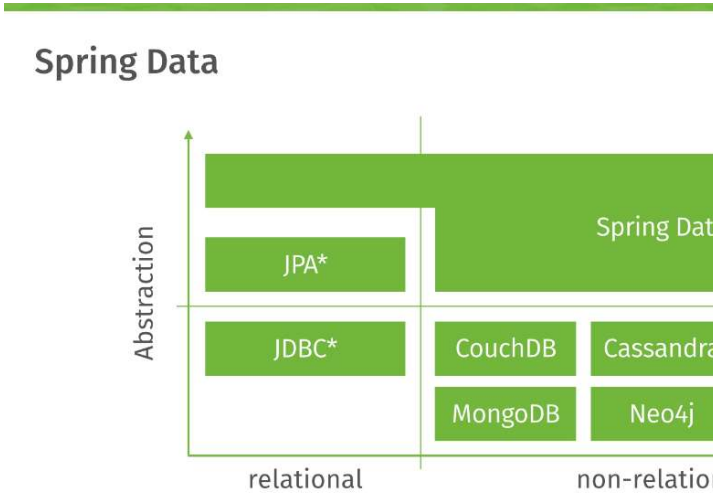
Spring



- architettura modulare
- I moduli di Core e Beans sono responsabili delle funzionalità di **Inversion Of Control (IoC**) e **Dependency Injection** ed hanno come compito principale la creazione, gestione e manipolazione di oggetti di qualsiasi natura che, in Spring, vengono detti beans.

- Il modulo context estende i servizi basilari del Core aggiungendo le funzionalità tipiche di un moderno framework. Tra queste troviamo JNDI, EJB, JMX, internazionalizzazione(I18N) e supporto

Data access, integration



- Fornisce un **livello di astrazione per l'accesso ai dati** mediante tecnologie eterogenee tra loro come ad esempio JDBC, Hibernate o JDO. Questo modulo tende a nascondere la complessità delle API di accesso ai dati, semplificando ed uniformando quelle che sono le problematiche legate alla gestione delle connessioni, delle transazioni e delle eccezzioni.
- Notevole attenzione è stata data all'integrazione del framework con i principali ORM in circolazione compresi JPA, JDO, Hibernate, e iBatis. Oltre a questo il Data Access si occupa di fornire supporto per il Java Message Service e

AOP

- Aggiunge al framework la funzionalità della **programmazione Aspect Oriented**. AOP offre un nuovo modo di programmare che come vedremo seguito porterà notevoli vantaggi in tutte quelle operazioni che sono trasversali tra più oggetti. In Spring l'utilizzo di AOP offre il meglio di sé nella **gestione delle transazioni**, permettendo di evitare l'utilizzo degli EJ tale scopo..

Web

- Come si intuisce dal nome, questo livello è responsabile delle caratteristiche Web del framework. Oltre alle funzionalità basilari, per la creazione di applicazioni Web, questo livello mette a disposizione un'implementazione del pattern MVC realizzando lo Spring MVC Framework (moduli Web-Servlet e Web-Portlet).

- Il framework MVC è ampiamente configurabile attraverso delle Strategy, può operare sia in contesti servlet che portlet e può essere utilizzato con numerose tecnologie di view comprese JSP e Velocity

Test

- Un’altra delle caratteristiche per il quale Spring si contraddistingue è il supporto al testing. Questo livello mette a disposizione un ambiente molto potente per il test delle componenti Spring, grazie anche alla sua integrazione con JUnit e TestNG e alla presenza di Mock objects per il testing del codice in isolamento.

Ambiente

- In particolare i componenti software utilizzati sono:
- **Java SDK** 1,8
- **Eclipse IDE**
- **Spring Framework** 5
- **Apache Derby**
- **Apache Tomcat** 10

- Spring https://spring.io/projects/

Installazione Spring

- Scaricare ed estrarre la cartella spring oppure con Maven o con Gradle
- In particolare l'archivio include tra le altre
  - la directory ***dist*** dove è presente il framework vero e proprio e
  - la directory ***lib*** dove sono contenute le dipendenze.
- Per utilizzare Spring, basterà semplicemente linkare nei propri progetti il **distspring.jar** e le opportune dipendenze necessarie di volta in volta.
- Per Eclipse abbiamo un plugin dal marketplace: http://marketplace.eclipse.org/content/spring-ide
- STS spring tools(IDE su base eclipse): https://spring.io/tools/sts/all

