Day18: Maven introduction, dependencies, ORM introduction, Entity class, Java Persistence

What are build tools?

> Build tools are programs that automate the creation of executable applications from source code (e.g., .apk for an Android app). Building incorporates compiling,linking and packaging the code into a usable or executable form.
>
> Basically build automation is the act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities like:
>
> 1. Downloading dependencies.
> 2. Compiling source code into binary code.
> 3. Packaging that binary code.
> 4. Running tests.
> 5. Deployment to production systems.

Why do we use build tools or build automation?

> In small projects, developers will often manually invoke the build process. This is not practical for larger projects, where it is very hard to keep track of what needs to be built, in what sequence and what dependencies there are in the building process. Using an automation tool allows the build process to be more consistent.

Various build tools available(Naming only few):

> For java - Ant,Maven,Gradle.

What is Build Tool

A build tool takes care of everything for building a process. It does following:

- Generates source code (if auto-generated code is used)
- Generates documentation from source code

- Compiles source code
- Packages compiled code into JAR of ZIP file
- Installs the packaged code in local repository, server repository, or central repository

Maven

Maven is a tool that can now be used for building and managing any Java-based project. It makes the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

Maven is a powerful *project management tool* that is based on POM (project object model). It is used for projects build, dependency and documentation.

Dependency is defined as a state of needing something or someone.

When you rely on coffee to get you through the day, this is an example of you having dependency on caffeine (coffee).

In Maven, a dependency is just another archive—JAR, ZIP, and so on—which our current project needs in order to compile, build, test, and/or run. These project dependencies are collectively specified in the pom. xml file, inside of a <dependencies> tag.
More details will be discussed later in these notes.

Understanding the problem without Maven

There are many problems that we face during the project development. They are discussed below:

1) Adding set of Jars in each project: In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

2**) Building and Deploying the project:** We must have to build and deploy the project so that it may work.

Maven's Objectives

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, Maven deals with several areas of concern:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices
-

To install maven on windows, you need to perform following steps:

1. Download maven and extract it

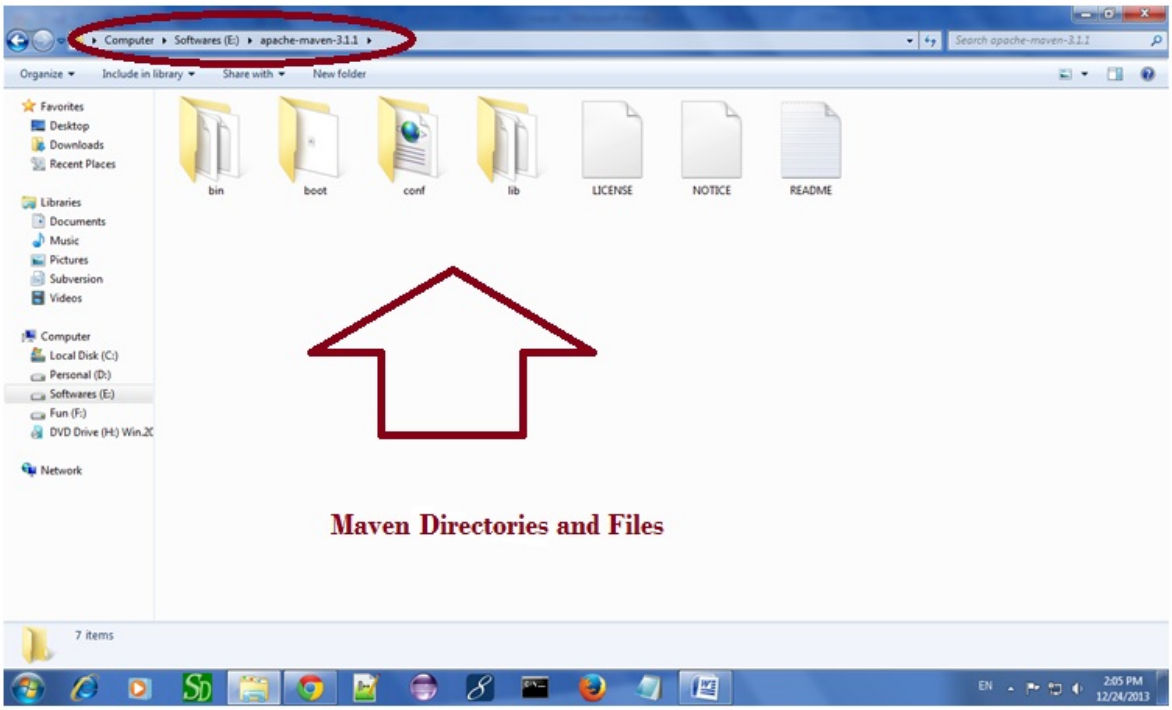
2. Add JAVA_HOME and MAVEN_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

