UNIT-V
Stream based I/O (java.io)

The Stream classes-Byte streams and Character streams:

Byte Streams: These streams read / write data in ASCII format, i.e. they read 1 byte and write 1 byte data at a time. The classes used for low level streaming are identified with a suffix InputStream / OutputStream in their names. These are used to write and read data from binary files. Generally the IO operations that have hardware interactivity are managed with byte streams.

Ex:FileInputStream, FileOutputStream, ByteArrayInputStream, ByteArrayOutPutStream, PrintStream etc.

Char streams: These are called High Level Streams. These streams read / write data in Unicode format, i.e. these read 2 bytes and write 2 bytes at a time. The classes that are used for high level streaming of data are identified with suffix Reader / Writer in their names. These are used to read and write data from text files. Generally the IO operations that do not have hardware interactivity are managed with char streams.

Ex: FileReader, FileWriter, BufferedReader, PrintWriter etc.
For Reading data from hard disk, either FileInputStream or FileReader are preferred.

The FileInputStream reads 1 byte at a time and FileReader reads 2 bytes at a time.

But if the file is a binary file, then FileInputStream must be preferred. All the input streaming classes are derived from the class InputStream and all data writing classes are derived from the class OutputStream. The InputStream and OutputStream classes are abstract classes, so they cannot be instantiated directly or they cannot be used directly by creating objects of them.

These provide what to be for reading and writing. All streaming classes are derived from these two and they provide how the data is to be read or write as per their specification.

For example the InputStream class has methods….
read() :- reads a single char /key stroke from key board and returns its ASCII value as an integer.
read(byte[]) : reads data into the given byte array
close() : used to close the stream.
available() : returns no. of bytes to be read.

All byte streams override these methods to read data 1 byte at a time. And all char streams override these methods to read 2 byte data at a time.

Similarly, the OutputStream class has methods…..
write(int): int specifying a char is written
write(byte[]): writes given byte array
All byte streams override these methods to write data 1 byte at a time. And all char streams override these methods to write 2 byte data at a time.
**All streams other than PrintStream (the System.out) throws IOException.

Q. Write about different types of streams in java

Reading console Input and Writing Console Output:

For reading data from console (system's keyboard), we need to use InputStream class object (System.in). But reading data with "in" object, returns the ASCII code or reads data into a byte[], which further needs to be converted into Unicode format to express it to user.

Rather, we can use a class BufferedReader object for reading data.
But, as BuferedReader is a char stream object, it cannot interact with hardware components directly. So we can read data by holding System.in object with BufferedReader object.

Filter streams are used for data conversions between different types of streams. Like the InputStreamReader class is used to convert ASCII formatted data into Unicode format.
Code :

```
BufferedReader br=new BufferedReader( new InputStreamReader(
System.in) );
String s=br.readLine();
```

Here the data read from System.in is in ASCII format and the readLine() method of BufferedReader class returns data in String (Unicode) format.
The class InputStreamReader acts as a filter between System.in object and br object , that converts data from ASCII standards into Unicode standards.

Q. write about reading writing data from console

Reading and writing Files using FileInputStream/ FileOutputStream and FileReader/FileWriter classes:

File Streams : when we develop a program , we concentrate on user interface process and data storage. The data storage is managed with variables, arrays, objects etc. but these are managed with in main memory. To use data in future, we must store data onto hard disk in the form of files. In java, we use file streams for data writing and reading.

We can use the classes FileInputStream / FileOutputStream or FileReader/ FileWriter to read/write data.

The byte stream classes read / write 1 byte at a time and the char streams read/write 2 bytes.

** for binary files, we must use only byte streams.
Ex:

```
class MyIO1{
public static void main(String a[]) throws
IOException,FileNotFoundException{
FileOutputStream fout=new FileOutputStream("c:/MyFolder/myfile.txt");
fout.write( new String("welcome to java files").getBytes() );
fout.close();
}
}
```

```
class MyIO2{
public static void main(String a[]) throws
IOException,FileNotFoundException{
FileWriter fw=new FileFileWriter("c:/MyFolder/myfile.txt");
fout.write("welcome to java files" );
fout.close();
}
}
```

In MyIO1, as we used byte streams, we wrote data by converting string into byte array and in MyIO2, as we use FileWriter which is a char stream, we wrote data as string itself. Instead of fout.write("welcome to java files"), we can also write as, fout.write(new String("welcome to java files").toCharArray()), by converting string into char array. File reading also very similar to above programs, but we use FileInputStream and FileReader classes.

Q. Write about File Streams in java

JDBC Introduction

Two-Tier Architecture,

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.

Definition of JDBC (Java Database Connectivity)

JDBC is an API(Application programming interface) used in java programming to interact with databases. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.

Purpose of JDBC

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC (Open database connectivity) driver. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.

The JDBC is java database connectivity a software mechanism used to connect to data. It is generally called JDBC driver. It is used to connect to a DB, through the ODBC of the DB.