Installazione Apache Derby

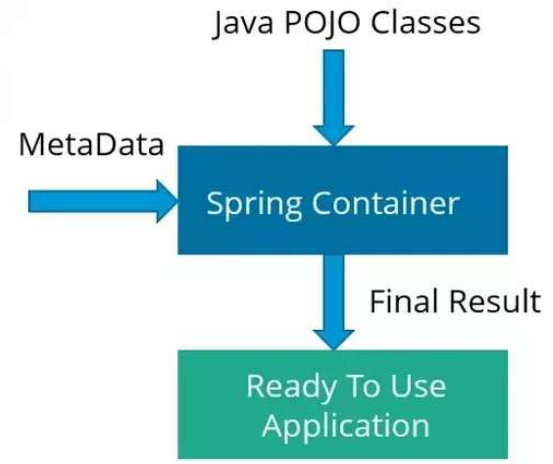
- Basandosi su JDBC per l'accesso ai dati, Spring può essere utilizzato con l maggior parte dei database presenti sul mercato e in particolare con tutt quelli che forniscono un driver per JDBC.
- http://db.apache.org/derby/derby_downloads.html
- L'installazione è molto semplice, una volta scaricato l'archivio zip conten l'ultima release del database dal sito del progetto (al momento la 10.5.3. basterà scompattarlo in una directory a scelta (ad esempio c:db-derby). Dopodiché sarà sufficiente creare la variabile d'ambiente DERBY_HOME impostando come valore il percorso della directory di Derby e aggiunger alla variabile d'ambiente PATH il valore %DERBY_HOME%bin.

Installazione Apache Tomcat

- Per poter testare la parte della guida inerente Spring MVC è necessario installare un web container. Grazie alla sua semplicità Apache Tomcat è l'ideale per questo scopo, anche se per applicazioni più complesse si suggerisce l'utilizzo di un application server come ad esempio JBoss AP.
- Per installare Apache Tomcat è sufficente eseguire il file d'installazione reperibile sul sito ufficiale. http://tomcat.apache.org/

IoC

- **L'Inversion of Control** è un principio architetturale nato alla fine degli an ottanta, basato sul concetto di invertire il controllo del flusso di sistema (Control Flow) rispetto alla programmazione tradizionale. Questo princip molto utilizzato nei framework e ne rappresenta una delle caratteristiche basilari che li distingue dalle API.
- Nella programmazione tradizionale la logica di tale flusso è definita esplicitamente dallo sviluppatore, che si occupa tra le altre cose di tutte operazioni di **creazione**, **inizializzazione** ed **in** dei metodi degli oggetti.
- IoC invece inverte il control flow facendo in m o sviluppatore a doversi preoccupare di questi ork, ch reagendo a qualche "stimolo" se ne occuperà o prin è anche conosciuto come Hollywood Princip hiam noi").



Spring Ioc +DI

dependency between the Employee and Address (tight coupling)

```
Class Employee{

Address address;

Employee(){

address=new Address();

} }
```

```
class Employee{
Address address;
Employee(Address address){
this.address=address;
} }
```

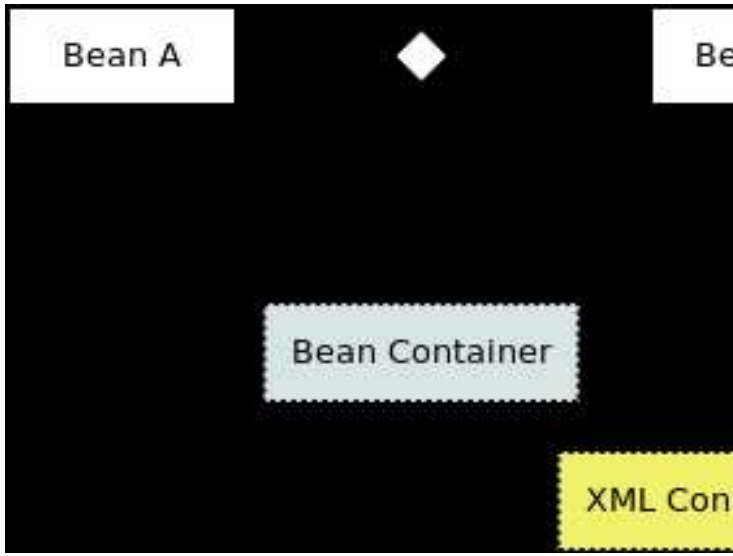
In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC co by XML file or annotation.

Advantage:

