Day-12

- **Agenda**
  - default & static methods
  - JDBC introduction and CRUD operation
  - Unit Testing- Junit test cases

Default & static methods

- **Default Methods** – Unlike other abstract methods these are the methods can have a default implementation. If you have default method in an interface, it is not mandatory to override (provide body) it in the classes that are already implementing this interface.

- **Static methods -** They are declared using the static keyword and will be loaded into the memory along with the interface. You can access static methods using the interface name. If your interface has a static method you need to call it using the name of the interface, just like static methods of a class.

- The **default methods** were introduced to provide backward compatibility so that existing interfaces can use the lambda expressions without implementing the **methods** in the implementation class. **Default methods** are also known as defender **methods** or virtual extension **methods**.

- Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default are known as default methods. These methods are non-abstract methods.

- Default methods are the method defined in the interfaces with method body and using the default keyword. Thus we can add instance methods to the interfaces. Default method can call methods from the interfaces they are enclosed in.

- **Static variable** in Java is variable which belongs to the class and initialized only once at the start of the execution. It is a variable which belongs to the class and not to object(instance). Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables.
- **Static method** in Java is a method which belongs to the class and not to the object. A static method can access only static data. It is a method which belongs to the class and not to the object(instance). A static method can access only static data. It cannot access non-static data (instance variables).
  - A static method can call only other static methods and can not call a non-static method from it.
  - A static method can be accessed directly by the class name and doesn’t need any object
  - A static method cannot refer to "this" or "super" keywords in anyway

Default Method Example

```
interface MyInterface{
   public static int num = 100;
   public default void display() {
      System.out.println("display method of MyInterface");
   }
}
public class InterfaceExample implements MyInterface{
   public static void main(String args[]) {
      InterfaceExample obj = new InterfaceExample();
      obj.display();
   }
}
```

Static Method Example

```
interface MyInterface{
  public void demo();
  public static void display() {
    System.out.println("This is a static method");
  }
}
public class InterfaceExample{
  public void demo() {
    System.out.println("This is the implementation of the demo method");
  }
  public static void main(String args[]) {
    InterfaceExample obj = new InterfaceExample();
    obj.demo();
    MyInterface.display();
  }
}
```

Difference between default and static interface method

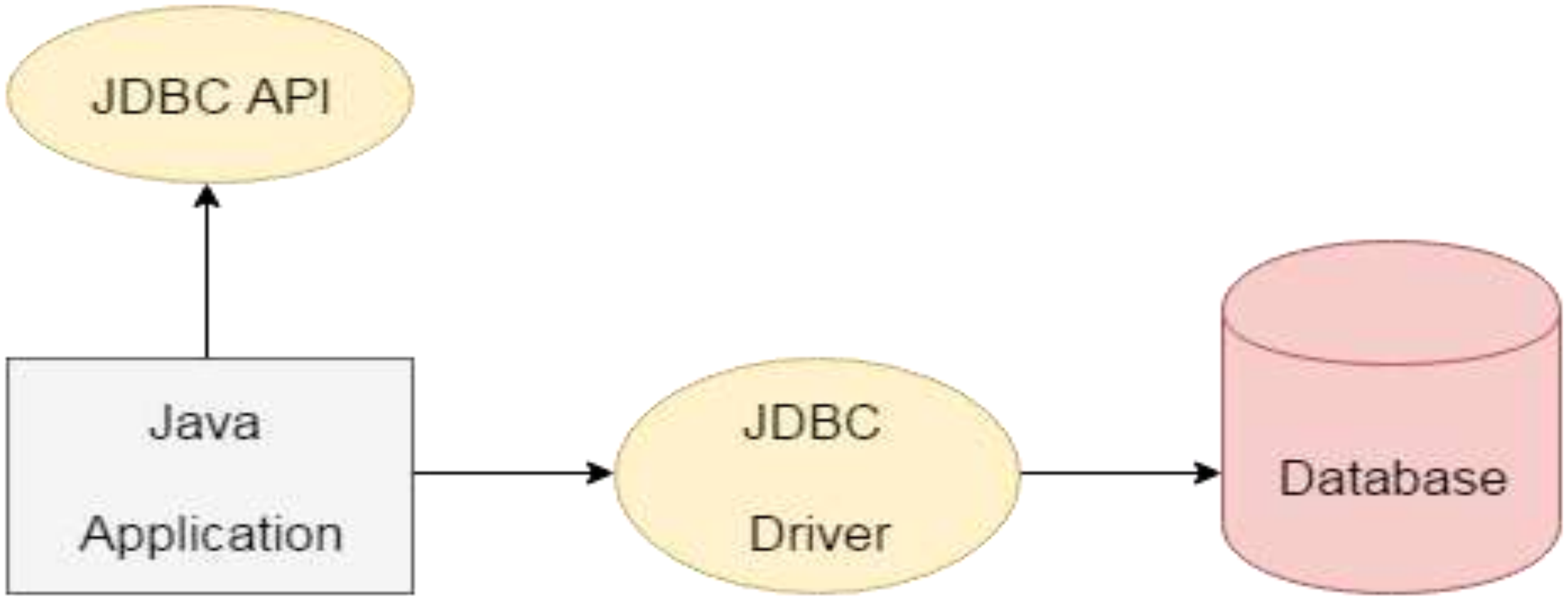
Sr. No.	Key	Static Interface Method	Default Method
1	Basic	It is a static method which belongs to the interface only. We can write implementation of this method in interface itself	It is a method with default keyword and class can override this method
2	Method Invocation	Static method can invoke only on interface class not on class.	It can be invoked on interface as well as class
3	Method Name	Interface and implementing class , both can have static method with the same name without overriding each other.	We can override the default method in implementing class
4.	Use Case	It can be used as a utility method	It can be used to provide common functionality in all implementing classes