1) Download Maven

Download Maven latest Maven software from Download latest version of Maven

For example: **apache-maven-3.1.1-bin.zip**

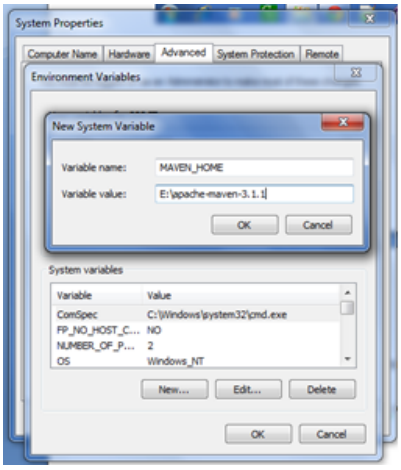
Extract it. Now it will look like this:



2) Add MAVEN_HOME in environment variable

Right click on **MyComputer** -> **properties** -> **Advanced System Settings** -> **Environment variables** -> **click new button**

Now **add MAVEN_HOME** in variable name and path of maven in variable value. It must be the home directory of maven i.e. outer directory of bin. For example: **E:\apache-maven-3.1.1** .It is displayed below:



Now click on **OK** button.

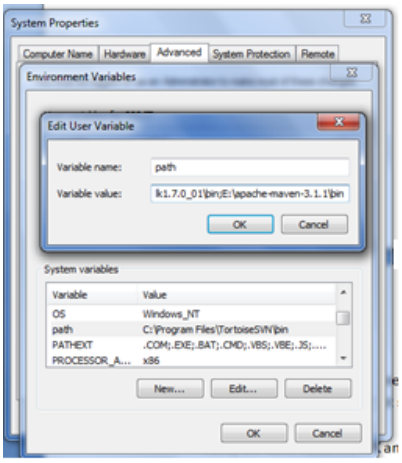
---

3) Add Maven Path in environment variable

Click on new tab if path is not set, then set the path of maven. If it is set, edit the path and append the path of maven.

Here, we have installed JDK and its path is set by default, so we are going to append the path of maven.

The path of maven should be **%maven home%/bin**. For example, **E:\apache-maven-3.1.1\bin** .



---

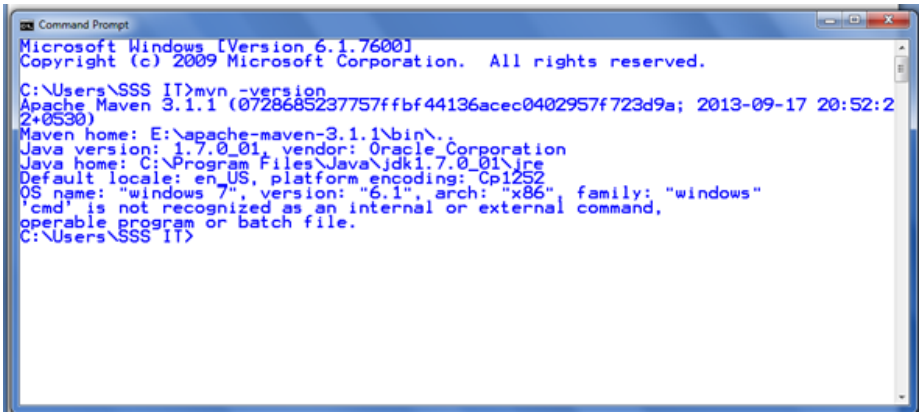
4)Verify maven

To verify whether maven is installed or not, open the command prompt and write:

1. mvn −version

Now it will display the version of maven and jdk including the maven home and java home.

