Java Database Connectivity

**JDBC** is a Java-based data access technology (Java Standard Edition platform) from [Oracle Corporation](http://en.wikipedia.org/wiki/Oracle_Corporation). This technology is an [API](http://en.wikipedia.org/wiki/Application_programming_interface) for the [Java programming language](http://en.wikipedia.org/wiki/Java_(programming_language)) that defines how a client may access a [database](http://en.wikipedia.org/wiki/Database). It provides methods for querying and updating data in a database. JDBC is oriented towards [relational databases](http://en.wikipedia.org/wiki/Relational_database). A JDBC-to-[ODBC](http://en.wikipedia.org/wiki/ODBC) bridge enables connections to any ODBC-accessible data source in the [JVM](http://en.wikipedia.org/wiki/JVM) host environment.

Functionality[[edit source](http://en.wikipedia.org/w/index.php?title=Java_Database_Connectivity&action=edit&section=2) | [editbeta](http://en.wikipedia.org/w/index.php?title=Java_Database_Connectivity&veaction=edit&section=2)]

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. JDBC represents statements using one of the following classes:

* [Statement](http://download.oracle.com/javase/7/docs/api/java/sql/Statement.html) – the statement is sent to the database server each and every time.
* [PreparedStatement](http://download.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html) – the statement is cached and then the [execution path](http://en.wikipedia.org/wiki/Query_plan) is pre-determined on the database server allowing it to be executed multiple times in an efficient manner.
* [CallableStatement](http://download.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html) – used for executing [stored procedures](http://en.wikipedia.org/wiki/Stored_procedures) on the database.

Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many [rows](http://en.wikipedia.org/wiki/Row_(database)) were affected in the database. These statements do not return any other information.

Query statements return a JDBC row result set. The row result set is used to walk over the [result set](http://en.wikipedia.org/wiki/Result_set). Individual [columns](http://en.wikipedia.org/wiki/Column_(database)) in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types.

There is an extension to the basic JDBC API in the [javax.sql](http://download.oracle.com/javase/7/docs/api/javax/sql/package-summary.html).

JDBC connections are often managed via a [connection pool](http://en.wikipedia.org/wiki/Connection_pool) rather than obtained directly from the driver. Examples of connection pools include [BoneCP](http://jolbox.com/), [C3P0](http://sourceforge.net/projects/c3p0) and [DBCP](http://commons.apache.org/dbcp)

JDBC drivers[[edit source](http://en.wikipedia.org/w/index.php?title=Java_Database_Connectivity&action=edit&section=3) | [editbeta](http://en.wikipedia.org/w/index.php?title=Java_Database_Connectivity&veaction=edit&section=3)]

JDBC drivers are client-side [adapters](http://en.wikipedia.org/wiki/Adapter_pattern) (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

**Types**[[edit source](http://en.wikipedia.org/w/index.php?title=Java_Database_Connectivity&action=edit&section=4) | [editbeta](http://en.wikipedia.org/w/index.php?title=Java_Database_Connectivity&veaction=edit&section=4)]

There are commercial and free drivers available for most relational database servers. These drivers fall into one of the following types:

* [Type 1](http://en.wikipedia.org/wiki/JDBC_driver#Type_1_Driver_-_JDBC-ODBC_bridge) that calls native code of the locally available ODBC driver.
* [Type 2](http://en.wikipedia.org/wiki/JDBC_driver#Type_2_Driver_-_Native-API_Driver_specification) that calls database vendor native library on a client side. This code then talks to database over network.
* [Type 3](http://en.wikipedia.org/wiki/JDBC_driver#Type_3_Driver_-_Network-Protocol_Driver), the pure-java driver that talks with the server-side middleware that then talks to database.
* [Type 4](http://en.wikipedia.org/wiki/JDBC_driver#Type_4_Driver_-_Native-Protocol_Driver), the pure-java driver that uses database native protocol.

There is also a type called [internal JDBC driver](http://en.wikipedia.org/w/index.php?title=Internal_JDBC_driver&action=edit&redlink=1), driver embedded with JRE in Java-enabled SQL databases. It's used for [Java stored procedures](http://en.wikipedia.org/wiki/Java_stored_procedure). This does not belong to the above classification, although it would likely be either a type 2 or type 4 driver (depending on whether the database itself is implemented in Java or not). An example of this is the KPRB driver supplied with Oracle RDBMS. "jdbc:default:connection" is a relatively standard way of referring making such a connection (at least Oracle and [Apache Derby](http://en.wikipedia.org/wiki/Apache_Derby) support it). The distinction here is that the JDBC client is actually running as part of the database being accessed, so access can be made directly rather than through network protocols.

