# Spring Annotations

- **ResponseEntity**: represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response.
- **@ExceptionHandler**: Provided by Spring Boot can be used to handle exceptions in particular Handler classes or Handler methods. Any method annotated with this is automatically recognized by Spring Configuration as an Exception Handler Method. An Exception Handler method handles all exceptions and their subclasses passed in the argument. It can also be configured to return a specific error response to the user. So let's create a custom ErrorResponse class so that the exception is conveyed to the user in a clear and concise way
- **@ControllerAdvice**: specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component. It can be viewed as an interceptor of exceptions thrown by methods annotated with @RequestMapping and similar.
- It declares @ExceptionHandler, @InitBinder, or @ModelAttribute methods to be shared across multiple @Controller classes.

# Spring Annotations

- ResponseEntityExceptionHandler is a convenient base class for @ControllerAdvice classes that wish to provide centralized exception handling across all @RequestMapping methods through @ExceptionHandler methods. It provides an methods for handling internal Spring MVC exceptions. It returns a ResponseEntity in contrast to DefaultHandlerExceptionResolver which returns a ModelAndView.

# JDBC

- JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.
- JDBC is an API(Application programming interface) used in java programming to interact with databases. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.
- Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC(Open database connectivity) driver. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.

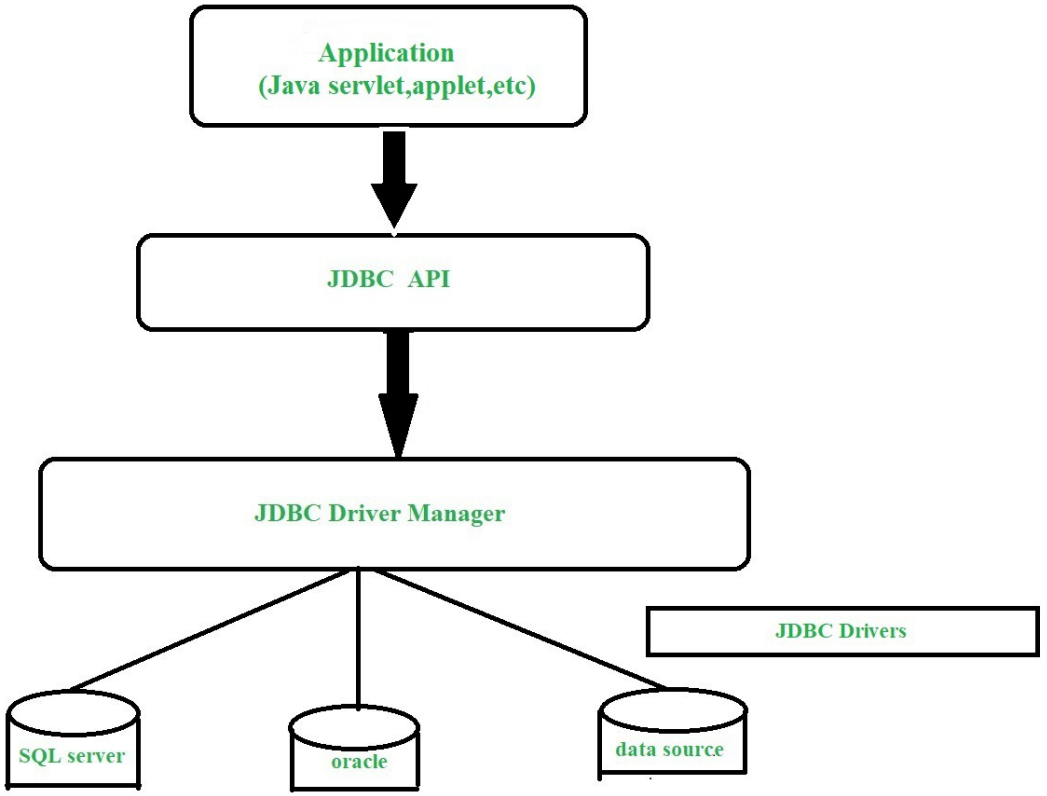
Fenil Shah

# JDBC

- **JDBC API**: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE(Standard Edition) and Java EE (Enterprise) platforms to exhibit WORA(write once run everywhere) capabilities. It also provides a standard to connect a database to a client application.
- **JDBC Driver manager**: It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
- **JDBC Test suite**: It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.
- **JDBC-ODBC Bridge Drivers**: It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the sun.jdbc.odbc package which includes a native library to access ODBC characteristics.