JDBC introduction and CRUD operation

- ➢JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:
  - JDBC-ODBC Bridge Driver,
  - Native Driver,
  - Network Protocol Driver, and
  - Thin Driver

- ➢JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.

- The JDBC API consists of classes and methods that are used to perform various operations like: connect, read, write and store data in the database.
- You can get idea of how JDBC connect Java Application to the database by following image.



- Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

- We can use JDBC API to handle database using Java program and can perform the following activities:
  1. Connect to the database
  2. Execute queries and update statements to the database
  3. Retrieve the result received from the database.

- JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
    - Making a connection to a database.
    - Creating SQL or MySQL statements.
    - Executing SQL or MySQL queries in the database.
    - Viewing & Modifying the resulting records.

- Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −
- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs).

- All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

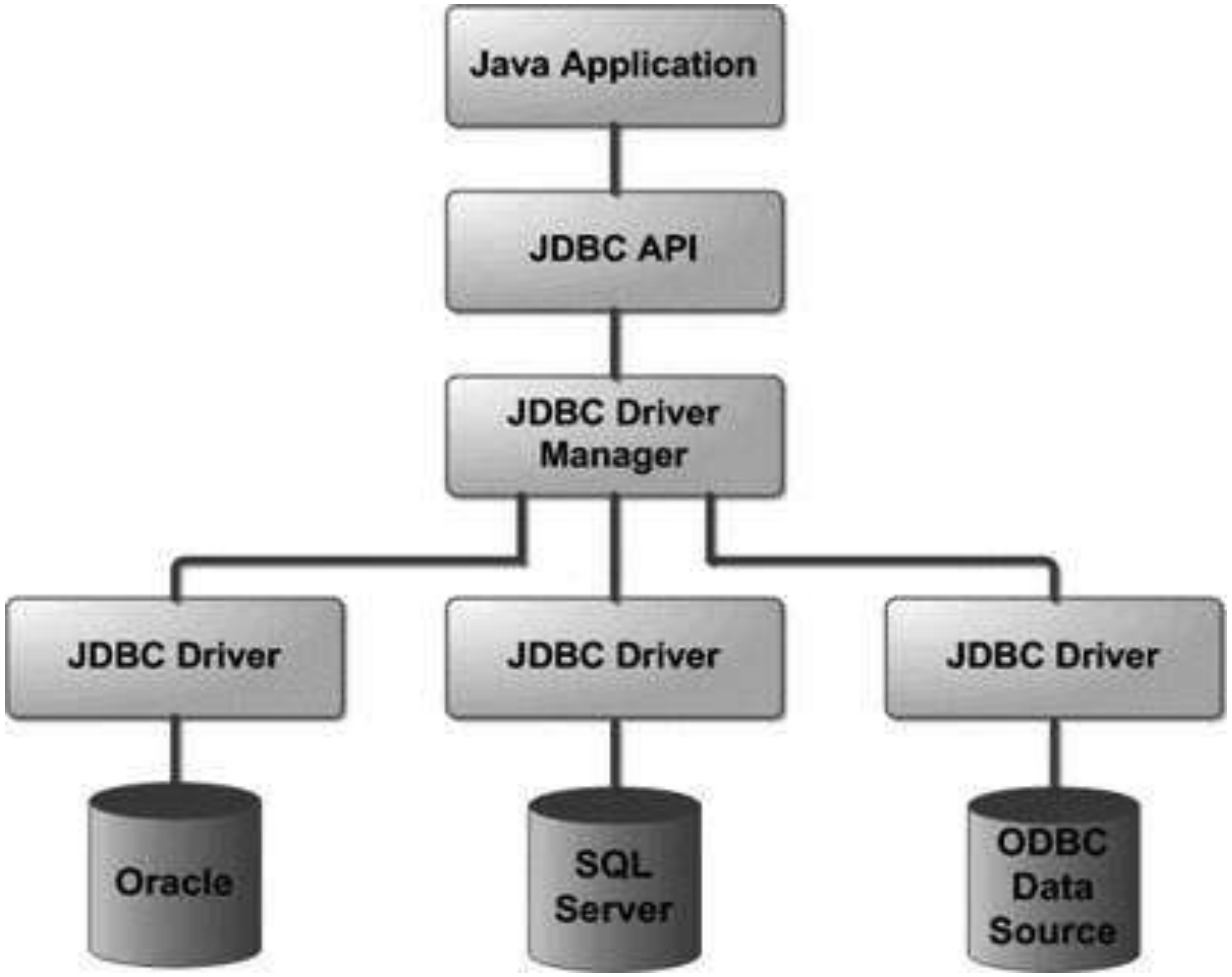
- JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

JDBC architecture overview:-

- Java application code calls a JDBC library; JDBC (DriverManager class) then loads a database driver and the driver then talks to a particular database.

- JDBC architecture can be 2-tier or 3-tier. In the 2-tier architecture, java apps directly communicate with the data source using JDBC API.

- In the 3-tier architecture, the client delivers the command to the middle tier; this then sends the command to the database or other data source. Here the middle tier implements the JDBC driver to interact with the particular database.


Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

- ➢JDBC Architecture consists of two layers −