JDBC actually has two levels of interface. In addition to the main interface, there is also an API from a JDBC "manager" that in turn communicates with individual database product "[driver](http://searchstorage.techtarget.com/definition/driver)s," the JDBC-ODBC bridge if necessary, and a JDBC network driver when the Java program is running in a network environment (that is, accessing a remote database).

When accessing a remote database, JDBC takes advantage of the Internet's file addressing scheme and a file name looks much like a Web page address (or [Uniform Resource Locator](http://searchnetworking.techtarget.com/definition/URL)). For example, a Java SQL statement might identify the database as:

JDBC specifies a set of object-oriented [class](http://whatis.techtarget.com/definition/class)es for the programmer to use in building SQL requests. An additional set of classes describes the JDBC driver API. The most common SQL[data type](http://searchsoa.techtarget.com/definition/data-type)s, mapped to Java data types, are supported. The API provides for implementation-specific support for Microsoft Transaction Server requests and the ability to [commit](http://searchsqlserver.techtarget.com/definition/commit) or [roll back](http://searchsqlserver.techtarget.com/definition/rollback) to the beginning of a transaction.

A specification developed by Sun Microsystems that defines how [Java](http://www.webopedia.com/TERM/J/Java.html) [objects](http://www.webopedia.com/TERM/O/object.html)interact. An object that conforms to this specification is called a JavaBean,and is similar to an [ActiveX control](http://www.webopedia.com/TERM/A/ActiveX_control.html). It can be used by any application that understands the JavaBeans format.

The principal difference between ActiveX controls and JavaBeans are that ActiveX controls can be developed in any [programming language](http://www.webopedia.com/TERM/P/programming_language.html) but executed only on a [Windows](http://www.webopedia.com/TERM/W/Windows.html) [platform](http://www.webopedia.com/TERM/P/platform.html), whereas JavaBeans can be developed only in Java, but can run on any platform.

JavaBeans

**JavaBeans** are [reusable](http://en.wikipedia.org/wiki/Code_reuse) [software components](http://en.wikipedia.org/wiki/Component-based_software_engineering) for [Java](http://en.wikipedia.org/wiki/Java_(programming_language)). Practically, they are classes that encapsulate many objects into a single object (the bean). They are [serializable](http://en.wikipedia.org/wiki/Serialization), have a [0-argument constructor](http://en.wikipedia.org/wiki/Nullary_constructor), and allow access to properties using [getter and setter methods](http://en.wikipedia.org/wiki/Mutator_method).

Advantages[[edit source](http://en.wikipedia.org/w/index.php?title=JavaBeans&action=edit&section=1) | [editbeta](http://en.wikipedia.org/w/index.php?title=JavaBeans&veaction=edit&section=1)]

* A bean provides all the benefits of Java's "[write once, run anywhere](http://en.wikipedia.org/wiki/Write_once,_run_anywhere)" paradigm.
* The properties, events, and methods of a bean that are exposed to another application can be controlled.
* A bean may register to receive events from other objects and can generate events that are sent to those other objects.
* Auxiliary software can be provided to help a person configure a java bean.
* The configuration setting of bean can be saved in a persistent storage and restored at a later time.

Disadvantages[[edit source](http://en.wikipedia.org/w/index.php?title=JavaBeans&action=edit&section=2) | [editbeta](http://en.wikipedia.org/w/index.php?title=JavaBeans&veaction=edit&section=2)]

.A class with a [nullary constructor](http://en.wikipedia.org/wiki/Nullary_constructor) is subject to being instantiated in an invalid state. If such a class is instantiated manually by a developer (rather than automatically by some kind of framework), the developer might not realize that the class has been improperly instantiated. The compiler can’t detect such a problem, and even if it’s documented, there’s no guarantee that the developer will see the documentation.

* Having to create a getter for every property and a setter for many, most, or all of them can lead to an immense quantity of [boilerplate code](http://en.wikipedia.org/wiki/Boilerplate_code).