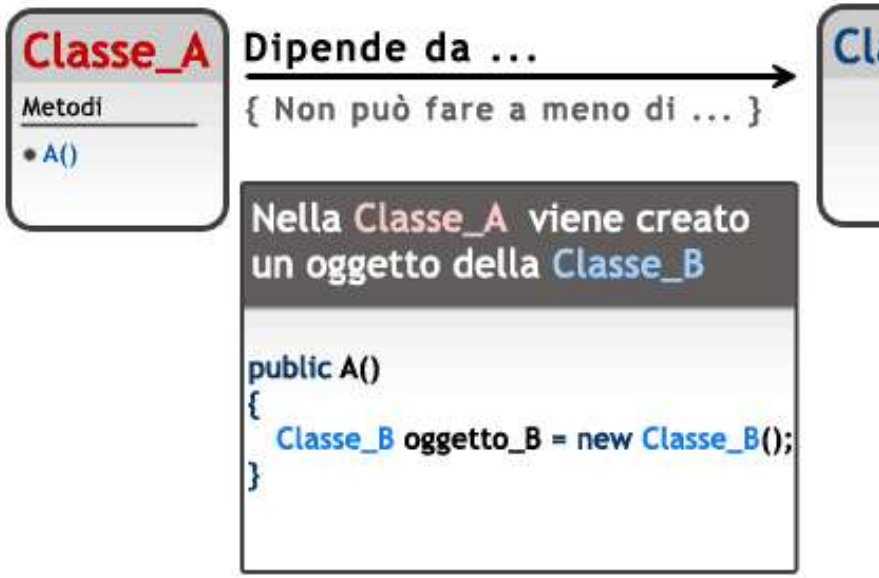
- makes the code loosely coupled so easy to maintain
- makes the code easy to test

Dependency Injection

- **Dependency Injection (DI)** per riferirsi ad una specifica implementazione
  IoC rivolta ad invertire il processo di risoluzione delle dipendenze, facend
  modo che queste vengano iniettate dall'esterno. Banalmente, nel caso d
  programmazione Object Oriented, una classe A si dice dipendente dalla
  se ne usa in qualche punto i servizi offerti.



IoC e DI



- Perché questo tipo di collaborazione abbia luogo la classe A ha diverse alternative:
  - istanziare e inizializzare (attraverso costruttore o metodi setter) la classe B,
  - ottenere un'istanza di B attraverso una factory oppure effettuare un lookup attra servizio di naming (es JNDI).
  - Ognuno di questi casi implica che nel codice di A sia "scolpita" la logica di risoluzi dipendenza verso B.
- Per chiarire introduciamo un esempio. Si vuole creare un semplice gener report in grado di generare output in formato testuale.

IoC e Dependency Inj

```
public class TxtReport {
  public void generate(String data) {
    System.out.println("genera txt report");
  } }
```

```
public class ReportGenerator {
  TxtReport report = null;
  public Report generate(String data) {
    report = new TxtReport();      // risoluzione della dipendenza
    report.generate(data);
    return report;
```

La classe ReportGenerator ha una di
verso TxtReport e questa è risolta ne
metodo generate().
Questo modo di operare, oltre a vinc
creazione della dipendenza nel codic
il riuso (cosa succede se in futuro vo
generare report in formato HTML?),
generare un forte accoppiamento tra
(**Coupling**)

Il problema risiede nel fatto che senz
apposito sistema la risoluzione delle
ad esclusivo appannaggio delle class
dovranno preoccuparsi di creare gli o
dipendono o di ottenerne delle refer
attraverso operazioni di lookup.

IoC e Dependency Injection

- L’idea alla base della Dependency Injection è quella di avere un compone esterno (assembler) che si occupi della creazione degli oggetti e delle lor relative dipendenze e di assemblarle mediante l’utilizzo dell’injection. In particolare esistono tre forme di injection:
  - **Constructor** Injection, dove la dipendenza viene iniettata tramite l’argomento de costruttore
  - **Setter** Injection, dove la dipendenza viene iniettata attraverso un metodo “set”
  - **Interface** Injection che si basa sul mapping tra interfaccia e relativa implementaz (non utilizzato in Spring)
- L’iniezione di dipendenza può essere realizzata in molteplici modi, tra cui più semplice consiste nell’utilizzo di una **factory**. Riprendiamo ora l’esem del generatore di report e cerchiamo di capire come realizzare una semp iniezione della dipendenza.

IoC e Dependency Injection

Creiamo un'interfaccia Report che definisce le operazioni di base che un deve avere e facciamola implementare alla classe TxtReport:

```
public interface Report {          // Interfaccia Report
  public void generate(String data);
  public void saveToFile();
}
public class TxtReport implements Report {      // Classe TxtReport
  String path;
  public TxtReport(String path) { this.path = path; }
  public void generate(String data) {
    System.out.println("genera txt report");
```

IoC e Dependency Injection

Modifichiamo il ReportGenerator in modo che sia in grado di utilizzare og
tipo Report e aggiungiamo il relativo metodo setter:

```
public class ReportGenerator {
  Report report;
  public Report generate(String data) {
    // report = new TxtReport();
    report.generate(data);
    return report;
  }
  public void setReport (Report report) {
```

IoC e Dependency Injection

Creiamo infine una classe factory che avrà il compito di istanziare oggetti ReportGenerator e di risolverne le dipendenze:

```
public class ReportGeneratorFactory {
  public static ReportGenerator createTxtReportGenerator() {
    ReportGenerator rg = new ReportGenerator();
    rg.setReport(new TxtReport());  // risoluzione della dipendenza
    return rg;
  }
}
```

