JDBC API

* JDBC API
- JDBC stands for java database connectivity.
- JDBC API is a set of classes & interfaces which can be used to work with database
- By using JDBC API we can perform crud operation easily

* Step to connect with database

Step: ① load a driver
- to load a driver, we can use forName() static method of class class.
- forName() method throws with classNotFoundExcep^n

e.g.

```
try {
    class.forName("com.mysql.cj.jdbc.Driver");
    System.out.println("driver loaded successfully");
} catch (classNotFoundException e)
    {
        e.printStackTrace();
    }
```

-NOTE: as per database driver can be varies

Step: ② create connection
- to create connection, we can use getConnection() static method of DriverManager class
- getConnection() method requires 3 parameter {url, username, password}
- ~~o~~ method returns instance of connect^ interface
- ~~u~~ throws with SQL exception

NOTE: as per database, url also can be varies

e.g.

```
connection con = null;
```

JDBC API

PAGE NO
DATE

★ JDBC API

- JDBC stands for java database connectivity.
- JDBC API is a set of classes & interfaces which can be used to work with database
- By using JDBC API we can perform crud operation easily

★ Step to connect with database

Step: ① load a driver

- to load a driver, we can use forName() static method of class class.
- forName() method throws with classNotFoundExcep^n

e.g.

```
try {
    class.forName("com.mysql.cj.jdbc.Driver");
    System.out.println("driver loaded successfully");
} catch (classNotFoundException e)
    {
        e.printStackTrace();
    }
```

-NOTE: as per database driver can be varies

Step: ② create connection

- to create connection, we can use getconnection() static method of DriverManager class
- getconnection() method requires 3 parameter {url, username, password}
- — method returns instance of connect^n interface.
- — throws with SQL exception

NOTE: as per database, url also can be varies

e.g.

```
connection con = null;
```

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql
//localhost:3306/jap77", "root", "root");
    S.O.p("database connected");
} catch (ClassNotFoundExcept^n | SQLException e){
        e.printStackTrace();
    }
```

Step: ③ create query
- to create a query we can use string object

eg.
```
String sql = "insert into t1(name, city) values(?,?)";
```

Step: ④ create statement
- to create a statement we can use createStatement() or use preparedStatement() method of connect^n interface
- createStatement() method returns instance of ~~preparedStatement~~ interface
- preparedStatement() method returns instance of preparedStatemat interface
- preparedStatement is a child interface of statement interface
- by using preparedstatement interface, we can execute complex query
- PreparedStatement doesn't create an object for redundance query
- createstatement() & preparedstatement() method throws with SQLException

eg.
```
Connection con = null;
preparedStatement pst = null;
try {
    class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://
localhost:3306/jap77", "root", "root");
String sql = "insert into t1(name, city) values(?,?)";
pst = con.preparedstatement(sql);
pst.setString(1,"manoj patil");
pst.setString(2,"nagpur");
} catch (classNotFoundException | SqlException e)
    {
        e.printStackTrace();
}
```

Step: ⑤ Execute Statement
- to execute statement we can use execute update or execute query
- execute update can be use if there is DML query
- executeUpdate() returns int value
- executeQuery() method can be used if there is DQL query
- executeUpdate() method returns instance of resultset interface
- executeUpdate() & executeQuery() method throws with SQLException

e.g.
```
int check = 0;
try {
    check = pst.executeUpdate();
} catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        System.out.println(check);
    }
```

Step; ⑥ close connection

- to close a connection close() method can be used
- to close a connection, finally block can be used
- connection should be closed in reverse order
- close() method throws with SQLException

e.g.

