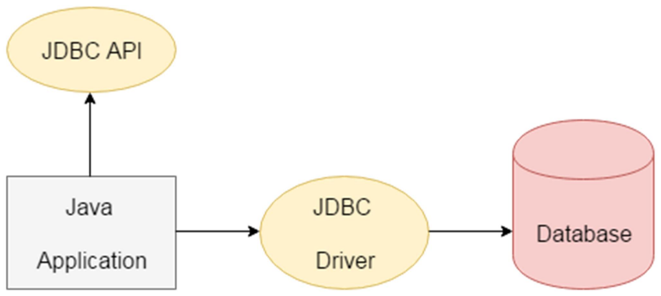
JDBC in Java

Java Database Connectivity (JDBC) is an Application Programming Interface (API) used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. It provides classes and interfaces to connect or communicate Java application with database. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database. The JDBC classes are contained in the Java Package java.sql and javax.sql.



JDBC 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 (Bridge):** This driver uses ODBC driver to connect to the database. It converts JDBC method calls into the ODBC function calls. It is also called Universal driver because it can be used to connect to any of the databases.

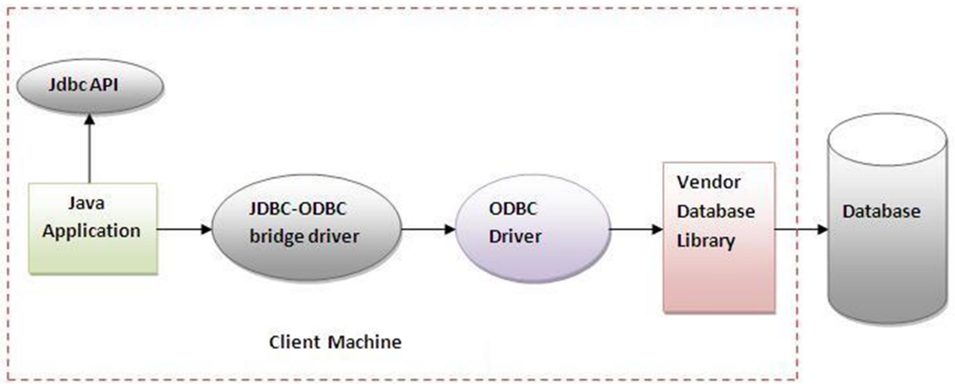


Figure- JDBC-ODBC Bridge Driver

2) **Native-API driver (Native):** The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API.

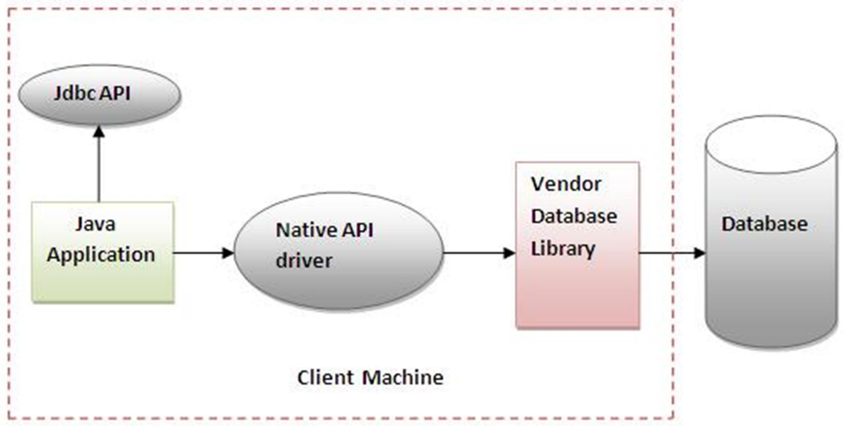


Figure- Native API Driver

3) **Network Protocol driver (Middleware):** This driver translates the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translates JDBC calls into database specific calls.