IoC e Dependency Injection

Il client della nostra applicazione si presenterà in questo modo:

```
public class ReportClient {
  public static void main(String[] args) {
    String data = null;        // reperimento dati
    ReportGenerator gen = ReportGeneratorFactory.createTxtReportGener
    gen.generate(data).saveToFile();
  }
}
```

La soluzione proposta ci ha permesso di disaccopiare le classi ReportGen
TxtGenerator attraverso l'utilizzo della classe ReportGeneratorFactory. In

IoC e Dependency Injection

Pur essendo efficace, questo tipo “manuale” di DI continua ancora a non risolvere il problema di avere scolpito nel codice la creazione del report. I problema è infatti stato semplicemente trasferito nella **classe factory**, che essere ogni volta modificata se si vorrà cambiare il tipo di report da utiliz

Un modo migliore per poter lavorare con la Dependency Injection è quell utilizzare come assembler uno **IoC Container** in grado di compiere operaz injection.

Per definizione un container è un componente esterno che si prende cari una serie di compiti esonerando così lo sviluppatore dal preoccuparsene. IoC Container non è altro che un container specializzato nella dependenc injection, che basandosi su apposite configurazioni definite dall’utente (generalmente attraverso l’utilizzo di un file xml) è in grado di compiere opportune operazioni di injection.

IoC Container

Considerato il cuore di Spring, lo **IoC Container** fornisce un contesto altar
configurabile per la creazione e risoluzione delle dipendenze di compone
qui vengono chiamati **bean** (da non confondere con i JavaBean).

in Spring un bean non deve aderire a nessun tipo di contratto e può esse
rappresentato da una qualunque classe Java.

Tecnicamente parlando, lo IoC Container è realizzato da due interfacce:

- **BeanFactory**, che definisce le funzionalità di base per la gestione dei be
- **ApplicationContext**, che estende queste funzionalità basilari aggiungen
  altre tipicamente enterprise come ad esempio la gestione degli eventi,
  l'internazionalizzazione e l'integrazione con AOP

IoC Container

Di seguito analizzeremo le funzionalità di base fornite dallo IoC Container da BeanFactory, tenendo presente che tutto ciò che verrà detto è valido a per ApplicationContext.

L'interfaccia BeanFactory rappresenta la forma più semplice di IoC Conta Spring e ha il compito di:

- creare i bean necessari all'applicazione
- inizializzare le loro dipendenze attraverso l'utilizzo dell'injection
- gestirne l'intero ciclo di vita

Per svolgere questi compiti, il container si appoggia a configurazioni impo dall'utente che, riflettendo lo scenario applicativo, specificano i bean che dovranno essere gestiti dal container, le dipendenze che intercorrono tra oltre alle varie configurazioni specifiche

XmlBeanFactory

In Spring esistono diverse implementazioni di BeanFactory, la più comune
quali è senza dubbio la **XmlBeanFactory** che permette di utilizzare uno o
XML per descrivere la configurazione da utilizzare. I file di configurazione
XmlBeanFactory hanno la seguente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/bea
                http://www.springframework.org/schema/beans/spring-b
3.0.xsd">
 <bean id="   " class="   ">
```

XmlBeanFactory

Dopo l'intestazione del file XML, racchiuse tra i tag <beans>, troviamo le definizioni dei bean e, per ognuno di questi, le eventuali proprietà che ne descriveranno la struttura e il comportamento. Osserviamone alcune:

l'**Id** associato al bean per essere richiamato tramite invocazioni al contair

Il nome della classe che implementa il bean in caso questa venga istanzia direttamente o della relativa factory che si occuperà della sua creazione

Risoluzione di dipendenze attraverso metodi setter o costruttori appositi

Proprietà comportamentali che definiscono come il bean deve essere tra container (scope, ciclo di vita, etc.)

XmlBeanFactory

IoC Container di Spring. I nostri **bean** sono rappresentati dalle classi **TxtR**
ReportGenerator visti in precedenza:

```
// Classe TxtReport
public class TxtReport implements Report {
  String path;
  public TxtReport(String path) {
     this.path = path;
}
  public void generate(String data) {
    System.out.println("genera txt report");
```

```
// Classe ReportGenerator
public class ReportGenerator {
  Report report;
  public Report generate(String data) {
    // report = new TxtReport();
    report.generate(data);
    return report;
  }
  public void setReport (Report report
    this.report = report;
  }
```

XmlBeanFactory

- **Creiamo un file bean.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/scher
http://www.springframework.org/schema/beans/spring-beans-3.

  <!-- Indica che il bean 'report' è implementato da TxtReport -->
  <bean id="report" class="it.spring.report.TxtReport">
    <!-- imposta il parametro per il costruttore-->
    <constructor-arg value="/report" />
  </bean>

  <!-- associa il bean 'reportGenerator' al nostro ReportGenerator
  <bean id="reportGenerator" class="it.spring.report.ReportGener
    <!-- indica al setter 'report' del reportGenerator di riferirsi ad og
    <!-- istanziati con il bean 'report' (quindi, in questo caso, TxtRep
    <property name="report" ref="report" />
  </bean>

</beans>
```

