JAR(Java Archive Files):

- JAR files consists of bytecode and some necessary files.
- JAR files is used to transfer java files.

Steps to Create the JAR file:

- Right click in then project which has to transferred and select export search for jar and select java jar and click on next,
- Select the destination to save the JAR files and click on finish.0

Steps to use JAR Files:

- Right click on the project where JAR has to be used.
- Select build path and config build path.
- Click on libraries and select class path.
- Click on add external JAR's.
- Select the JAR files from the file system and open.
- After success build path JAR file will be present under reference libraries.

Steps to Connect to JAVA with the Database:

We have 5steps to achieve,

- Load or Register a Driver
- Establish a Connection
- Create Statement
- Execute Query
- Close the Connection.

We can found these in java.SQL.*.package

MySQL Basic Commands:

Command line Interface:

Show databases - It will list all the databases present in mySQL.

Create schema schema name(database name) - it will create a new database

Drop database database name - For deleting the database from SQL.

Use database name - it is used to change the database and make that as a current database.

Create table tablename

Show tables - will shows all the tables present in current database.

Insert into tablename values(1,'John'); - Inserting the values into the tables. We can get 1 rows affected.

Select *from tablename;

We were performing insert, update and delete we will get row affected message with integer values.

Load/Register Driver:

Load/Register Driver can be achieved in two ways,

- Implicitly
- Explicitly

Explicitly:

Loading the driver is done using following methods present in java.sql.DriverManager class

Syntax:

```
Public static void registerDriver(Driver object){}
```

Explicit Way:

```
java.sql.Driver d= new com.mysql.cj.jdbc.Driver();
DriverManager.registerDriver(d);
```

Implicit Way:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

- It will throw the class not found exception we handle with the try catch block.

Example Driver classes:

MySQL - com.mysql.jdbc.Driver

ORACLE - oracle.jdbc.driver.OracleDriver

Driver Manager:

Driver Manager is a class present in java.sql.*;

It manage the set of java database connectivity(JDBC) driver softwares that are available for an application to use.

Application can use multiple JDBC drivers.

To Establish a connection:

Get the connection:

To get the connection we need a critical information about DBURL.

DBURL -->(Database Uniform Resource Locator)

DBURL is used to identify database present in RDBMS application

Structure of DBURL:

Protocol: subprotocol:subname -->under subname we have host name(ip), port number, dbname/username and password

Protocol:

Protocol is a set of rules which helps to connect between two application is our case java application with RDBMS application.

It's mandatory information in the database url.

The protocol used is jdbc.

SubProtocol:

It's also a set of rules which gives information about database connectivity mechanism.

It's mandatory information in the database url.

Sub-Protocol depend on the RDBMS application used

The Sub-Protocol usded is mysql.

SubName:

It will give physical location of RDBMS application.

It consists of four parts.

- Host Name: It's uniquely identifies the computer where RDBMS application is installed, usually it will be ip address in our case we will see localhost since RDBMS and java application are present in same system.
- Port Number: It's used to identifies each RDBMS application, port number differ from one application to another application.
- Database Name: It's uniquely identifies database present in RDBMS server.
- Credentials: Username& Password.

DBURL EG:

Jdbc:mysql://localhost:3306/DatabaseName" user=root*password=root;

Java.sql.Statement -----> Its used to Carry Static Query

Java.sql.PreparedStatement ---> Its used to carry dynamic query

Java.sql.callableStatement ---> Its used to perform pl/sql queries

Types of Execute Methods:

Int ---> executeUpdate(String sql);

ResultSet ---> executeQuery(String sql);

Int[] ---> executeBatch(String sql);

Boolean ---> execute();

ExecuteQuery:

Methods are used to execute our queries.

Ex:

Public int executeUpdate(String sql) throws sqlException{}; -->Using this query for INSERT, UPDATE, DELETE Type of queries.

Public ResultSet executeQuery(String sql) throws sqlException; ---> SELECT Type of Queries.

Executing Dynamic Queries:

Since the query is dynamic we should use prepared statement.

To use prepare statement interface following the methods are used.

Syntax:

Public java.sql.PreparedStatement
prepareStatement(String sql) throws SqlException;

To give the values during runtime use setter method in prepared statement using object.

Public void setInt(PlaceHolderPosition, value);
Public void setString(PlaceHolderPosition.value)
Public void setDouble(PlaceHolderPosition,value);
Public void setLomg(PlaceHolderPosition,value);
Public void setShort(PlaceHolderPosition,value);

Code to ExecuteQueries:

```
PreparedStatement p=c.prepareStatement("insert into table_name
value(?,?,?,?,?)");
p.setString(1,values);
p.setString(2,values);
p.setString(3, values);
p.setString(4, values);
p.setString(5,values);
```

Execute the Select Type of Query:

Since the query is select type use following methods to execute.