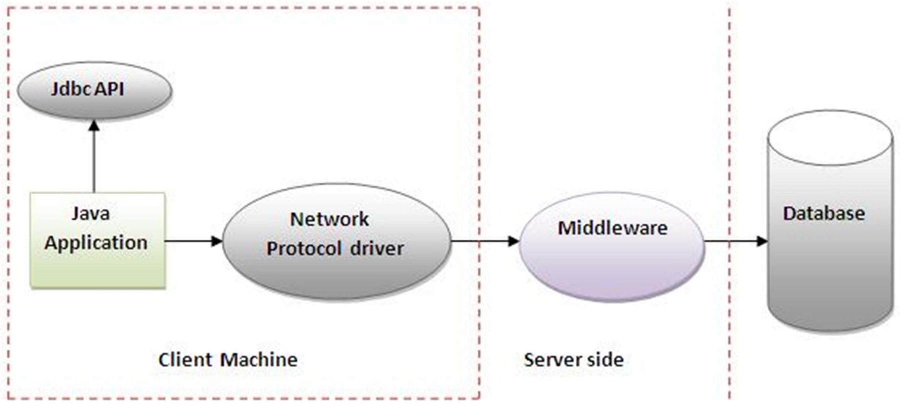


Figure- Network Protocol Driver

4) **Thin driver (Pure):** This driver is called Pure Java Driver because this driver interacts directly with database. It does not require any native database library that is why it is also known as Thin Driver.

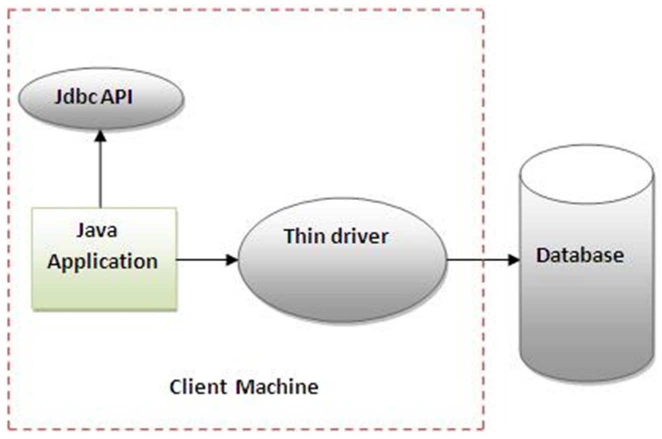
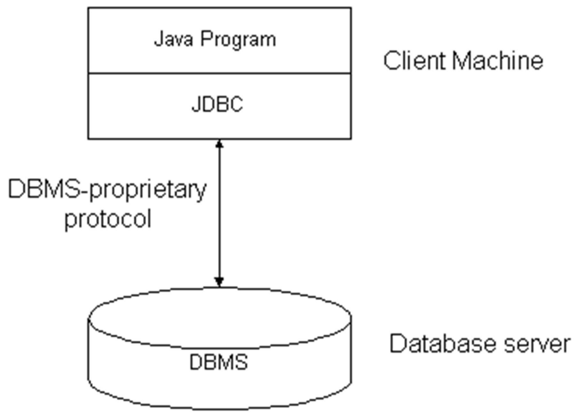


Figure- Thin Driver

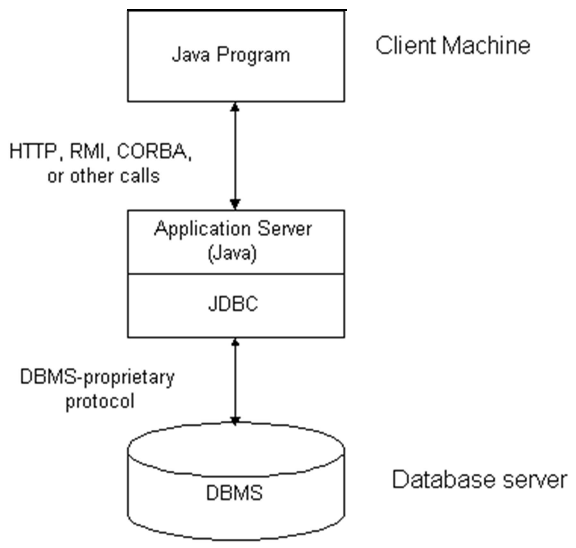
Two Tier and Three Tier client server model

The JDBC API supports two-tier and three-tier architecture for database access.

In a two-tier model, a Java application/applet communicates directly with the database, via the JDBC driver. The Java application/applet and the database can be on the same machine, or the database can be on a server and the Java application/applet can be on a client machine using any network protocol.



In a three-tier model, a Java application/applet communicates with a middle tier component that functions as an application server. In the three-tier model, commands are sent to a middle tier of services, which then send SQL statements to the database. The database processes the SQL statements and sends the results back to the middle tier, which then sends them to the user.


