JDBC

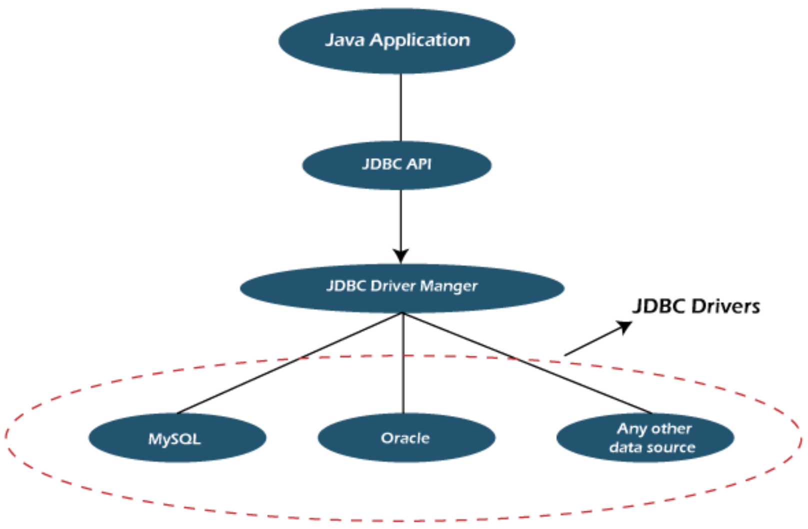
Components of JDBC :

Main Components :

1. The JDBC API – Provides various methods and interfaces for easy and effective communication with databases. (java.sql.*,javax.sql.*)
DriverManager
Driver
Connection
Statement
PreapredStatement
CallableStatement
ResultSet
DatabaseMetaData
Blob
Clob

2. JDBC DriverManager
JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database.
3. JDBC Test suite: JDBC Test suite facilitates the programmer to test the various operations such as deletion, updation, insertion that are being executed by the JDBC Drivers or not.
4. JDBC-ODBC Bridge Drivers: JDBC-ODBC Bridge Drivers are used to connect the database drivers to the database. The bridge does the translation of the JDBC method calls into the ODBC method call.

JDBC ARCHITECTURE



JDBC Versions(Specifications) :
Versions Of JDBC

Initially, Sun Microsystems had released JDBC in JDK 1.1 on Feb 19, 1997. After that, it has been part of the Java Platform.

The following table contains JDBC versions and implementations:

JDBC Version	JDK Implementation	Year
JDBC 1.2	JDK 1.1	1997
JDBC 2.1	JDK 1.2	1999
JDBC 3.0	JDK 1.4	2001
JDBC 4.0	Java SE 6	2006
JDBC 4.1	Java SE 7	2011
JDBC 4.2	Java SE 8	2014
JDBC 4.3	Java SE 9	2017

Drivers for Different Databases

Database	JDBC Driver Provider Name	JAR File Name
MySQL	Oracle Corporation	mysql-connector-java-VERSION.jar
Oracle	Oracle Corporation	ojdbc8.jar
SQL Server	Microsoft Corporation	sqljdbc41.jar, sqljdbc42.jar
Postgre SQL	PostgreSQL Global Development Group	postgresql-VERSION.jar
SQLite	Xerial.org	sqlite-jdbc-VERSION.jar
MS Access	UCanAccess.com	ucanaccess-VERSION.jar

Types of Drivers:

There are 4 different types of Drivers available in JDBC. They are classified based on the technique which is used to access a Database.
They are as follows:
Type I : JDBC- ODBC Bridge
Type II: Native API Partly Java Driver
Type III: Network Protocol(middleware Server Driver)- Fully Java Driver
Type IV: Thin Driver- Fully Java Driver

