JAVA.LANG.SECURITYMANAGER.CHECKEXEC METHOD EXAMPLE

Description

The **java.lang.SecurityManager.checkExec***Stringcmd* method throws a SecurityException if the calling thread is not allowed to create a subprocess. This method is invoked for the current security manager by the exec methods of class Runtime.

This method calls checkPermission with the FilePermission*cmd*, " *execute* " permission if cmd is an absolute path, otherwise it calls checkPermission with FilePermission "<< *ALLFILES* >>" , " *execute* " . If you override this method, then you should make a call to super.checkExec at the point the overridden method would normally throw an exception.

Declaration

Following is the declaration for **java.lang.SecurityManager.checkExec** method

```
public void checkExec(String cmd)
```

Parameters

- **cmd** -- the specified system command.

Return Value

This method does not return a value.

Exception

- **SecurityException** -- if the calling thread does not have permission to create a subprocess.
- **NullPointerException** -- if the cmd argument is null.

Example

Our examples require that the permissions for each command is blocked. A new policy file was set that allows only the creating and setting of our Security Manager. The file is in C:/java.policy and contains the following text:

```
grant {
  permission java.lang.RuntimePermission "setSecurityManager";
  permission java.lang.RuntimePermission "createSecurityManager";
  permission java.lang.RuntimePermission "usePolicy";
};
```

The following example shows the usage of lang.SecurityManager.checkExec method.

```
package com.tutorialspoint;

public class SecurityManagerDemo {

   public static void main(String[] args) {

   // set the policy file as the system securuty policy
   System.setProperty("java.security.policy", "file:/C:/java.policy");

   // create a security manager
   SecurityManager sm = new SecurityManager();

   // set the system security manager
   System.setSecurityManager(sm);
```

```
    // perform the check
    sm.checkExec("notepad.exe");

    // print a message if we passed the check
    System.out.println("Allowed!");
    }
}
```

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

```
Exception in thread "main" java.security.AccessControlException: access denied
(java.io.FilePermission <<ALL FILES>> execute)
```