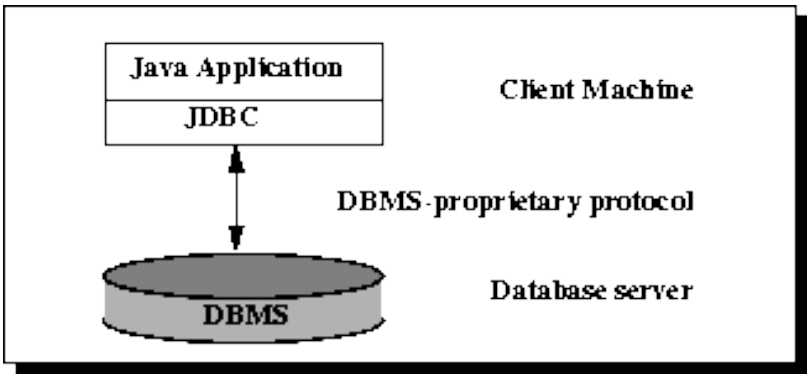
When we develop an application, we concentrate on 3 things viz., user interface, process and data storage. As data is vital in the application, if all three components are managed with one application, the data is logically associated with logic of the program and if we modify logic, it affects the data. To overcome this problem, the application can be divided into 2 parts, in which one part is managed with java program and the other part with a DBMS like oracle.

The java program manages user interface and business logic and the oracle manages the data. Here the application runs on 2 layers viz., java layer and

oracle layer otherwise program layer and DB layer. So it is called 2tiered architecture.

It works as client-server application. When user uses this application, he can see only the java program, so it is called front end and the DB is never worked by user directly as it stays at back of the application, so it is called backend.

A 2-tiered application looks as :



In above 2-tiered applications, the user interface and the logic are bound with each other strongly. Where if the user interface is modified it may affect logic or the user interface need to be updated in all systems to make use of updations. It is practically impossible to update front end in all the systems. And to make use of data, every system must have the front end installed.

** Explain 2 tiered architecture.

Types of JDBC Drivers & Loading Driver and Connecting to DB:

First we need to identify the drivers to be used for connecting java with DB. There are different types of drivers

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

**Q. There can be a question like Write about different JDBC drivers

The oracle thin driver is managed with a class “OracleDriver” of “oracle.dbc.driver” package.

In case of oracle 10g, it is available in the path of oracle installation as a jar file.

For ex:

C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

This need to be set in classpath for making use of it.

Once classpath is set, then we can load the above thin driver into memory for establishing connection. It can be done with the class “java.lang.Class” and its static method forName().

Code:

Class.forName(“oracle.jdbc.driver.OracleDriver”);

In database terminology it is said that the driver is registered.

The different classes and interface of java.sql package are used for fetching data from DB into java program viz.,

DriverManager : it is a class, used to establish connection to the DB whose URL is specified using above loaded drivers, and returns the connection as an instance of java.sql.Connection interface.

The DB URL for oracle using thin drivers is :

“jdbc:oracle:thin:@localhost:1521:xe”

The DriverManager has a static method getConnection(), that establish connection.

Code :

Connection con=

DriverManager.getConenction(“jdbc:oracle:thin:@localhost:1521:xe”,”userid”, ”password”);

Now…. The DriverManager’s getConnection() establish connection to oracle DB with name “xe” running on localhost with port number 1521 with given user name and password and returns the connection and that is held by con.

Core Interfaces in java.sql: Connection, Statement, PreparedStatement, ResultSet,

Connection : this interface is instantiated by the getConnection() of DriverManager and has methods to create Statement , PreparedStatement objects. These two objects are used to convert String query into SQL standard query and execute on DB.

Statement : this interface has methods execute() and executeUpdate() etc to execute SQL queries. This object is given with query while executing the query. So each time the query is compiled.

Ex:

Statement st=con.createStatement();

ResultSet rs= st.executeQuery("select * from emp");

st.executeUpdate("update emp set sal=3000 where eid=101");

PreparedStatemenet : this interface also has same methods execute() and executeUpdate() as of in Statement interface. This object is given with query while getting created. Query is not given while executing. So query is compiled once and can be executed any number of times.

The PreparedStatement has a facility of giving with "place holders" for values not known, and later the place holders modified with values. The place holder is represented with a "?".

Ex1:

PreparedStatement pst1=con.prepareStatement("select * from emp where eid=?");

pst1.setInt(1, Integer.parseInt(tf.getText()));

ResultSet rs1=pst1.executeQuery();

Ex2:

PreparedStatement pst2=con.prepareStatement("update emp set ename=?, sal=? where eid=?");

pst2.setString(1, tf.getText());

pst2.setInt(2, Integer.parseInt(tf_sal.getText()));

pst2.setInt(3, Integer.parseInt(tf_num.getText()));

pst1.executeUpdate();

*irrespective of Statement or PreparedStatement used, the executeQuery() returns no. of rows data as instance of ResultSet interface. And the executeUpdate() returns an int representing no. of rows affected by query.

ResultSet : it is an interface that is used to move across the data fetched by executeQuery() of either Statement or PreparedStatement objects.