Let's see the output:



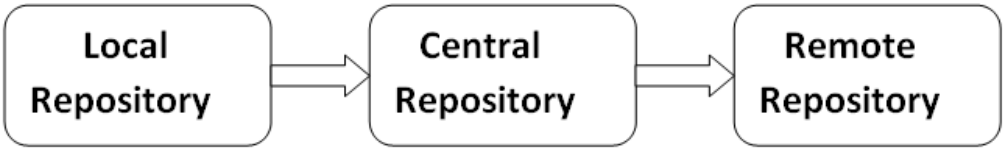
Maven Repository

A **maven repository** is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

Local repository then **Central repository** then **Remote repository**.



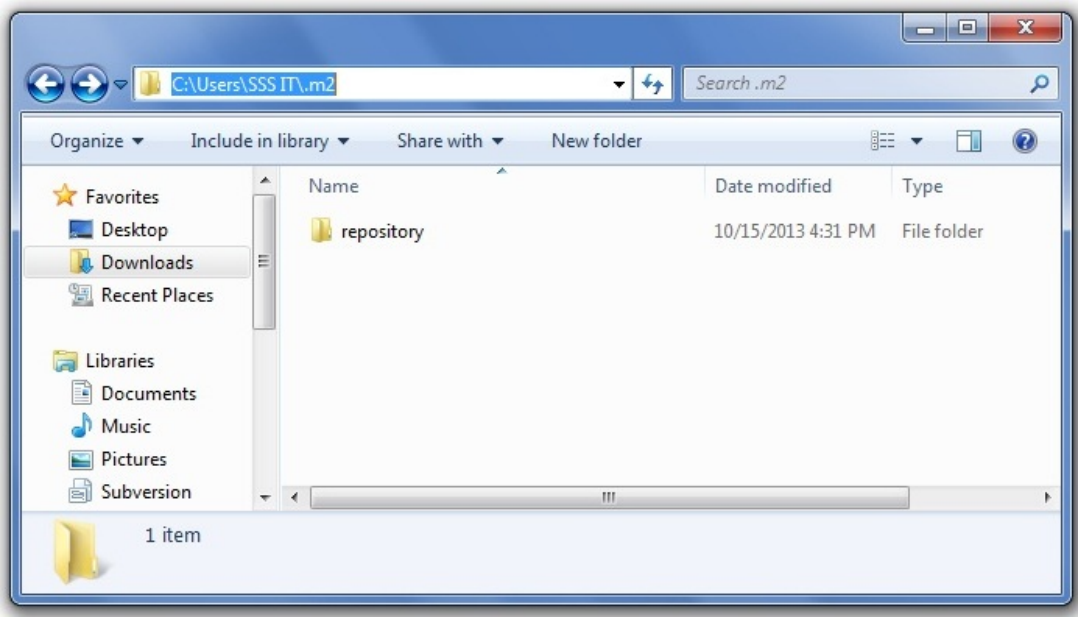
If dependency is not found in these repositories, maven stops processing and throws an error.

---

1) Maven Local Repository

Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.

By default, maven local repository is %USER_HOME%/.m2 directory. For example: **C:\Users\SSS IT\.m2**.



Update location of Local Repository

We can change the location of maven local repository by changing the **settings.xml** file. It is located in **MAVEN_HOME/conf/settings.xml**, for example: **E:\apache-maven-3.1.1\conf\settings.xml**.