Type -1 Driver - JDBC ODBC Bridge Driver

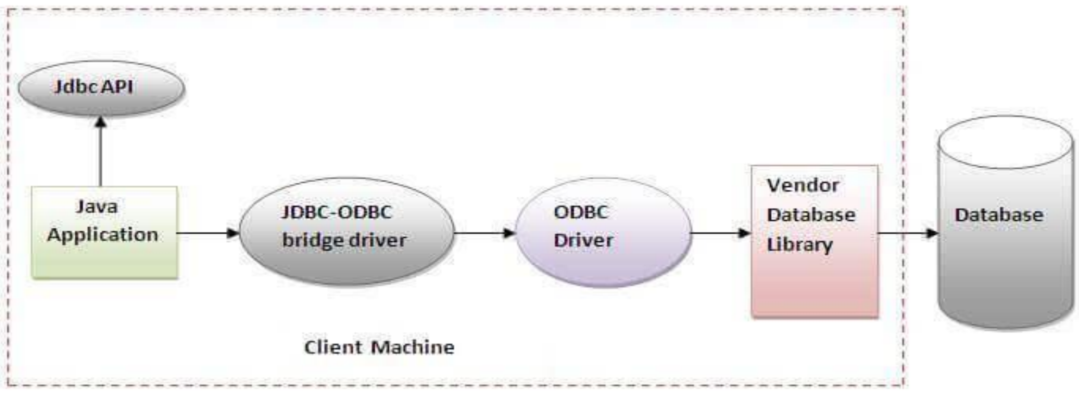


Figure- JDBC-ODBC Bridge Driver

Type -2 Driver - Native API Driver

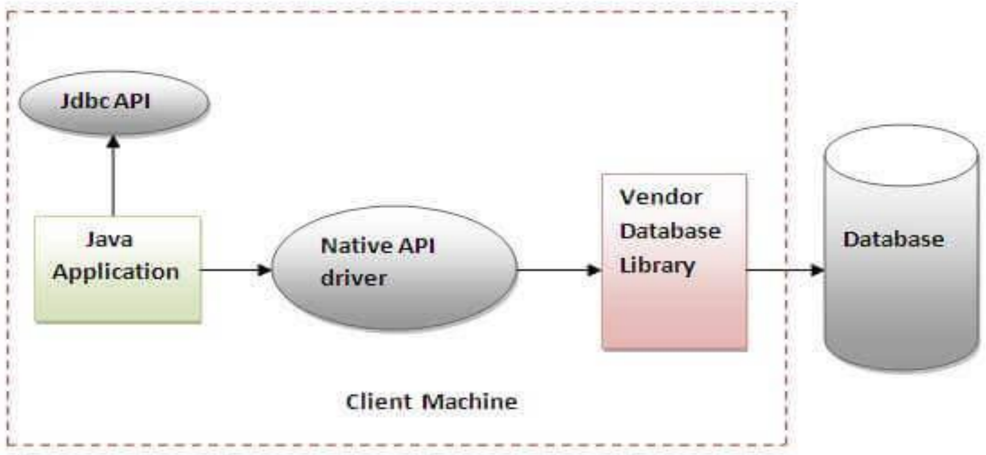


Figure- Native API Driver

Type -3 Driver- Network Protocol Driver

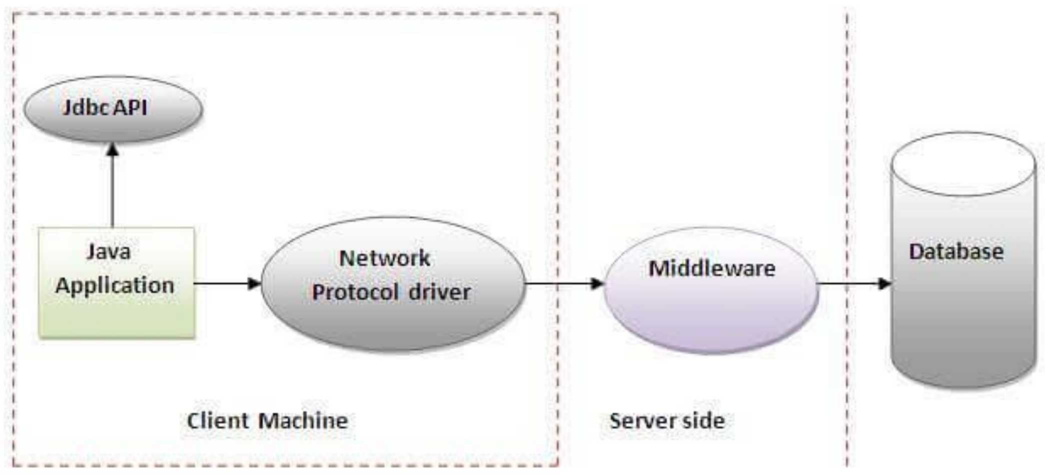


Figure- Network Protocol Driver

Type -4 Driver – Thin Driver

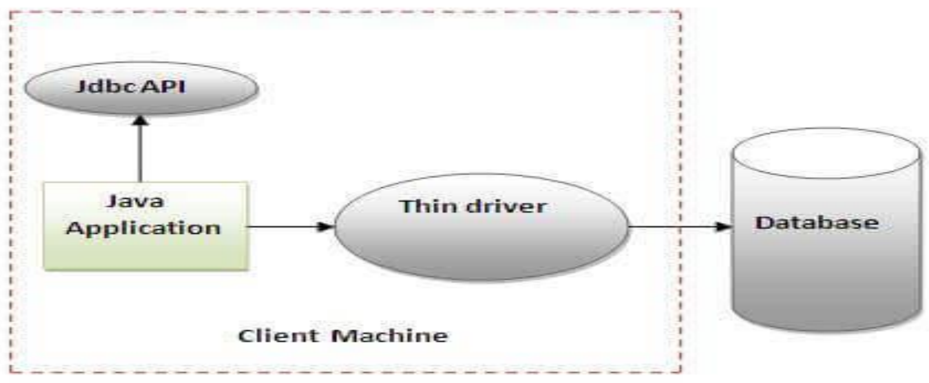


Figure- Thin Driver

Pros & Cons

1. JDBC-ODBC bridge driver:
JDBC-ODBC bridge driver is a native code driver which uses ODBC driver to connect with the database. It converts JDBC method calls into ODBC function calls. It is also known as Type 1 driver.
Advantages:
It can be used with any database for which an ODBC driver is installed.
Disadvantages:
Performance is not good as it converts JDBC method calls into ODBC function calls.
ODBC driver needs to be installed on the client machine.
Platform dependent.

2. Native-API driver:
Native-API driver uses the client-side libraries of the database. It converts JDBC method calls into native calls of the database API. It is partially written in java. It is also known as Type 2 driver.
Advantages:
It is faster than a JDBC-ODBC bridge driver.
Disadvantages:
Platform dependent.
The vendor client library needs to be installed on the client machine.

3. Network-Protocol driver:
Network-Protocol driver is a pure java driver which uses a middle-tier to converts JDBC calls directly or indirectly into database specific calls. Multiple types of databases can be accessed at the same time. It is a platform independent driver. It is also known as Type 3 or MiddleWare driver.
Advantages:
Platform independent.
Faster from Type1 and Type2 drivers.
It follows a three tier communication approach.
Multiple types of databases can be accessed at the same time.
Disadvantages:
It requires database-specific coding to be done in the middle tier.

4. Thin driver:
Thin driver is a pure java driver which converts JDBC calls directly into the database specific calls. It is a platform independent driver. It is also known as Type 4 or Database-Protocol driver.
Advantages:
Platform independent.
Faster than all other drivers.
Disadvantages:
It is database dependent.
Multiple types of databases can’t be accessed at the same time.

Steps:

1. Load and Register the Driver
2. Establish a connection
3. Create the statement and execute the statement
4. Process the results
5. Close the connection

Connection (using NOTEPAD)

