**1Z0-816 - Java SE 11 Programmer II: Core Java Libraries**

In this course, explore the features, libraries, and process application programming interfaces (APIs) introduced in JDK 11. Discover how to work with Java serialization filters, the JDEPRScan tool, and streams, along with several other topics. Begin this 11-video course by taking a look at the new features and libraries introduced in JDK 11, and also the core Java libraries. You will then look into working with Java serialization filters, the object of which is to control how to allow incoming streams of object serialized data to be filtered. This leads learners into observing how to create pattern-based filters and custom filters and discovering how to work with JDEPRScan. Next, you will delve into creating unmodifiable collections including lists, sets, and maps in Java 11, and then recall the essential process API capabilities in Java 11. You will examine the approach to creating processes in Java 11 and retrieving process information in Java 11. In the final tutorial in this course you will explore filtering processes in Java 11, including filtering processes with streams. This is one course is a series to prepare for the 1Z0-816: Java SE 11 Programmer II certification exam.

**Course Overview**

[Video description begins] *Topic title: Course Overview. The presenter is Niranjan Pandey. He is a Mentor and Big Data Solution Architect.* [Video description ends]

Hi, my name is Niranjan Pandey, and I'm part of Alchemy Software Solutions eLearning and Training Content Development Team. I specialize in application development, system architecture, IT security, database design and big data solutions, virtualizations and cloud solutions, machine and deep learning solutions, automation and robotics, DevOps and IoT. In this course, I will explore the new features, libraries and process APIs that were introduced in JDK 11. Learn to work with Java serialization filters, JDEPR scan tools and streams. Create pattern-based filters and custom filters. I will also discuss how to create unmodifiable collections, like list, set, and maps. Create processes, retrieve process information, and work with streams to filter processes.

**JDK 11 Features and Libraries**

[Video description begins] *Topic title: JDK 11 Features and Libraries.* [Video description ends]

The objective of this video is to recognize the new features and libraries that are introduced in JDK 11. Apart from coming up with improvements at the lower end in terms of JVM, there are various different features which has been upgraded in Java 11. To start with, it provides capability of negating a predicate method reference. It also helps us to transform empty strings into empty optional, that helps us to manage empty context in much better way. Apart from that, it has also improvised access control by providing nest-based access control. And finally, Java 11 provides local variable syntax in order to bring in simplification to derive Lambda parameters when we write Lambda expressions. Now let's try to understand some of the essential feature upgrade which was made in order to ensure that your application is optimized, and it takes care of all the essential features which are required in enterprise applications. And out of which first is free application class-data sharing feature. In other words, they came up with concept of providing you CDS in order to ensure that applications are able to share data in much better way. They also removed overhead of flight recorder and heap profiler. Apart from that, they provided heap allocation on alternative memory devices, that provides you extensibility of memory usage to application developers. From perspective of security, they also provided new default set of root authority certificates. And in order to provide better garbage collection for developers when they are writing their application and they're planning to test, they came up with new garbage collectors. Two garbage collectors which are added in Java 11 are ZGC and Epsilon. Further, in order to ensure that it provides better compilation, it came up with concept of ahead-of-time compilation and GraalVM. And finally, in order to help us to implement and adopt Transport Layer Security 1.3, they started support to TLS 1.3 as well. Now, let's try to understand some of the additional libraries and enhancements which was made in Java 11. First enhancement was they came up with new native unmodifiable collection API. They added new Reactive Stream API in order to improve stream predicate on optional API. And they also Improvised Process API by providing simplification to the API set. Apart from that, they also enhanced the way you work with File API. Started support for HTTP/2 protocol, provided Standard Java Async HTTP client which was missing in previous versions. And they also enhanced multi-release JARs. Now let's talk about some of the JVM enhancements which was made in Java 11. And first is, they focused on low-overhead, and this feature can be enabled by default continuously. Second, they also provided accessibility by using well-defined programmatic interface. Third, now you can sample all the allocations before finally deciding. Apart from that, they also enhanced the way JVM was implemented by providing capability of defining an implementation in independent way. And apart from that, they also improvised the way information about Java objects are captured by providing capability of capturing information for live Java objects and dead Java objects.

