# RowSet Interface- added in Java 7

## The RowSet Interface

The interface javax.sql.RowSet extends the ResultSet interface to provide support for the JavaBean component model. The RowSet interface defines getters and setters for different data types.

RowSet also supports methods to add and remove event listeners (since it is a JavaBean component). Other Java objects may use the event notification mechanism supported by RowSet.

The RowSet interface implements ***the Observer design pattern*** (as a Subject). A Java object that wants to receive event notification from RowSet must implement the RowSetListener interface and must be registered with the RowSet object. RowSet notifies all the registered objects on the occurrence of one of the following events: change in cursor location, change in a row, and change in the entire RowSet object.

Java provides five different flavors of the RowSet interface (see Figure below). The interfaces of these flavors can be found in the javax.sql.rowset package. **The five interfaces are JdbcRowSet, JoinRowSet, CachedRowSet, WebRowSet, and FilteredRowSet**.

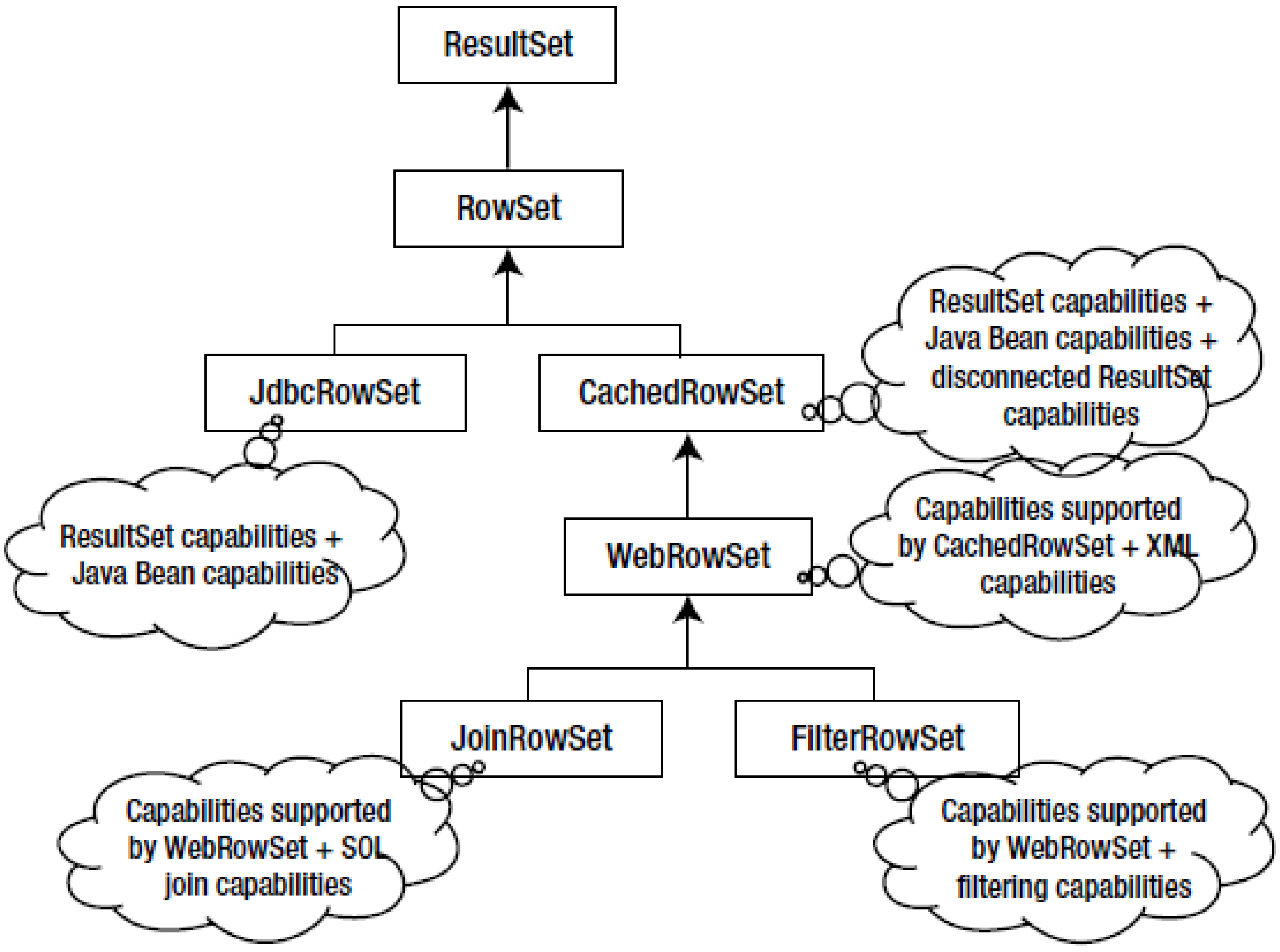
ResultSet
RowSet
JdbcRowSet
CachedRowSet
ResultSet capabilities + Java Bean capabilities + disconnected ResultSet capabilities
ResultSet capabilities + Java Bean capabilities
WebRowSet
Capabilities supported by CachedRowSet + XML capabilities
JoinRowSet
FilterRowSet
Capabilities supported by WebRowSet + SQL join capabilities
Capabilities supported by WebRowSet + filtering capabilities