Fenil Shah

# JDBC




Fenil Shah

# Why Hibernate?

- JDBC code is dependent upon the Database software being used i.e. our persistence logic is dependent, because of using JDBC. Here we are inserting a record into Person table but our query is Database software-dependent i.e. Here we are using MySQL. But if we change our Database then this query won't work.
- If working with JDBC, changing of Database in middle of the project is very costly.
- JDBC code is not portable code across the multiple database software.
- In JDBC, Exception handling is mandatory. Here We can see that we are handling lots of Exception for connection.
- While working with JDBC, There is no support Object-level relationship.
- In JDBC, there occurs a Boilerplate problem i.e. For each and every project we have to write the below code. That increases the code length and reduce the readability.
- By using Hibernate we can avoid all the above problems and we can enjoy some additional set of functionalities.

# Hibernate

- Hibernate is a framework which provides some abstraction layer, meaning that the programmer does not have to worry about the implementations, Hibernate does the implementations for you internally like Establishing a connection with the database, writing query to perform CRUD operations etc.
- It is a java framework which is used to develop persistence logic. Persistence logic means to store and process the data for long use. More precisely Hibernate is an open-source, non-invasive, light-weight java ORM(Object-relational mapping) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE.
- **Framework** means it is special install-able software that provides an abstraction layer on one or more technologies like JDBC, Servlet, etc to simplify or reduce the complexity for the development process.
- **Open Source**: The source code of Hibernate is also available on the Internet and we can also modify the code.
- **Non-invasive**: The classes of Hibernate application development are loosely coupled classes with respect to Hibernate API i.e. Hibernate class need not implement hibernate API interfaces and need not extend from Hibernate API classes.
- Used for Only **Relational** Databases.

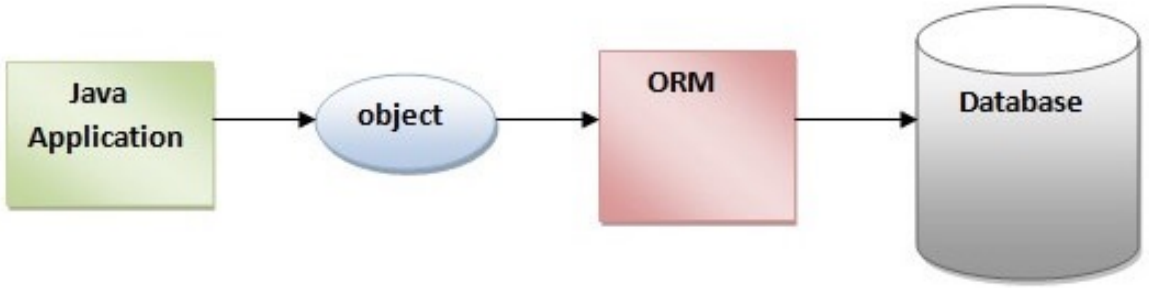
# Hibernate

- **Light-weight**: Hibernate is less in size means the installation package is not big is size. Hibernate does not require any heavy container for execution. Hibernate can be used alone or we can use Hibernate with other java technology and framework.
- Hibernate framework support **Auto DDL** operations. In JDBC manually we have to create table and declare the data-type for each and every column. But Hibernate can do DDL operations for you internally like creation of table,drop a table,alter a table etc.
- Hibernate supports **Auto Primary key** generation. It means in JDBC we have to manually set a primary key for a table. But Hibernate can this task for you.
- Hibernate framework is independent of Database because it supports **HQL** (Hibernate Query Language) which is not specific to any database, whereas JDBC is database dependent.
- In Hibernate, **Exception Handling** is not mandatory, whereas In JDBC exception handling is mandatory. Hibernate supports **Cache Memory** whereas JDBC does not support cache memory.
- **Simplifies Complex Join**: Fetching data from multiple tables is easy in hibernate framework.