**Core Java Libraries**

[Video description begins] *Topic title: Core Java Libraries. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to recognize the essential core Java libraries which are provided in JDK 11. As a Java developer, I need to work with various different components. Some of the components drives the language. Some of the components provides us utilities. Some of the components will help us to work with input and output. Some components are there that helps us to write applications which are network based. And yes, finally, we need to also take care of security. In order to help us in taking care of various different facets of applications, Java has provided certain packages.

[Video description begins] *Core Java Packages and Internal Libraries.* [Video description ends]

Java.lang is one of them, it is imported by default once you write your Java program. Apart from that it provides Java.util which provides various different utility, classes, and interfaces to improvise our application. Then we have Java.io which facilitates us to write applications where input and output are required, where input source and output source can be files, or it can even be various different input and output sources. Finally, in order to help us in writing network-based applications, Java provides Java.net. And in order to help us to write applications to secure our applications we need Java.security. Let's go ahead and try to understand what all are the important responsibilities which are taken care by these packages. Let's start with Java.lang. It contains classes and interfaces of core Java language. It provides two important classes, first is the object class and second is the class class itself. Apart from that Java.lang also provides us capability of representing and identifying features of classes at run time. Java.util is an important package, it provides generic Java utilities, help us to work with collection framework. It also provides us classes, interfaces, and methods in order to generate formatted output for printing, scanning, and these are present in Java.util. In order to work with arrays it provides us array manipulation utilities. In order to help us in writing event driven applications, it provides us event model. In order to work with date and time, it provides us date and time facilities. And finally, in order to write applications from perspective of internationalization, it provides us classes that helps us to internationalize our applications. Now let's try to understand what is role of Java.io package. It provides us classes that helps us to read input from data stream, write output to data stream. Also helps us to manipulate files which are present on local system, and facilitates in object serialization and de-serialization. Now, in order to help us in writing network-based applications, Java.net provides various different classes and interfaces. It can be divided into two different sections. First is the low-level API that provides InetAddress class that communicates directly with the low-level implementation of network. And in order to make them available in our application at high level, it provides us classes like URL, URI, and URL Connection. And finally, Java.security. Whenever you want to encrypt files with provided keys, you will go ahead and you'll use Java.security package. It supports implementation of cryptography, as well as digital signatures in order to secure the resources.

**Java Serialization Filters**

[Video description begins] *Topic title: Java Serialization Filters. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to demonstrate how to work with Java serialization filters. Primary objective of working with Java serialization filter is to ensure that we are able to control how to allow incoming streams of object serialized data to be filtered. In order to improve security and also get system robustness. In order to work with serialization and provide a flexible mechanism to narrow the classes that can be deserialized from any class available to an application down to the context appropriate set of classes. We can go and we can set the serialization filter process. It can also be used in order to provide metrics to the filter for graph size and complexity during deserilization to validate normal graph behavior. Finally, you can also utilize capability of identifying a mechanism for RMI exported objects to validate the classes expected in the invocation. In short, the filter mechanism must not require subclassing or modification to existing subclasses, specifically when we talk about object input stream.

[Video description begins] *He opens the Program Files folder, which includes a Java folder.* [Video description ends]

Let's go ahead and see where we will be doing the settings. In order to set serialization filter for your processes, you have to go and you have to open your jdk directory, which is present in Program Files Java in case of normal installation.

[Video description begins] *The Java folder contains a jdk-11.0.4 folder.* [Video description ends]

Then go ahead and open the directory of jdk as well, and you will get a folder called as conf.

[Video description begins] *He opens the jdk-11.0.4 folder which includes a conf folder. He opens the conf folder which further includes management and security folders.* [Video description ends]

Double click and open conf, you will get net.properties, logging.properties, sound.properties. Your serialization is present under security in java.security file.

[Video description begins] *He opens the security folder which includes a java.security file.* [Video description ends]

You'll open it by double-clicking on it and selecting the right editor.

