**Naming Guidelines**

It’s one of the Java coding guidelines which depends on the context you are in. Let’s read more about this.

**Arguments Or Parameters**

* Use a related name for the value/object being passed, and prefixing with **<arg> or <param>**.

**e.g.** argEmpName, paramSalary etc.

**Fields And Variables**

* Start field/variable name in lower case and then continue in sentence case.

**e.g.** clickCheckBox, viewInfo, openWindow.

* Don’t use underscores to start or separate the words.

**Constants**

* Use upper case and underscores to form constants.

**e.g.** static final int MAX\_SIZE = 256;

static final string BROWSER\_TYPE = “Chrome”;

**Classes And Interface**

* Always begin class/interface names with a capital letter.

**e.g.** Class Name: PageFactory or PageObject.

Interface Name: IPageObjectModel

**Compilation Unit Files**

* Use the name of the class or interface prefixed with <.java> to represent it is a source code file.

**e.g.** TestPage.java, UIMap.java, LoginPage.java.

**Component**

* Use a meaningful name with a proper suffix.

**e.g.** categoryMenu, listView etc.

**Packages**

* Start package name with unique top-level domain names like com, edu, gov, etc. Follow the ISO Standards 3166, 1981. Remaining part may vary according to an organization’s internal naming structure.

**e.g.** com.techbeamers.testpackage.

**Member Function**

* Have a function name that relates to the task is meant for. Start it with an active verb whenever possible.

**e.g.**showStatus(), drawCircle(), addLayoutComponent().

**Boolean getter member functions**

* It’s a good practice to prefix boolean getter functions with <is>.

**e.g.** isVisible(), isChecked(), isNumeric().

**Getter member functions**

* Usually all getter functions should start with <get> prefix.

**e.g.** getLocalDate(), getMonth(), getDayOfMonth().

**Setter member functions**

* Usually all setter functions should start with <set> prefix.

**e.g.**setLocalDate(), setMonth(), setDayOfMonth().

**Exception Handling Coding Practices**

* Throwable and Error classes should not be caught
* Throwable.printStackTrace(…) should never be called
* Generic exceptions Error, RuntimeException, Throwable and Exception should never be thrown
* Exception handlers should preserve the original exception
* System.out or System.err should not be used to log exceptions

**Encryption and decryption**

Encryption and decryption are fundamental requirements of every secure-aware application, therefore the Java platform provides strong support for encryption and decryption through its **Java Cryptographic Extension (JCE)** framework which implements the standard cryptographic algorithms such as AES, DES, DESede and RSA. This tutorial shows you how to basically encrypt and decrypt files using the *Advanced Encryption Standard (AES)*algorithm. AES is a symmetric-key algorithm that uses the same key for both encryption and decryption of data.

Here are the general steps to encrypt/decrypt a file in Java:

* Create a **Key** from a given byte array for a given algorithm.
* Get an instance of **Cipher** class for a given algorithm transformation.
* Initialize the **Cipher** with an appropriate mode (encrypt or decrypt) and the given **Key**.
* Invoke **doFinal(input\_bytes)** method of the **Cipher** class to perform encryption or decryption on the **input\_bytes**, which returns an encrypted or decrypted byte array.
* Read an input file to a byte array and write the encrypted/decrypted byte array to an output file accordingly.

**JDBC**

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

* Making a connection to a database
* Creating SQL or MySQL statements
* Executing that SQL or MySQL queries in the database
* Viewing & Modifying the resulting records

## **Creating JDBC Application:**

There are six steps involved in building a JDBC application which I'm going to brief in this tutorial:

## **Import the packages:**

This requires that you include the packages containing the JDBC classes needed for database programming. Most often, using import java.sql.\* will suffice as follows:

//STEP 1. Import required packages

import java.sql.\*;

## **Register the JDBC driver:**

This requires that you initialize a driver so you can open a communications channel with the database. Following is the code snippet to achieve this:

//STEP 2: Register JDBC driver

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

## **Open a connection:**

This requires using the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with the database as follows:

//STEP 3: Open a connection

// Database credentials

static final String USER = "username";

static final String PASS = "password";

System.out.println("Connecting to database...");

conn = DriverManager.getConnection(DB\_URL,USER,PASS);