Syntax;

Public ResultSet executeQuery(String sql) throws sqlException;

The return type of this method is ResultSet.

The output of select type of query will be in the form of table and it will be given in the form of resultset object.

To fetch the data from ResultSet following operation has to be done.

First check the next data is present or not using following method
Public boolean next() throws sqlException
The above method returns true if condition the next record are present.
If the records are present in ResultSet get the data using getter method present inside ResultSet object.
Public int getInt(int columnIndex);
Public int getLong(int columnIndex);
Public int getString(int columnIndex);
Public int getDouble(int columnIndex);
Public int getShort(int columnIndex);

Ex:

```
ResultSet rs=s.executeQuery("select *from table_name);
While(rs.next()){
System.out.println("StudentName:"+rs.getString(1));
System.out.println("Student Id:"+rs.getInt(2));
System.out.println("Percentage:"+rs.getDouble(3));
```

Note:

Column index always starts from 1.

Column index depend on result of SqlQuery.
Ex: column 1 2
Select sal, ename from emp;

Closing a Connection:

To use memory efficiently we must close the connection following methods are used.

Syntax:

Void close() throws sqlException;

Ex:

(i) c.close();

(ii) finally{ //Make the connection c as globally variable.

```
Try{
c.close();
} catch (SQLException e){
e.printStackTrace();
}}
```

ExecuteBatch():

ExecuteBatch() is used to execute multiple queries in single transaction.

Batch processing Methods:

(i) addBatch() ---> It's used to add the statement to the batch.

Syntax:

Public void addBatch(String query) throws SQLException

(ii) executeBatch(): ----> It's used to execute the batch.

Syntax:

Public int[] executeBatch() throws SQLException

ClearBatch():

It's used to remove all statement from the batch.

Syntax:

Public void clearBatch() throws SQLException

Execute();

If we don't know what type of query it is whether its insert, update, delete type of query or select type of query. In this case we use execute method.

Important Note:

What is JDBC?

What is driver software?

Component of JDBC?

(i) Interfaces:

- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet

(ii)Classes:

- DriverManager
- BLOB(Binary Large Object)
- CLOB

Steps to Connect Java with Database?
What do you mean by statement? And explain its types?
Difference between Statement and PreparedStatement?
What do you mean by DriverManager?
What do you mean connection interface?
Explain ResultSet?
Explain about Batch Processing and explain its types?
How many ways we can load/Road Register?Explain internally/Externally?
Difference type of Execute Methods?

Disadvantage of Java Project:

We need to add multiple JAR file we can't remember 'n' number of JAR files. To Overcomes this we use Maven project

Maven Project:

To overcome the disadvantage of Java project we use Maven project.

Steps To create Maven Project:

- Click on file
- Go for new
- Go for other
- And search for maven project and choose it.
- Click on Checkbox -> Create Simple project and click next
- Group id -> Stands for package name
- Artifact id -> Stands for project name
- Click Finish

How to update Maven project:

- Right click on the project.
- Choose maven
- Click on update project
- Choose which project you want to update
- Click on checkbox called as force update
- And click on okay.

Hibernate:

Hibernate is a java framework that simplifies the development of java application, On to interact with the database. It is an open source, lightweight, ORM(Object Relational Mapping) tool.

Hibernate implements the specifications of JPA(Java persistence API) for data persistence.

Advantage of Hibernate:

- Open source and lightweight
- Fast performance
- Database independent Query
- Automatic table creation
- Simplifies complex join.

JPA(Java Persistence API):

Java Persistence API(JPA) is a java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

ORM TOOL:

An ORM tool simplifies the data creation, data manipulation and data access. It's a programming technique that maps the object as one row in the table in the database.
Java Application ->Object->ORM->Database.

Entity Manager Factory:

Entity manager factor provides instance of entity manager for connecting to same database. All the instances are configured to use the same setting as defined by the default implementation. Several entity manager facilities can be prepared for connecting to different database.
The EntityManagerFactory interface present in javax.persistence package is used to provide an entity manager.
Persistence:

The persistence is a bootstrap class whch is used to obtain an Entity Manager Factor interface.

Syntax:

```
EntityManagerFtory emf=
Persistence.CreateEntityManagerFactory("String");//String is persistence unit name.
```

Entity Manager(EM):

Entity Manager is an interface provided by Java Persistence API specification.
Entity Manager is used to manage the lifecycle of entity instance, such as,

- Create and remove persistence entity instance.
- Find entity by their primary key.
- Query over entities.

Method Present in EM:

Persist() --> It is used to save the instance.
Merge() -> It is used to merge the state of given entities into the current persistence context.
Remove() --> It is used to remove the instance.
Find() --> It is used to fetch the entities based on primary key

Entity Transaction:

Entity Transaction is a interface is used to control the transactions on resource local entity manager.