Java Program
Client Machine
HTTP, RMI, CORBA, or other calls
Application Server (Java)
JDBC
DBMS-proprietary protocol
DBMS
Database server

Steps to connect a Java Application to Database Using JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

1. **Import JDBC Packages**

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

```
import java.sql.*;
```

2. **Load and Register the JDBC Driver**

In this step we load the JDBC driver class into JVM. This step is also called as registering the JDBC driver. The forName() method of class Class is used to register the driver class. This method is used to dynamically load the driver class. This step can be completed in two ways.

- Class.forName("fully qualified classname")

Example: Class.forName("oracle.jdbc.driver.OracleDriver");

- DriveManager.registerDriver(object of driver class)

Example: DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

3. **Open a connection to the database**

In this step connection between a java program and a database will be opened. To open the connection, we call getConnection() method of DriverManager class.

For getConnection() method we need to pass three parameters.

- url
- username
- password

url: url is used to select one register JDBC driver among multiple registered driver by DriverManager class.

username and password: username and password are used for authentication purpose.

Connection con=new DriverManager.getConnection(url, username, password);

Example:

Connection con=new DriverManager.getConnection(Jdbc:Odbc:< dsn >", "scott","tiger");

4. **Create a statement object to perform a query**

To transfer sql commands from java program to database we need statement object. To create a statement object we call createStatement() method of connection interface. The object of statement is responsible to execute queries with the database.

Example:

Statement stmt=new createStatement();

5. **Execute the statement object and return a query ResultSet**

Once a Statement object has been constructed, the next step is to execute the query.

Call any one of the following three methods of Statement interface is used to execute queries to the database and to get the output.

- executeUpdate(): Used for non-select operations.
- executeQuery(): Used for select operation.

- execute(): Used for both select or non-select operation.

These methods return the object of ResultSet that can be used to get all the records of a table.

Example:

```
ResultSet rs=s.executeQuery("select * from user");
 while(rs.next())
 {
  System.out.println(rs.getString(1)+" "+rs.getString(2));
 }
```

6. Process the ResultSet

Once the query has been executed, there are two steps to be carried out:

- Processing the output ResultSet to fetch the rows
- Retrieving the column values of the current row

The first step is done using the next() method of the ResultSet object.

The second step is done by using the getXXX() methods of the JDBC rset object. Here getXXX() corresponds to the getInt(), getString() etc with XXX being replaced by a Java datatype.

The following code demonstrates the above steps:

```
String str;
while (rset.next())
 {
 str = rset.getInt(1)+ " "+ rset.getString(2)+ "
        "+rset.getFloat(3)+ " "rset.getInt(4)+ "\n";
 }
byte buf[] = str.getBytes();
OutputStream fp = new FileOutputStream("query1.lst");
fp.write(buf);
fp.close();
```

Here the 1, 2, 3, and 4 in rset.getInt(), rset.getString(), getFloat(), and getInt() respectively denote the position of the columns in the SELECT statement.

7. Closing the ResultSet and Statement

Once the ResultSet and Statement objects have been used, they must be closed explicitly. This is done by calls to the close() method of the ResultSet and Statement classes. The following code illustrates this:

```
rset.close();

sql_stmt.close();
```

8. Closing the Connection

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the close() method of the Connection class.

The following line of code does this:

```
conn.close();
```

ResultSet Interface in JDBC

ResultSet also an interface and is used to retrieve SQL select query results. A default ResultSet object is not updatable and the cursor moves only in forward direction.

The methods of the ResultSet interface can be broken down into three categories −

Navigational methods: Used to move the cursor around.

Get methods: Used to view the data in the columns of the current row being pointed by the cursor.

Update methods: Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

Types of ResultSet Interface

1. ResultSet.TYPE_FORWARD_ONLY: The ResultSet can only be navigating forward.

2. ResultSet.TYPE_SCROLL_INSENSITIVE: The ResultSet can be navigated both in forward and backward direction. It can also jump from current position to another position. The ResultSet is not sensitive to change made by others.

3. ResultSet.TYPE_SCROLL_SENSITIVE: The ResultSet can be navigated in both forward and backward direction. It can also jump from current position to another position. The ResultSet is sensitive to change made by others to the database.

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

ResultSetMetaData is an interface which provides information about a result set that is returned by an executeQuery() method. This interface provides the metadata about the database. It includes the information about the names of the columns, number of columns etc.