***Figure*** *The RowSet hierarchy*

**JdbcRowSet is a connected RowSet**, which means a JdbcRowSet implementation is always connected to the corresponding database.

**The other four interfaces are disconnected RowSets**, which means an object of any one of these four RowSet implementations (except JdbcRowSet) connects to the database only when they want to read or write; all the other times they are disconnected from the database.
This property of implementation of these four interfaces along with the capability of being serializable make them suitable for sending over the network.

Figure above shows the hierarchical relationship among various RowSet interfaces. As you already know, RowSet is a subinterface of the ResultSet interface. JdbcRowSet is a subinterface of RowSet; JdbcRowSet has all the features ResultSet supports plus Java Bean capabilities. CachedRowSet is also a subinterface of the RowSet interface; it has all the features JdbcRowSet supports plus it has the capabilities of a disconnected ResultSet. WebRowSet adds XML capabilities to the CachedRowSet features. Similarly, JoinRowSet adds SQL join capabilities to WebRowSet, and FilteredRowSet adds result filtering capabilities to WebRowSet.

***Let's now discuss the RowSetProvider class and the RowSetFactory interface, which were introduced in Java 7***. RowSetProvider provides APIs to get a RowSetFactory implementation that can be used to instantiate a proper RowSet implementation. It provides the following two methods:

• **RowSetFactory newFactory():** This API creates a new instance of a RowSetFactory implementation. So which factory implementation will this method instantiate? It is a good question; the answer is that this API infers the type of factory implementation to instantiate from the environment settings. It first looks in the system property **javax.sql.rowset.RowSetFactory**. If the API could not infer the factory implementation, then it uses ServiceLoader API to determine the type of the factory implementation to instantiate, and finally it looks for the platform default implementation of the RowSetFactory instance. If the API could not infer the factory implementation type, it raises a SQLException.

**RowSetFactory newFactory(String factoryClassName, ClassLoader classloader):** If it is unclear which driver will be loaded when you call the plain newFactory() method due to multiple drivers in the scope, you can use the overloaded method newFactory(), which takes the class name of the factory and the class loader and instantiates the appropriate factory.

RowSetFactory defines five methods; each method creates a type of RowSet implementation. So why are RowSetFactory and RowSetProvider required to create a RowSet implementation when you can create them using the traditional plain way?
**The answer is *flexibility*;** you can create the RowSet object without specifying the details that you need to provide during a traditional RowSet object; you can then set other details once the object gets created.

**I am writing a program to see how to use RowSetFactory and RowSetProvider ?**

```
import javax.sql.rowset.*;
import java.sql.*;
// To illustrate how to use RowSet, RowSetProvider, and RowSetFactory
class DbQueryRowSetTest {
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/addressBook";
String userName = "root";
String password = "mysql123";
try {
// first, create a factory object for rowset
RowSetFactory rowSetFactory = RowSetProvider.newFactory();
// create a JDBC rowset from the factory
JdbcRowSet rowSet = rowSetFactory.createJdbcRowSet();
rowSet.setUrl(url);
rowSet.setUsername(userName);
rowSet.setPassword(password);
rowSet.setCommand("SELECT * FROM contact");
rowSet.execute();
System.out.println("id \tfName \tlName \temail \t\tphoneNo");
while (rowSet.next()){
System.out.println(rowSet.getInt("id") + "\t"
+ rowSet.getString("firstName") + "\t"
+ rowSet.getString("lastName") + "\t"
+ rowSet.getString("email") + "\t"
+ rowSet.getString("phoneNo"));
}
} catch (SQLException sqle) {
sqle.printStackTrace();
}
}
}
```

```
Id      fName    lName   email             phoneNo
1       Mohan    Taylor  mohan@abc.com     +919976543210
2       William  Becker  william@abc.com   +919876543210
3       Samuel   Uncle   sam@abc.com       +919955331100
```