Fenil Shah

# Hibernate

- Hibernate is a **ORM tool** means it support Object relational mapping. Whereas JDBC is not object oriented moreover we are dealing with values means primitive data. In hibernate each record is represented as a Object but in JDBC each record is nothing but a data which is nothing but primitive values.
- **Fast Performance**: The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.
- Hibernate Maintains the **Connections** and allocates threads accordingly. **Associations** like one-to-one, one-to-many, many-to-one, and many-to-many can be acquired easily with the help of annotations.
- Hibernate is an implementation of JPA guidelines. It helps in mapping Java data types to SQL data types. It is the contributor of JPA.



# Hibernate Lifecycle

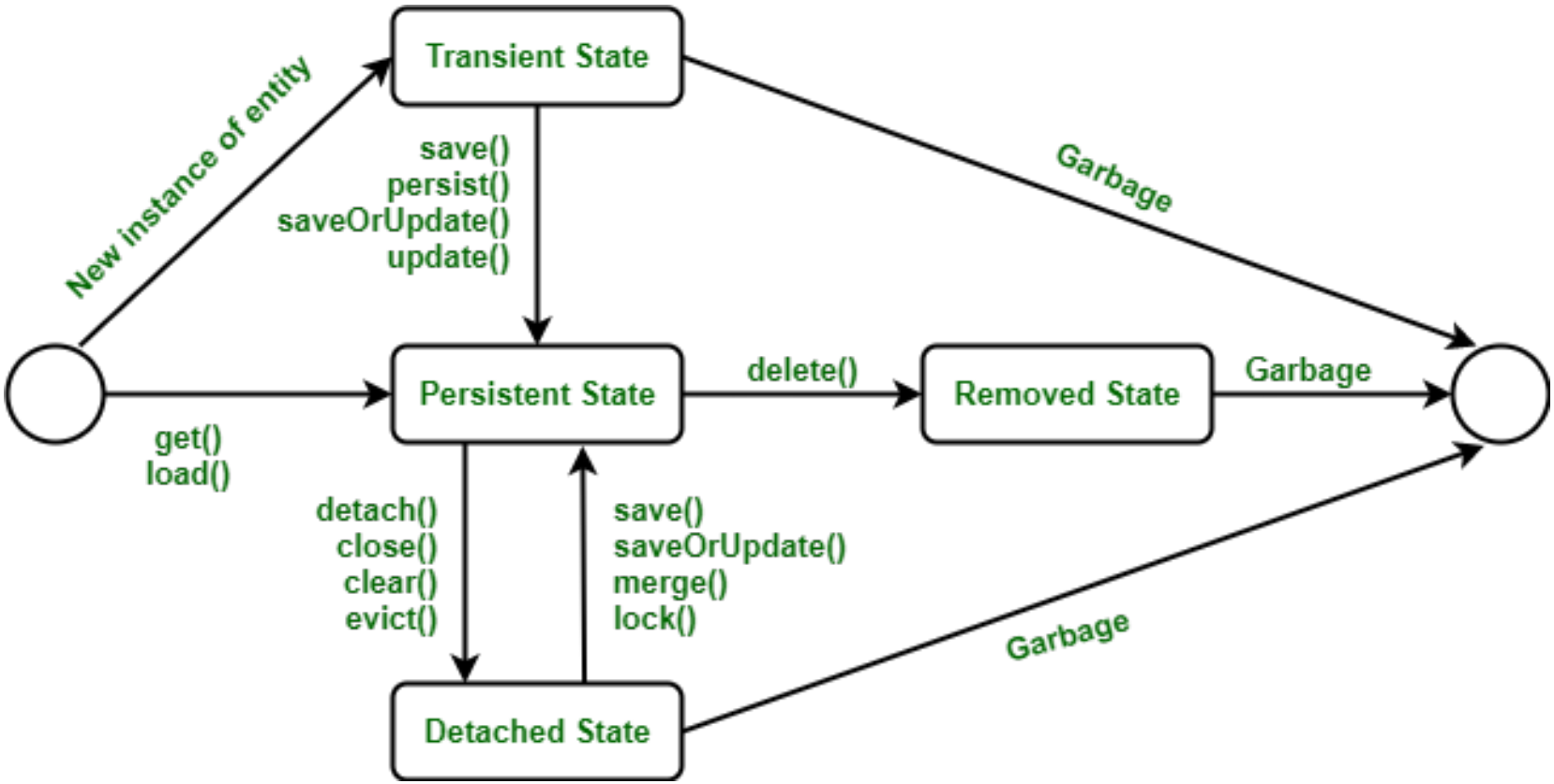
- The **transient** state is the first state of an entity object. When we instantiate an object of a POJO class using the new operator then the object is in the transient state. This object is not connected with any hibernate session. As it is not connected to any Hibernate Session, So this state is not connected to any database table. So, if we make any changes in the data of the POJO Class then the database table is not altered. Transient objects are independent of Hibernate, and they exist in the heap memory. Eg- When objects are generated by an application but are not connected to any session & The objects are generated by a closed session.
- Once the object is connected with the Hibernate Session then the object moves into the **Persistent** State. So, there are two ways to convert the Transient State to the Persistent State :
- Using the hibernated session, save the entity object into the database table. & Using the hibernated session, load the entity object into the database table.
- In this state. each object represents one row in the database table. Therefore, if we make any changes in the data then hibernate will detect these changes and make changes in the database table.
- Eg- session.persist(e);  session.save(e); session.saveOrUpdate(e); session.update(e);