[Video description begins] *He opens the java.security file using Notepad application.* [Video description ends]

We have selected Notepad, and in Notepad, we went to Serialization process-wide filter. Now you can go and you can do the configuration, and that can be done by specifying the right pattern and the system property indicated by jdk.serialFilter. To do that, you need to ensure that your pattern includes it, = and it also sets a limit. For example, if you want to set maxdepth, maxrefs, maxbytes, maxarrays. In order to control depth of the graph, internal references, number of bytes in the input stream, and the maximum length allowed, you have to uncomment it and you have to change the value. Once you will change the value and launch your JRE, it will ensure that it follows the pattern that you have specified in order to control the limits. Similarly, if you want to apply pattern specifically when it matches the class or the package name and outcome of method whenever it is called like Class.getName. You can go and you can identify the pattern. And using the documentation which we are going through you can set that particular pattern. One of the most important pattern that helps you to manage serial filter is jdk.serialFilter. We can clearly see in the file it is commented and it is equal to pattern;pattern.

[Video description begins] *He points to the following code line: jdk.serialFilter=pattern;pattern.* [Video description ends]

This pattern can be replaced by some actual pattern of your choice, and you can uncomment it. For example, in case if you want to ensure that it matches all the classes. In order to ensure that your serial filter indicates match to every class in a particular package. You can go and you can specify packageName, followed by dot. And then you have to specify \* that indicates that it will apply the serial filter on all the classes which are currently present in the package.

[Video description begins] *He alters the aforementioned code line to as follows: jdk.serialFilter=packageName.\*;pattern.* [Video description ends]

Apart from that, you do get certain inbuilt serial filters. One of them is RMI Registry Serial Filter. You can go and you can set. Some of them are maxarray, maxdepth, String, Number, Proxy, Remote, UnicastRef. You can go ahead and you can uncomment it, in order to set the registry filter for your RMI. So depending on your need, you can go and you can edit java.security file. After editing, save it. After saving, whenever you will invoke JRE, you'll find that that pattern is applied.

**Pattern-based and Custom Filters**

[Video description begins] *Topic title: Pattern-based and Custom Filters. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to create pattern based and custom filters. In order to specify pattern based filters while executing your class, you can go ahead and you can send the class on which you want to define a pattern based filter on the current application. We can define it by using system property for the current application. A system property will always supersede with the values that you have defined in security property under con folder of your JDK. In order to create filter that only applies to the current application and only a single invocation of Java, we need to define the jdk.serialFilter system property in the command line. If you are using Eclipse, in order to define it, you need to select the class under Package Explorer.

[Video description begins] *He opens the java - Eclipse IDE window. It is divided into five parts. The first part contains a menu bar and a toolbar. The menu bar includes File, Edit, and Source menus. The toolbar includes Cut, Copy, Paste, and Run as icons. The second part is a Package Explorer pane which contains an it\_sdjsep\_06\_enus\_05 folder which further includes a src subfolder. The src subfolder includes a it\_sdjsep\_06\_enus\_05 subfolder. The it\_sdjsep\_06\_enus\_05 subfolder contains a CustomFilter.java file. The third part is a content pane. It doesn't contain any code. The fourth part contains two tabs: Tasks List and Outline. The fifth part includes Console and Declaration tabs and clear and comment icons. The Console tab is selected. Then he right-clicks on the CustomFilter.java file in the navigation pane.* [Video description ends]

Right-click on it, select Run As from the context menu and click on Run Configuration. Once you'll click on Run Configuration, it will launch a dialog box.

[Video description begins] *The Run Configuration dialog box opens. It is divided into three parts. The first part is a navigation pane, the second part is a content pane, and the third part contains Run and Close buttons. The navigation pane includes Java Application option which further includes ProcessTest(1) suboption. The ProcessTest(1) suboption is selected. The content pane includes Main and Arguments tabs and Show Command Line, Revert, and Apply buttons.* [Video description ends]

Go to the second tab which is immediately after Main called Arguments. And under VM arguments, specify the following pattern.

