Process API Updates (JEP-102)

Unitl java 8, communicating with processor/os/machine is very difficult. We required to write very complex native code and we have to use 3rd party jar files.

The way of communication with processor is varied from system to system (i.e. os to os). For example, in windows one way, but in Mac other way. Being a programmer we have to write code based on operating system, which makes programming very complex.

To resolve this complexity, JDK 9 engineers introduced several enhancements to Process API. By using this Updated API, we can write java code to communicate with any processor very easily. According to worldwide Java Developers, Process API Updates is the number 1 feature in Java 9.

With this Enhanced API, we can perform the following activities very easily.

1. Get the Process ID (PID) of running process.
2. Create a new process
3. Destroy already running process
4. Get the process handles for processes
5. Get the parent and child processes of running process
6. Get the process information like owner, children,...
etc...

What's New in Java 9 Process API:

1. Added several new methods (like pid(),info() etc) to Process class.

2. Added several new methods (like startPipeline()) to ProcessBuilder class. We can use ProcessBuilder class to create operating system processes.

3. Introduced a new powerful interface ProcessHandle. With this interface, we can access current running process, we can access parent and child processes of a particular process etc

4. Introduced a new interface ProcessHandle.Info, by using this we can get complete information of a particular process.

Note: All these classes and interfaces are part of java.lang package and hence we are not required to use any import statement.

How to get ProcessHandle object:

It is the most powerful and useful interface introduced in java 9.
We can get ProcessHandle object as follows

1. ProcessHandle handle=ProcessHandle.current();
   Returns the ProcessHandle of current running Process

2. ProcessHandle handle=p.toHandle();
   Returns the ProcessHandle of specified Process object.

3. Optional<ProcessHandle> handle=ProcessHandle.of(PID);
   Returns the ProcessHandle of proccess with the specified pid.
   Here, the return type is Optional, because PID may exist or may not exist.

Use Case-1: To get the process ID (PID) of current process

```
1)  public class Test
2)  {
3)     public static void main(String[] args) throws Exception
4)     {
5)        ProcessHandle p=ProcessHandle.current();
6)        long pid=p.pid();
7)        System.out.println("The PID of current running JVM instance :"+pid);
8)        Thread.sleep(100000);
9)     }
10) }
```

We can see this process id in Task Manager (alt+ctrl+delete in windows)

ProcessHandle.Info:

We can get complete information of a particular process by using ProcessHandle.Info object.
We can get this Info object as follows.

```
ProcessHandle p = ProcessHandle.current();
ProcessHandle.Info info = p.info();
```

Once we got Info object, we can call the following methods on that object.

1. user():
Return the user of the process.
public Optional<String> user()

2. command():
Returns the command,that can be used to start the process.
public Optional<String> command()

3. startInstant():

Returns the start time of the process.

public Optional<String> startInstant()

4. totalCpuDuration():

Returns the total cputime accumulated of the process.

public Optional<String> totalCpuDuration()

Use Case-2: To get snapshot of the current running process info

```
1)  public class Test
2)  {
3)     public static void main(String[] args) throws Exception
4)     {
5)        ProcessHandle p=ProcessHandle.current();
6)        ProcessHandle.Info info=p.info();
7)        System.out.println("Complete Process Inforamtion:\n"+info);
8)        System.out.println("User: "+info.user().get());
9)        System.out.println("Command: "+info.command().get());
10)       System.out.println("Start Time: "+info.startInstant().get());
11)       System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration().get());
12)    }
13) }
```

Use Case-3: To get snapshot of the Particular Process Based on id

```
1)  import java.util.*;
2)  public class Test
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Optional<ProcessHandle> opt=ProcessHandle.of(7532);
7)        ProcessHandle p=opt.get();
8)        ProcessHandle.Info info=p.info();
9)        System.out.println("Complete Process Inforamtion:\n"+info);
10)       System.out.println("User: "+info.user().get());
11)       System.out.println("Command: "+info.command().get());
12)       System.out.println("Start Time: "+info.startInstant().get());
13)       System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration().get());
14)    }
15) }
```

ProcessBuilder:

We can use ProcessBuilder to create processes.
We can create ProcessBuilder object by using the following constructor.

```
public ProcessBuilder(String... command)
```

The argument should be valid command to invoke the process.