Fenil Shah

# Hibernate Lifecycle

- For converting an object from Persistent State to **Detached** State, we either have to close the session or we have to clear its cache. As the session is closed here or the cache is cleared, then any changes made to the data will not affect the database table. Whenever needed, the detached object can be reconnected to a new hibernate session.
- Eg- session.detach(e);  session.evict(e); session.clear(); session.close();
- In the hibernate lifecycle it is the last state. In the **removed** state, when the entity object is deleted from the database then the entity object is known to be in the removed state. It is done by calling the delete() operation. As the entity object is in the removed state, if any change will be done in the data will not affect the database table. Eg - session.delete().

# Hibernate Lifecycle



# Hibernate Properties

- **@GeneratedValue**: Specifies how to generate values for the given column. This annotation will help in creating primary keys values according to the specified strategy.
- **GenerationType.AUTO**: Indicates that the persistence provider should select an appropriate strategy based on the specified database dialect. The AUTO will automatically set the generated values. It is the default GenerationType. If no strategy is defined, it will consider AUTO by default. All the databases support this strategy.

  Eg- insert into person_tab (age, dob, first_name, last_name, id) values (?, ?, ?, ?, ?)
- **GenerationType.IDENTITY**: Indicates that the persistence provider must assign primary keys for the entities using a database identity column. It is very similar to AUTO generationType. This strategy is not available in all the databases. If the particular database doesn't support IDENTITY generationtype, the persistence provider will itself choose an alternative strategy.

  Eg- insert into person_tab (age, dob, first_name, last_name) values (?, ?, ?, ?)

**GenerationType.TABLE**: The GenerationType.TABLE indicates that the persistence provider must assign primary keys for the entities using a database table. It uses a database table to store the latest primary key value. This strategy is available in all the databases.