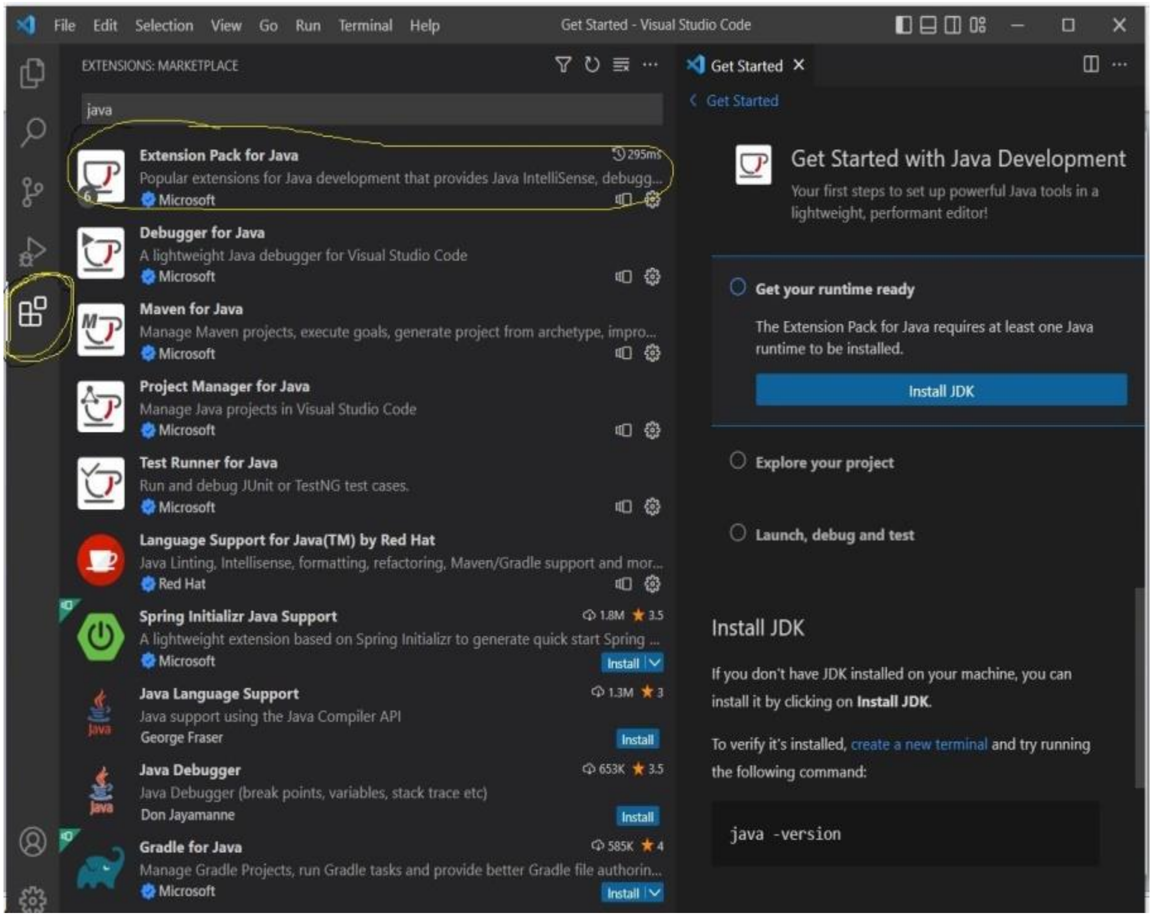
Installation
To connect java application with the mysql (database, mysqlconnector.jar file is required to be loaded.
download the jar file mysql-connector.jar
Two ways to load the jar file:
Paste the mysqlconnector.jar file in jre/lib/ext folder
Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here. (if JRE is not available please in JAVA folder)
Set classpath
There are two ways to set the classpath:
Temporary

```
C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;
```

Permanent
Go to environment variable then click on new tab. In variable name
write classpath and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

ECLIPSE

Store connector.jar file in some location
Click right click on Project folder,
Chose Build Path
Chose
Add External Archives /Configure Build Path
Configure Build Path
In Libraries, click on Classpath, Add External JARs
Visual Studio Code
Download and Install
Add JAVA extension pack
Chose Explorer
Chose Create JAVA Project


File Edit Selection View Go Run Terminal Help
Get Started - Visual Studio Code
EXTENSIONS: MARKETPLACE
java
Get Started

Extension Pack for Java 295ms
Popular extensions for Java development that provides Java IntelliSense, debugg...
Microsoft

Debugger for Java
A lightweight Java debugger for Visual Studio Code
Microsoft

Maven for Java
Manage Maven projects, execute goals, generate project from archetype, impro...
Microsoft

Project Manager for Java
Manage Java projects in Visual Studio Code
Microsoft

Test Runner for Java
Run and debug JUnit or TestNG test cases.
Microsoft

Language Support for Java(TM) by Red Hat
Java Linting, Intellisense, formatting, refactoring, Maven/Gradle support and mor...
Red Hat

Spring Initializr Java Support 1.8M ★ 3.5
A lightweight extension based on Spring Initializr to generate quick start Spring ...
Microsoft Install

Java Language Support 1.3M ★ 3
Java support using the Java Compiler API
George Fraser Install

Java Debugger 653K ★ 3.5
Java Debugger (break points, variables, stack trace etc)
Don Jayamanne Install

Gradle for Java 585K ★ 4
Manage Gradle Projects, run Gradle tasks and provide better Gradle file authorin...
Microsoft Install

Get Started with Java Development
Your first steps to set up powerful Java tools in a lightweight, performant editor!

Get your runtime ready
The Extension Pack for Java requires at least one Java runtime to be installed.
Install JDK

Explore your project

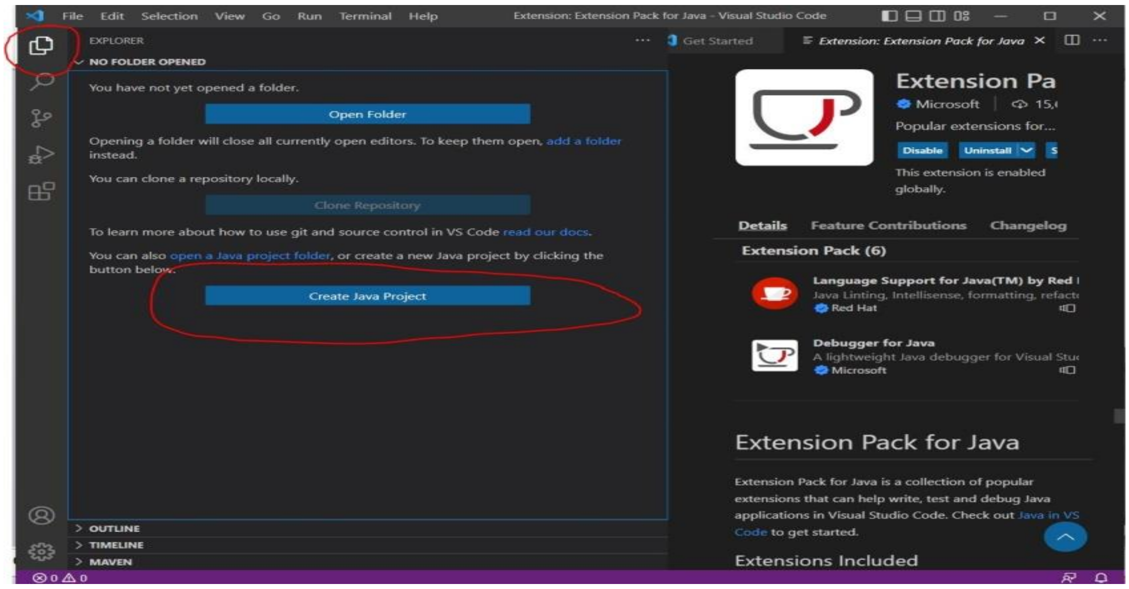
Launch, debug and test

Install JDK

If you don't have JDK installed on your machine, you can install it by clicking on **Install JDK**.

To verify it's installed, create a new terminal and try running the following command:

```
java -version
```



NO FOLDER OPENED

You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

To learn more about how to use git and source control in VS Code read our docs.

You can also open a Java project folder, or create a new Java project by clicking the button below.