Let's see the default code of settings.xml file.

settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->

...
</settings>
```

```
<! -- Now change the path to local repository. After changing the path of local repository, it will look like this: -->

<! -- settings.xml -->

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
   <localRepository>e:/mavenlocalrepository</localRepository>

...
</settings>
```

As you can see, now the path of local repository is e:/mavenlocalrepository.

2) Maven Central Repository

Maven **central repository** is located on the web. It has been created by the apache maven community itself.

The path of central repository is: http://repo1.maven.org/maven2/.

The central repository contains a lot of common libraries that can be viewed by this url http://search.maven.org/#browse.

3) Maven Remote Repository

Maven **remote repository** is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in pom.xml file.

Let's see the code to add the jUnit library in pom.xml file.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.javatpoint.application1</groupId>
  <artifactId>my-application1</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

You can search any repository from Maven official website **mvnrepository.com**.

Maven pom.xml file

POM is an acronym for **Project Object Model**. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

Before maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

Elements of maven pom.xml file

For creating the simple pom.xml file, you need to have following elements:

40.6M

903

Features of Java - Javatpoint

Element	Description
project	It is the root element of pom.xml file.
modelVersion	It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0.
groupId	It is the sub element of project. It specifies the id for the project group.
artifactId	It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.
version	It is the sub element of project. It specifies the version of the artifact under given group.

IMP

File: pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javatpoint.application1</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>

</project>
```

Maven pom.xml file with additional elements

Here, we are going to add other elements in pom.xml file such as:

Element	Description
packaging	defines packaging type such as jar, war etc.
name	defines name of the maven project.
url	defines url of the project.
dependencies	defines dependencies for this project.

Element	Description
dependency	defines a dependency. It is used inside dependencies.
scope	defines scope for this maven project. It can be compile, provided, runtime, test and system.

File: pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.javatpoint.application1</groupId>
  <artifactId>my-application1</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

Maven Eclipse Example

In eclipse, click on File menu → New → Project → Maven → Maven Project. → Next → Next → Next. Now write the group Id, artifact Id, Package as shown in below figure → finish.

Now you will see a maven project with complete directory structure. All the files will be created automatically such as Hello Java file, pom.xml file, test case file etc.

Now you can see the code of App.java file and run it. It will be like the given code:

