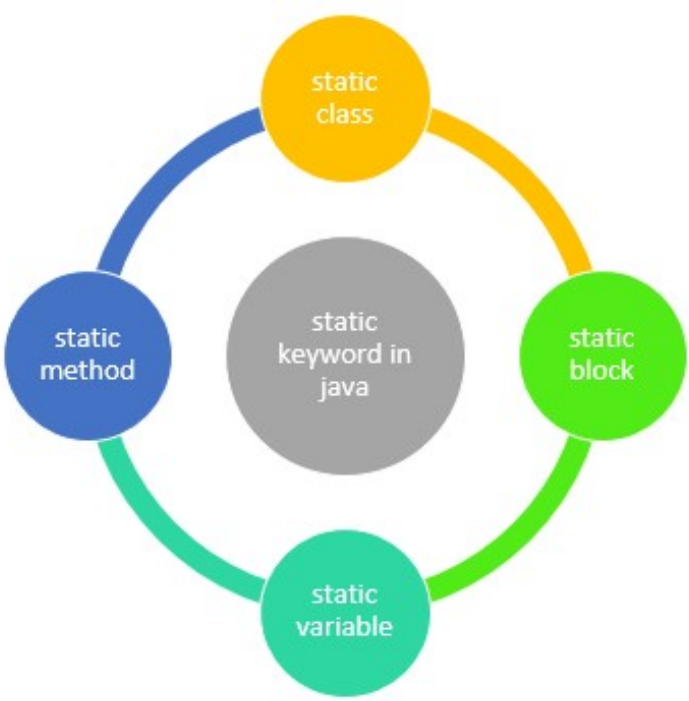
JDBC Day 2: Static, Non-Static Block, Statement, and PreparedStatement.



Static and Non-Static Block

The static and non-static blocks are used as a member, in a class, and both of these have their specific use in Java language.

```
package com.masai;

public class A {

  {
    System.out.println("inside non-static block of A class");
  }

  static{
    System.out.println("inside static block of A class");
  }

  public A() {
    System.out.println("inside constructor of class A");
  }

  public void funA() {
```

```
    System.out.println("inside funA of A");
  }

  public static void main(String[] args) {
    System.out.println("inside main of class A");

    A a1= new A();

    a1.funA();



  }

}
```

Output:

```
inside static block of A class
inside main of class A
inside non-static block of A class
inside constructor of class A
inside funA of A
```

Why the output is coming in this way. Let's understand first difference between Static and Non-Static Block:

Static Block:

- The static block will be executed before the main method of our application.
- As soon as our application is loaded into the memory, static blocks will be executed.
- Multiple static blocks would execute in the order they are defined in the class.
- A typical static block looks like

```
static{
// code for static block
}
```

Non-Static Block:

- A non-static block executes when the object is created, before the constructor
- **Non-static** blocks will never execute if we don't create an object of a class.
- In the case of multiple non-static blocks, the block executes the order in which it is defined in the class.
- All non-static block executes every time an object of the containing class is created.
- A typical non-static block looks like below

```
{
// non static block
```

```
}
```

Example:

```
package com.masai;

public class A {

static{
  System.out.println("inside static block of A class");

  //we can write any type of code inside the static block

  A a1= new A();
  a1.funA();

}



public void funA() {

  System.out.println("inside funA of A");
  }
}
```

forName():

```
package com.masai;

public class Demo{

public static void main(String[] args) {

  try {
    Class.forName("com.masai.A");
  } catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  }
}
```

When the **forName()** method of our Jdbc application executes, 3 things happen in the background.

1. The driver-related main class will be loaded into the primary memory.
2. Driver-related main class objects will be created.
3. That driver object will be registered with the DriverManager class.

Example:

```
    try {
      Driver d = new Driver();
      DriverManager.registerDriver(d);
```

```
  } catch (SQLException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
  }
```

The above line of code is written inside the static block of every Driver class.

Note: from Java 1.6 onwards, no need to load the driver class into the memory also, it is optional (**but recommended to have**).