# Hibernate Properties

- **GenerationType.SEQUENCE**: Indicates that the persistence provider must assign primary keys for the entities using the database sequence. This strategy consists of two parts, defining a sequence name and using the sequence name in the classes. It also builds a sequence generator in the database and not supported by all the databases. A SEQUENCE GenerationType is global to the application; it can be accessed by one or more entities in the application.
- **spring.jpa.hibernate.ddl-auto**:
- validate: Only checks if the Schema matches the Entities. If the schema doesn't match, then the application startup will fail. Makes no changes to the database.
- update: Updates the schema only if necessary. For example, If a new field was added in an entity, then it will simply alter the table for a new column without destroying the data. Doesn't perform delete.
- create: Drops and creates the schema at the application startup. With this option, all your data will be gone on each startup.
- create-drop: drop the schema when the SessionFactory is closed explicitly, typically when the application is stopped. Creates schema at the startup and destroys the schema on context closure. Useful for unit tests.
- none: does nothing with the schema, makes no changes to the database

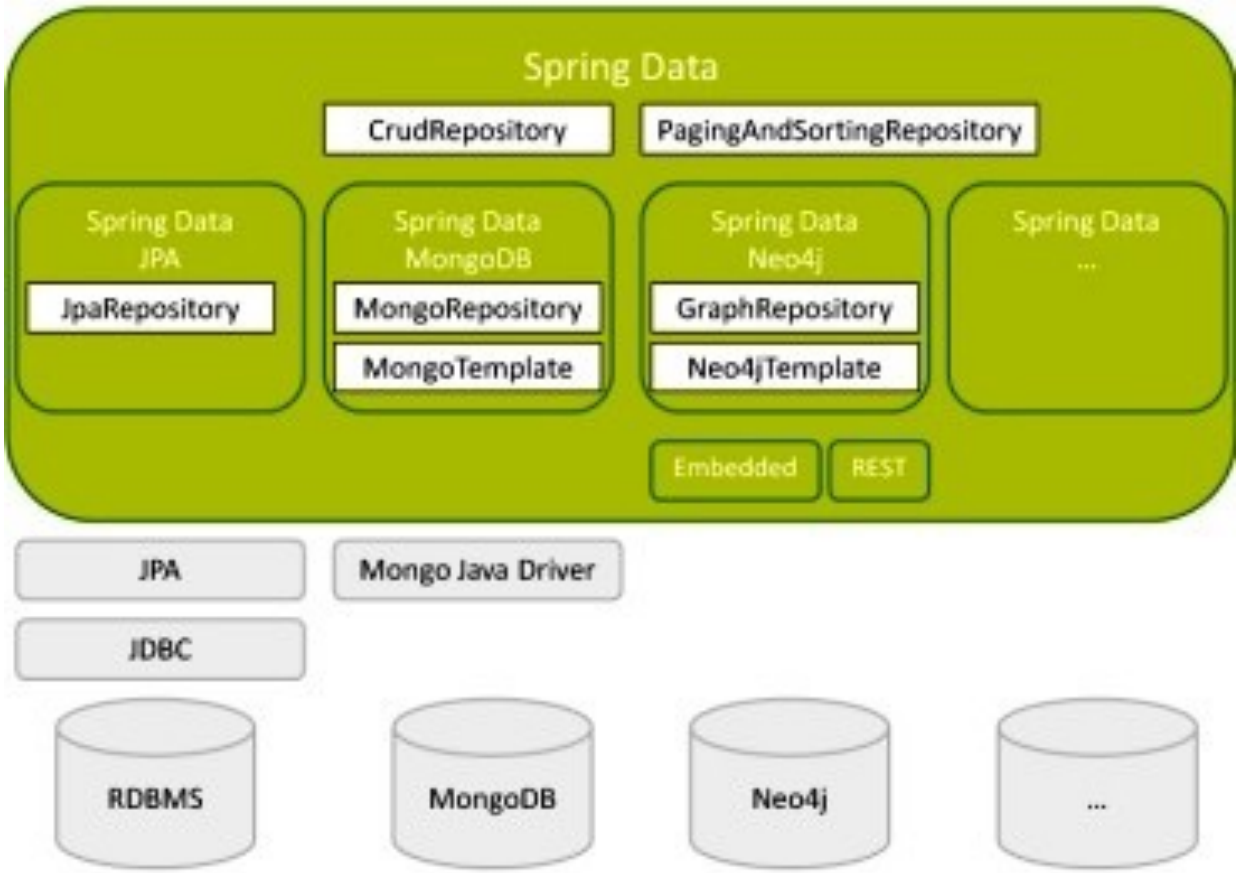
# JPA

- 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. It is used to examine, control, and persist data between Java objects and relational databases. It is observed as a standard technique for Object Relational Mapping. For data persistence, the javax.persistence package contains the JPA classes and interfaces.
- JPA is only a specification, it is not an implementation. It is a set of rules and guidelines to set interfaces for implementing object-relational mapping, .
- It needs a few classes and interfaces. It supports simple, cleaner, and assimilated object-relational mapping. It supports polymorphism and inheritance. Dynamic and named queries can be included in JPA.
- No Particular specification is available for NoSql databases as they are unstructured and stores data differently.

# JPA

- All ORM tools (such as Hibernate) follow the common standards, by executing the same specification. Subsequently, if we need to switch our application from one ORM tool to another then we can easily do it.
