Standard Steps followed for developing JDBC(JDBC4.X) Application
==========
1. Load and register the Driver
2. Establish the Connection b/w java application and database
3. Create a Statement Object
4. Send and execute the Query
5. Process the result from ResultSet
6. Close the Connection

1. Load and register the Driver

In earlier version of JDBC 3.X we were loading and registering the driver using the following approach

```
Driver driver = new Driver();
DriverManager.registerDriver(driver);
```

Alternate to this we can also load the driver as shown below

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

```
class Driver{
      static{
            Driver driver = new Driver();
            DriverManager.registerDriver(driver);
      }
}
```

Note: We say a class represents a Driver, iff the class has implemented an interface called "java.sql.Driver(I)".

```
MySQL       => Driver(c) implements Driver(I)
Oracle        => OracleDriver(c) implements Driver(I)
Postgresql => PostgreSqlDriver(c) implements Driver(I)
```

From JDBC4.X onwards loading and registering would happen automatically depending upon the jar added in the classpath
location of the project.

Note:
1. JVM will search for the jar in the classpath
2. It will open the jar,move to META-INF folder
3. It will open services folder
4. It will search for java.sql.Driver file
5. Whatever value which is present inside Driver file that would be loaded automatically using
        Class.forName(value)

The above feature of JDBC4.X is called as "AutoLoading".

Formatting the String query to accept the dynamic inputs
==========
```
int sage = scanner.nextInt();
String sname = scanner.next();
String saddr = scanner.next();

sname = " ' " + sname + " ' ";
saddr  =  " ' "    + saddr   + " ' ";
```

In DB specific query
   String => Varchar ===> ' '

```
   int       =>  int           ===> direct values
String query = "insert into student(`sname`,`sage`,`saddr`) values
("+sname+","+sage+","+saddr+")";
System.out.println(query);
```

To resolve the problem of the above approach we use a inbuilt class called "String".

```
int sage = scanner.nextInt();
String sname = scanner.next();
String saddr = scanner.next();

public static String format(String format, Object... args) {
        return new Formatter().format(format, args).toString();
 }
```

note:

String use format specifier as '%s'
int use format specifier as %d
flaot use format specifier as %f

```
String query =String.format( "insert into student(`sname`,`sage`,`saddr`) values
('%s',%d,'%s')",sname,sage,saddr );
```

Through JDBC we have performed CRUD opeartion along with dynamic inputs from the user

Insert => Create
Select => Read
Update => Update
Delete => Delete

Assignment
=========
Give the menu to the user as the operation listed below on student table
1. Create 2. Read 3. Update 4. Delete