**Spring**

* A powerful, open-source, application framework to reduce the complexity of enterprise application development
* It has layered architecture and loosely coupled
* Enables POJO programming that aids in continuous integration and testability
* Supports AOP
* Spring container – creates and manages the life cycle of application objects (components)
* Spring modules:



* Spring container creates objects, wires them together, configures them and manages their complete life cycle
* Spring configuration information configures the spring container. These metadata are provided in form of:
  + XML
  + Java Annotations
  + Java code
* DI in spring is done in two ways:
  + Constructor injection
  + Setter injection
* IoC containers in Spring
  + BeanFactory
  + ApplicationContext (built over bean factory plus some other features)
* How configuration metadata is provided to container
  + XML based configuration
  + Annotation based configuration
  + Java based configuration
* Bean property values can be

* + value (in case of simple primitive values or String)
  + ref (in case it is an object)
  + Spring configurations can be split into several XMLs using import tag
    - <import resource=”service-layer-config.xml” />
* Five bean scopes are supported by Spring
  + Singleton
  + Prototype
  + Request
  + Session
  + Global-session
* Bean life cycle



* Autowiring has the following modes
  + no (the default)
  + byName
  + byType
  + constructor
  + autodetect

* To turn on annotation based wiring , you must add the following line in beans definition XML as
  + <context: annotation-config />

* Bean initialization types
  + If lazy-init is false (default) bean is loaded when application comes up
  + If true, bean is loaded when first explicit call is made to it
* Main annotations for spring beans



* @Required annotation

* + Is applied to bean property setter methods
  + Should be populated at configuration time with an explicit property value
* @Qualifier

* + To remove ambiguity between beans of same types to be used

* @RequestMapping
  + Class level
  + Method level

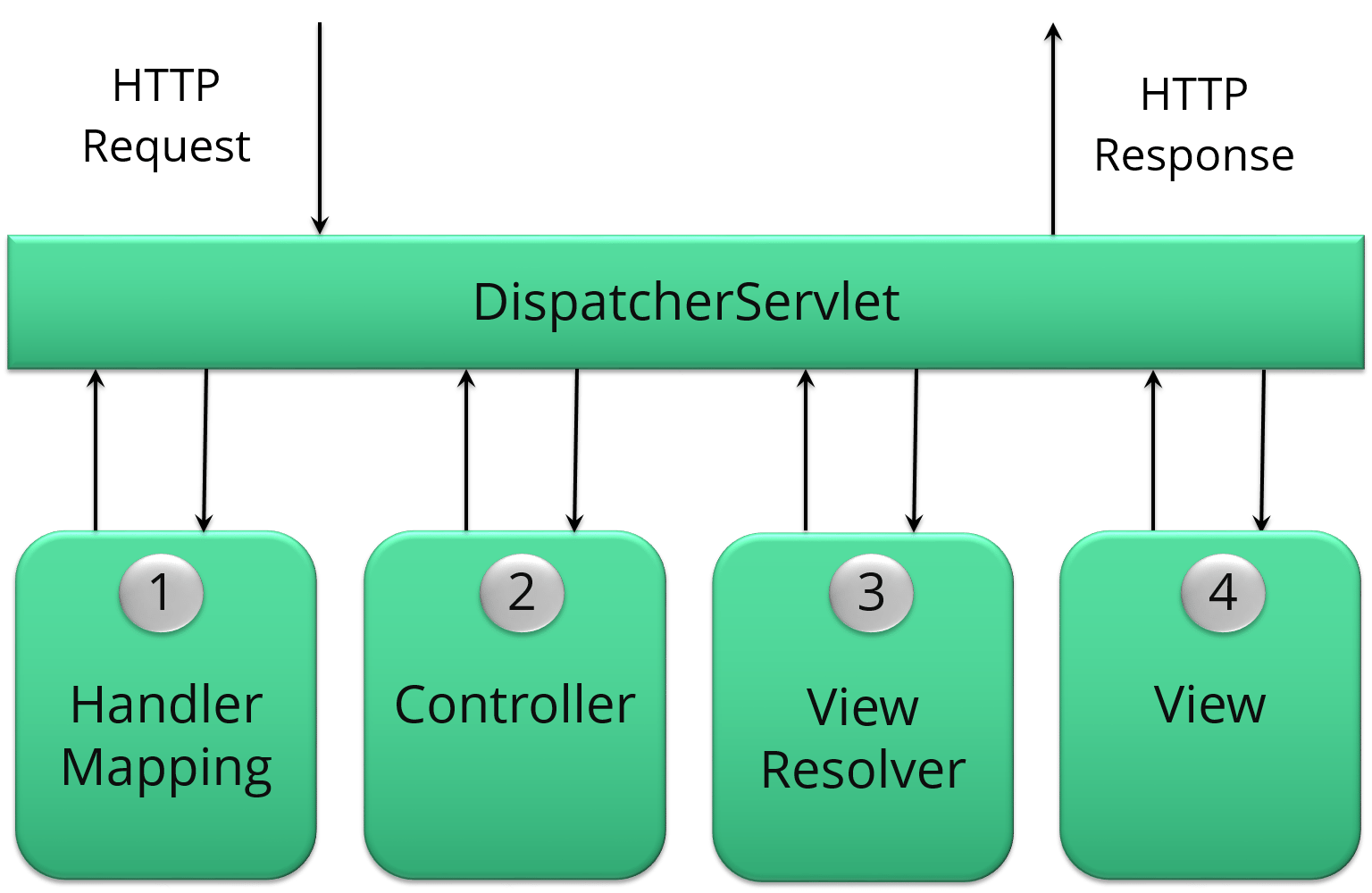
* Spring DAO layer allows you to switch between persistence technologies easily
* Ways by which Hibernate can be accessed using Spring
  + IoC with a hibernate template and callback
  + Extending Hibernate DAO support and applying an AOP interceptor node

