Lesson Proper for Week 15

What is Java Database Connectivity (JDBC) , its architecture and functions?

JDBC (Java Database Connectivity) is the Java Application Programming Interface (API) that manages connecting to a database, issuing queries and commands, and handling result sets obtained from the database. Released as part of JDK 1.1 in 1997, JDBC was one of the first components developed for the Java persistence layer.

JDBC was initially conceived as a client-side API, enabling a Java client to interact with a data source. That changed with JDCB 2.0, which included an optional package supporting server-side JDBC connections. Every new JDBC release since then has featured updates to both the client-side package (java.sql) and the server-side package (javax.sql). JDBC 4.3, the most current version as of this writing, was released as part of Java SE 9 in September 2017.

How JDBC works

Developed as an alternative to the C-based ODBC (Open Database Connectivity) API, JDBC offers a programming-level interface that handles the mechanics of Java applications communicating with a database or RDBMS. The JDBC interface consists of two layers:

1. The JDBC API supports communication between the Java application and the JDBC manager.
2. The JDBC driver supports communication between the JDBC manager and the database driver.

JDBC is the common API that your application code interacts with. Beneath that is the JDBC-compliant driver for the database you are using.

What are the JDBC Drivers

1. JDBC Drivers

    1. JDBC-ODBC (Java Database Connectivity – Open Database Connectivity)bridge driver

2. Native-API driver

3. Network Protocol driver

4. Thin 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.

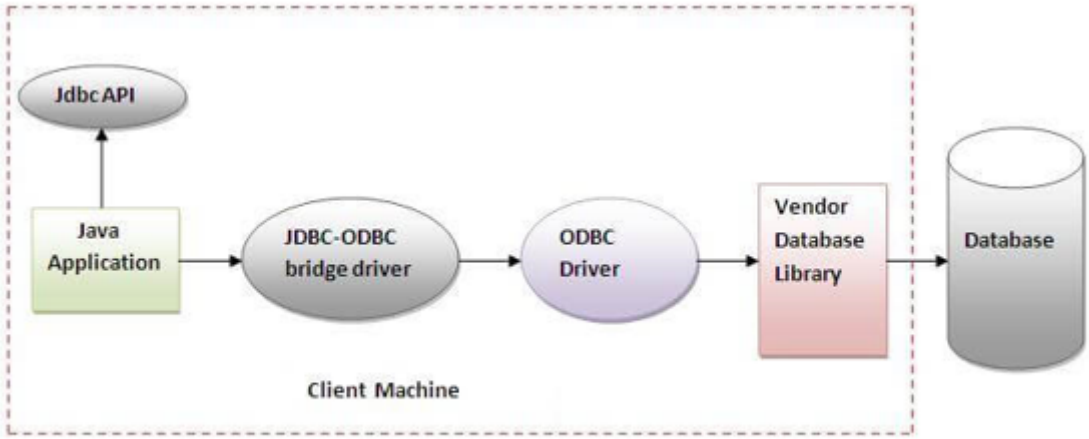


Figure- JDBC-ODBC Bridge Driver

In Java 8, the JDBC-ODBC Bridge has been removed.

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.

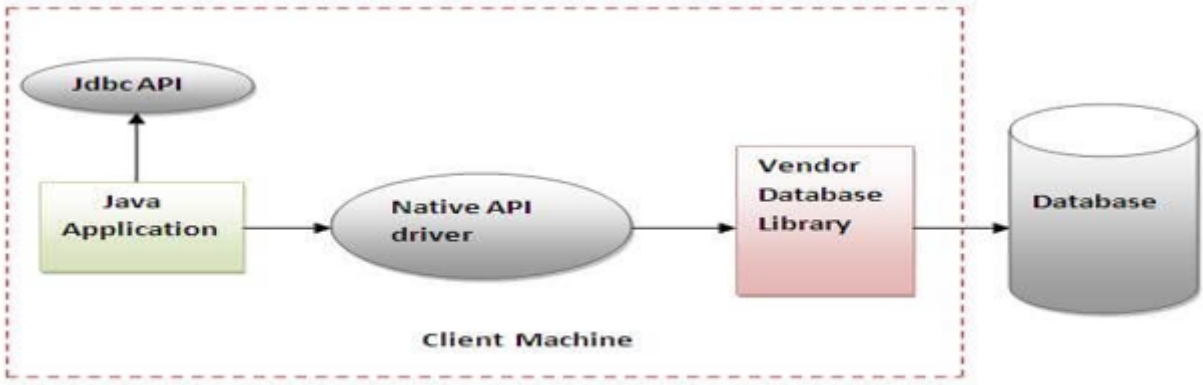


Figure- Native API Driver

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.