```
finally {
        try {
            pst.close();
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

crud appⁿ

```
package com.model;
public class Student {
    private int id;
    private String name;
    private String city;
    private double percentage;
    // setter & getter
    // toString
```

```
package com.dao;

public class StudentDao {
    public int insertStudent(Student s)
    {
        int check = 0;
        Connection con = null;
        PreparedStatement pst = null;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jsp77", "root", "root");
            String sql = "insert into student (name, city, percentage) values (?, ?, ?)";
            pst = con.prepareStatement(sql);
            pst.setString(1, s.getName());
            pst.setString(2, s.getCity());
            pst.setDouble(3, s.getPercentage());
            check = pst.executeUpdate();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                pst.close();
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return check;
    }
}
```

```
package com.demo;

public class App {
    public static void main (String[] args) {
        StudentDao sd = new StudentDao();
        Student s1 = new student();
        s1.setName ("nisha patil");
        s1.setcity ("udaipur");
        s1.setPercentage (79.10);
        System.out.println (sd.insertStudent(s1));
    }
}
```

student dao

```
public int deleteStudentById (int id)
{
    int check = 0;
    connection con = null;
    preparedStatement pst = null;
    try {
        Class.forName ("com.mysql.cj.jdbc.Driver");
        con = DriverManager.getconnection ("jdbc:mysql://localhost:3306/jap77", "root", "root");
        String sql = "delete from student where id=?";
        pst = con.preparedStatement (sql);
        pst.setInt (1, id);
        check = pst.executeUpdate();
    } catch (classNotFoundException | SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            pst.close();
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return check;
}
```

app.java

```
System.out.println (sd.deleteStudentById(35));
```

student dao

```
public student findStudentById (int id)
{
    student s = null;
    connection con = null;
    preparedStatement pst = null;
    ResultSet rs = null;
    try {
        Class.forName ("com.mysql.cj.jdbc.Driver");
        con = DriverManager.getconnection ("jdbc:mysql://localhost:3306/jap77", "root", "root");
        String sql = "select id, name, city, percentage from student where id = ?";
        pst = con.preparedStatement (sql);
        pst.setInt (1, id);
        rs = pst.executequery();
        while (rs.next())
        {
```

```
        S = new Student();
        S.setId(rs.getInt("id"));
        S.setName(rs.getString("name"));
        S.setCity(rs.getString("city"));
        S.setPercentage(rs.getDouble("percentage"));
    }
} catch (ClassNotFoundException | SQLException e) {
    e.printStackTrace();
} Finally
  {
    try {
        rs.close();
        pst.close();
        con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
return S;
}
```

App.java

```
Student S1 = sd.findStudentById(30);
System.out.println(S1);
```

StudentDao

```
public List<Student> findAllStudent()
  {
    List<Student> list = new ArrayList();
    Connection con = null;
    PreparedStatement pst = null;
    ResultSet rs = null;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
con = DriverManager.getConnection("jdbc:mysql://localhost
:3306/jsp 9t", "root", "root");
        String sql = "select id, name, city, percentage from
                    student";
        pst = con.preparedStatement(sql);
        rs = pst.executeQuery();
        while (rs.next())
        {
        Student S1 = new Student();
        S1.setId(rs.getInt("id"));
        S1.setName(rs.getString("name"));
        S1.setCity(rs.getString("city"));
        S1.setPercentage(rs.getDouble("percentage"));
        list.add(S1);
    }
} catch (ClassNotFoundException | SQLException e)
  {
    e.printStackTrace();
} finally {
    try {
        rs.close();
        pst.close();
        con.close();
} catch (SQLException e) {
    e.printStackTrace();
  }
}
return list;
}
```

App.java

```
List<Student> list = sd.findAllStudent();
    for(Student s: list)
      System.out.println(s);
    }
}
```

Updat studentdao

```
public in updateStudent(Student s){
    int check =0;
    Connection con = null;
    PreparedStatement pst = null;
    try{
  class.forname("com.mysql.cj.jdbc.Driver");
  con = DriverManager.getConnection("jdbc:mysql://localhost
:3306/jap77", "root", "root");
  String sql = "update student set name=?, city=?,
Percentage=? where id=?";
    pst = con.preparedStatement(sql);
    pst.setString(1, s.getName());
    pst.setString(2, s.getCity());
    pst.setDouble(3, s.getPercentage());
    pst.setInt(4, s.getId());
    check = pst.executeUpdate();
} catch(classNotFoundException | SQLException e){
    e.printStackTrace();
} finally
  {
  try
  {
    pst.close();
    con.close();
} catch(SQLException e)
  {
    e.printStackTrace()
  }
}
  return check;
}
```