## **Execute a query:**

This requires using an object of type Statement or PreparedStatement for building and submitting an SQL statement to the database as follows:

//STEP 4: Execute a query

System.out.println("Creating statement...");

stmt = conn.createStatement();

String sql;

sql = "SELECT id, first, last, age FROM Employees";

ResultSet rs = stmt.executeQuery(sql);

If there is an SQL UPDATE,INSERT or DELETE statement required, then following code snippet would be required:

//STEP 4: Execute a query

System.out.println("Creating statement...");

stmt = conn.createStatement();

String sql;

sql = "DELETE FROM Employees";

ResultSet rs = stmt.executeUpdate(sql);

## **Extract data from result set:**

This step is required in case you are fetching data from the database. You can use the appropriate ResultSet.getXXX() method to retrieve the data from the result set as follows:

//STEP 5: Extract data from result set

while(rs.next()){

//Retrieve by column name

int id = rs.getInt("id");

int age = rs.getInt("age");

String first = rs.getString("first");

String last = rs.getString("last");

//Display values

System.out.print("ID: " + id);

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);

}

## **Clean up the environment:**

You should explicitly close all database resources versus relying on the JVM's garbage collection as follows:

//STEP 6: Clean-up environment

rs.close();

stmt.close();

conn.close();

**Servlet**

# **Steps to create a servlet example**

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

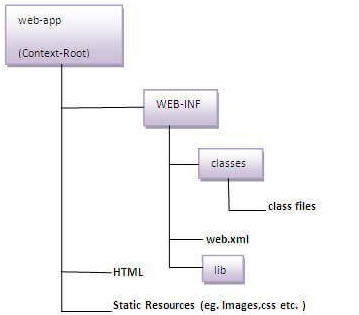
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

### 1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



### 2)Create a Servlet

|  |
| --- |
| There are three ways to create the servlet.   1. By implementing the Servlet interface 2. By inheriting the GenericServlet class 3. By inheriting the HttpServlet class   The HttpServlet class is widely used to create the servlet because it provides methods to handle http  requests such as doGet(), doPost, doHead() etc. |
| In this example we are going to create a servlet that extends the HttpServlet class. In this example,  we are inheriting the HttpServlet class and providing the implementation of the doGet() method.  Notice that get request is the default request. |

**3)Compile the servlet**

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

|  |  |
| --- | --- |
| **Jar file** | **Server** |
| 1) servlet-api.jar | Apache Tomcat |
| 2) weblogic.jar | Weblogic |
| 3) javaee.jar | Glassfish |
| 4) javaee.jar | JBoss |

**Two ways to load the jar file**

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

**4)Create the deployment descriptor (web.xml file)**

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

**web.xml file**

1. **<web-app>**
3. **<servlet>**
4. **<servlet-name>**sonoojaiswal**</servlet-name>**
5. **<servlet-class>**DemoServlet**</servlet-class>**
6. **</servlet>**
8. **<servlet-mapping>**
9. **<servlet-name>**sonoojaiswal**</servlet-name>**
10. **<url-pattern>**/welcome**</url-pattern>**
11. **</servlet-mapping>**
13. **</web-app>**

**Description of the elements of web.xml file**

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

|  |
| --- |
| **<web-app>** represents the whole application. |
| **<servlet>** is sub element of <web-app> and represents the servlet. |
| **<servlet-name>** is sub element of <servlet> represents the name of the servlet. |
| **<servlet-class>** is sub element of <servlet> represents the class of the servlet. |
| **<servlet-mapping>** is sub element of <web-app>. It is used to map the servlet. |
| **<url-pattern>** is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet. |
|  |

**5)Start the Server and deploy the project**

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

**One Time Configuration for Apache Tomcat Server**

You need to perform 2 tasks:

1. set JAVA\_HOME or JRE\_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

**How to change port number of apache tomcat**

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

**5) How to deploy the servlet project**

Copy the project and paste it in the webapps folder under apache tomcat.

But there are several ways to deploy the project. They are as follows:

* By copying the context(project) folder into the webapps directory
* By copying the war folder into the webapps directory
* By selecting the folder path from the server
* By selecting the war file from the server

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

1. projectfolder> jar cvf myproject.war \*

Creating war file has an advantage that moving the project from one location to another takes less time.

**6) How to access the servlet**

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

1. http://localhost:9999/demo/welcome