[Video description begins] *The Arguments tab includes a VM arguments text box linked with a Variables button.* [Video description ends]

In order to specify pattern, you'll be writing -Djdk, followed by .serialFilter. After specifying jdk.serialFilter, you have to use equal to followed by what filter you want to apply. For example, assume that we want to apply maxarray, so we'll specify maxarray=100, because we want limit to be 100. Apart from that you can also define other filters by using semicolon. We'll use semicolon, we'll specify maxdepth, and we'll set it to 10. After specifying we can click on Show Command Line.

[Video description begins] *The Apply Changes dialog box opens.* [Video description ends]

You'll find that it will ask you to change, you will click Apply, and it will display you the Command Line.

[Video description begins] *The Apply Changes dialog box closes and the Command Line dialog box opens.* [Video description ends]

Command Line clearly indicates that it invokes javaw.exe. And it applies jdk.serialFilters setting up max array to 100 and max depth to 10. Apart from that you can use other elements as well on the Command Line.

[Video description begins] *He closes the Command Line dialog box.* [Video description ends]

You can close it and you can click on run in order to run the program on which the current limits will be set.

[Video description begins] *The Run Configuration dialog box closes.[Video description ends]*

*Now, we can clearly see that it shows Java class not found.*

*[Video description begins] He points to the error displayed in the Console tab.* [Video description ends]

It is because in the main tab, we haven't selected the right class, so we'll select the class again.

[Video description begins] *He right-clicks on the CustomFilter.java file in the Package Explorer pane. A flyout opens.* [Video description ends]

We'll go to Run As menu option. Click on Run Configuration, and we'll ensure that it selects the right class for main.

[Video description begins] *The Run Configuration dialog box opens. The Main tab is selected in the content pane. It includes a Main class text box which is linked with a Search button. The text box contains it\_sdjsep\_06\_enus\_05.ProcessTest text value. He clicks the Search button. The Select Main Type dialog box opens. It includes a Select type (? = any character, \* = any String, TZ = TimeZone): drop-down list box. The default \*\* symbols are displayed in the drop-down list box. He clicks the Cancel button and the Select Main Type dialog box closes.* [Video description ends]

As of now we don't have ProcessTest, and we don't have any class with main function. So we'll close it, and we'll clear the console.

[Video description begins] *He closes the Run Configuration dialog box.* [Video description ends]

But yes, this is how you will apply this limit on a single application. Now, let's go ahead and try to understand how to define CustomFilter. In order to define CustomFilter, we can specify it in application code as well. And they are set on individual stream or on all stream in the current process. In order to do that, we can create a custom filter, apply the pattern, method, or a lambda expression. Apart from lambda expression, we can also implement custom filter as a pattern on a class. We have already written a class called CustomFilter. We'll double click on it, and we'll try to understand what it does.

[Video description begins] *The CustomFilter.java opens in the content pane.* [Video description ends]

In line number 1, we have specified the package to which this class should belong. Line number 3 is importing ObjectInputFilter, which is present in java.io.

[Video description begins] *He points to the following code line: package it\_sdjsep\_06\_enus\_05;.[Video description ends]*

*Now, we are going to apply CustomFilter on the current ObjectInputFilter.*

*[Video description begins] He points to the following code line: import java.io.ObjectInputFilter;.* [Video description ends]

To do that, we have defined a method name dateTimeFilter in line number 6, which returns ObjectInputFilter.Status, and method is of type static.

[Video description begins] *He points to the following code line, code starts: static ObjectInputFilter.Status dateTimeFilter(ObjectInputFilter.FilterInfo info) {. Code ends.* [Video description ends]

To the current method, we are passing ObjectInputFilter.FilterInfo info. And in line number 7, we are getting information of the serialClass and we are populating serialClass with that.

[Video description begins] *He points to the following code line: Class<?> serial = info.serialClass();.* [Video description ends]

Finally, in line number 8, we are checking the condition we have serial is not equal to null.

[Video description begins] *He points to the following code line: if (serial != null) {.* [Video description ends]

If it is not equal to null, you'll get the package name, which would be equal to java.net.