App.java

```
Student s1 = sd.findStudentById(2);
s1.setPercentage(32.78);
System.out.println(sd.updateStudent(s1));
```

package com.util

```
public class myDatabase{
    public static connection createConnect"(){
    connection con = null;
    try{
      class.forName("com.mysql.cj.jdbc.Driver");
con = DriverManager.getConnection("jdbc:mysql://localhost:
3306/jap77", "root", "root");
} catch(classNotFoundException | SQLException e){
    e.printStackTrace();
  }
    return con;
}
public static void closeConnection(preparedStatement
pst, connection con){
```

```
try{
    pst.close();
    con.close();
}catch(SQLException e){
    e.printStackTrace();
}
}
public static void closeConnection(ResultSet rs, PreparedStatement pst, Connection con){
    try{
        rs.close()
        closeConnection(pst, con);
    }catch(SQLException e){
        e.printStackTrace();
    }
}
```

StudentDao.java

```
public class StudentDao{
    public int insertStudent(Student s)
    {
        int check=0;
        Connection con=MyDatabase.createConnection();
        String sql="insert into student(name,city,percentage)
        values(?,?,?)";
        PreparedStatement pst=null;
        try{
            pst=con.prepareStatement(sql);
            pst.setString(1,s.getName());
            pst.setString(2,s.getCity());
            pst.setDouble(3,s.getPercentage());
            check=pst.executeUpdate();
        }catch(SQLException e){
            e.printStackTrace();
        }finally
        {
            MyDatabase.closeConnection(pst,con);
        }
        return check;
    }

public int deleteStudentById(int id){
    int check=0;
    Connection con=MyDatabase.createConnection();
    String sql="delete from Student where id=?";
    PreparedStatement pst=null;
    try{
        pst=con.prepareStatement(sql);
        pst.setInt(1,id);
        check=pst.executeUpdate();
    }catch(SQLException e){
        e.printStackTrace();
    }finally{
        MyDatabase.closeConnection(pst,con);
    }
    return check;
}
}
```

StudentDao.java

```
public Student FindStudentById(int id){
connection con = MyDatabase.createConnection();
String sql = "select id, name, city, percentage from student
where id=?";
PreparedStatement pst = null;
List<Student> list = new ArrayList();
ResultSet rs = null;
try{
    pst = con.preparedStatement(sql);
    pst.setInt(1, id);
    rs = pst.executeQuery();
    list = MyDatabase.studentRowMapped(rs);
} catch(SqlException e){
    e.printStackTrace();
} finally{
    MyDatabase.closeConnection(rs, pst, con);
}
return (!list.isEmpty()) ? list.get(0) : null;
}
}

public List<Student> FindAllStudent()
{
    List<Student> list = new ArrayList();
    connection con = MyDatabase.createConnection();
    String sql = "select id, name, city, percentage from student";
    PreparedStatement pst = null;
    ResultSet rs = null;
    try{
        pst = con.preparedStatement(sql);
        rs = pst.executeQuery();
        list.addAll(MyDatabase.studentRowMapped(rs));
    } catch(SqlException e){
        e.printStackTrace();
    } finally{ MyDatabase.closeConnection(rs, pst, con); }
    return list;
}
}

public int updateStudent(Student s)
{
    int check = 0;
    connection con = MyDatabase.createConnection();
    String sql = "update student set name=?, city=?, percentage=?
where id=?";
    PreparedStatement pst = null;
    try{
        pst = con.preparedStatement(sql);
        pst.setString(1, s.getName());
        pst.setString(2, s.getCity());
        pst.setDouble(3, s.getPercentage());
        pst.setInt(4, s.getId());
        check = pst.executeUpdate();
    }catch(SqlException e){
        e.printStackTrace();
    }finally
    {
        MyDatabase.closeConnection(pst, con);
    }
    return check;
}
```