Important Methods:

Begin():

Begin() is used to start the transaction.

Commit():

This method is used to commit the transaction.

JPQL(Java Persistence Query Language):

It's an platform independent object oriented query language define as an part of java persistence API specification.
JPQL is used to make queries against an entity stored in relational database.
JPQL inspired by SQL.
Note: In JPQL we worked with entities and collections of entities.
While in sql we worked with column and rows
SQL: select * from table_name;
JPQL: select alias_name from entity_class_name alias_name;
SQL: select *from table_name where name=?;
JPQL: select alias_name from entity_class_name where aliasname.name=?;

Types of Query Parameter:

Similar to JDBC prepared statement parameters, JPA specifies two different way or writing parameterized queries by using,

- Positional Parameter
- Named Parameter

Difference between Persist and merge method:

Persist	Merge
Persist should be called only on new entities.	Merge is means to re-attach entities
It will persist the entity object in database.	It will update the object in the database for duplicate key.
If you pass the object with duplicate primary key it will throw exception.	If the primary key is not matches it will insert the object as new record in table.

Mapping:

The mapping of association between entity classes and relationship between soul of ORM. Hibernate mapping are one of the key features of hibernate they establish the relationship between two database tables as an attributes in your model.

You can establish either uni directional or bi-directional.

Different Mapping Notations Are:

- @OneToOne(Uni-Direction)
- @OneToMany(Uni-Direction)
- @ManyToOne(Uni-Direction)
- @ManyToMany(Uni-Di/rection)
- @OneToOne(Bi-Direction)
- @OneToMany(Bi-Direction)
- @ManyToMany(Bi-Direction)

One to One(Bi-Directional):

Bi-Directional associates allows us to fetch details of dependent object from both side. In such case, we have the reference of two classes in each other. Let's take an example of same Person and Pan, now Person class has-a-reference of Pan and Pan has a reference of Person. Here you can retrieve the data in bi-direction. If you have if you can get pan details and also you can get person details with pan id.

Ex: One person has one pan, a pan is associated with a single person.

Many to One Bi-Directional:

In bi-direction, in many-to-one mapping many instances of entity is associated with one instance of other entity. For example many branches have one hospital.

In one-to-many mapping one instance of entity is associated with many instance of other entity. For Example one hospital have many branches.

Here you can retrieve the data in bi-direction. If you have branch id you can get the hospital details and you can get branch details with hospital id. For Example, many branches are associated with one hospital.

Many-to-Many Uni Directional:

In many-to-Many mapping many instances of entity is associated with many instances of other entity. For example many students have many courses.'

Here you can retrieve the data in uni-direction. If you have student id you can't get the all the details of courses but you cannot get student details with course id.

Ex: Many students are associated with many courses.

Many-to-Many Bi-Directional:

In many-to-many mapping many instances of entity is associated with many instances of other entity. For example, many students have many courses and many courses have many students.

Here you can retrieve the data in bi-direction. If you have student id you can get all the details of the courses and you can get all student details with course id.

Ex: Many students are associated with many courses.

Persistent Unit in Hibernate:

A persistence unit defines a set of all entity classes that are managed by entity manager instance in an application.

It contains configurations file which is required for connection.

It consists of url, username, password.

It's used to perform actions on database.

Cascading in Hibernate:

Cascading is a feature in Hibernate, which is used to manage the state of the mapped entity whenever the state of its relationship owner (superclass) affected. When the relationship owner(superclass) is saved/deleted, then the mapped entity associated with it should also be saved/deleted automatically.

When we perform some action on the target entity, the same action will be applied to the associated entity.

Fetching in Hibernate:

The process of data fetching or loading called as fetch type in hibernate.

There are two types of fetches,

- Lazy
- Eager

Default fetching:

- One-to-One --> Eager
- One-to-Many --> Lazy
- Many-to-One --> Eager
- Many-to-Many --> Lazy

Lazy Fetching:

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 uses case.

Eager Fetching:

The FetchType.EAGER tells hibernate to get all elements of a relationship when selecting the root entity.

Caching in Hibernate:

Hibernate caching improves the performance of the application by pooling the object in the cache.

It's useful when we have to fetch the same data multiple times.

There are mainly two types of caching,

- First Level Cache
- Second Level Cache

First-Level Cache:

Session object holds the first-level cache data. It's enables by default.

The first-level cache data will not be available for entire application. An application can use many session object.

Each and every Entity Manager has its own first level cache memory.

Second-Level Cache:

SessionFactor object holds the second level cache data. The data stored in the second level-cache will be available for an entire application. But we need to enable it explicitly.
Three steps to enable second level cache:

- Add hibernate - ehCache dependency(version should be same as hibernate core version).
- Add @cacheable annotation to entity class.
- Enable second -level cache by configuring in persistence.xml file.