XmlBeanFactory

Analizzando il file di configurazione possiamo notare che è stato chiesto al container di gestire due classi:

- it.spring.report.ReportGenerator (con id reportGenerator)
- it.spring.report.TxtReport (con id report).

sono state anche inserite le direttive per indicare al container come risolvere le dipendenze di queste classi:

- Al bean report è stata iniettata, tramite una injection di tipo constructor, una stringa indicante il percorso in cu report
- A ReportGenerator è stato iniettato il bean report, con una injection di tipo setter.
- Infine istanziamo il container e vediamo come interagire con esso per utilizzare i bean configurati:

```
public class ReportClient {
  public static void main(String[] args) {
    String data = null;       // ...  reperimento dati
    BeanFactory ctx = new XmlBeanFactory(new ClassPathResource("beans.xml"));
    ReportGenerator gen = (ReportGenerator) ctx.getBean("reportGenerator");
    gen.generate(data);
  }
}
```

XmlBeanFactory

Una volta istanziato il container è possibile utilizzarlo alla stregua di una factory (da qui il nome BeanFactory) repe
attraverso il metodo getBean (nomebean).

l'esempio completo (progetto Eclipse) Esempio_01.zip

Come si può ben capire questo modo di operare ci offre enormi vantaggi in termini di riuso dei bean. Se in seguit
necessario di un nuovo report che produca file in formato HTML basterà semplicemente scrivere una classe che a
all'interfaccia Report e modificare il bean corrispondente nel file di configurazione.

Accesso ai dati

- Spring supporta:
  - JDBC
  - Java Persistence API (JPA)
  - Java Data Objects (JDO)
  - Hibernate
  - Common Client Interface (CCI)
  - iBATIS SQL Maps
  - Oracle TopLink

Accesso ai dati

- Creiamo il nostro modello da persistere: è composto da una semplice c
  Book contenente le proprietà necessarie per rappresentare un libro e c
  relativi metodi accessori.

```
public class Book {
  String isbn;
  String author;
  String title;
  public String getIsbn() { return isbn; }
  public void   setIsbn(String isbn) { this.isbn = isbn; }
  public String getAuthor() { return author; }
```

Accesso ai dati

La persistenza del modello sarà effettuata nella tabella books del databa
library; di seguito i dettagli di connessione e creazione della tabella.

```
CONNECT 'jdbc:derby://localhost:1527/library;create=true';


CREATE TABLE BOOKS ( ISBN VARCHAR(13) NOT NULL,
                     AUTHOR VARCHAR(20),
                     TITLE VARCHAR(20),
                     PRIMARY KEY (ISBN));
```

Accesso ai dati

Una possibile scelta è Apache Derby, un database open source estremam
leggero e di facile configurazione.

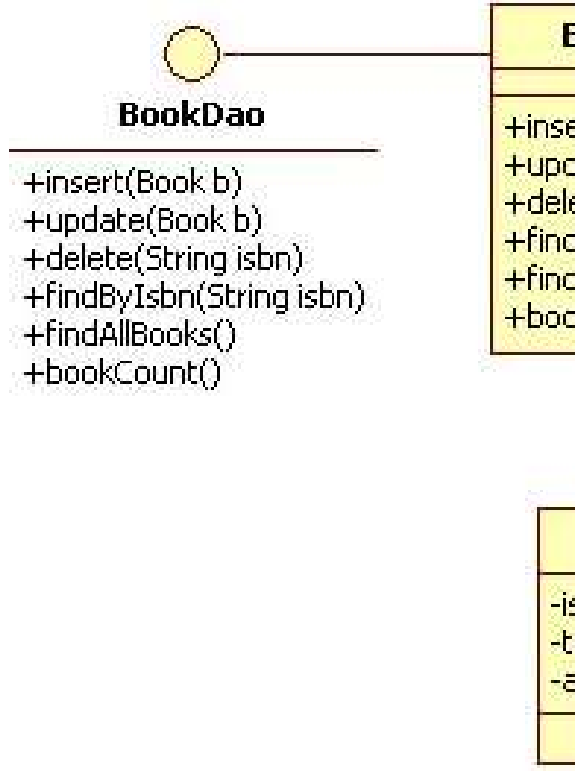
Per avviare il server basterà semplicemente eseguire il file
startServerNetwork.bat presente nella sottodirectory bin del path dove è
installato Apache.