Extension Pa
Microsoft 15,
Popular extensions for...
This extension is enabled globally.

Extension Pack (6)

Language Support for Java(TM) by Red
Java Linting, Intellisense, formatting, refact
Red Hat

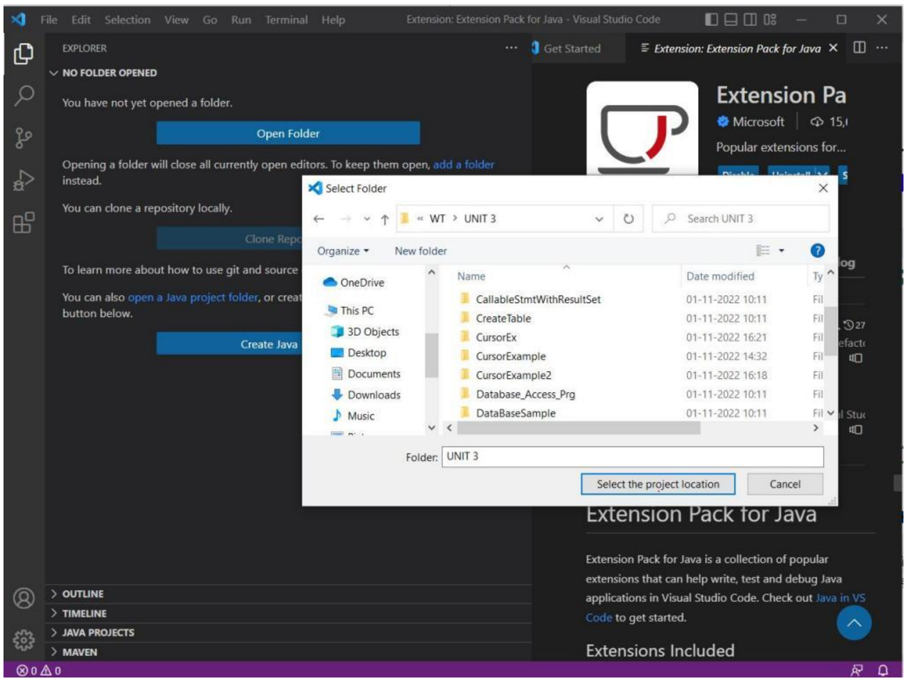
Debugger for Java
A lightweight Java debugger for Visual Stu
Microsoft

Extension Pack for Java

Extension Pack for Java is a collection of popular extensions that can help write, test and debug Java applications in Visual Studio Code. Check out Java in VS Code to get started.

Extensions Included

⊗ 0 ⚠ 0



You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

To learn more about how to use git and source

You can also open a Java project folder, or creat button below.

Name	Date modified
CallableStmtWithResultSet	01-11-2022 10:11
CreateTable	01-11-2022 10:11
CursorEx	01-11-2022 16:21
CursorExample	01-11-2022 14:32
CursorExample2	01-11-2022 16:18
Database_Access_Prg	01-11-2022 10:11
DataBaseSample	01-11-2022 10:11

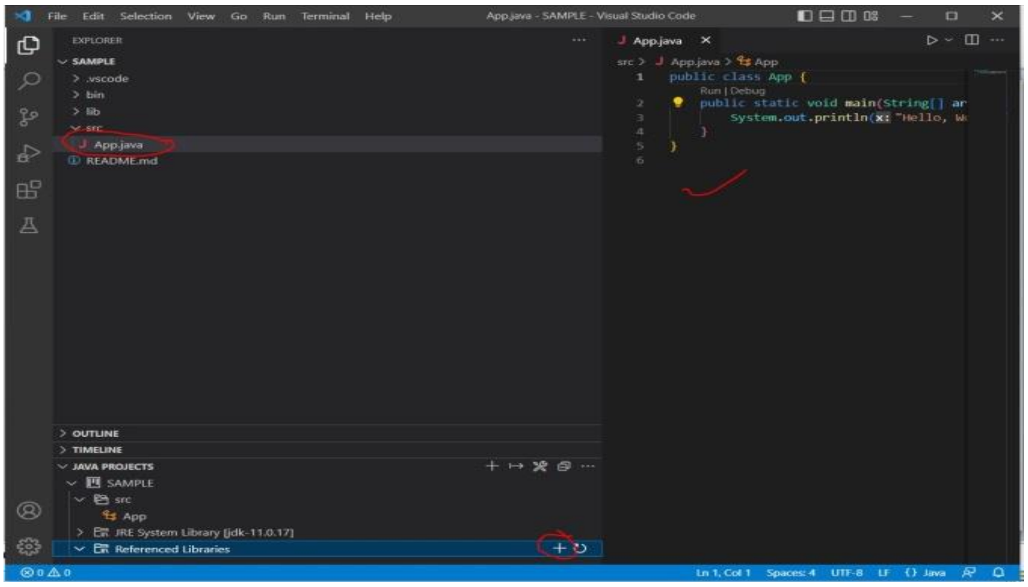
Folder: UNIT 3

Extension Pack for Java

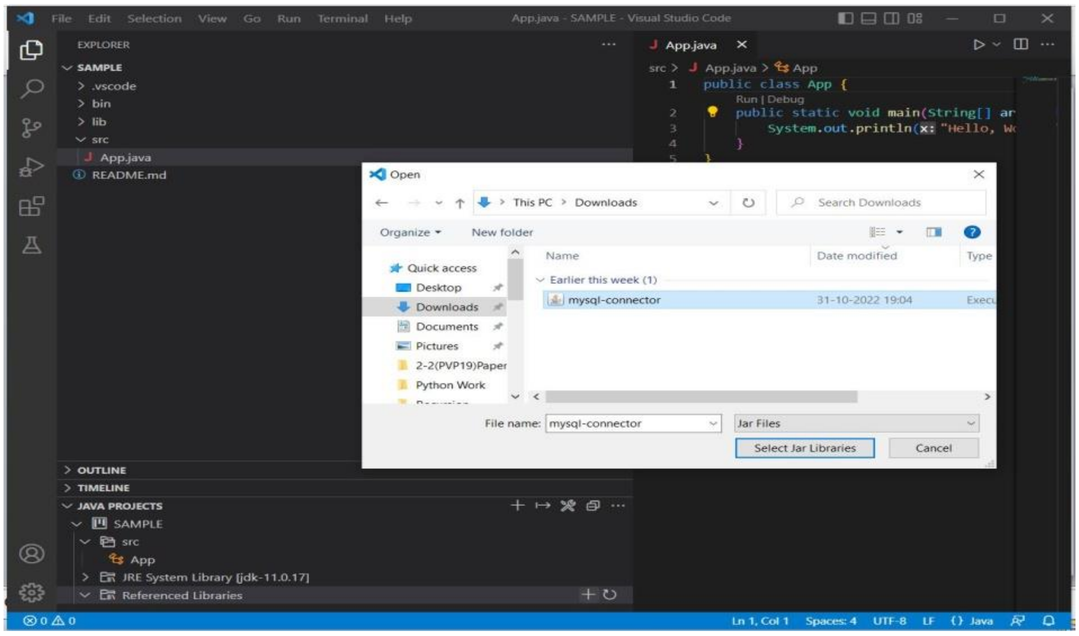
Extension Pack for Java is a collection of popular extensions that can help write, test and debug Java applications in Visual Studio Code. Check out Java in VS Code to get started.

