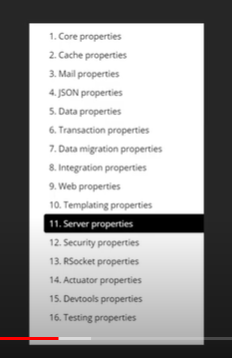
Application.properties file //// application.yml



**ANNOTATIONS:**

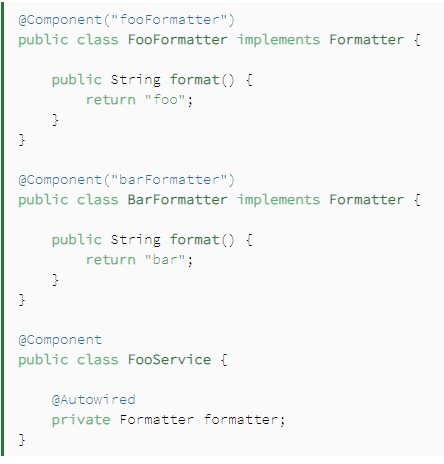
**The Spring @Qualifier Annotation:**

Autowire Need for Disambiguation

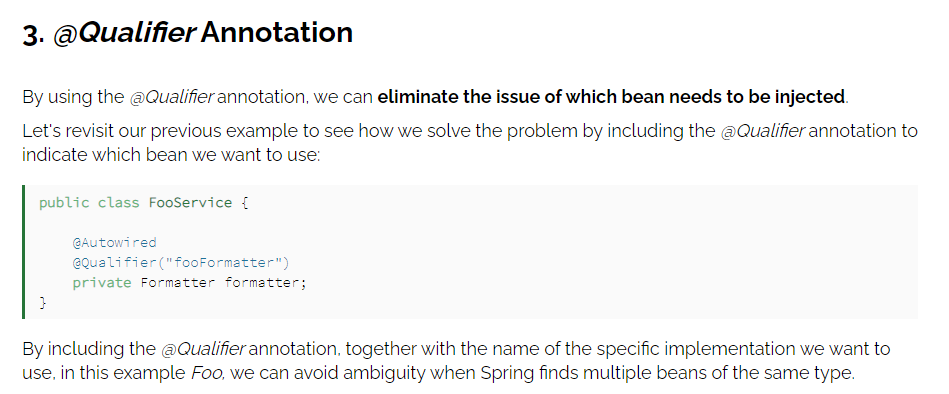
The [*@Autowired*](https://www.baeldung.com/spring-autowire) annotation is a great way of making the need to inject a dependency in Spring explicit. Although it's useful, there are use cases for which this annotation alone isn't enough for Spring to understand which bean to inject.

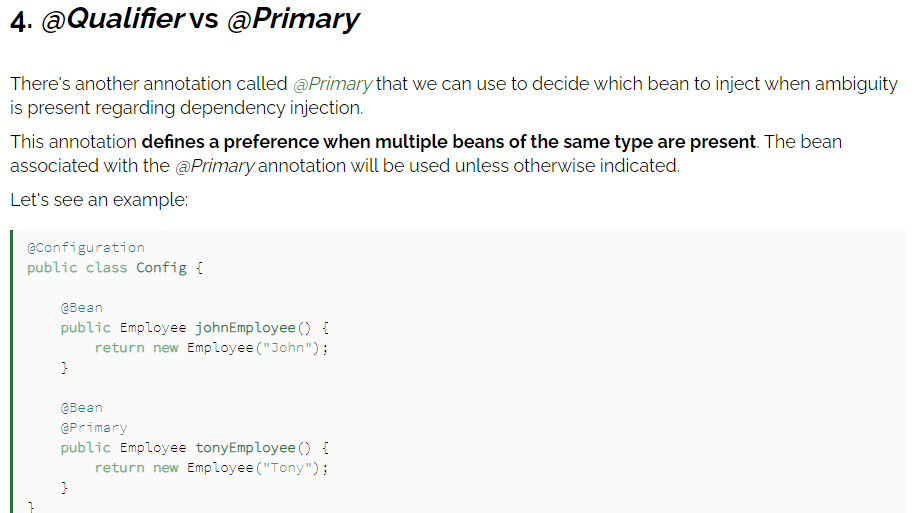
By default, Spring resolves autowired entries by type.