Eg:

```
ProcessBuilder pb = new ProcessBuilder("javac","Test.java");
ProcessBuilder pb = new ProcessBuilder("java","Test");
ProcessBuilder pb = new ProcessBuilder("notepad.exe","D:\\names.txt");
```

Once we create a ProcessBuilder object,we can start the process by using start() method.

```
pb.start();
```

Use Case-4: To create and Start a process by using ProcessBuilder

FrameDemo.java:

```
1)  import java.awt.*;
2)  import java.awt.event.*;
3)  public class FrameDemo
4)  {
5)     public static void main(String[] args)
6)     {
7)        Frame f = new Frame();
8)        f.addWindowListener( new WindowAdapter()
9)        {
10)          public void windowClosing(WindowEvent e)
11)          {
12)                   System.exit(0);
13)          }
14)       });
15)       f.add(new Label("This Process Started from Java by using ProcessBuilder !!!"));
16)       f.setSize(500,500);
17)       f.setVisible(true);
18)    }
19) }
```

Test.java

```
1)  public class Test
2)  {
3)     public static void main(String[] args) throws Exception
4)     {
5)        ProcessBuilder pb=new ProcessBuilder("java","FrameDemo");
6)        pb.start();
7)     }
8)  }
```

Use Case-5: To open a file with notepad from java by using ProcessBuilder

```
1)  public class Test
2)  {
3)     public static void main(String[] args) throws Exception
4)     {
5)        new ProcessBuilder("notepad.exe","FrameDemo.java").start();
6)     }
7)  }
```

Use Case-6: To start and destroy a process from java by using ProcessBuilder

```
1)  class Test
2)  {
3)     public static void main(String[] args) throws Exception
4)     {
5)        ProcessBuilder pb=new ProcessBuilder("java","FrameDemo");
6)        Process p=pb.start();
7)        System.out.println("Process Started with id:"+p.pid());
8)        Thread.sleep(10000);
9)        System.out.println("Destroying the process with id:"+p.pid());
10)       p.destroy();
11)    }
12) }
```

Use Case-7: To destroy a process which is not created from Java

```
1)  import java.util.*;
2)  class Test
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Optional<ProcessHandle> optph=ProcessHandle.of(5232);
7)        ProcessHandle ph=optph.get();
8)        ph.destroy();
9)     }
10) }
```

Use Case-8: To display all running process information

```
1)  import java.util.*;
2)  import java.util.stream.*;
3)  import java.time.*;
4)  class Test
5)  {
6)     public static void dumpProcessInfo(ProcessHandle p)
7)     {
8)        ProcessHandle.Info info=p.info();
9)        System.out.println("Process Id:"+p.pid());
10)       System.out.println("User: "+info.user().orElse(""));
11)       System.out.println("Command: "+info.command().orElse(""));
12)       System.out.println("Start Time: "+info.startInstant().orElse(Instant.now()).toString());
13)       System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration().
14)                                                   orElse(Duration.ofMillis(0)).toMillis());
15)       System.out.println();
16)    }
17)    public static void main(String[] args) throws Exception
18)    {
19)       Stream<ProcessHandle> allp=ProcessHandle.allProcesses();
20)       allp.limit(100).forEach(ph->dumpProcessInfo(ph));
21)    }
22) }
```

Use Case-9: To display all child process information

```
1)  import java.util.stream.*;
2)  import java.time.*;
3)  class Test
4)  {
5)     public static void dumpProcessInfo(ProcessHandle p)
6)     {
7)        ProcessHandle.Info info=p.info();
8)        System.out.println("Process Id:"+p.pid());
9)        System.out.println("User: "+info.user().orElse(""));
10)       System.out.println("Command: "+info.command().orElse(""));
11)       System.out.println("Start Time: "+info.startInstant().orElse(Instant.now()).toString());
12)       System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration()
13)                                                   .orElse(Duration.ofMillis(0)).toMillis());
14)       System.out.println();
15)    }
16)    public static void main(String[] args) throws Exception
17)    {
18)       ProcessHandle handle=ProcessHandle.current();
19)       Stream<ProcessHandle> childp=handle.children();
20)       childp.forEach(ph->dumpProcessInfo(ph));
21)    }
```

```
22) }
```

Note: If Current Process not having any child processes then we won't get any output

Use Case-10: To perform a task at the time of Process Termination

```
1)  import java.util.concurrent.*;
2)  public class Test
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        ProcessBuilder pb=new ProcessBuilder("java","FrameDemo");
7)        Process p=pb.start();
8)        System.out.println("Process Started with id:"+p.pid());
9)        CompletableFuture<Process> future=p.onExit();
10)       future.thenAccept(p1->System.out.println("Process Terminated with Id:"+p1.pid()));
11)       System.out.println(future.get());
12)    }
13) }
```

Output [In Normal Termination]:

```
D:\durga_classes>java Test
Process Started with id:4828
Process[pid=4828, exitValue=0]
Process Terminated with Id:4828
```

Output [In Abnormal Termination alt+ctrl+delete]:

```
D:\durga_classes>java Test
Process Started with id:12512
Process[pid=12512, exitValue=1]
Process Terminated with Id:12512
```

Observe exit value: 0 for normal termination and non-zero for abnormal termination