using Resource Block

```
public int insertStudent(Student s)
{
    int check = 0;
    String sql = "insert into student(name, city, percentage) values
    (?,?,?)";
    try (Connection con = MyDatabase.createConnection();
     PreparedStatement pst = con.prepareStatement(sql);){
    pst.setString(1, s.getName());
    pst.setString(2, s.getCity());
    pst.setDouble(3, s.getPercentage());
    check = pst.executeUpdate();
    } catch (SQLException e)
    {
     e.printStackTrace();
    }
    return check;
}

public int deleteStudentById(int id)
{
    int check = 0;
    String sql = "delete from student where id = ?";
    try (Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);)
    {
     pst.setInt(1, id);
     check = pst.executeUpdate();
    } catch (SQLException e) {
     e.printStackTrace();
    }
    return check;
}
```

```
public Student findStudentById(int id)
{
    String sql = "select id, name, city, percentage from student
    where id = ?";
    List<Student> list = new ArrayList();
    try (Connection con = MyDatabase.createConnection();
     PreparedStatement pst = con.prepareStatement(sql);) {
     pst.setInt(1, id);
     ResultSet rs = pst.executeQuery();
     list.addAll(MyDatabase.StudentRowMapper(rs));
    } catch (SQLException e)
    {
     e.printStackTrace();}
     return (!list.isEmpty()) ? list.get(0) : null;
}

public List<Student> findAllStudent()
{
    List<Student> list = new ArrayList();
    String sql = "select id, name, city, percentage from student";
    try (Connection con = MyDatabase.createConnection();
     PreparedStatement pst = con.prepareStatement(sql);
     ResultSet rs = pst.executeQuery();){
     list.addAll(MyDatabase.createRowMapper(rs));
    } catch (SQLException e)
    {
     e.printStackTrace();
    }
    return list;
}
```

using Resource Block

```
public int insertStudent(Student s)
{
    int check = 0;
    String sql = "insert into student(name, city, percentage) values
    (?,?,?)";
    try(Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);){
    pst.setString(1, s.getName());
    pst.setString(2, s.getCity());
    pst.setDouble(3, s.getPercentage());
    check = pst.executeUpdate();
    }catch(SQLException e)
    {
    e.printStackTrace();
    }
    return check;
}

public int deleteStudentById(int id)
{
    int check = 0;
    String sql = "delete from student where id=?";
    try(Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);)
    {
    pst.setInt(1, id);
    check = pst.executeUpdate();
    }catch(SQLException e){
    e.printStackTrace();
    }
    return check;
}

public Student findStudentById(int id)
{
    String sql = "select id, name, city, percentage from student
    where id=?";
    List<Student> list = new ArrayList();
    try(Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);){
    pst.setInt(1, id);
    ResultSet rs = pst.executeQuery();
    list.addAll(MyDatabase.studentRowMapper(rs));
    }catch(SQLException e)
    {
    e.printStackTrace();}
    return (!list.isEmpty())? list.get(0): null;
}

public List<Student> findAllStudent()
{
    List<Student> list = new ArrayList();
    String sql = "select id, name, city, percentage from student";
    try(Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);
    ResultSet rs = pst.executeQuery();){
    list.addAll(MyDatabase.studentRowMapper(rs));
    }catch(SQLException e)
    {
    e.printStackTrace();
    }
    return list;
}
```

```
public int updateStudent(Student s){
    int check = 0;
    String sql = "update student set name=?, city=?
percentage=?, where id=?";
    try(Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);)
    {
      pst.setString(1, s.getName());
      pst.setString(2, s.getCity());
      pst.setDouble(3, s.getPercentage());
      pst.setInt(4, s.getId());
       check = pst.executeUpdate();
    }catch(SQLException e){
        e.printStackTrace();
    }
    return check;
}
```

App.java

```
StudentDao sd = new StudentDao();
  String[] cities = {

        }

  Random r = new Random();
  for(int i=1; i<=10000; i++)
  {
    Student s = new Student();
    StringBuilder sb = new StringBuilder();
    for(int j=1; j<=15; j++)
```

```
    sb.append((char)(r.nextInt(26)+65));
    s.setName(sb.toString());
    s.setCity(cities[r.nextInt(cities.length)]);
    s.setPercentage(r.nextDouble()*100);
    System.out.println(sd.insertStudent(s));
  }
}
```