  - **JDBC API** − This provides the application-to-JDBC Manager connection.
  - **JDBC Driver API** − This supports the JDBC Manager-to-Driver Connection.

- ➢The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

- ➢The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Common JDBC Components

- DriverManager − This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- Driver − This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- Connection − This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- Statement − You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- ResultSet − These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- SQLException − This class handles any errors that occur in a database application.

JDBC driver

- A JDBC driver is a set of classes/interfaces that implements the interfaces that are provided in the JDBC. API such as java.sql.Driver, in database specific way. Usually, each vendor such as MySQL, Oracle etc. provides these drivers.

- A Java application uses JDBC to interact with relational databases without knowing about the underlying JDBC driver implementations.

- JDBC drivers can be classified as Type-1, Type-2, Type-3 and Type-4:
  - Type-1 drivers are bridge drivers (like JDBC-ODBC Bridge).
  - Type-2 drivers are partly java and partly native drivers (like Type-2 driver for sybase) that uses JNI calls for DB specific native client APIs.
  - Type-3 (like Weblogic RMI driver) uses a middleware to connect to a database and
  - Type-4 (like thin driver for oracle) directly connects to a database. We will be using type-4 drivers for our

Basic steps for using JDBC in an application

1. Obtaining the connection
2. Creating a JDBC statement object that can hold an SQL statement
3. Executing the statement and retrieving the result.
4. Closing the connection

- We have seen the terms Database, relational database, SQL, ODBC etc. in our discussion. We will quickly see what they are.