- We can annotate classes to the extent that we would like with JPA annotations, although, nothing will take place without an implementation. Suppose JPA as the guidelines that should be followed, however, Hibernate is a JPA implementation code that unites the API as described by the JPA specification and gives the anonymous functionality.
- As an object-oriented query language, it uses Java Persistence Query Language (JPQL) to execute database operations. To interconnect with the entity manager factory for the persistence unit, it uses EntityManagerFactory interface. Thus, it gives an entity manager.
- To make, read, and remove actions for instances of mapped entity classes, it uses **EntityManager** interface. This interface interconnects with the persistence condition. The entity manager implements the API and encapsulates all of them within a single interface. Entity manager is used to read, delete and write an entity. An object referenced by an entity is managed by entity manager.

# SpringData Repository



# JAR, WAR, EAR

- **JAR**:  Java Archive – is a package file format. JAR files have the .jar extension and may contain libraries, resources, and metadata files. Essentially, it's a zipped file containing the compressed versions of .class files and resources of compiled Java libraries and applications. JAR files allow us to package multiple files in order to use it as a library, plugin, or any kind of application. we can run a JAR from the command line if we build it as an executable JAR without using additional software. We can create JAR with any structure
- **WAR**: Web Application Archive or Web Application Resource. These archive files have the .war extension and are used to package web applications that we can deploy on any Servlet/JSP container. it also contains the WEB-INF public directory with all the static web resources, including HTML pages, images, and JS files. Moreover, it contains the web.xml file, servlet classes, and libraries. WAR files are used only for web applications. we need a server to execute a WAR. We use the WAR file in Java to package the web application which may contains servlet, xml, jsp, image, html, css, js etc. WAR has a predefined structure with WEB-INF and META-INF directories
- **EAR:** Enterprise Application Archive is standard JAR file that represents the modules of the application, and a metadata directory called META-INT which contains one or more deployment descriptors. It allows deploying different modules onto an application server simultaneously.