A questo punto sarà possibile aprire la console(Apache Derby Console)
applicativa attraverso il comando ij, da dove lanciare i comandi mostrati i

```
C:\>ij
Versione ij 10.5
ij> CONNECT 'jdbc:derby://localhost:1527/library;create=true';
ij> CREATE TABLE BOOKS ( ISBN VARCHAR(13) NOT NULL, AUTHOR VAR
CHAR(20), TITLE VARCHAR(20), PRIMARY KEY (ISBN));
0 righe inserite/aggiornate/eliminate
ij>
```

Accesso ai dati

Una delle peculiarità principali di Spring è quella di favorire la scrittura di codice modulare favorendone il riuso. Per questa ragione Spring incoraggia l'utilizzo del **DAO** (Data Access Object), un pattern architetturale che ha come scopo quello di separare le logiche di business da quelle di accesso ai dati.



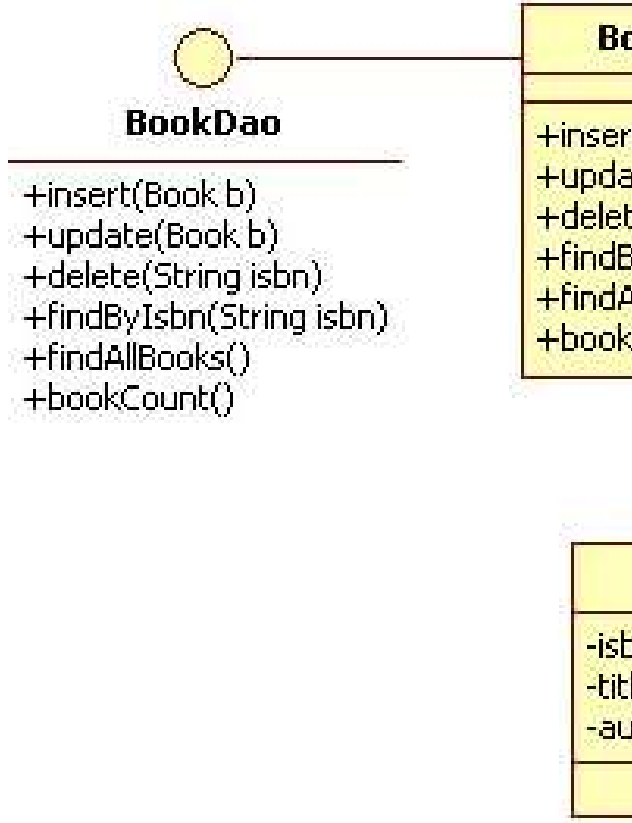
L'idea alla base di questo pattern è quello di descrivere le operazioni necessarie per la persistenza del modello in un'interfaccia e di implementare la logica specifica di accesso ai dati in apposite classi.

Accesso ai dati

La logica di business necessaria per la persistenza del nostro modello è descritta nella classe **BookDao**, mentre nella classe **BookJdbcDao** troviamo un'implementazione specifica di questa interfaccia rivolta a gestire le logiche di accesso ai dati mediante tecnologia JDBC.

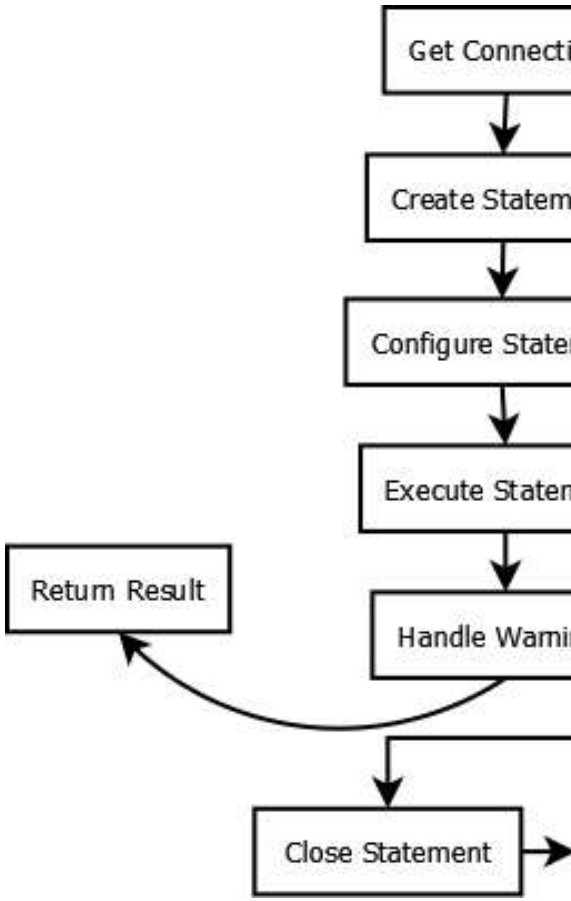
In questo scenario è possibile notare come grazie all'utilizzo del pattern DAO sia possibile con poco sforzo fornire ulteriori implementazioni della classe BookDao per introdurre nuove logiche di accesso ai dati (ad esempio per l'utilizzo di Hibernate un ipotetico BookHibernateDao).