Create JAVA Project :

Then Chose the type of Project and Location
Once the location is selected, Specify the name of the Project, then JAVA project will be created and by default APP.java file will be created, Next Add jar file in referenced libraries.


File Edit Selection View Go Run Terminal Help
App.java - SAMPLE - Visual Studio Code
EXPLORER
SAMPLE
.vscode
bin
lib
src
App.java
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
SAMPLE
src
App
JRE System Library [jdk-11.0.17]
Referenced Libraries
App.java
src > App.java > App
Ln 1, Col 1 Spaces: 4 UTF-8 LF Java

```
public class App {
    Run | Debug
    public static void main(String[] ar
        System.out.println(x: "Hello, Wo
    }
}
```



```
public class App {
    Run | Debug
    public static void main(String[] ar
        System.out.println(x: "Hello, Wo
    }
}
```

JDBC API

Provides classes and interfaces that are used by Java Applications to communicate databases.
The JDBC driver communicates with a database for any requests made by a Java application by using the JDBC API.
The JDBC driver not only process SQL commands, but also sends back the result of processing of these SQL commands.
JDBC follows write once run anywhere behaviour of JAVA.
The JDBC API is part of Java SE and is available to Java Platform EE.

JDBC 4.0 mainly uses two packages:
i)Java.sql   ii) Javax.sql

java.sql package

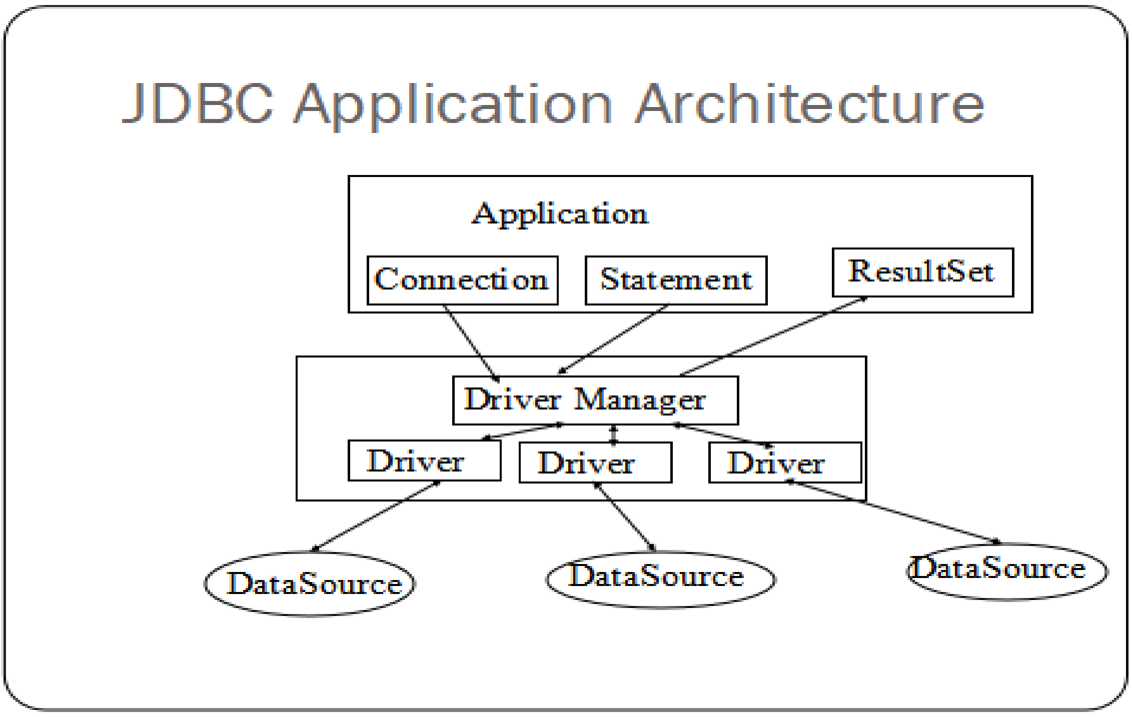
Also called as JDBC Core API.
Package contains classes and interfaces to perform JDBC operations such as creating and executing SQL queries.
These classes and interfaces further classified into:
Connection management – establish a connection with database
Database access – Execution of SQL Queries- after connection is established
Data types- SQL Datatypes (Ex: BLOB,CLOB, UDT....)
Database metadata – is used to retrieve info about Database
Exceptions and warnings – handle unwanted exceptions raised by the application

javax.sql package

Also called as JDBC Extension API (supplement of java.sql package).
Which provides Classes and interfaces to access server-side data sources.
 Classified into
 DataSource
Connection and statement pooling – establish number of connections
Distributed transaction – supports accessing of data from multiple servers
Rowsets – is used to retrieve data from a network (java-bean)

Exploring Major Classes and Interfaces

Major classes and interfaces:
DriverManager Class
Driver Interface
Connection Interface
Statement Interface
ResultSet Interface



DriverManager Class

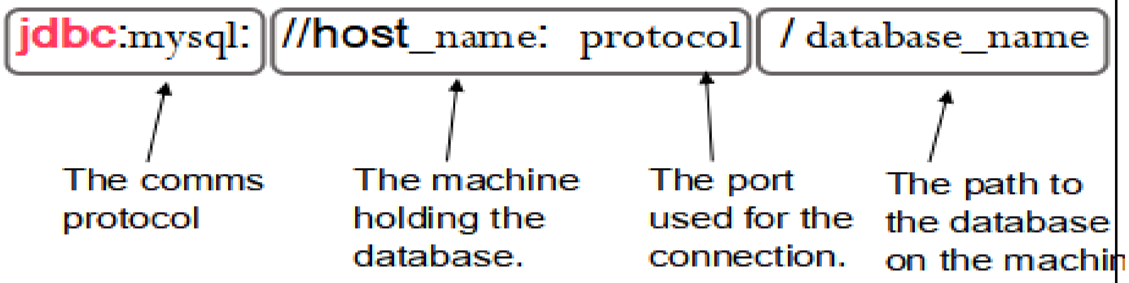
- The task of the DriverManager class is to keep track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- All are static methods.