Fenil Shah

# JAR, WAR, EAR

- **JAR**: EJB modules which contain enterprise java beans (class files) and EJB deployment descriptor are packed as JAR files with .jar extension. The main difference between an application jar and a library jar is that an application will have a starting point (main class with a main method) to run the code as an application, and configuration in the jar file metadata to tell the java environment where to find that starting point. Note that jar files can contain other jar files, in fact this is quite commonly done for jar files that contain an application that in turn needs some java libraries, which are packaged as jar files.
- **WAR**: Web modules which contain Servlet class files, JSP Files, supporting files, GIF and HTML files are packaged as a JAR file with .war (web archive) extension. Files that are supposed to be directly loadable — HTML files, JSP files, and image files, for example — go in the top directory of the WAR file. Everything internal to the application, like metadata, config files, class files, and libraries, goes under the WEB-INF subdirectory.
- **EAR**: All the above files (.jar and .war) are packaged as a JAR file with .ear (enterprise archive) extension and deployed into Application Server. Standalone web containers such as Tomcat and Jetty do not support EAR files — these are not full-fledged Application servers. Enterprise applications means applications that meet the needs particular to large companies (aka enterprises), which usually means things like connecting to a lot of systems, being used by a lot of people, and being high availability and high reliability.

# Associations

- Association in Java defines the connection between two classes that are set up through their objects. Association manages one-to-one, one-to-many, and many-to-many relationships. In Java, the multiplicity between objects is defined by the Association. It shows how objects communicate with each other and how they use the functionality and services provided by that communicated object. Association manages one-to-one, one-to-many, many-to-one and many-to-many relationships.
- Composition and Aggregation are the two forms of association.
- **Aggregation**: It represents Has-A's relationship.
- It is a unidirectional association i.e. a one-way relationship. For example, a department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, both the entries can survive individually which means ending one entity will not affect the other entity. Code reuse is best achieved by aggregation.

Fenil Shah

# Associations

- **Composition**: a restricted form of Aggregation in which two entities are highly dependent on each other. It represents part-of relationship. In composition, both entities are dependent on each other. When there is a composition between two entities, the composed object cannot exist without the other entity.
- Eg- a library can have no. of books on the same or different subjects. So, If Library gets destroyed then All books within that particular library will be destroyed. i.e. books can not exist without libraries. That's why it is composition. Book is Part-of Library.
- Ref: https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate/

# JPA FetchTypes

- FetchType defines when Hibernate gets the related entities from the database, and it is one of the crucial elements for a fast persistence tier.
- **FetchType.EAGER** – Fetch it so you'll have it when you need it. The FetchType.EAGER tells Hibernate to get all elements of a relationship when selecting the root entity. This is the default for to-one relationships.
- EAGER fetching tells Hibernate to get the related entities with the initial query. This can be very efficient because all entities are fetched with only one query. But in most cases it just creates a huge overhead because you select entities you don't need in your use case.
- **FetchType.LAZY** – Fetch it when you need it. The FetchType.LAZY tells Hibernate to only fetch the related entities from the database when you use the relationship. This is a good idea in general because there's no reason to select entities you don't need for your uses case.
- This tells Hibernate to delay the initialization of the relationship until you access it in your business code. The drawback of this approach is that Hibernate needs to execute an additional query to initialize each relationship.

Fenil Shah

# YAML (.yml) File

- YAML is a configuration language. Languages like Python, Ruby, Java heavily use it for configuring the various properties while developing the applications.
- If you have ever used Elastic Search instance and MongoDB database, both of these applications use YAML(.yml) as their default configuration format. Hierarchical Structure.
- Supports key/val, basically map, List and scalar types (int, string etc.). While retrieving the values from .yml file we get the value as whatever the respective type (int, string etc.) is in the configuration
- If you are using spring profiles, you can have multiple profiles in one single .yml file. Since version 2.4.0, Spring Boot supports creating multi-document properties files. Simply put, we can split a single physical file into multiple logical documents.
- .yml file is advantageous over .properties file as it has type safety, hierarchy and supports list but if you are using spring, spring has a number of conventions as well as type conversions that allow you to get effectively all of these same features that YAML provides for you.
- One advantage that you may see out of using the YAML(.yml) file is if you are using more than one application that read the same configuration file. you may see better support in other languages for YAML(.yml) as opposed to .properties.