Entity Life Cycle of Hibernate:

Transient State:

The transient state is the initial state of an object.
Once we create an instance of POJO class, then the object entered in the transient state.
Here, an object is not associated with the Session. So, the transient state is not related to any database.
Hence, modifications in the data don't affect any changes in the database.
The transient objects exists in the heap memory. They are independent of hibernate.

Persistent State:

Soon as the object is associated with the Session, it entered in the persistence state.
Hence. We can say that an object is in the persistence state when we save or persist it.
Here, each object represents the row of the database table.
So modifications in the data make changes in the database.

Detached State:

Once we either close the session or clear its cache, then the object entered into the detached state.
As an object is no more associated with the Session, modifications in the data don’t affect any changes in the database.
However, the detached object still has representation in the database.
If we want to persist the changes made to a detached object, it is required to reattach the application to a calid Hibernate Session.
To associate the detached object with the new Hibernate Session, use any of these methods load(), merge(), refresh(), update() or save() on a new session with the reference of the detached object.

Removed State:

It's the last state in the hibernate .
Whenever the entity object is deleted from the database then the entity object is saved to be removed state. It can achieved by calling remove()

WebServer:

Like any other application like (media player, adobe reader…)
Web server is also an application which runs on operating system.
Web server has a name implies(Serves request to a web application).
In other words, It helps both web browser and web application to interact with each other.

Hence every application is directly under the control of a web server.
Few Examples of Web Server such as,

- Apache TomCat
- RedHat Jboss
- IBM WebSpear
- Oracle WebLogic

URL Format:

Protocol://hostname:portnumber/application name/resource locator
http:// localhost :8080 /ServletDemo /resource filename(HTML,CS,JS file)
8080 is default port number for http
ServletDemo is a project name where we're currently working on it.

Servlets:

Servlet is an technology used to create an dynamic web application.
Servlet is an API that provides many interfaces and classes.
Servlet is an interface that must be implemented for creating any servlet.
Servlet will take request from the user and generate response for that request.

Servlet is an interface in javax.servlet.package
It has 5 Abstract methods.

- Init()
- Service()
- Destroy()
- getServiceInfo()
- getServletConfig()

You can create a servlet by implementing the servlet interface and providing the implementation for all these methods.

How to Achieve Servlets:

- Class A implements Servlet(Interface)
- Class A extends GenericServlet(Sub Abstract Class)
- Class A extends HttpServlet(Sub Abstract Class).

Deployment Descriptor/Web.xml:

- Another name for web.xml is deployment descriptor.
- So it's a pipe used by Servlet container to define which servlet class is mapped with which URL.

Ex:

```
<servlet>
<servlet-name>Static</servlet-name>
<servlet-class>com.application.swiggy.StaticServlet</servlet-
class>
</servlet>

<servlet-mapping>
<servlet-name>Static</servlet-name>
<url-pattern>/static</url-pattern>
</servlet-mapping>
```

Query Parameter(Query String):

Query parameter is used to transfer your data from front-end to back-end.

It will be in the form of key and value pairs.
To separate URL and Query parameter we use placeholder(?)
One key and value pair will be separated with & symbol.
Ex: http://localhost:8080/ServletDemo/add?num1=10&num2=20

Get Parameter():

Get Parameter() is used to fetch the data from the URL based on key.
The return type of get parameter() is String.
Ex:

```
String num1= req.getParameter("num1");
String num2= req.getParameter("num2");
```

PrintWriter:

Print Writer is used to print on browser.
Print writer object can be obtain with the help of getWriter() present in ServletResponse.
Ex:

```
PrintWriter out = res.getWriter();
Out.print(html);
```

Web Container:

A web container is the component of a web server that interacts with the java servlet.
A web container manages the life cycle of servlets and it maps a URL to a particular servlet while ensuring that the requester has relevant access rights.
Java Servlet do not have main method, so a container is required to load them. **The servlet gets deployed on the web container.**

ServletLifeCycle:

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

- Servlet class is loaded.
- Servlet instance of object is created.
- Init() is invoked
- Service() is invoked
- Destroy() is invoked

LifeCycle Methods:

The web container calls the init() only once after creating the servlet instance.
Init() is used to initialize the servlet.
Ex:

```
@Override
 public void init() throws ServletException {
 System.out.println("Hi im init Method");
 }
```

Service():

Service() is the main method to perform the actual task.
It will take request from the user and create response.
Ex:

```
@Override
 public void service(ServletRequest req, ServletResponse res)
throws ServletException, IOException {

System.out.println("Hi im service method");

}
```

Destroy():

The web container calls the destroy() before removing servlet object from the service.
Ex:

```
@Override
 public void destroy() {
 System.out.println("Hi im destroy method");
 }
```