Method Name	Description
static Connection void getConnection (String url)	It tries to establish the connection to a given database URL
static Connection getConnection (String url, Sting user, String password)	It tries to establish the connection to a given database URL.

static Connection getConnection(String url, Properties info)	It tries to establish the connection to a given database URL. Property is in the form of Object .
	
	 Properties p=new Properties();
	 p.put("user","root");
	 p.put("password","admin");
	 C o n n e c t i o n cn=DriverManager.getConnecton("URL",p);

static Drivers[] getDrivers()	It retrieves the enumeration of the drivers which has been registered with the DriverManager class
static Driver getDriver(String URL)	Given a URL, this method returns a driver that can understand URL
Static void deregisterDriver(Driver)	Unregisters the driver
Sttic void registerDriver(Driver)	Registers the driver

JDBC Database URL


jdbc:mysql:
//host_name: protocol
/ database_name
The comms protocol
The machine holding the database.
The port used for the connection.
The path to the database on the machin

e.g. jdbc:mysql://localhost:3306/suresh

Driver Interface

- Implemented by DriverManager Class

Method	Description
public boolean acceptsURL(String url)	Checks whether the format of the given URL is according to the format or not.
public abstract Connection connect(String url, Properties info)	Try to make a database connection to the given URL.
public abstract int getMajorVersion()	Get the driver's major version number.
public abstract int getMinorVersion()	Get the driver's minor version number.
public abstract boolean jdbcCompliant()	Report whether the Driver is a genuine JDBC COMPLIANT driver.

Connection Interface

- To communicate with a database using JDBC, we must first establish a connection to the database through the appropriate driver.
- This can be done with **java.sql.Connection** interface.
- Within the context of a Connection, SQL statements are executed and results are returned
- The connection object is obtained by the DriverManager.getConnection() method by supplying the Database location and authentication details.

- We can use the Connection object(ref) for the following things:

  1. It creates the Statement, PreparedStatement and CallableStatement objects for executing the SQL statements.

     Statement

     It is used to execute SQL statements

     Prepared Statement

     Used to prepare statements with place holders(?) to set the values at run time

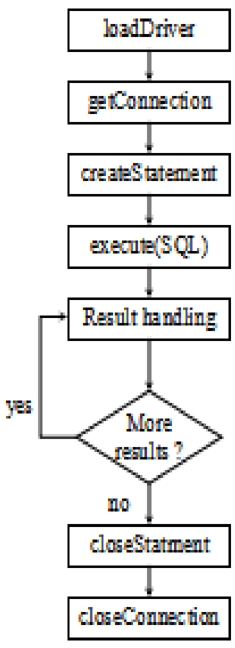
     Callable Statement

     Used to execute functions or procedures available in data base

  2). It helps us to **Commit** or **roll back** a jdbc transaction

Return Type	Method Name	Description
void	close()	Releases a Connection's database and JDBC resources immediately
void	commit()	Makes all changes made since the previous commit or rollback permanent and releases any database locks currently held by the Connection.
Statement	createStatement()	Creates a Statement object for sending SQL statements to the database.
boolean	isClosed()	Tests to see if a Connection is closed.
CallableStatement	prepareCall(String sql)	Creates a CallableStatement object for calling database stored procedures.
PreparedStatement	prepareStatement(String sql)	Creates a PreparedStatement object for sending parameterized SQL statements to the database.
void	rollback().	Drops all changes made since the previous commit or rollback and releases any database locks currently held by this Connection
Void	savePoint()	Sets unamed save point
SavePoint	savePoint(String name)	Sets save point with specified name.

A Simple JDBC application



```
import java.sql.*;
public class jdbctest {
 public static void main(String args[]){
   try{
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection
      ("jdbc:mysql://localhost:3306/suresh?", "user", "passwd");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery
      ("select no,sal from EMP where no < 50");
    while(rs.next())
                        System.out.println(rs.getInt(1) + + rs.getInt(2) );
        stmt.close()
    con.close();
   } catch(Exception e) {
    System.err.println(e);
}}}
```

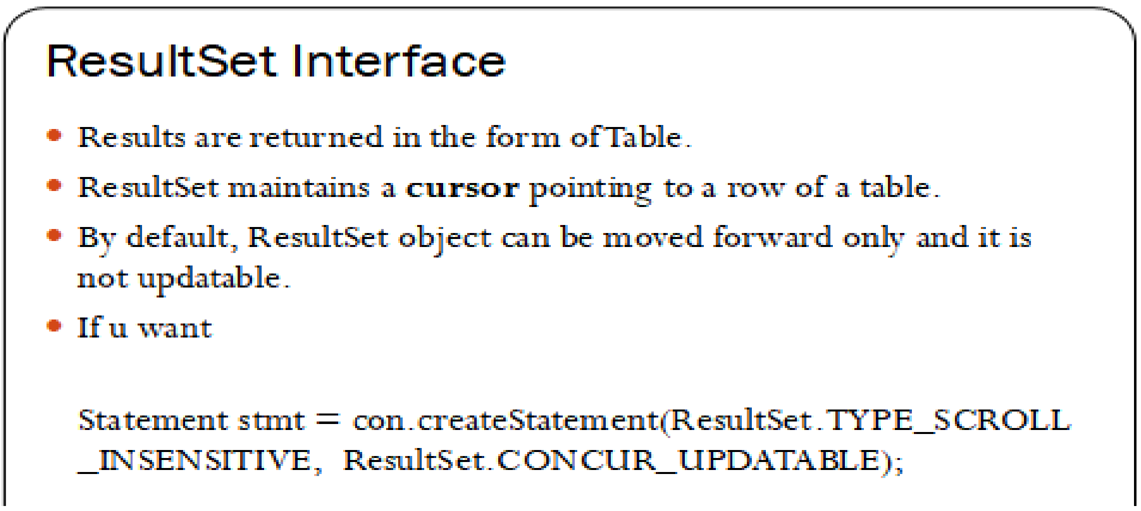
Statement Interface

- The Statement interface creates an object that is used to execute a static SQL statement and obtain the results produced by it.

public interface **Statement**

Return Type	**Method**	**Description**
void	close()	Releases this Statement object's database and JDBC resources immediately
boolean	execute("SQL Query")	Executes an SQL statement that might return multiple results. -DDL
ResultSet	executeQuery("SELECT Queries")	Executes an SQL statement that returns a single ResultSet object.
int	executeUpdate("DML Queries")	Executes an SQL INSERT, UPDATE, or DELETE statement.
ResultSet	getResultSet()	Retrives the result set generated by the execute() method.
void	addBatch("SQL QUERY") executeBatch()	Add the commands to the existing list of commands for the statement obejct