Here at runtime, our JVM will load the appropriate Driver class into the memory by taking it from the classpath

Connection

```
public interface Connection
extendsWrapper,AutoCloseable
```

A Connection is a session between a Java application and a database. It helps to establish a connection with the database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData, i.e., an object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback(), setAutoCommit(), setTransactionIsolation(), etc.

Commonly used methods of Connection interface:

1) public Statement createStatement():
creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType,int resultSetConcurrency):
Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) public void setAutoCommit(boolean status): is used to set the commit status. By default, it is true.

4) public void commit(): saves the changes made since the previous commit/rollback is permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

Inserting record into the table using Java Application:

1: Create a table inside the database:

```
create table student
(
roll int primary key,
name varchar(12),
address varchar(12),
```

```
marks int
);
```

2: Get Connection Object

```
import java.sql.Connection;

import java.sql.DriverManager;
import java.sql.SQLException;

public class Demo {

public static void main(String[] args) throws ClassNotFoundException {

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

  String url = "jdbc:mysql://localhost:3306/b21sb101";

  try {
    Connection con = DriverManager.getConnection(url,"root","root");


  } catch (SQLException e) {
    e.printStackTrace();
  }

   }
}
```

Once we get the Connection object then we can execute any SQL statement from our Java application.

For that, we require a "java.sql.Statement(I)" object.

Statement

The *Statement* interface accepts strings as SQL queries. Thus, **the code becomes less readable** when we concatenate SQL strings:

The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.

Fourthly, **the *Statement* interface is suitable for DDL queries like CREATE, ALTER, and DROP**:

There are different types of statements that are used in JDBC as follows:

- • Create Statement
- • Prepared Statement
- • Callable Statement

Create Statement:

From the connection interface, you can create the object for this interface. It is generally used for general–purpose access to databases and is useful while using static SQL statements at runtime.

Syntax:

```
Statement statement = connection.createStatement();
```

```
try {
    Connection connection = DriverManager.getConnection(url,"root","root");
    Statement statement = connection.createStatement();
    statement.executeUpdate(query);

    } catch (SQLException e) {
       e.printStackTrace();
  }
```

Once we get the Statement object, in order to execute any SQL statement from our Java application we need to use one of the following methods of Statement object.

1. public int executeUpdate(String dml)throws SQLException;
2. public ResultSet executeQuery(String drl)throws SQLException;
3. public boolean execute(String any sql)throws SQLException

Now Let’s insert one record into the student table:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.cj.jdbc.Driver;

public class Demo {

  public static void main(String[] args) {

    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String url="jdbc:mysql://localhost:3306/sb101db";


    try {
      Connection conn= DriverManager.getConnection(url,"root","root");

      Statement st= conn.createStatement();

      int x= st.executeUpdate("insert into student values(10,'Ram','delhi',850)");

      if(x > 0)
        System.out.println("record inserted sucessfully..");
      else
        System.out.println("not inserted...");

    } catch (SQLException e) {
```

```
      // TODO Auto-generated catch block
      e.printStackTrace();
    }


  }
}
```

Closing the connection solution 1: closing the connection inside finally block

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.cj.jdbc.Driver;

public class Demo {

  public static void main(String[] args) {


    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }


    String url="jdbc:mysql://localhost:3306/sb101db";


    Connection conn=null;

    try {
      conn = DriverManager.getConnection(url,"root","root");

      Statement st= conn.createStatement();

      int x= st.executeUpdate("insert into student values(10,'Ram','delhi',850)");

      if(x > 0)
        System.out.println("record inserted sucessfully..");
      else
        System.out.println("not inserted...");



    } catch (SQLException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();

    }
    finally {

      try {
        conn.close();
      } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }

    }
```

```
  }


}
```

Closing the connection solution 2:

Note: from Java 1.7 onwards we can use try with resource to close the connection.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.cj.jdbc.Driver;

public class Demo {

public static void main(String[] args) {


  try {
    Class.forName("com.mysql.cj.jdbc.Driver");
  } catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }


  String url="jdbc:mysql://localhost:3306/sb101db";

  try(Connection conn= DriverManager.getConnection(url, "root", "root")){


    Statement st= conn.createStatement();

    int x= st.executeUpdate("insert into student values(10,'Ram','delhi',850)");

    if(x > 0)
      System.out.println("record inserted sucessfully..");
    else
      System.out.println("not inserted...");



  } catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();

  }

  }
}
```

Using Statement object inserting dynamic details inside the student table:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
```

```
import com.mysql.cj.jdbc.Driver;

public class Demo {

  public static void main(String[] args) {


    Scanner sc = new Scanner(System.in);

    System.out.println("Enter Roll: ");
    int roll= sc.nextInt();

    System.out.println("Enter Name: ");
    String name= sc.next();

    System.out.println("Enter Address: ");
    String address= sc.next();

    System.out.println("Enter Marks: ");
    int marks= sc.nextInt();

    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }


    String url="jdbc:mysql://localhost:3306/sb101db";

    try(Connection conn= DriverManager.getConnection(url, "root", "root")){


      Statement st= conn.createStatement();

      int x= st.executeUpdate("insert into student values("+roll+",'"+name+"','"+address+"',"+marks+")");

      if(x > 0)
        System.out.println("record inserted sucessfully..");
      else
        System.out.println("not inserted...");



    } catch (SQLException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();

    }

  }


}
```

Finally, **the *Statement* interface can't be used for storing and retrieving files and arrays**.

Note: If we use the Statement(I) object to perform any DB operation then the
The statement object has the following limitations:

1. Complexities to concatenate the dynamic variables.

2. Whenever we pass any query to the DB using the Statement object, the DB engine will perform the following tasks every time to execute that query:

a. query compilation
b. query plan generation
c. query optimization

Example: **select * from student where roll =?**

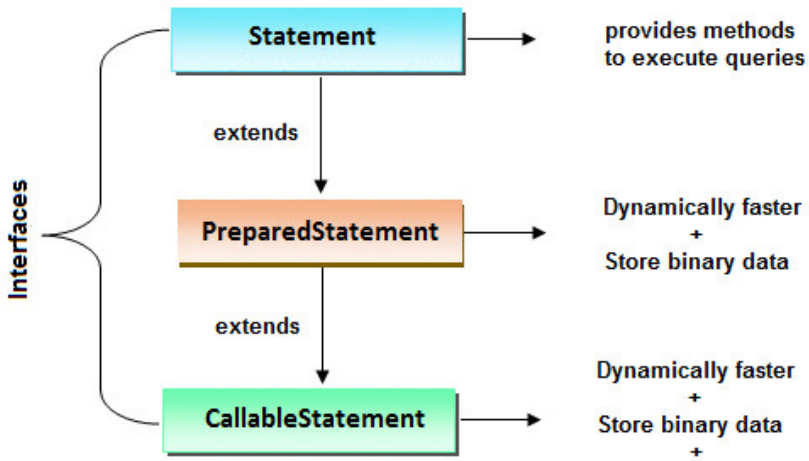
For the same query doing these operations every time, it will degrade the performance.

Tell the DB engine, to perform the above 3 tasks only the first time with the value or without the value (In case of without the value, make use of placeholders ?) and put that query inside the cache, next time onwards just add the dynamic values and execute that query.

Note: For that, we need to use **"java.sql.PreparedStatement(I)"** object instead of using **"java.sql.Statement(I)"** object.

PreparedStatement

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized queries.



Note: To get the PreparedStatement object we need to call "prepareStatement(String sql)" method on the Connection object, with the value or without the value.

In the case without the value, we need to use placeholders?

```
PreparedStatement ps = conn.prepareStatement("insert into student values(30,'arun','pune',600)")
```

Let's see the example of a parameterized query:

```
PreparedStatement ps = conn.prepareStatement("insert into student values(?,?,?,?)");
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

In the case of the parameter (?), before executing the query we need to bind the appropriate values to the corresponding placeholders by calling various types of setXXX(--) on the PreparedStatement object.