JavaBeans API[[edit source](http://en.wikipedia.org/w/index.php?title=JavaBeans&action=edit&section=3) | [editbeta](http://en.wikipedia.org/w/index.php?title=JavaBeans&veaction=edit&section=3)]

The JavaBeans functionality is provided by a set of classes and interfaces in the java.beans package.

| **Interface** | **Description** |
| --- | --- |
| AppletInitializer | Methods in this interface are used to initialize Beans that are also [applets](http://en.wikipedia.org/wiki/Java_applet). |
| BeanInfos | This interface allows the designer to specify information about the events, methods and properties of a Bean. |
| Customizer | This interface allows the designer to provide a graphical user interface through which a bean may be configured. |
| DesignMode | Methods in this interface determine if a bean is executing in design mode. |
| ExceptionListener | A method in this interface is invoked when an exception has occurred. |
| PropertyChangeListener | A method in this interface is invoked when a bound property is changed. |
| PropertyEditor | Objects that implement this interface allow the designer to change and display property values. |
| VetoableChangeListener | A method in this interface is invoked when a Constrained property is changed. |
| Visibility | Methods in this interface allow a bean to execute in environments where the GUI is not available. |

JavaBean conventions[[edit source](http://en.wikipedia.org/w/index.php?title=JavaBeans&action=edit&section=4) | [editbeta](http://en.wikipedia.org/w/index.php?title=JavaBeans&veaction=edit&section=4)]

In order to function as a JavaBean [class](http://en.wikipedia.org/wiki/Class_(computer_science)), an object class must obey certain conventions about method naming, construction, and behaviour. These conventions make it possible to have tools that can use, reuse, replace, and connect JavaBeans.

The required conventions are as follows:

* The class must have a public [default constructor](http://en.wikipedia.org/wiki/Default_constructor) (with no arguments). This allows easy instantiation within editing and activation frameworks.
* The class [properties](http://en.wikipedia.org/wiki/Property_(programming)) must be accessible using *get*, *set*, *is* (used for boolean properties instead of get), and other methods (so-called [accessor methods](http://en.wikipedia.org/wiki/Accessor) and [mutator methods](http://en.wikipedia.org/wiki/Mutator_method)) according to a standard [naming convention](http://en.wikipedia.org/wiki/Naming_conventions_(programming)). This allows easy automated inspection and updating of bean state within frameworks, many of which include custom editors for various types of properties. Setters can have one or more than one argument.
* The class should be [serializable](http://en.wikipedia.org/wiki/Serialization#Java). [This allows applications and frameworks to reliably save, store, and restore the bean's state in a manner independent of the [VM](http://en.wikipedia.org/wiki/Virtual_machine) and of the platform.]

**TestPersonBean.java**:

**import** player.PersonBean;

**public** **class** TestPersonBean {

**public** **static** **void** main(String[] args) {

PersonBean person = **new** PersonBean();

person.setName("Bob");

person.setDeceased(**false**);

*// Output: "Bob [alive]"*

System.out.print(person.getName());

System.out.println(person.isDeceased() ? " [deceased]" : " [alive]");

}

}

Remote Method Invocation (RMI)

1. [Remote Method Invocation (RMI)](http://www.javatpoint.com/RMI)
2. [Understanding stub and skeleton](http://www.javatpoint.com/RMI#rmistubandskeleton)
   1. [stub](http://www.javatpoint.com/RMI#stub)
   2. [skeleton](http://www.javatpoint.com/RMI#skeleton)
3. [Understanding requirements for the distributed applications](http://www.javatpoint.com/RMI#reqdistributed)
4. [Steps to write the RMI program](http://www.javatpoint.com/RMI#rmisteps)
5. [Example of creating simple RMI application](http://www.javatpoint.com/RMI#rmiex)
   1. [Create the remote interface](http://www.javatpoint.com/RMI#rmiexst1)
   2. [Provide the implementation of the remote interface](http://www.javatpoint.com/RMI#rmiexst2)
   3. [create the stub and skeleton objects using the rmic tool](http://www.javatpoint.com/RMI#rmiexst3)
   4. [Start the registry service by rmiregistry tool](http://www.javatpoint.com/RMI#rmiexst4)
   5. [Create and start the remote application](http://www.javatpoint.com/RMI#rmiexst5)
   6. [Create and start the client application](http://www.javatpoint.com/RMI#rmiexst6)

|  |
| --- |
| The Remote Method Invocation (RMI) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. |
| The RMI provides remote communication between the applications using two objects stub and skeleton. |

Understanding stub and skeleton

|  |
| --- |
| RMI uses stub and skeleton object for communication with the remote object. A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects: |

stub

|  |
| --- |
| The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:   1. It initiates a connection with remote Virtual Machine (JVM), 2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM), 3. It waits for the result 4. It reads (unmarshals) the return value or exception, and 5. It finally, returns the value to the caller. |

skeleton

|  |
| --- |
| The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:   1. It reads the parameter for the remote method 2. It invokes the method on the actual remote object, and 3. It writes and transmits (marshals) the result to the caller.   In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons. |