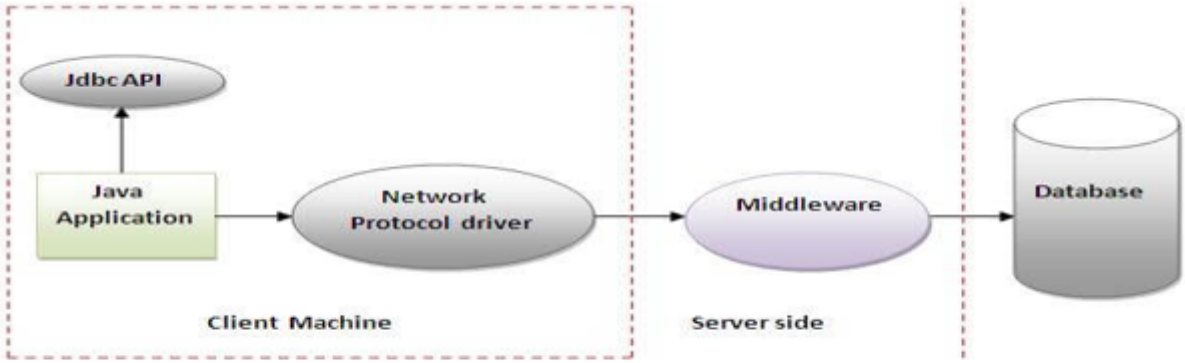


Figure- Network Protocol Driver

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.

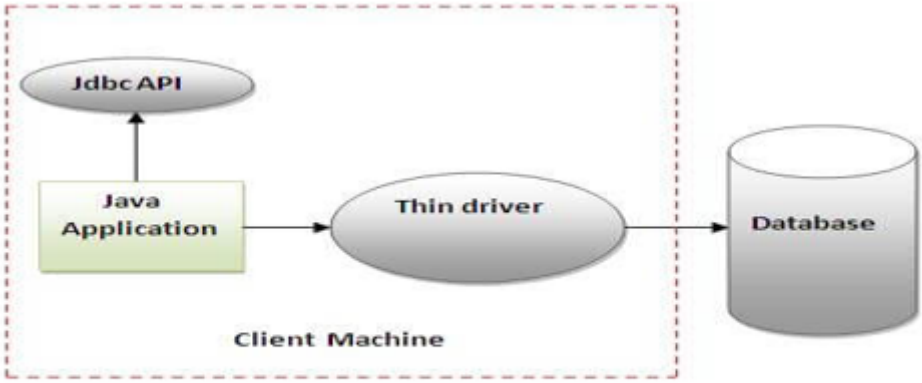


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depend on the Database

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

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).

JDBC Architecture

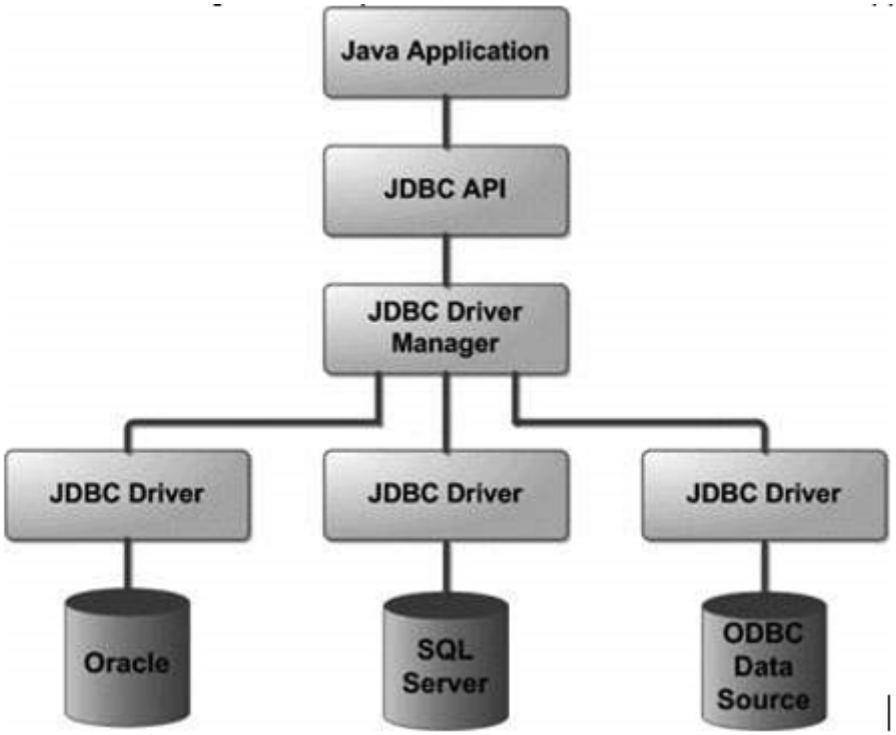
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDB Architecture consists of two layers −