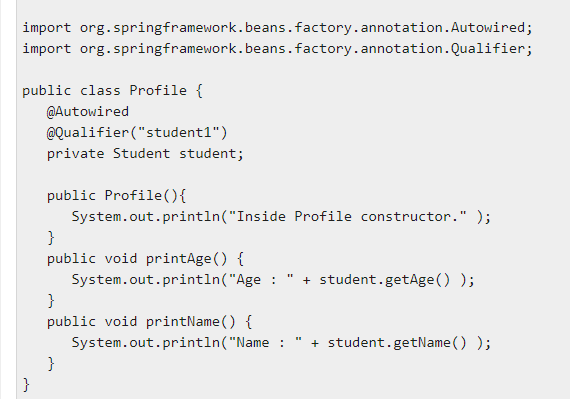
**If more than one bean of the same type is available in the container, the framework will throw *NoUniqueBeanDefinitionException****,* indicating that more than one bean is available for autowiring.

****

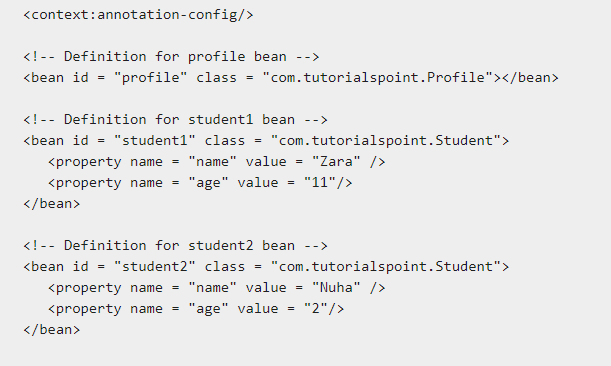
If we try to load FooService into our context, the Spring framework will throw a NoUniqueBeanDefinitionException. This is because **Spring doesn't know which bean to inject**. To avoid this problem, there are several solutions; the @Qualifier annotation is one of them.

****

****

****

**Beans.xml**

****

**@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

**@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

**@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

**@SpringBootApplication=@ComponentScan+@EnableAutoConfiguration+@Configuration**

**@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

## spring Boot Annotations

* **@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. **@SpringBootApplication**.
* **@SpringBootApplication:** It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.
* **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
* **@PostMapping:** It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
* **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
* **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**
* **@PatchMapping:** It maps the **HTTP PATCH**requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**
* **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
* **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
* **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
* **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
* **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.**For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
* **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody**annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
* **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

**Spring JPA**

# **JPA Introduction**

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence

# **JPA Object Relational Mapping**

Object Relational Mapping (ORM) is a functionality which is used to develop and maintain a relationship between an object and relational database by mapping an object state to database column. It is capable to handle various database operations easily such as inserting, updating, deleting etc.



## ORM Frameworks

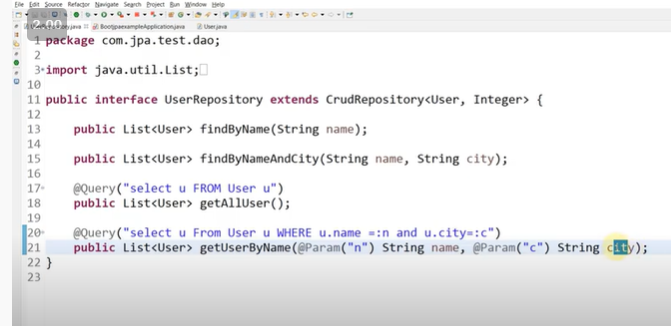
Following are the various frameworks that function on ORM mechanism: -

* Hibernate
* TopLink
* ORMLite
* iBATIS
* JPOX

# **Spring Boot JPA**

## What is JPA?

**Spring Boot JPA**is a Java specification for managing **relational** data in Java applications. It allows us to access and persist data between Java object/ class and relational database. JPA follows **Object-Relation Mapping**(ORM). It is a set of interfaces. It also provides a runtime **EntityManager** API for processing queries and transactions on the objects against the database. It uses a platform-independent object-oriented query language JPQL (Java Persistent Query Language