Syntax:

```
ResultSetMetaData rsmd=rs.getMetaData();
```

Commonly used methods of ResultSetMetaData interface

Method Name	Description
int getColumnCount() throws SQLException	Returns the number of columns in a ResultSet.
String getColumnName(int column) throws SQLException	Returns the column name.
String getColumnTypeName(int column) throws SQLException	Returns the database specific datatype of the column.
String getTableName(int column) throws SQLException	Returns the column's table name.
String getSchemaName(int column) throws SQLException	Returns the name of the schema of the column's table.

Java.lang package in Java

It provides classes that are fundamental to the design of the Java programming language. The most important classes are Object, which is the root of the class hierarchy, and Class, instances of which represent classes at run time.

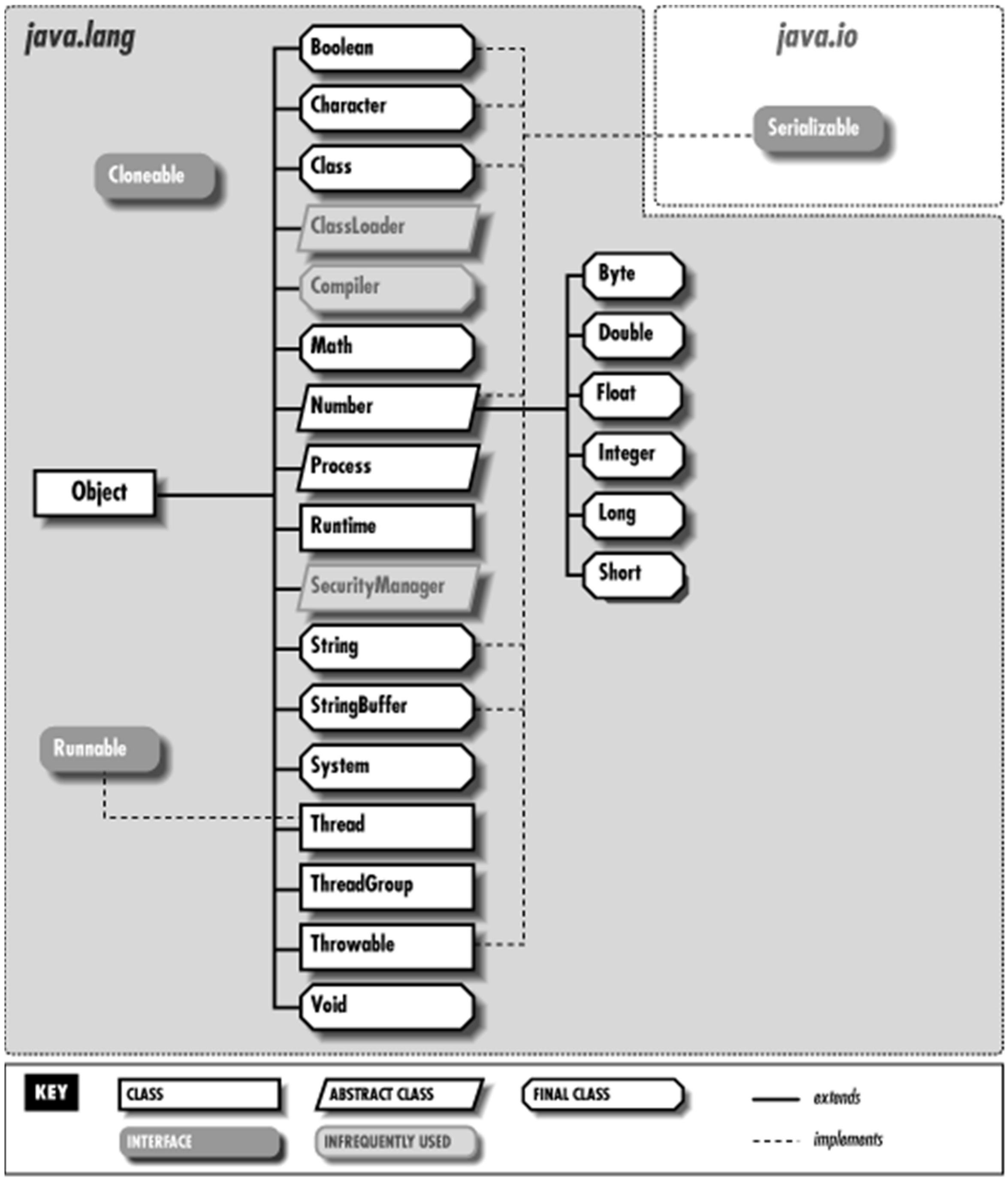
Java.util Package in Java

It contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

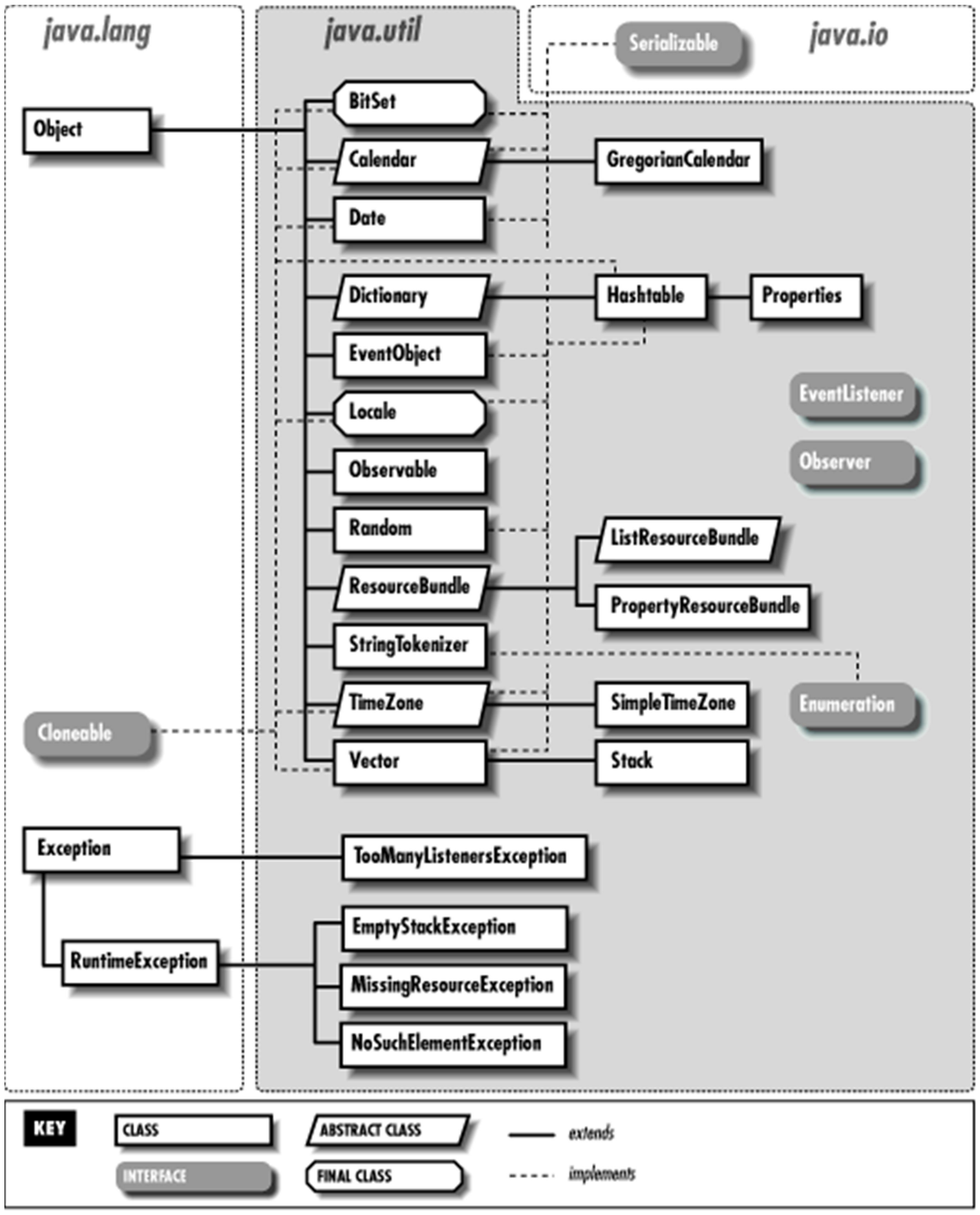
Java.io Package in Java

This package provides for system input and output through data streams, serialization and the file system. Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package will cause a NullPointerException to be thrown.

