PROGRAMMING IN JAVA

UNIT-4

Thread:

A Thread is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

In order to perform complicated tasks in the background, we used the Thread concept in Java. All the tasks are executed without affecting the main program. In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent.

In a simple way, a Thread is a:

- Feature through which we can perform multiple activities within a single process.
- Lightweight process.
- Series of executed statements.
- Nested sequence of method calls.

Types of Thread:

1. **User thread:**

This is created by the Java programmer or user. JVM wait for these threads to finish their task. These threads are foreground threads.

2. **Daemon thread:**

This is created by the operating system. JVM doesn't wait for daemon threads to finish their task. These threads are used to perform some background tasks like garbage collection.

Thread Priorities:

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Thread class is the main class on which Java's Multithreading system is based. Thread class, along with its companion interface Runnable will be used to create and run threads for utilizing Multithreading feature of Java.

Thread Model/ Thread life cycle:

Just like a process, a thread exists in several states. These states are as follows:

1. New (Ready to run)

A thread is in New when it gets CPU time.

2. Running

A thread is in a Running state when it is under execution.

3. Suspended

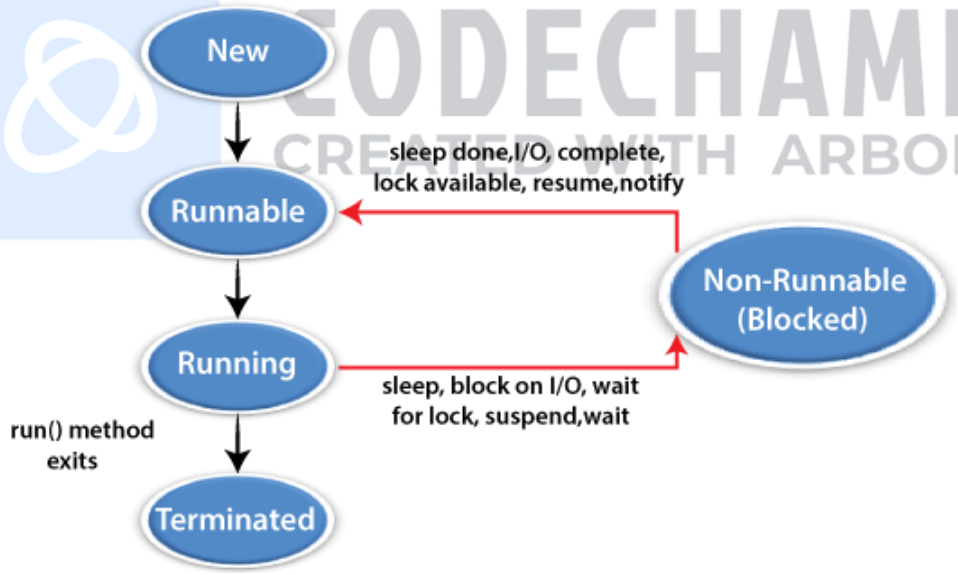
A thread is in the Suspended state when it is temporarily inactive or under execution.

4. Blocked

A thread is in the Blocked state when it is waiting for resources.

5. Terminated

A thread comes in this state when at any given time, it halts its execution immediately.



Multitasking:

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

1. Process-based Multitasking (Multiprocessing)
2. Thread-based Multitasking (Multithreading)

1. Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2. Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

Multithreading:

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading:

- It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
- You can perform many operations together, so it saves time.
- Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Creating Thread:

A thread is created either by "creating or implementing" the **Runnable Interface** or by extending the **Thread class**. These are the only two ways through which we can create a thread.

Let's dive into details of both these way of creating a thread:

Thread Class:

A **Thread class** has several methods and constructors which allow us to perform various operations on a thread. The Thread class extends the **Object** class. The **Object** class implements the **Runnable** interface. The thread class has the following constructors that are used to perform various operations.

- Thread()

- Thread(Runnable, String name)
- Thread(Runnable target)
- Thread(ThreadGroup group, Runnable target, String name)
- Thread(ThreadGroup group, Runnable target)
- Thread(ThreadGroup group, String name)
- Thread(ThreadGroup group, Runnable target, String name, long stackSize)

Runnable Interface(run() method)

The Runnable interface is required to be implemented by that class whose instances are intended to be executed by a thread. The runnable interface gives us the **run()** method to perform an action for the thread.

start() method

The method is used for starting a thread that we have newly created. It starts a new thread with a new callstack. After executing the **start()** method, the thread changes the state from New to Runnable. It executes the **run() method** when the thread gets the correct time to execute it.