- 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.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- 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.

Important Developments in JDBC-ODBC

Stands for "Open Database Connectivity." With all the different types of databases available, such as Microsoft Access, Filemaker, and MySQL, it is important to have a standard way of transferring data to and from each kind of database. For this reason, the SQL Access group created the ODBC standard back in 1992. Any application that supports ODBC can access information from an ODBC-compatible database, regardless of what database management system the database uses.

For a database to be ODBC-compatible, it must include an ODBC database driver. This allows other applications to connect to and access information from the database with a standard set of commands. The driver translates standard ODBC commands into commands understood by the database's proprietary system. Thanks to ODBC, a single application (such as Web server program) can access information from several different databases using the same set of commands.

JDBC with MS Access – UCanAccess driver

UCanAccess is a pure Java JDBC Driver implementation which allows Java developers and JDBC client programs to read/write Microsoft Access database (.mdb and .accdb) files. No ODBC needed. JDBC-ODBC bridge has been discontinued in latest Java releases so here is an alternative method to connect to Microsoft Access Database using pure java based driver.

Database connectivity

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps

1. **Import JDBC Packages:** Add import statements to your Java program to import required classes in your Java code.

2. **Register JDBC Driver:** This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.

3. **Database URL Formulation:** This is to create a properly formatted address that points to the database to which you wish to connect.

4. **Create Connection Object:** Finally, code a call to the *DriverManager* object's *getConnection( )* method to establish actual database connection.

1. Import JDBC Packages

The Import statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.

To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following *imports* to your source code −

```
import java.sql.* ;  // for standard JDBC programs
```

2. Register JDBC Driver

Approach I - Class.forName()

The most common approach to register a driver is to use Java's Class.forName() method, to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");

Approach II - DriverManager.registerDriver()

```
   try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
```

3. Database URL Formulation

After you've loaded the driver, you can establish a connection using the DriverManager.getConnection() method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods −

- **getConnection(String url)**
- **getConnection(String url, Properties prop)**
- **getConnection(String url, String user, String password)**

Here each form requires a database URL. A database URL is an address that points to your database.

connection = DriverManager.*getConnection*(dbURL);

Using Only a Database URL

A second form of the DriverManager.getConnection( ) method requires only a database URL

String dbURL =

"jdbc:ucanaccess://C:/Users/MyComputer/Desktop/Student.accdb";

4. Using a Database URL and a Properties Object A third form of the DriverManager.getConnection( ) method requires a database URL and a Properties object connection = DriverManager.getConnection(dbURL);

Closing JDBC Connections

connection.close();

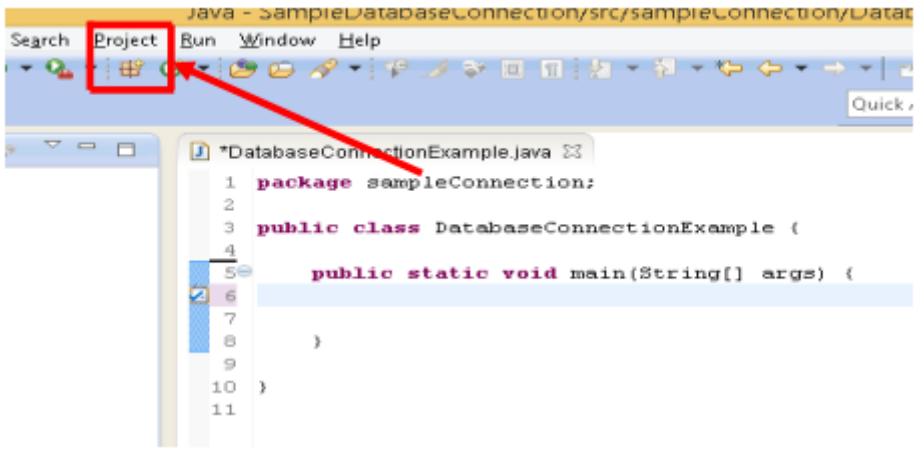
A Sample Java program connecting to created Student.accdb for simulation and clear understanding of the use of JDBC UcanaccessDriver (JDBC). The first illustration are the steps of installing of the UcanaccessDriver in Elipse IDE.