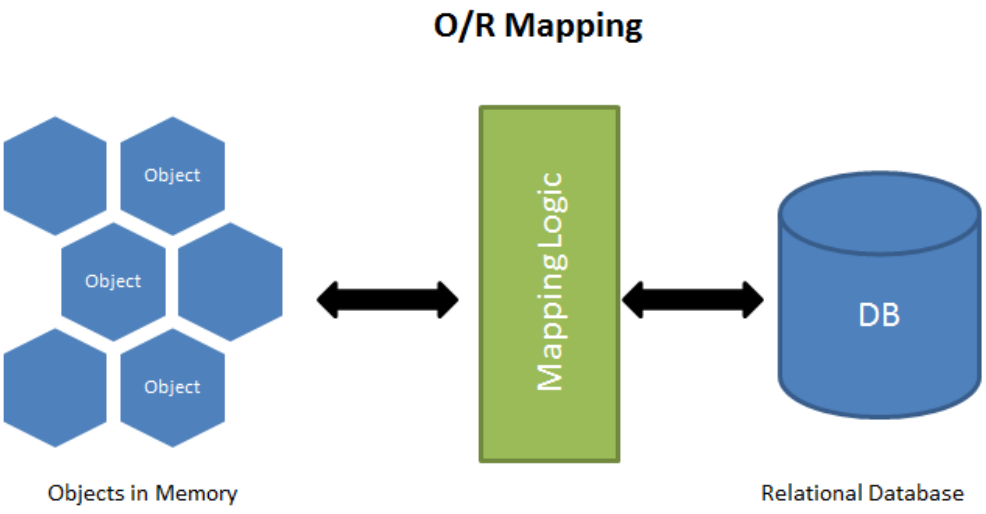
```
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

If you right click on the project → Run As, you will see the maven options to build the project.

What is an ORM?

Before getting into the proper definition of ORM, first let's take an example which will explain the term in a more better way. So, have you ever used *SQL* database in your application and have you ever used SQL queries to update, insert or retrieve the data, no matter how tough the queries are. So, here comes the idea of Object Relational Mapping(ORM). And it is a programming technique for converting data between incompatible type systems using object-oriented programming languages.

It means you can write database queries using the object oriented paradigm of your preferred language and there are many free and commercial packages available that perform object relational mapping.



ORM sets the mapping between the set of objects which are written in the preferred programming language like *JavaScript* and relational database like SQL. It hides and encapsulates the SQL queries into objects and instead of SQL queries we can use directly the objects to implement the SQL query.

Why to use ORM and What are the benefits of ORM?

Now the question arises that if we can use directly the queries then why to include the ORM frameworks in between.

So, first of all you get to write in the language you are already using. It is sometime tough to write SQL queries directly as they are complicated in some cases. So, to maintain the fluency we use ORM, so that we can write in the language we know.

Second, it hides the SQL or any other database query away from your application logic.

Third, for heavy database usage like creating 10+ tables and using many queries in them, then it is good to use ORM as it reduce the code and give better understanding of the code to you and as well as to your team mates and it makes your application faster and easier to maintain.

What are some commonly used ORM Frameworks?

There are many different ORM frameworks that can be used with different languages like *Enterprise Java Beans(EJB)*, *Hibernate*, etc. can be used with Java.

And with JavaScript we can use *bookshelf.js* which is built on *Knex SQL* Query Builder. It is designed to work with *PostgreSQL, MySQL* and *SQLite3*. And the other framework used with JavaScript is *mongoose.js* which is designed to work with *MongoDB.*

> So, lets conclude the article in a very simple line that, ORM is used to write Database queries in the preferred languages instead of writing the Raw Database queries.

JPA(Java Persistence API)

The java persistence API is a specification in Java. It is used to persist data between Java object and relational database. It acts as a bridge between Java object and database.

Entity An entity is a group of states associated together in a single unit. On adding behavior, an entity behaves like an object and becomes a major constituent of the object-oriented paradigm. So, an entity is an application-defined object in Java Persistence Library.

Entity Properties: These are the properties of an entity that an object must have: -

- **Persistability** — An object is called persistent if it is stored in the database and can be accessed anytime.
- **Persistent Identity** — In Java, each entity is unique and represents an object identity. Similarly, when the object identity is stored in a database then it is represented as persistence identity. This object identity is equivalent to the primary key in the database.
- **Transactionality** — Entity can perform various operations such as create, delete, update. Each operation makes some changes in the database. It ensures that whatever changes made in the database either succeed or fail atomically.
- **Granuality** — Entities should not be primitives, primitive wrappers or built-in objects with single dimensional state.

Entity Metadata: Each entity is associated with some metadata that represents the information of it. Instead of a database, this metadata is exist either inside or outside the class. This metadata can be in following forms: -

- **Annotation** — In Java, annotations are the form of tags that represents metadata. This metadata persists inside the class.
- **XML** — In this form, metadata persist outside the class in XML file.

Entity Creation: For the creation of an Entity from a class, we need only two things.
1. No arguments Constructor
2. Annotation

```
  import javax.persistence.Entity;
import javax.persistence.Id;

@Entity // To create a Enitiy
public class Employee {

    @Id  // for primary key
    private int id;
    private String name;
    private String address;
    private int age;
```

```
    public Employee() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Entity Manager:

Entity manager encapsulates all of them within a single interface. it is used to read, write and delete an entity. An object referenced by an entity is managed by the entity manager.

Hibernate

It is an object-relational mapping (**ORM**) tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Also handles object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.