Student class.java

```
public List<Student> findStudentByPercentageLessThan
(double percentage)
{
    List<Student> list = new ArrayList();
    String sql = "select id, name, city, percentage, from
student where percentage<?";
    try(Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);){
     pst.setDouble(1, percentage);
     ResultSet rs = pst.executeQuery();
     list.addAll(MyDatabase.studentRowMapper(rs));
    }catch(SQLException e)
     { e.printStackTrace(); }
    return list;
  }
}
```

```
public Student findStudentByNameAndCity(String name, String
                                        city)
{
    String sql = "select id, name, city, percentage from student
        where name=? and city=?";
    List<Student> list = new ArrayList();
```

```
try (Connection con = MyDatabase.createConnection();
PreparedStatement pst = con.prepareStatement(sql);)
{
  pst.setString(1, name);
  pst.setString(2, city);
  ResultSet rs = pst.executeQuery();
  list.addAll(MyDatabase.studentRowMapped(rs));
}
catch(SqlException e)
{ e.printStackTrace(); }
return (!list.isEmpty()) ? list.get(0) : null;
}

public List<Student> findStudentByNameOrCity(String
name, String city)
{
    List<Student> list = new ArrayList();
    String sql = "select id, name, city, percentage from
student where name = ? OR city = ?";
  try (Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql)
  {
    pst.setString(1, name);
    pst.setString(2, city);
    ResultSet rs = pst.executeQuery();
    list.addAll(MyDatabase.studentRowMapper(rs));
  } catch (SqlException e) { e.printStackTrace(); }
    return list;
}
```

```
public List<Student> findStudentByPercentageBetween(double low,
double high)
{
  List<Student> list = new ArrayList();
  String sql = "select id, name, city, percentage from student where
  percentage between ? & ?";
  try (Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql)){
    pst.setDouble(1, low);
    pst.setDouble(2, high);
    ResultSet rs = pst.executeQuery()
    list.addAll(MyDatabase.studentRowMapper(rs));
  } catch(SQLException e) { e.printStackTrace(); }
    return list;
}

public List<Student> findStudentByIdNotBetween(int low, int high)
{
  List<Student> list = new ArrayList();
  String sql = "select id, name, percentage from student where
id not between ? & ?";
  try (Connection con = MyDatabase.createConnection();
    PreparedStatement pst = con.prepareStatement(sql);)
  {
    pst.setInt(1, low);
    pst.setInt(2, high);
    ResultSet rs = pst.executeQuery()
    list.addAll(MyDatabase.studentRowMapper(rs));
  } catch(SQLException e) { e.printStackTrace(); }
    return list;
}
```

```
public List<Student> findStudentByNameContains(String name)
{
  List<Student> list = new ArrayList();
  String sql = "select id, name, percentage, city from student where
      name like ?";
  try (Connection con = MyDatabase.createConnection()) {
    PreparedStatement pst = (con).prepareStatement(sql);){
    pst.setString(1, "%" + name + "%");
    ResultSet rs = pst.executeQuery();
    list.addAll(MyDatabase.StudentRowMapper(rs));
  } catch (SQLException e) { e.printStackTrace(); }
  return list;
}

public List<String> findAllNames()
{
  List<String> list = new ArrayList();
  String sql = "select name from students";
  try (Connection con = MyDatabase.createConnection();
  PreparedStatement pst = con.prepareStatement(sql);
  ResultSet rs = pst.executeQuery();){
  while (rs.next())
    list.add(rs.getString("name"));
  } catch (SQLException e) { e.printStackTrace(); }
  return list;
}
}
```

```
public Map<String, List> findAllNamesAndPercentage()
{
  Map<String, List> m = new HashMap();
  String sql = "select name, percentage from student";
  try (Connection con = MyDatabase.createConnection();
  PreparedStatement pst = con.prepareStatement(sql);
  ResultSet rs = pst.executeQuery();){
  List<String> list1 = new ArrayList();
  List<Double> list2 = new ArrayList();
  while (rs.next())
  {
    list1.add(rs.getString("name"));
    list2.add(rs.getDouble("percentage"));
  }
  m.put("ListOfNames", list1);
  m.put("ListOfPercentage", list2);
} catch (SQLException e) { e.printStackTrace(); }
return m;
}
}
```

App.java

```
Public static void main(String Args[]) {
  StudentDao sd = new StudentDao();
  Map<String, List> m = sd.findAllNamesAndPercentage();
  List list1 = m.get("ListOfNames");
  List list2 = m.get("ListOfPercentage");
  for (int i=0; i<list1.size(); i++)
  System.out.println(list1.get(i) + "\t" + list2.get(i)
  );
}
}
```