1.Create table

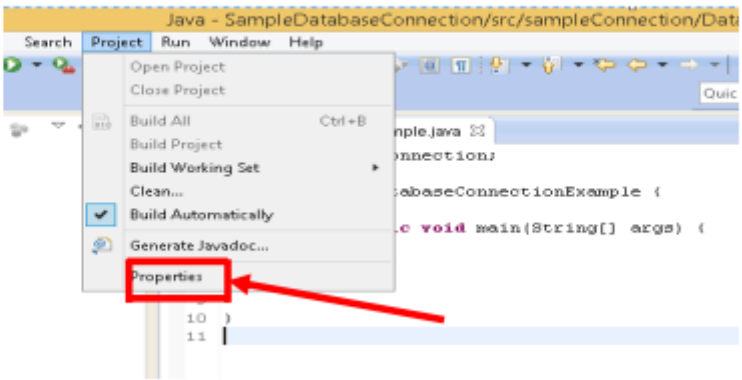
• Steps to create database connection using MsAccess

• Import JDBC packages

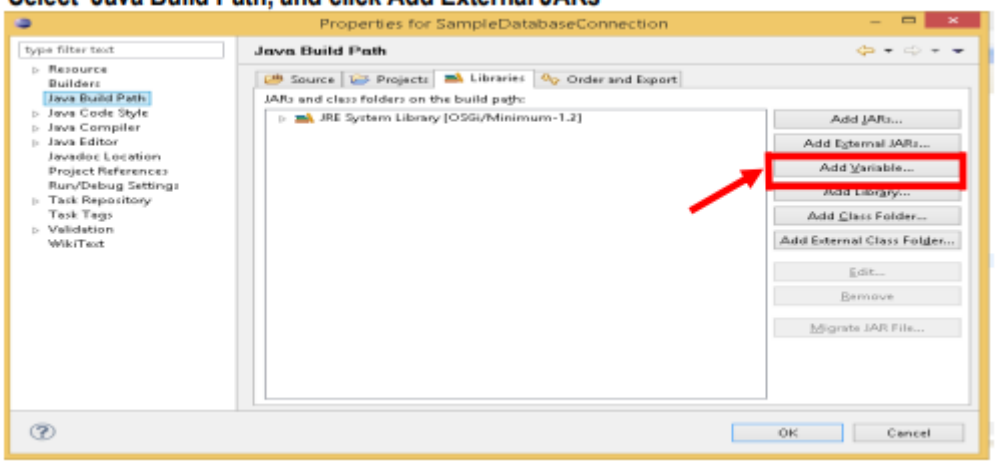
• Click the Project Menu



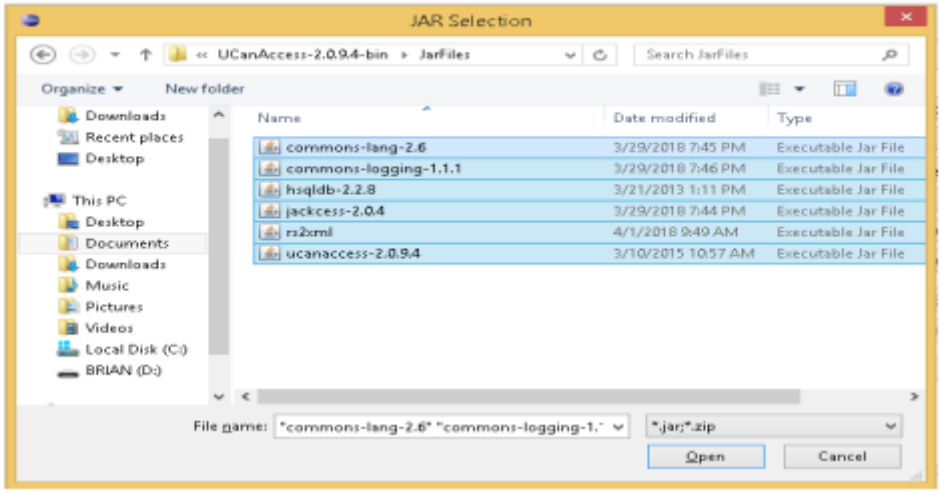
- **Select Properties**



- **Select Java Build Path, and click Add External JARs**



- **Select all the JAR files and click Open, then click OK in the Java build path menu**