Understanding requirements for the distributed applications

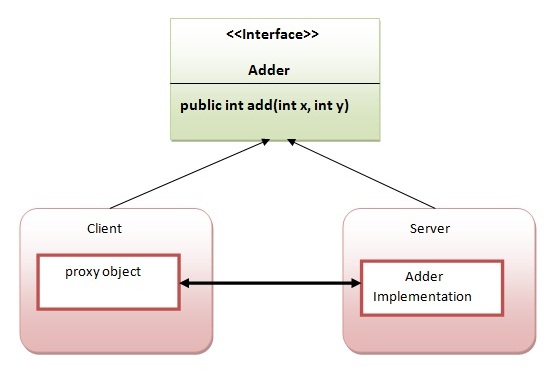
|  |
| --- |
| If any application performs these tasks, it can be distributed application.   1. The application need to locate the remote method 2. It need to provide the communication with the remote objects, and 3. The application need to load the class definitions for the objects.   The RMI application have all these features, so it is called the distributed application. |

Steps to write the RMI program

|  |
| --- |
| The is given the 6 steps to write the RMI program.   1. Create the remote interface 2. Provide the implementation of the remote interface 3. Compile the implementation class and create the stub and skeleton objects using the rmic tool 4. Start the registry service by rmiregistry tool 5. Create and start the remote application 6. Create and start the client application |

*Example of creating simple RMI application*

|  |
| --- |
| In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application. |



**1) create the remote interface**

|  |
| --- |
| For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException. |

1. **import** java.rmi.\*;
2. **public** **interface** Adder **extends** Remote{
4. **public** **int** add(**int** x,**int** y)**throws** RemoteException;
5. }

**2) Provide the implementation of the remote interface**

|  |
| --- |
| Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to   * Either extend the UnicastRemoteObject class, * or use the exportObject() method of the UnicastRemoteObject class   In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException. |

1. **import** java.rmi.\*;
2. **import** java.rmi.server.\*;
4. **public** **class** AdderRemote **extends** UnicastRemoteObject **implements** Adder{
6. AdderRemote()**throws** RemoteException{
7. **super**();
8. }
10. **public** **int** add(**int** x,**int** y){**return** x+y;}
12. }

**3) create the stub and skeleton objects using the rmic tool.**

|  |
| --- |
| Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects. |

1. rmic AdderRemote

**4) Start the registry service by the rmiregistry tool**

|  |
| --- |
| Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000. |

1. rmiregistry 5000

**5) Create and run the server application**

|  |
| --- |
| Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.   1. **public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it returns the reference of the remote object. 2. **public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it binds the remote object with the given name. 3. **public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;** it destroys the remote object which is bound with the given name. 4. **public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;** it binds the remote object to the new name. 5. **public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;** it returns an array of the names of the remote objects bound in the registry. |

**1a)Developing Web Page Using Basic HtmlAlgorithm Steps:**

Begin:Step 1: Create a HTML document.Step 2: Including the image of the college in the html document using the <IMG> tag.Step 3: Using <FRAMESET> tag divide the html page as preferred.Step 4: Create relevant web pages for the college and display the menu in the left side of the window.Step 5: Make the respective forms to be displayed in the right side of web page when themenu is clickedEnd

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Servlet** technology is used to create web application (resides at server side and generates dynamic web page).  **Servet** technology is robust and scalable as it uses the java language. Before Servlet, CGI (Common Gateway Interface) scripting language was used as a server-side programming language. But there were many disadvantages of this technology. We have discussed these disadvantages below.  There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.  What is a Servlet?   |  | | --- | | Servlet can be described in many ways, depending on the context.   * Servlet is a technology i.e. used to create web application. * Servlet is an API that provides many interfaces and classes including documentations. * Servlet is an interface that must be implemented for creating any servlet. * Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests. * Servlet is a web component that is deployed on the server to create dynamic web page. |   servlet   |  | | --- | |  |   What is web application?   |  | | --- | | A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request. |   Advantage of Servlet  **advantage of servlet**   |  |  | | --- | --- | | There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lighweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:   1. **better performance:** because it creates a thread for each request not process. 2. **Portability:** because it uses java language. 3. **Robust:** Servlets are managed by JVM so no need to worry about momory leak, garbage collection etc. 4. **Secure:** because it uses java language | | |  | |