Working of JDBC

- Java application that needs to communicate with the database has to be programmed using JDBC API.
- JDBC Driver supporting data sources such as Oracle and SQL server has to be added in java application for JDBC support which can be done dynamically at run time.
- This JDBC driver intelligently communicates the respective data source.
- *Lets implement ………*

CRUD operation

- CRUD - Create, Retrieve, Update and Delete

- These CRUD operations are equivalent to the INSERT, SELECT, UPDATE and DELETE statements in SQL language. Although the target database system is MySQL, but the same technique can be applied for other database systems as well because the query syntax used is standard SQL which is supported by all relational database systems.

- Lets write code and practice this with JDBC………

Unit Testing- Junit test cases

- ➢Unit testing is a form of white box testing, in which test cases are based on internal structure. The tester chooses inputs to explore particular paths, and determines the appropriate output.

- ➢The purpose of unit testing is to examine the individual components or pieces of methods/classes to verify functionality, ensuring the behaviour is as expected.

- ➢Types of unit testing
  - 1) Manual Testing:-If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.
  - 2) Automated Testing:- If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

- Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.

- JUnit is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code

- JUnit is an open source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests.

- As the name implies, it is used for Unit Testing of a small chunk of code.

- Once you are done with code, you should execute all tests, and it should pass. Every time any code is added, you need to re-execute all test cases and makes sure nothing is broken.

- Why you need JUnit testing
    - It finds bugs early in the code, which makes our code more reliable.
    - JUnit is useful for developers, who work in a test-driven environment.
    - Unit testing forces a developer to read code more than writing.
    - You develop more readable, reliable and bug-free code which builds confidence during development.

Annotations for Junit testing

- **@Test** annotation specifies that method is the test method.
- **@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).
- **@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.
- **@Before** annotation specifies that method will be invoked before each test.
- **@After** annotation specifies that method will be invoked after each test.
- **@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests.

Assert class

- Assert is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class org.junit.Assert which extends java.lang.Object class.

- There are various types of assertions like Boolean, Null, Identical etc.

- Junit provides a class named Assert, which provides a bunch of assertion methods useful in writing test cases and to detect test failure.

- The assert methods are provided by the class **org.junit.Assert** which extends **java.lang.Object** class.

- JUnit Assert methods
  - Boolean
  - Null object
  - Identical
  - Assert Equals
  - Assert Array Equals
  - Fail Message

JUnit Assert methods :Boolean

- If you want to test the boolean conditions (true or false), you can use following assert methods:-

    1. assertTrue(condition)
    2. assertFalse(condition)

    Here the condition is a boolean value.

JUnit Assert methods : Null Object

- ➢If you want to check the initial value of an object/variable, you have the following methods:

  1. assertNull(object)
  2. assertNotNull(object)

  Here object is Java object

JUnit Assert methods : Identical

- ➢If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.
  - assertSame(expected, actual), It will return true if expected == actual
  - assertNotSame(expected, actual)

Assert Equals, Assert Array Equals

- If you want to test equality of two objects, you have the following methods
  - **assertEquals(expected, actual)**
  - It will return true if: **expected.equals(actual)** returns true.

- If you want to test equality of arrays, you have the following methods as given below:
  - **assertArrayEquals(expected, actual)**

- Above method must be used if arrays have the same length, for each valid value for **i**, you can check it as given below:
  - **assertEquals(expected[i],actual[i])**
  - **assertArrayEquals(expected[i],actual[i])**

Fail Message

- ➢If you want to throw any assertion error, you have **fail()** that always results in a fail verdict.
    - **Fail(message);**

- ➢You can have assertion method with an additional **String** parameter as the first parameter. This string will be appended in the failure message if the assertion fails. E.g. **fail(message)** can be written as
    - **assertEquals(message, expected, actual)**

Lets Practice……….