It has methods first(), previous(), last() and next() to move across each row of data fetched. The methods getInt(), getString() etc are used to fetch data of each column in current row.

Ex:

System.out.println(rs.getInt(1) +" "+ rs.getString(2)+" "+ rs.getInt(3));

Q. explain working with JDBC

Write about classes and interfaces used for JDBC connectivity:

Connection : this interface is instantiated by the getConnection() of DriverManager and has methods to create Statement , PreparedStatement objects. These two objects are used to convert String query into SQL standard query and execute on DB.

Statement : this interface has methods execute() and executeUpdate() etc to execute SQL queries. This object is given with query while executing the query. So each time the query is compiled.

Ex:

Statement st=con.createStatement();

ResultSet rs= st.executeQuery("select * from emp");

st.executeUpdate("update emp set sal=3000 where eid=101");

PreparedStatemenet : this interface also has same methods execute() and executeUpdate() as of in Statement interface. This object is given with query while getting created. Query is not given while executing. So query is compiled once and can be executed any number of times.

The PreparedStatement has a facility of giving with “place holders” for values not known, and later the place holders modified with values. The place holder is represented with a “?”.

Ex1:

PreparedStatement pst1=con.prepareStatement(“select * from emp where eid=?”);

pst1.setInt(1, Integer.parseInt(tf.getText()));

ResultSet rs1=pst1.executeQuery();

Ex2:

PreparedStatement pst2=con.prepareStatement(“update emp set ename=?, sal=? where eid=?”);

pst2.setString(1, tf.getText());

pst2.setInt(2, Integer.parseInt(tf_sal.getText()));

pst2.setInt(3, Integer.parseInt(tf_num.getText()));

pst1.executeUpdate();

*irrespective of Statement or PreparedStatement used, the executeQuery() returns no. of rows data as instance of ResultSet interface. And the executeUpdate() returns an int representing no. of rows affected by query.

CallableStatement : used to execute built in stored procedures of the SQL. i.e. PL/SQL part queries like procedures, functions etc are execute by this CallableStatement object, to increase the speed of execution of the query.

ResultSet : it is an interface that is used to move across the data fetched by executeQuery() of either Statement or PreparedStatement objects.

It has methods first(), previous(), last() and next() to move across each row of data fetched. The methods getInt(), getString() etc are used to fetch data of each column in current row.

Ex:

System.out.println(rs.getInt(1) +" "+ rs.getString(2)+" "+ rs.getInt(3));

Q. explain working with JDBC

Write about classes and interfaces used for JDBC connectivity

Explain how to get data from a DB into a java program.

Basic CRUD Operations using JDBC (Create, Read, Update, Delete) :
JDBC (Java Database Connectivity) is used to perform database operations from a Java program. The four basic database operations are Create, Read, Update, and Delete (CRUD). These operations are performed using Statement or PreparedStatement interfaces. The methods executeQuery() and executeUpdate() are mainly used to execute SQL statements.

1. Create Operation (Insert)

The Create operation is used to insert new records into a database table. In JDBC, this is done using the SQL INSERT statement. The method executeUpdate() is used to execute the insert query. It returns an integer value representing the number of rows inserted.

PreparedStatement is preferred because the query is precompiled and supports placeholders (?) for dynamic values.

Example:

```
PreparedStatement pst = con.prepareStatement(
"insert into emp(eid, ename, sal) values(?, ?, ?)");
pst.setInt(1, 101);
pst.setString(2, "Ravi");
pst.setInt(3, 3000);
int rows = pst.executeUpdate();
```

If rows > 0, the record is inserted successfully.

2. Read Operation (Select)

The Read operation is used to retrieve data from the database. It uses the SQL SELECT statement. The executeQuery() method is used to execute select queries. It returns a ResultSet object that contains the retrieved data.

The ResultSet interface provides methods like next(), getInt(), and getString() to access the data row by row.

Example:

```
PreparedStatement pst = con.prepareStatement(
"select * from emp");
ResultSet rs = pst.executeQuery();

while(rs.next()) {
   System.out.println(rs.getInt(1) + " " +
                      rs.getString(2) + " " +
                      rs.getInt(3));
}
```

The next() method moves the cursor to the next row until all records are processed.

3. Update Operation

The Update operation is used to modify existing records in a table. It uses the SQL UPDATE statement. The executeUpdate() method executes the update query and returns the number of rows affected.

Example:

```
PreparedStatement pst = con.prepareStatement(
"update emp set sal=? where eid=?");
pst.setInt(1, 5000);
pst.setInt(2, 101);
int rows = pst.executeUpdate();
```

If the returned value is 1, one record is updated successfully.

4. Delete Operation

The Delete operation removes records from a database table. It uses the SQL DELETE statement. The executeUpdate() method executes the delete query and returns the number of rows deleted.

Example:

```
PreparedStatement pst = con.prepareStatement(
"delete from emp where eid=?");
pst.setInt(1, 101);
int rows = pst.executeUpdate();
```

If rows > 0, the record is deleted successfully.

**** Q. Explain basic CRUD operations**