L'interfaccia DAO che utilizeremo sarà così fatta:



Spring JDBC

La forma primaria di accesso ai dati in Java è senza dubbio **JDBC**, un'API creata per rendere platform-independent l'interazione con i RDBMS. Il suo utilizzo offre notevoli vantaggi in termini di performances e facilità di utilizzo, richiedendo per contro allo sviluppatore di farsi carico dell'intera gestione dei processi di accesso ai dati, ovvero:



- Richiesta della connessione al **datasource**
- Creazione del **PreparedStatement** e valorizzazione dei parametri
- Esecuzione del PreparedStatement
- Estrazione dei risultati dal **ResultSet** (in caso di

Spring Jdbc

- Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

- Problems of JDBC API
- The problems of JDBC API are as follows:

- We need to write a lot of code before and after executing the quer such as creating connection, statement, closing resultset, connecti etc.

Spring Jdbc

- Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

- Problems of JDBC API
- The problems of JDBC API are as follows:

- We need to write a lot of code before and after executing the quer such as creating connection, statement, closing resultset, connecti etc.

Spring Jdbc Approaches

- Spring framework provides following approaches for JDBC databas access:

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcTemplate
- SimpleJdbcInsert and SimpleJdbcCall

Spring JdbcTemplate class

- It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of conne object etc. So it will not lead to any problem if you forget to close the connection.
- It handles the exception and provides the informative exception messa the help of excepion classes defined in the org.springframework.dao pa
- We can perform all the database operations by the help of JdbcTempla such as insertion, updation, deletion and retrieval of the data from the database.
- Let's see the methods of spring JdbcTemplate class.

- No. Method Description

Spring JdbcTemplate class

- It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of conne object etc. So it will not lead to any problem if you forget to close the connection.
- It handles the exception and provides the informative exception messa the help of excepion classes defined in the org.springframework.dao pa
- We can perform all the database operations by the help of JdbcTempla such as insertion, updation, deletion and retrieval of the data from the database.
- Let's see the methods of spring JdbcTemplate class.

- No. Method Description

Spring JdbcTemplate example

We are assuming that you have created the following table inside the Or
database.

create table **employee**(

id number(10),

name varchar2(100),

salary number(10)

);

Employee.java

This class contains 3 properties with constructors and setter and getters.

Spring JdbcTemplate example

```
import org.springframework.jdbc.core.JdbcTemplate;
public class EmployeeDao {
private JdbcTemplate jdbcTemplate;
public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}
public int saveEmployee(Employee e){
    String query="insert into employee values(
    '"+e.getId()+"','"+e.getName()+"','"+e.getSalary()+"')";
    return jdbcTemplate.update(query);
```

Spring JdbcTemplate example

Applicationcontext.xml

The DriverManagerDataSource is used to contain the information about th
database such as driver class name, connnection URL, username and pass

There are a property named datasource in the JdbcTemplate class of
DriverManagerDataSource type. So, we need to provide the reference of
DriverManagerDataSource object in the JdbcTemplate class for the dataso
property.

Here, we are using the JdbcTemplate object in the EmployeeDao class, so
pass

Spring JdbcTemplate example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
   xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">


<bean id="ds"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDrive
```

Spring JdbcTemplate example

Test.java

This class gets the bean from the applicationContext.xml file and calls the saveEmployee() method. You can also call updateEmployee() and deleteEmployee() method by uncommenting the code as well.

Spring JdbcTemplate example

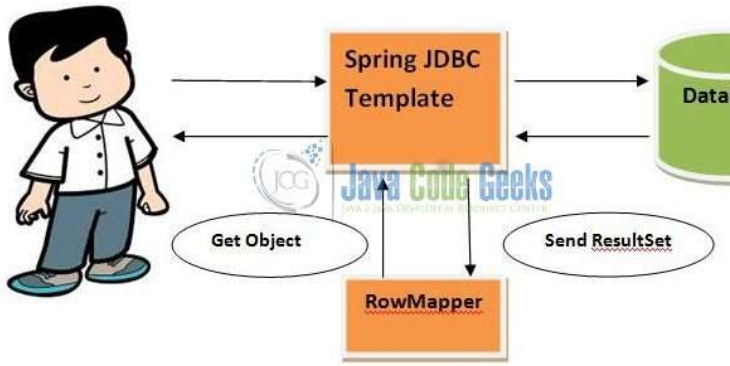
Test.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationC
public class Test {

public static void main(String[] args) {
    ApplicationContext ctx=new
ClassPathXmlApplicationContext("applicationContext.xml");

    EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
```

Spring JDBC Template



Spring offre diverse possibilità per la persistenza dei dati mediante JDBC, la
principale delle quali è l'utilizzo della **classe JDBCTemplate**.

Questa classe implementa l'intero processo di accesso ai dati attraverso **te
methods**, rendendo possibile la personalizzazione di tutte le fasi di tale pr
mediante l'ovverride dei metodi specifici.