```
   String query = "INSERT INTO persons(id, name) VALUES( ?, ?)";

    PreparedStatement preparedStatement = connection.prepareStatement(query);
    preparedStatement.setInt(1,45);
    preparedStatement.setString(2,"Rakesh Kumar");
}
```

After binding the appropriate placeholders we need to execute the query using one of the following methods of the PreparedStatement object.

```
public int executeUpdate()throws SQLException;

public ResultSet executeQuery()throws SQLException;

public boolean execute()throws SQLException;
```

Example: Inserting the record into the Student table using PreparedStatement

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

import com.mysql.cj.jdbc.Driver;

public class Demo {

  public static void main(String[] args) {


    Scanner sc = new Scanner(System.in);

    System.out.println("Enter Roll: ");
    int roll= sc.nextInt();

    System.out.println("Enter Name: ");
    String name= sc.next();

    System.out.println("Enter Address: ");
    String address= sc.next();

    System.out.println("Enter Marks: ");
    int marks= sc.nextInt();

    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    String url="jdbc:mysql://localhost:3306/sb101db";

    try(Connection conn= DriverManager.getConnection(url, "root", "root")){
```

```
    PreparedStatement ps = conn.prepareStatement("insert into student values(?,?,?,?)");

      ps.setInt(1, roll);
      ps.setString(2, name);
      ps.setString(3, address);
      ps.setInt(4, marks);

      int x= ps.executeUpdate();

      if(x >0)
        System.out.println("record inserted successfully..");
      else
        System.out.println("Not inserted...");

    } catch (SQLException e) {
      // TODO Auto-generated catch block
      //e.printStackTrace();

      System.out.println(e.getMessage());

    }
  }

}
```

WE Probelm

Updating **the record into the Student table using PreparedStatement: Give 50 grace marks to all the students whose marks are less than 600**

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

import com.mysql.cj.jdbc.Driver;

public class Demo {
public static void main(String[] args) {


  Scanner sc = new Scanner(System.in);

  System.out.println("Enter the Grace marks : ");
  int gMarks= sc.nextInt();


  try {
    Class.forName("com.mysql.cj.jdbc.Driver");
  } catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }


  String url="jdbc:mysql://localhost:3306/sb101db";
```

```
  try(Connection conn= DriverManager.getConnection(url, "root", "root")){


    PreparedStatement ps = conn.prepareStatement("update student set marks = marks + ? where marks < 60
0");

    ps.setInt(1, gMarks);

    int x= ps.executeUpdate();

    if(x > 0)
      System.out.println(x +" records are updated sucessfully");
    else
      System.out.println("No record updated...");







  } catch (SQLException e) {
    // TODO Auto-generated catch block
    //e.printStackTrace();

    System.out.println(e.getMessage());

  }

   }
}
```

WE Problem:

***Deleting* the record from the Student table using PreparedStatement: Delete those on the basis of roll no.**

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

import com.mysql.cj.jdbc.Driver;

public class Demo {

public static void main(String[] args) {


  Scanner sc = new Scanner(System.in);

  System.out.println("Enter the roll to delete the student");
  int roll= sc.nextInt();


  try {
    Class.forName("com.mysql.cj.jdbc.Driver");
  } catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
```

```
  String url="jdbc:mysql://localhost:3306/sb101db";

  try(Connection conn= DriverManager.getConnection(url,"root","root"))  {

    PreparedStatement ps= conn.prepareStatement("delete from student where roll = ?");

    ps.setInt(1, roll);


    int x= ps.executeUpdate();

    if(x > 0)
      System.out.println(x+" record deleted sucessfully..");
    else
      System.out.println("Student does not exist with the Roll "+roll);

    } catch (SQLException e) {
      e.printStackTrace();

     System.out.println(e.getMessage());
  }

   }
}
```

You Problem:

1. Create the employee table and grace their salary by 5000 using the Prepared Statement
2. Delete those employees from the employee table whose salary is less than 20000.

References:

https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html
https://www.javatpoint.com/java-jdbc

https://www.baeldung.com/java-statement-preparedstatement

https://www.roseindia.net/tutorial/java/jdbc/dataaccessobjectdesignpattern.html

https://www.oracle.com/java/technologies/data-access-object.html