- **Creating Student.accdb**

Student

StudentId	StudentNam	Age	Course	Click to Add
2019-MN-1	Lopez, Gerald	20	BSCS	
2019-MN-2	Reyes, Luigi	18	BSIT	
2019-MN-3	Legaspi, Sean	19	BSIS	
2019-MN-4	Abandia, Bea	21	BSA	
2019-MN-5	Vergara, Kate	20	BSCE	

- **Example program to connect java to database with SELECT query using MsAccess**

```
package sampleConnection;

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

public class DatabaseConnectionExample {

 public static void main(String[] args) {

 // variables
   Connection connection = null;
   Statement statement = null;
   ResultSet resultSet = null;

 // Step 1: Loading or registering JDBC driver class
            try {
    Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
    System.out.println("Loading Success!");
    }//end of try
  catch(ClassNotFoundException cnfex) {
       System.out.println("Problem in loading MS Access JDBC driver");
       cnfex.printStackTrace();
    }//end of catch

            // Step 2: Opening database connection
```

```
        try {
        String dbURL =
"jdbc:ucanaccess://C:/Users/MyComputer/Desktop/Student.accdb";


    // Step 2.A: Create and get connection using DriverManager class
          connection = DriverManager.getConnection(dbURL);

         // Step 2.B: Creating JDBC Statement
           statement = connection.createStatement();

         // Step 2.C: Executing SQL &amp;amp; retrieve data into ResultSet
           resultSet = statement.executeQuery("SELECT * FROM PLAYER");

           System.out.println("ID\t\tName\t\tAge\tMatches");
           System.out.println("==\t\t=============\t===\t=======");

         // processing returned data and printing into console
         while(resultSet.next()) {
          System.out.println(resultSet.getInt(1) + "\t" +
                    resultSet.getString(2) + "\t" +
                    resultSet.getString(3) + "\t" +
                    resultSet.getString(4));
         }// end of while loop
        }// end of try
        catch(SQLException sqlex){
           sqlex.printStackTrace();
        }// end of catch
        finally {

           // Step 3: Closing database connection
           try {
              if(null != connection) {

                 // cleanup resources, once after processing
                 resultSet.close();
                 statement.close();


                  // and then finally close connection
                  connection.close();
              }
           }//end of try
           catch (SQLException sqlex) {
              sqlex.printStackTrace();
           }//end of catch
        }//end of finally
     }//end of main
```

}//end of DatabaseConnectionExample.java

- **Program Output**

```
Loading Success!
ID              Name            Age     Course
==              =============   ===     =======
2019-MN-1       Lopez, Gerald   20      BSCS
2019-MN-2       Reyes, Luigi    18      BSIT
2019-MN-3       Legaspi, Sean   19      BSIS
2019-MN-4       Abandia, Bea    21      BSCE
2019-MN-5       Vergara, Kate   20      BSA
```

EXERCISE 1

A student is to create a simple student registration system. What are the possible columns or fields of th entity student?

EXERCISE 2

Your school would like to create simple registration system to help students enrolled in the college. Aside from the entity student, what will be the other entities to be included in the database to complete the system?

◀ Preliminary Activity for Week 15

Jump to...

Analysis, Application, and Exploration for Week 15 ▶

Navigation

Home
Dashboard
Site pages
My courses
121 - CC106
Participants
Grades
General
MODULE 1: WHAT IS APPLICATION DEVELOPMENT?
MODULE 2: WHAT ARE THE TECHNICAL SKILLS REQUIRED I...
MODULE 3: WHAT ARE THE PROGRAMMING LANGUAGES USED ...
MODULE 4: WHAT IS JAVA PROGRAMMING LANGUAGE AS APP...
MODULE 5: HOW TO WRITE JAVA PROGRAMMING LANGUAGE A...
MODULE 6: PRELIMINARY EXAMINATION
MODULE 7: HOW TO WRITE JAVA PROGRAM USING INTEGRAT...

Fair Warning

2nd Semester Enrollment

Activities

- Assignments
- Forums
- Quizzes
- Resources