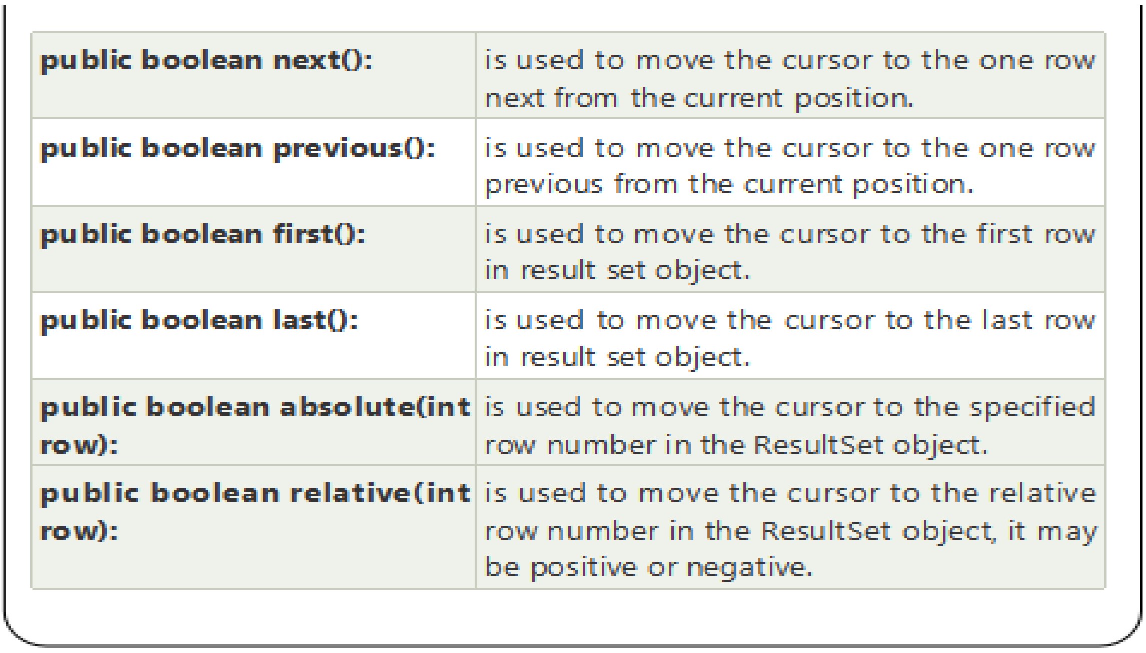
```
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO COFFEES " +
"VALUES('Amaretto', 49, 9.99, 0, 0)");
stmt.addBatch("INSERT INTO COFFEES " +
"VALUES('Hazelnut', 49, 9.99, 0, 0)");
stmt.addBatch("INSERT INTO COFFEES " +
"VALUES('Amaretto_decaf', 49, 10.99, 0, 0)");
stmt.addBatch("INSERT INTO COFFEES " +
"VALUES('Hazelnut_decaf', 49, 10.99, 0, 0)");
int [] updateCounts = stmt.executeBatch();
```



ResultSet Interface

- Results are returned in the form of Table.
- ResultSet maintains a **cursor** pointing to a row of a table.
- By default, ResultSet object can be moved forward only and it is not updatable.
- If u want

  Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);



public boolean next():	is used to move the cursor to the one row next from the current position.
public boolean previous():	is used to move the cursor to the one row previous from the current position.
public boolean first():	is used to move the cursor to the first row in result set object.
public boolean last():	is used to move the cursor to the last row in result set object.
public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.

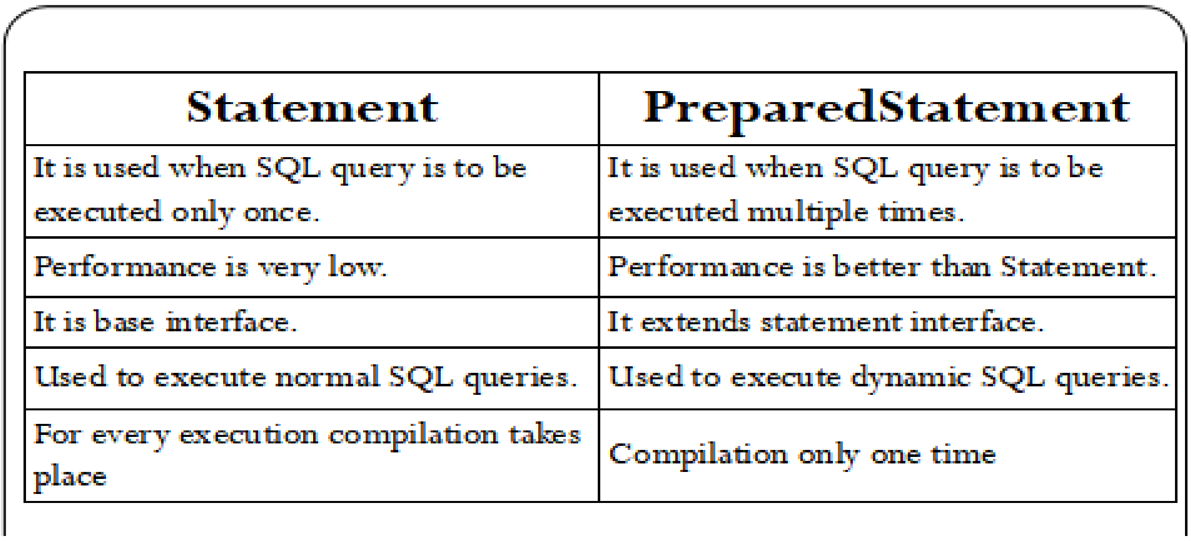
public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

PreparedStatement interface

- The PreparedStatement interface creates an object that represents a precompiled SQL statement.
- A SQL statement is pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.
  - **Note:** The setter methods for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter. For instance, if the IN parameter has SQL type INTEGER, then the method setInt should be used.