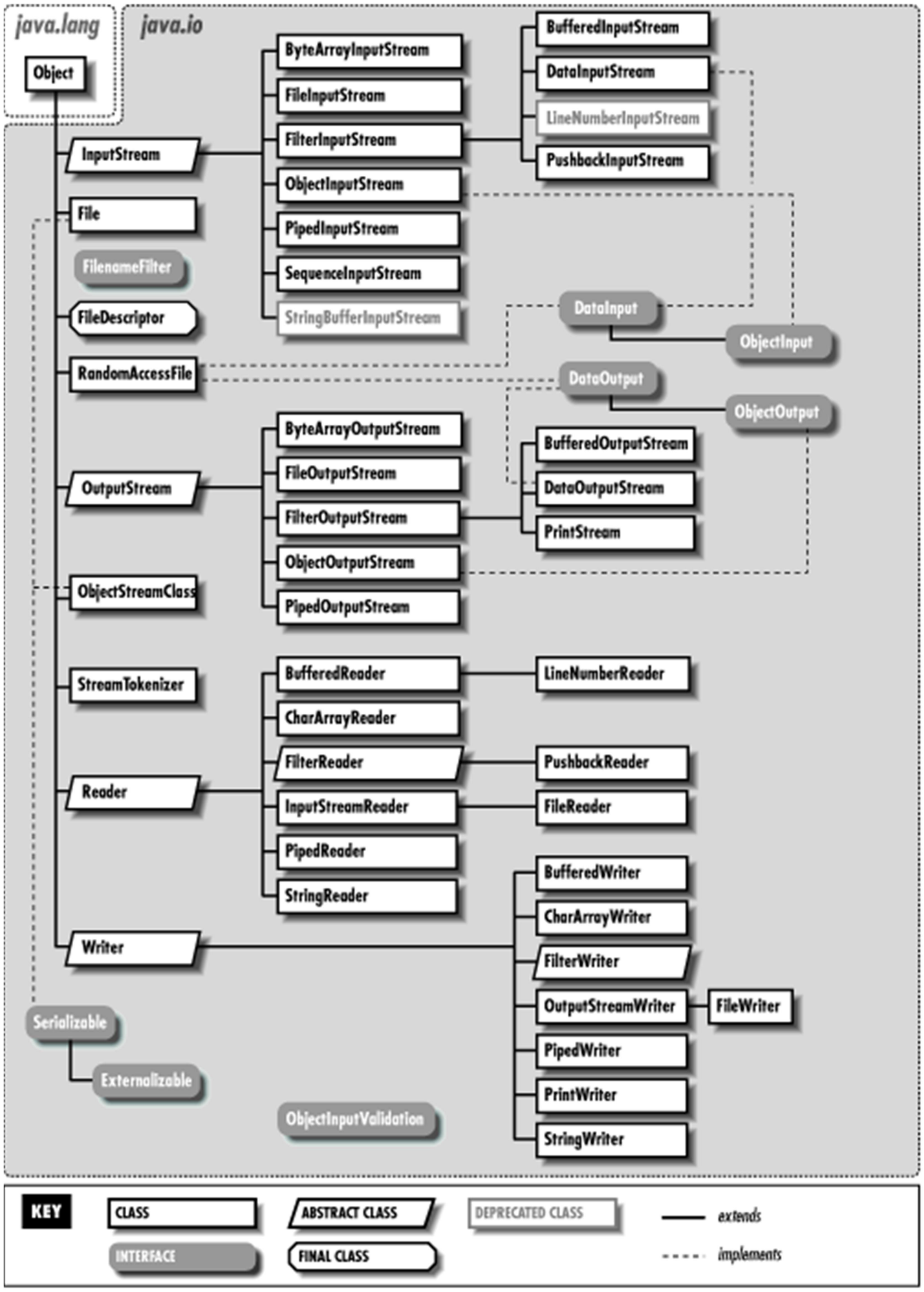
Java.lang package in Java



Java.util Package in Java



Java.io Package in Java


java.lang
java.io
Object
InputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
File
FilenameFilter
FileDescriptor
RandomAccessFile
DataInput
ObjectInput
DataOutput
ObjectOutput
OutputStream
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
ObjectOutputStream
PipedOutputStream
ObjectStreamClass
StreamTokenizer
Reader
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
PrintWriter
StringWriter
Serializable
Externalizable
ObjectInputValidation
KEY
CLASS
ABSTRACT CLASS
DEPRECATED CLASS
INTERFACE
FINAL CLASS
extends
implements