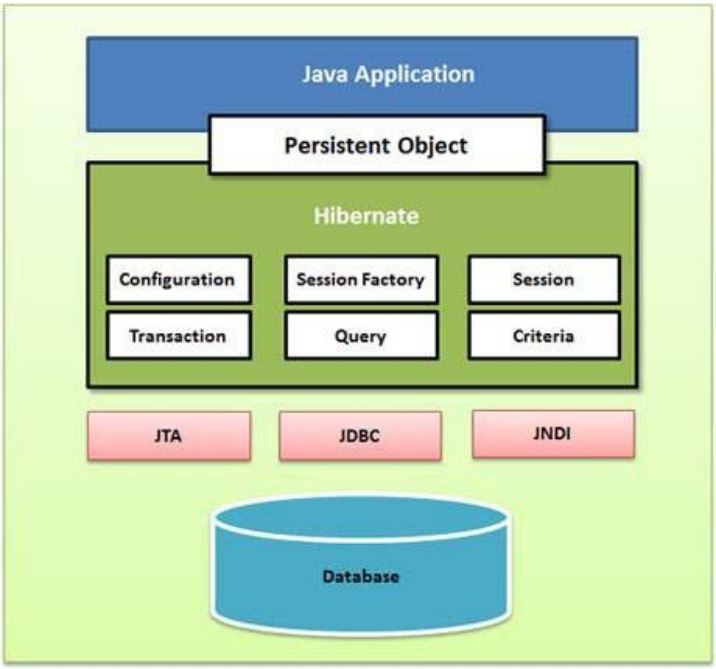
Primary feature is mapping from Java classes to database tables and mapping from Java data types to SQL data types. Also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

Hibernate Architecture

Hibernate is an ORM framework with a layered architecture that helps the user to operate without having to know the underlying APIs. It also makes use of the database and configuration data to provide persistence services to the application.

There are basically 3 layers in this architecture:-

- Application Layer
- Hibernate Core Layer
- Database Layer



Hibernate Application Architecture

Core Components of Hibernate

Let's have a brief description of each of the class objects involved in this Architecture.

1. **Configuration:**

- **org.hibernate.cfg** package contains the **Configuration** class.
- **Configuration cfg = new Configuration()** used to activate the Framework.
- It reads both cfg file and mapping files **cfg.configure();**
- Configuration object created once, at the time of initialization.

1. **SessionFactory:**

- **org.hibernate.sessionFactory** package contains the **SessionFactory** Interface.
- It is immutable and thread-safe in nature.

- **SessionFactory factory = cfg.buildSessionFactory()** used while creation of SessionFactory object
- From cfg object it takes the JDBC information and create a JDBC Connection.

1. **Session**:

- **org.hibernate.session** package contains the **Session** interface
- Session object created based upon SessionFactory object i.e. factory.
- It is a lightweight object and not thread-safe.
- Session object used to execute CRUD operations.
- **Session session = factory.buildSession()** used while the creation of Session object

1. **Transaction:**

- **org.hibernate.transaction** package contains the **Transaction** interface
- Transaction object used whenever we perform any operation and based upon that operation there is some change in the database.
- It is a short-lived single-threaded object.
- **Transaction tx = session.beginTransaction();tx.commit()**; used while creation of Transaction object.

1. **Query:**

- **org.hibernate.query** package contains the **Query** interface.
- The session objects are used to create query objects.
- Query objects use Hibernate Query Language (HQL) to get data from the database.
- **Query query = session.createQuery()** used while initialisation of Query object.

1. **Criteria:**

- **org.hibernate.criteria** package contains the **Criteria** interface.
- The session objects are used to create Criteria Objects.
- **Criteria criteria = session.createCriteria()** used while initialisation of Criteria object.

References:

https://maven.apache.org/what-is-maven.html

https://www.javatpoint.com/maven-tutorial

https://stackoverflow.com/questions/7249871/what-is-a-build-tool

https://medium.com/@grover.vinayak0611/what-is-orm-why-to-use-it-and-brief-introduction-of-orm-frameworks-b61b16d02a3c

https://medium.com/@grover.vinayak0611/what-is-orm-why-to-use-it-and-brief-introduction-of-orm-frameworks-b61b16d02a3c

https://medium.com/programming-geek/jpa-java-persistence-api-c93b648fc931