JAVA.LANG.RUNTIME.LOADLIBRARY METHOD

Description

The **java.lang.Runtime.loadLibrary***Stringfilename* method loads the dynamic library with the specified library name. A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent. The mapping from a library name to a specific filename is done in a system-specific manner.

First, if there is a security manager, its checkLink method is called with the libname as its argument. This may result in a security exception. The method System.loadLibrary*String* is the conventional and convenient means of invoking this method. If native methods are to be used in the implementation of a class, a standard strategy is to put the native code in a library file *callitLibFile* and then to put a static initializer:

```
static { System.loadLibrary("LibFile"); }
```

within the class declaration. When the class is loaded and initialized, the necessary native code implementation for the native methods will then be loaded as well. If this method is called more than once with the same library name, the second and subsequent calls are ignored.

Declaration

Following is the declaration for **java.lang.Runtime.loadLibrary** method

```
public void loadLibrary(String libname)
```

Parameters

- **libname** -- the name of the library.

Return Value

This method does not return a value.

Exception

- **SecurityException** -- if a security manager exists and its checkLink method doesn't allow loading of the specified dynamic library
- **UnsatisfiedLinkError** -- if the library does not exist
- **NullPointerException** -- if libname is null

Example

The following example shows the usage of lang.Runtime.loadLibrary method.

```
package com.tutorialspoint;

public class RuntimeDemo {

   public static void main(String[] args) {

   // print when the program starts
   System.out.println("Program starting...");

   // load a library that is Windows/System32 folder
   System.out.println("Loading Library...");
   Runtime.getRuntime().loadLibrary("C:/Windows/System32/crypt32.dll");
   System.out.println("Library Loaded.");
```

```
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
Program starting...
Loading Library...
Library Loaded.
```