* Transaction management types supported by Spring
  + Programmatic TM (extreme flexibility but difficult to maintain)
  + Declarative TM (TM separated from business code)

* Spring AOP

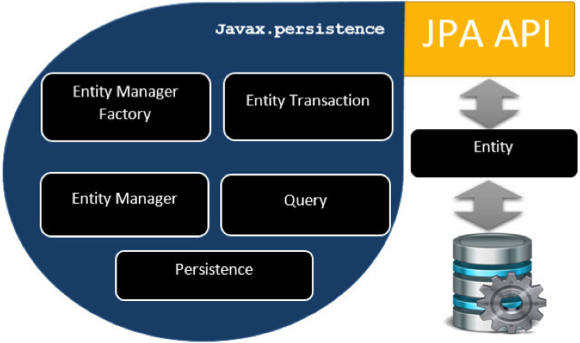
* + Joinpoints
  + Advice (5 types)
  + Aspects
  + Cross-cutting concerns

* Spring web MVC framework



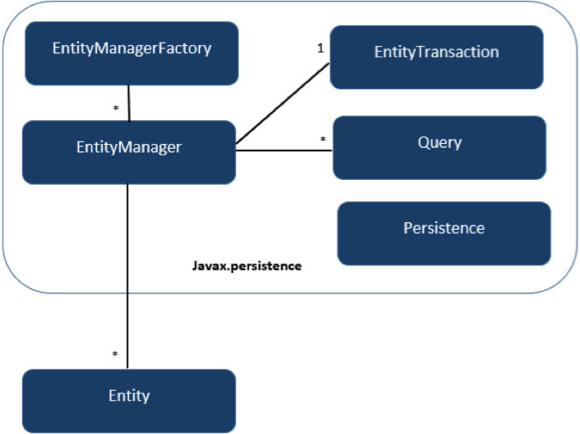
**JPA and Hibernate**

* Hibernate is one of the most widely used ORM tools for Java applications. It is a reference implementation of Java Persistence API (JPA)
* JPA forms a bridge between object models (Java codes) and relational models (database). First introduced by Oracle with EJB 3.0
* JPA is a specification. Spring data, Hibernate, iBatis are some of its implementations

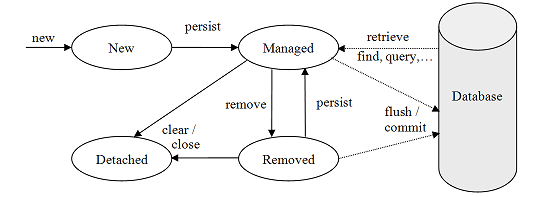


**Class level architecture of JPA**

* The following diagram shows JPA class relationships



* The main concept of JPA is to make a duplicate copy of database in cache memory. While transacting with the database, first it will affect the duplicate data and only when it is committed using entity manager, the changes affect the database
* The main role of **EntityManagerFactory** instance is to support instantiation of **EntityManager** instances. Instantiation of EntityManagerFactory is one-time job. Once instantiated it can serve the entire application
* A connection to database is represented by the **EntityManager** instance. It also provides functionality for performing operations on a database
* **Entities** are POJO representing fields of a DB table with setters and getters
* Operations that modify the content of a database require active transactions. Transactions are managed by an **EntityTransaction** instance obtained from the EntityManager
* EntityManager also acts as a factory for **Query** instances, that are needed for executing queries on the database
* **Persistence** is the JPA bootstrap class. It contains many static methods to perform important operation in JPA context
* DAO layer contains classes that have operations to perform on entities (e.g. create employee, get employee, update employee, delete employee etc.)



**Entity object life cycle**

* To use JPA in a Spring project, the EntityManager needs to be set up. JPA can now be fully setup with no XML
* We can map POJOs to traditional DB tables using JPA annotations or XML-based configurations
* Hibernate implicitly provides transaction management
* Important interfaces of Hibernate
  + SessionFactory (initialized by hibernate configuration file – single per application)
  + Session (core interface between Java application layer and hibernate)
  + Transaction (one session can have multiple transactions)
* Three states of an entity beans
  + Transient
  + Persistent
  + Detached