**Java Runtime class**

**Java Runtime** class is used *to interact with java runtime environment*. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.

The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

**Important methods of Java Runtime class**

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public static Runtime getRuntime() | returns the instance of Runtime class. |
| 2) | public void exit(int status) | terminates the current virtual machine. |
| 3) | public void addShutdownHook(Thread hook) | registers new hook thread. |
| 4) | public Process exec(String command)throws IOException | executes given command in a separate process. |
| 5) | public int availableProcessors() | returns no. of available processors. |
| 6) | public long freeMemory() | returns amount of free memory in JVM. |
| 7) | public long totalMemory() | returns amount of total memory in JVM. |

**Java Runtime exec() method**

1. **public** **class** Runtime1{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("notepad");//will open a new notepad
4. }
5. }

**How to shutdown system in Java**

You can use *shutdown -s* command to shutdown system. For windows OS, you need to provide full path of shutdown command e.g. c:\\Windows\\System32\\shutdown.

Here you can use -s switch to shutdown system, -r switch to restart system and -t switch to specify time delay.

1. **public** **class** Runtime2{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("shutdown -s -t 0");
4. }
5. }

**How to shutdown windows system in Java**

1. **public** **class** Runtime2{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("c:\\Windows\\System32\\shutdown -s -t 0");
4. }
5. }

**How to restart system in Java**

1. **public** **class** Runtime3{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("shutdown -r -t 0");
4. }
5. }

**Java Runtime availableProcessors()**

1. **public** **class** Runtime4{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. System.out.println(Runtime.getRuntime().availableProcessors());
4. }
5. }

**Java Runtime freeMemory() and totalMemory() method**

In the given program, after creating 10000 instance, free memory will be less than the previous free memory. But after gc() call, you will get more free memory.

1. **public** **class** MemoryTest{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime r=Runtime.getRuntime();
4. System.out.println("Total Memory: "+r.totalMemory());
5. System.out.println("Free Memory: "+r.freeMemory());
7. **for**(**int** i=0;i<10000;i++){
8. **new** MemoryTest();
9. }
10. System.out.println("After creating 10000 instance, Free Memory: "+r.freeMemory());
11. System.gc();
12. System.out.println("After gc(), Free Memory: "+r.freeMemory());
13. }
14. }

Total Memory: 100139008

Free Memory: 99474824

After creating 10000 instance, Free Memory: 99310552

After gc(), Free Memory: 100182832