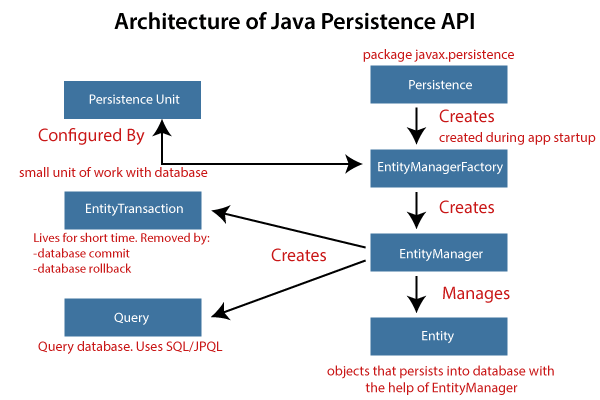
****

## JPA Architecture

JPA is a source to store business entities as relational entities. It shows how to define a POJO as an entity and how to manage entities with relation.

The following figure describes the class-level architecture of JPA that describes the core classes and interfaces of JPA that is defined in the **javax persistence** package. The JPA architecture contains the following units:

* **Persistence:** It is a class that contains static methods to obtain an EntityManagerFactory instance.
* **EntityManagerFactory:** It is a factory class of EntityManager. It creates and manages multiple instances of EntityManager.
* **EntityManager:** It is an interface. It controls the persistence operations on objects. It works for the Query instance.
* **Entity:** The entities are the persistence objects stores as a record in the database.
* **Persistence Unit:** It defines a set of all entity classes. In an application, EntityManager instances manage it. The set of entity classes represents the data contained within a single data store.
* **EntityTransaction:** It has a **one-to-one** relationship with the EntityManager class. For each EntityManager, operations are maintained by EntityTransaction class.
* **Query:** It is an interface that is implemented by each JPA vendor to obtain relation objects that meet the criteria.



## JPA Implementations

JPA is an open-source API. There is various enterprises vendor such as Eclipse, RedHat, Oracle, etc. that provides new products by adding the JPA in them. There are some popular JPA implementations frameworks such as **Hibernate, EclipseLink, DataNucleus,** etc. It is also known as **Object-Relation Mapping** (ORM) tool.

## Object-Relation Mapping (ORM)

In ORM, the mapping of Java objects to database tables, and vice-versa is called **Object-Relational Mapping.** The ORM mapping works as a bridge between a **relational database** (tables and records) and **Java application** (classes and objects).

In the following figure, the ORM layer is an adapter layer. It adapts the language of object graphs to the language of SQL and relation tables.

## Spring Boot jpa

# **Spring Boot H2 Database**

## What is the in-memory database

In-memory database relies on system memory as oppose to disk space for storage of data. Because memory access is faster than disk access. We use the in-memory database when we do not need to persist the data. The in-memory database is an embedded database. The in-memory databases are volatile, by default, and all stored data loss when we restart the application.

The widely used in-memory databases are **H2, HSQLDB**(HyperSQL Database)**,**and**Apache Derby.**It creates the configuration automatically.

## What is the H2 Database

**H2** is an **embedded, open-source,**and**in-memory** database. It is a relational database management system written in [Java](https://www.javatpoint.com/java-tutorial). It is a **client/server** application. It is generally used in **unit testing**. It stores data in memory, not persist the data on disk.

**Advantages**

* Zero configuration
* It is easy to use.
* It is lightweight and fast.
* It provides simple Configuration to switch between a real database and in-memory database.
* It supports standard SQL and JDBC API.
* It provides a web console to maintain in the database

# **Spring Boot Thymeleaf**

## What is Thymeleaf?

The**Thymeleaf** is an open-source Java library that is licensed under the **Apache License 2.0**. It is a **HTML5/XHTML/XML** template engine. It is a **server-side Java template**engine for both web (servlet-based) and non-web (offline) environments. It is perfect for modern-day HTML5 JVM web development. It provides full integration with Spring Framework.