Let's take an example to understand how we can create a Java thread by extending the Thread class:

ThreadExample1.java


```
1.  public class ThreadExample1 extends Thread {
2.       public void run()
3.      {
4.         int a= 10;
5.         int b=12;
6.         int result = a+b;
7.         System.out.println("Thread started running..");
8.         System.out.println("Sum of two numbers is: "+ result);
9.      }
10.     public static void main( String args[] )
11.     {
12.           ThreadExample1 t1 = new  ThreadExample1();
13.            t1.start();
14.     }
15. }
```

Inter-thread Communication:

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:

wait()-It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify().

notify()-It wakes up one single thread that called wait() on the same object.

notifyAll()-It wakes up all the threads that called wait() on the same object.

Difference between wait() and sleep()

wait()	sleep()
called from synchronised block	no such requirement
monitor is released	monitor is not released
gets awake when notify() or notifyAll() method is called.	does not get awake when notify() or notifyAll() method is called
not a static method	static method
wait() is generaly used on condition	sleep() method is simply used to put your thread on sleep.

Thread Deadlock:

Deadlock is a situation of complete Lock, when no thread can complete its execution because lack of resources. Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order.

JDBC (Java Database Connectivity):

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).

Why Should We Use JDBC:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)

4. Thin driver (fully java driver)

1. JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

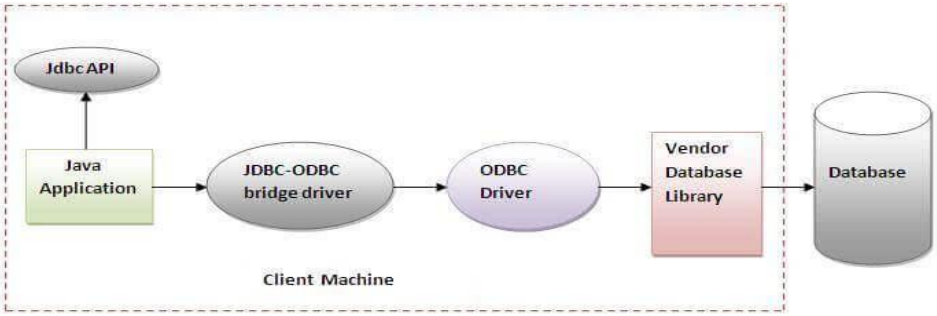


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2. Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

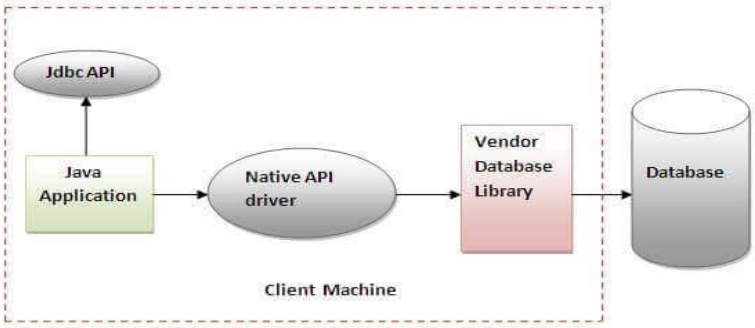


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3. Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

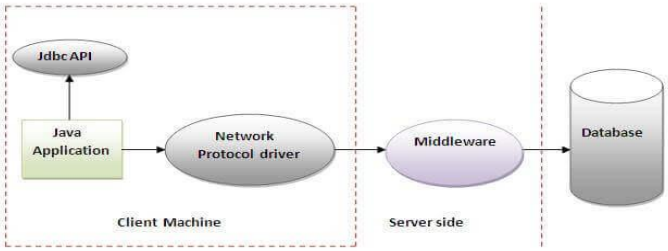


Figure- Network Protocol Driver

Advantage:

- No client-side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4. Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

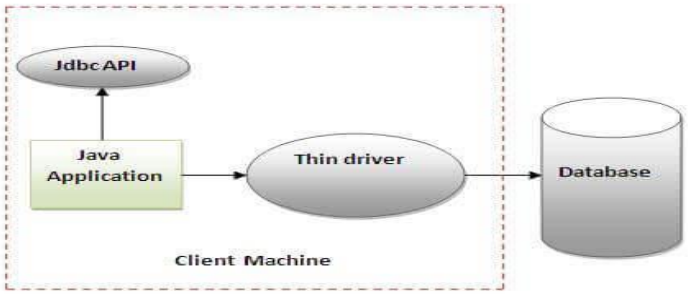


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

JDBC connection with a Database:

5 Steps to connect to the database in java

1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

1. Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)throws ClassNotFoundException

Example:

Here, Java program is loading oracle driver to esteblish database connection.