Vediamo ora attraverso degli snippet di codice come eseguire operazioni s
Database mediante JDBCTemplate.

Iniziamo con la creazione dell'implementazione specifica per JDBC del **Boo
visto prima. Come è possibile notare dal codice, questo possiede un riferim
alla classe JDBCTemplate, inizializzata attraverso il metodo setDataSource(
provvede a fornire il DataSource necessario.

Spring JDBC Template: Update

Attraverso il metodo **update** è possibile eseguire tutte quelle operazioni che comportano una modifica dei dati nel database.

```
public class BookJdbcDao implements BookDao {
  // ...
public void insert(Book book)    //Inserimento
  {
    jdbcTemplate.update("insert into books (isbn, autore, titolo) values (?, ?, ?)",
                        new Object[] { book.getIsbn(), book.getAutore(), book.getTitolo() });
  }
```

Spring JDBC Template: Ricerca

Passando alle interrogazioni, questo è il caso di ricerca di un intero.

```
public class BookJdbcDao implements BookDao {
  // ...
  // Query di un intero
  public int bookCount() {
        int rowCount = jdbcTemplate.queryForInt("select count(1) from books");
        return rowCount;
      }
}
```

Spring JDBC Template: Ricerca

```
public class BookJdbcDao implements BookDao {

  ...

public Book findByISBN(String isbn) {      //Query di un singolo oggetto

    Book book = (Book) jdbcTemplate.queryForObject("select * from books
where isbn = ?«,   new Object[] { isbn }, new BookRowMapper());

    return book;

  }

public List<Book> findAllBooks() {      // Query di una lista

    List<Book> books = (List<Book>) jdbcTemplate.query("select * from
books", new BookRowMapper());

    return books;

                        }
```

la classe BookRowMapper contiene le reg
tra una riga del ResultSet e classe Book ec
jdbcTemplate per ritornare i valori della q

Jdbc Template Injection

Grazie all'utilizzo **dell'Inversion of Control** è possibile iniettare nella nostra classe un oggetto di tipo **JdbcTemplate** già inizializzato con un datasource. Esaminiamo il **file di configurazione**:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  xsi:schemaLocation="http://www.springframework.org/schema/beans    http://www.springframework.org/schema/beans/spring-beans-2.0.xs

  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.apache.derby.jdbc.ClientDriver" />
    <property name="url" value="jdbc:derby://localhost:1527/library;create=true" />
    <property name="username" value="app" />
    <property name="password" value="app" />
  </bean>

  <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource" />
  </bean>

  <bean id="bookDao" class="it.spring.book.BookJdbcDao">
```

Jdbc Template Injection

La classe **JdbcDaoSupport** permette di agevolare l'utilizzo di JdbcTemplate implementando direttamente i metodi setDatasour
setJdbcTemplate().

Facendo estendere questa classe ai propri DAO sarà sufficente iniettare un datasource per poter utilizzare un JdbcTemplate att
metodo **getJdbcTemplate**() (ereditato anch'esso da **JdbcDaoSupport**).

```
public class BookJdbcDao extends JdbcDaoSupport implements BookDao {

  ...


  //Inserimento

  public void insert(Book book)

  {

    getJdbcTemplate().update("insert into book (isbn, autore, titolo) values (?, ?, ?)",

                    new Object[] { book.getIsbn(), book.getAuthor(), book.getTitle() });
```

Simple Jdbc Template

La classe **SimpleJdbcTemplate** aggiunge alcune funzionalità offerte da Java 1.5 alla classe base JdbcTemplate, semplificando ul
procedure di accesso ai dati. Grazie al supporto dei parametri a lunghezza variabile è possibile passare direttamente i valori ai
senza dover utilizzare un array di oggetti.

```
public class BookJdbcDaoSupport extends SimpleJdbcDaoSupport implements BookDao {

  public void insert(Book book) {

    getSimpleJdbcTemplate().update("insert into book (isbn, autore, titolo) values (?, ?, ?)", book.getIsbn(), book.getAutore(), bo

    ...

  }
```

Con l'utilizzo dei generics e dell'autoboxing anche le query vengono notevolmente semplificate.

Simple Jdbc Template

In particolare Spring fornisce la classe **ParameterizedBeanPropertyRowMapper**, un'implementazione dell'interfaccia RowMapp
di effettuare le operazioni di mapping viste in precedenza in maniera transparente allo sviluppatore.

```
public Book findByISBN(String isbn) {                    //Query di un singolo oggetto

  Book book = getSimpleJdbcTemplate().queryForObject("select * from books where isbn =?",
ParameterizedBeanPropertyRowMapper.newInstance(Book.class), isbn);

  return book;

}
```

```
public List<book> findAllBooks() {        // Query di una lista di oggetti

  List<book> books = (List<book>) getJdbcTemplate().query( "select * from
books",ParameterizedBeanPropertyRowMapper.newInstance(Book.class));

  return books;

}
```