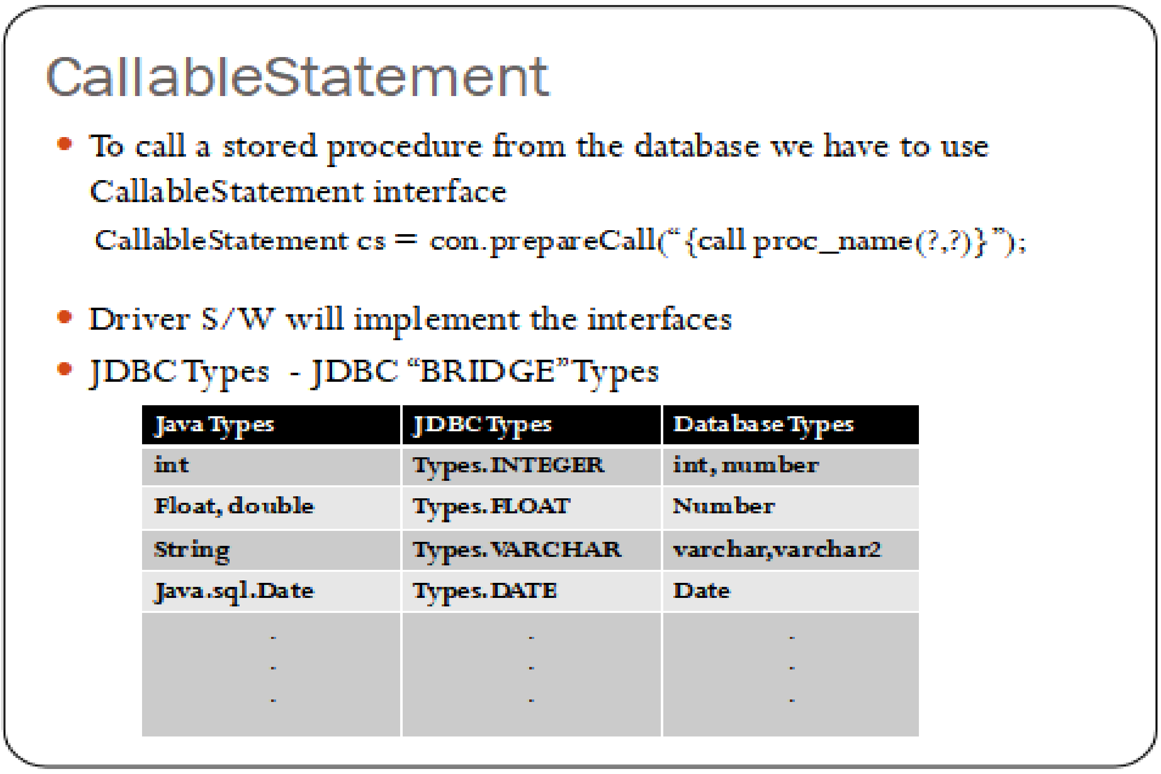
public interface **PreparedStatement** extends Statement

Return Type	Method	Description
boolean	execute()	Executes any kind of SQL statement.
ResultSet	executeQuery()	Executes the SQL query in this PreparedStatement object and returns the result set generated by the query.
int	executeUpdate()	Executes the SQL INSERT, UPDATE or DELETE statement in this PreparedStatement object.
void	setBoolean (int parameterIndex, boolean x)	Sets the designated parameter to a Java boolean value.
void	setDate(int parameterIndex, Date x)	Sets the designated parameter to a java.sql.Date value.
void	setDouble(int parameterIndex, double x)	Sets the designated parameter to a Java double value.
void	setFloat(int parameterIndex, float x)	Sets the designated parameter to a Java float value.
void	setInt (int parameterIndex, int x)	Sets the designated parameter to a Java int value.
void	setLong(int parameterIndex, long x)	Sets the designated parameter to a Java long value.
void	setNull (int parameterIndex, int sqlType)	Sets the designated parameter to SQL NULL.
void	setString (int parameterIndex, String x)	Sets the designated parameter to a Java String value.
void	setTime (int parameterIndex, Time x)	Sets the designated parameter to a java.sql.Time value.



Statement	PreparedStatement
It is used when SQL query is to be executed only once.	It is used when SQL query is to be executed multiple times.
Performance is very low.	Performance is better than Statement.
It is base interface.	It extends statement interface.
Used to execute normal SQL queries.	Used to execute dynamic SQL queries.
For every execution compilation takes place	Compilation only one time

Procedure
Module
IN
OUT
INOUT



CallableStatement

- To call a stored procedure from the database we have to use CallableStatement interface

  CallableStatement cs = con.prepareCall("{call proc_name(?,?)}");

- Driver S/W will implement the interfaces
- JDBC Types - JDBC "BRIDGE" Types

Java Types	JDBC Types	Database Types
int	Types.INTEGER	int, number
Float, double	Types.FLOAT	Number
String	Types.VARCHAR	varchar,varchar2
Java.sql.Date	Types.DATE	Date
. . .	. . .	. . .

Procedure

```
DELIMITER &
CREATE [or REPLACE]
  PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_na
  me datatype [, parameter datatype]) ]
BEGIN
  Declaration_section
  Executable_section
END;
&

CALL procedure_name ( parameter(s)) delimiter
```

PROCEDURE WITH IN AND OUT

```
delimiter //
create procedure emp_sal(IN var1 INT, OUT var2
  INT)
begin
select sal into var2 from EMP1 where eno=var1;
end;
//
```

Procedures with OUT Parameter

```
DELIMITER //
CREATE PROCEDURE data15 (OUT var1 INT)
BEGIN
  SELECT max(sal) INTO var1 FROM EMP1;

END;
//

mysql> CALL data15(@M);
mysql> SELECT @M;
```

Procedures with INOUT Parameter

```
DELIMITER //
CREATE PROCEDURE display_sal (INOUT var1 INT)
BEGIN
   SELECT sal INTO var1 FROM EMP1 WHERE eno=var1;
END;
 //

SET @M=2000
Call display_sal(@M)
SELECT @M
```

Steps to call stores Procedure

- Make sure stored procedure is in database.
- Create a callable statement interface object
- Provide values for every IN parameter by using corresponding setter methods.
- For every OUT parameters we have to register with JDBC types.
- Execute Procedure call.
- Get results by using getter methods from OUT parameter.

- Create a callable statement interface object

  CallableStatement cs = con.prepareCall("{call emp_sal(?,?)}");

- Provide values for every IN parameter by using corresponding setter methods.

  cs.setXXX(1, value);
- For every OUT parameters we have to register with JDBC types.

  cs.registerOutParameter(2,Types.XXX);
- Execute Procedure call.

  cs.execute()
- Get results by using getter methods from OUT parameter.

  cs.getXXX(2)

Function Syntax

```
CREATE FUNCTION Function_Name(input_arguments)
  RETURNS data_type
[DETERMINISTIC |READS SQL DATA | NO SQL CONTAINS SQL]
BEGIN
 declare variables;
 statements . . . . . . . . . .
 return variable;
END
```