Class.forName("oracle.jdbc.driver.OracleDriver");

2. Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1) public static Connection getConnection(String url)throws SQLException

2) public static Connection getConnection(String url,String name,String password)

throws SQLException

Example:

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

3. Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Example:

Statement stmt=con.createStatement();

4. Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

Example:

ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

5. Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

public void close()throws SQLException

Example:

con.close();

1. Two-Tier Database Architecture:

In two-tier, the application logic is either buried inside the User Interface on the client or within the database on the server (or both). With two-tier client/server architectures, the user system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients.

2. Three-Tier Database Architecture:

In three-tier, the application logic or process lives in the middle-tier, it is separated from the data and the user interface. Three-tier systems are more scalable, robust and flexible. In addition, they can integrate data from multiple sources. In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers.

Difference Between Two-Tier And Three-Tier Database Architecture:

S.NO	Two-Tier Database Architecture	Three-Tier Database Architecture
1	It is a Client-Server Architecture.	It is a Web-based application.
2	In two-tier, the application logic is either buried inside the user interface on the client or within the database on the server (or both).	In three-tier, the application logic or process resides in the middle-tier, it is separated from the data and the user interface.
3	Two-tier architecture consists of two layers : Client Tier and Database (Data Tier).	Three-tier architecture consists of three layers : Client Layer, Business Layer and Data Layer.
4	It is easy to build and maintain.	It is complex to build and maintain.
5	Two-tier architecture runs slower.	Three-tier architecture runs faster.

6	It is less secured as client can communicate with database directly.	It is secured as client is not allowed to communicate with database directly.
7	It results in performance loss whenever the users increase rapidly.	It results in performance loss whenever the system is run on Internet but gives more performance than two-tier architecture.
8	Example – Contact Management System created using MS-Access or Railway Reservation System, etc.	Example – Designing registration form which contains text box, label, button or a large website on the Internet, etc.

Creating and executing SQL statements:

This article is going to help you in learning how to do basic database operations using JDBC (Java Database Connectivity) API. These basic operations are **INSERT, SELECT, UPDATE and DELETE** statements in SQL language. Although the target database system is Oracle Database, but the same techniques can be applied to other database systems as well because of the query syntax used is standard SQL is generally supported by all relational database systems.

Principal JDBC interfaces and classes

Let's take an overview look at the JDBC's main interfaces and classes which we'll use in this article. They are all available under the *java.sql* package:

Class.forName() : Here we load the driver's class file into memory at the runtime. No need of using new or creation of object.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

DriverManager: This class is used to register driver for a specific database type (e.g. Oracle Database in this tutorial) and to establish a database connection with the server via its **getConnection()** method.

Connection: This interface represents an established database connection (session) from which we can create statements to execute queries and retrieve results, get metadata about the database, close connection, etc.

```
Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
```

Statement and **PreparedStatement**: These interfaces are used to execute static SQL query and parameterized SQL query, respectively. **Statement** is the super interface of the **PreparedStatement** interface. Their commonly used methods are:

A. **boolean execute(String sql):** executes a general SQL statement. It returns *true* if the query returns a *ResultSet*, false if the query returns an update count or returns nothing. This method can be used with a *Statement* only.
B. **int executeUpdate(String sql):** executes an INSERT, UPDATE or DELETE statement and returns an update account indicating number of rows affected (e.g. 1 row inserted, or 2 rows updated, or 0 rows affected).

```
Statement stmt = con.createStatement();
    String q1 = "insert into userid values
    ('" +id+ "', '" +pwd+ "', '" +fullname+ "', '" +email+ "')";
    int x = stmt.executeUpdate(q1);
```

ResultSet executeQuery(String sql): executes a SELECT statement and returns a *ResultSet* object which contains results returned by the query.

```
Statement stmt = con.createStatement();
    String q1 = "select * from userid WHERE id = '" + id + "'
    AND pwd = '" + pwd + "'";
    ResultSet rs = stmt.executeQuery(q1);
```

ResultSet: contains table data returned by a SELECT query. Use this object to iterate over rows in the result set using next() method.

SQLException: this checked exception is declared to be thrown by all the above methods, so we have to catch this exception explicitly when calling the above classes' methods.