[Video description begins] *He points to the following code line: return serial.getPackageName().equals("java.net").* [Video description ends]

Instead of using java.net you can pass your own packages on which you want this filter to work. If it returns true, it indicates that ObjectInputFilter.Status.ALLOWED else it will not allow or it will reject. Which is indicated after colon, which is the false part by specifying ObjectInputFilter.Status.REJECTED in line number 11. And finally, in line number 13, we are returning ObjectInputFilter.Status.UNDECIDED.

[Video description begins] *He points to the following code line: return ObjectInputFilter.Status.UNDECIDED;.* [Video description ends]

This is the way how you will be creating the filter. After creating the filter, you can apply the filter wherever you are using object input stream.

**jdeprscan tool**

[Video description begins] *Topic title: jdeprscan tool. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to demonstrate how to work with JDEPRScan tool. One of the important aspect of ensuring that programs are quality programs is to ensure that we are doing static analysis. JDEPRScan is one of the static analysis tools that scan a JAR file or a class and finds out what all APIs are deprecated. In order to utilize JDEPRScan, we can go and we can use the command called JDEPRScan, followed by various options and we can pass directory, a JAR file or a class. Responsibility of JDEPRScan command is to scan every argument and check whether APIs are deprecated. For example, assume that we have a JAR file which may contain various different classes. Now you want to find out what all part of the current APIs which are there in JAR file are deprecated. You can go ahead and you can utilize various different options in order to ensure that you are able to test it with different release of Java SE. Let's go ahead and try it out.

[Video description begins] *The Select Administrator: Command Prompt window is open in which the following prompt is displayed: D:\javasamples>.* [Video description ends]

In other to utilize the command, you have to go to the right folder where you have your JAR files or class files. After moving to that particular folder in your command prompt, you can go and you can type the command. After typing the command called jdeprscan, you have to go and you have to select the release for which you are testing. For example, let's go ahead and test for version 6 and then we'll specify the JAR file. Current JAR file that we are analyzing as Javassist version is 3.12.0.GA.jar we'll press Enter. Once you'll press Enter, as of now you can ignore the warning.

[Video description begins] *He executes the following command: jdeprscan --release 6 javassist-3.12.0.GA.jar. A warning displays in the output and the same prompt displays.* [Video description ends]

You'll find that there are no messages, it means that there are no APIs which are present in javassit 3.12.0.GA.jar which is deprecated from perspective of version 6. Now let's go ahead and use the same command with version 7. We need to just go and change 6 to 7 in the existing command and press Enter. Once you'll press Enter it will check again and you'll find that even in version 7, no APIs of javassist 3.12.0.GA.jar is deprecated. Now let's go ahead and increase the version to 9. Once we'll increase the version to 9 by changing 7 to 9 and press Enter, it will go and it will find out and you will get a huge list of classes which are deprecated. In other words, it helps you to identify what all APIs are there which are using some or other deprecated methods. Now let's go ahead and clear the screen by writing cls, and write custom class file. You can use any editor in order to write a custom Java file. For example, we have written Sample.java.

[Video description begins] *He executes the following command: type Sample.java. The output displays the following code: import java.rmi.RMISecurityManager; public class sample { SecurityManager sm = new RMISecurityManager(); Boolean b2 = new Boolean(true); }.* [Video description ends]

First statement in Sample.java is import java.rmi.RMISecurityManager. Then we are using public class Sample, where sample is name of the class. We are creating an instance of RMISecurityManager and we are initializing object of SecurityManager which is indicated by sm. Next, we are creating an instance of boolean by writing Boolean b2 equal to new Boolean and we are passing true. In other words, we are creating an object of type boolean. Now let's go and use the command in order to find out are we using some deprecated methods or APIs in our current class. In order to do that, we'll be using jdeprscan followed by class path, we'll specify class hyphen path, we'll specify dot because we want it to start its search from the current path, give a space specify the release. Let's go ahead and specify release as release 6 and then we'll specify the name of the class that is sample. We'll press Enter. Once you press Enter you'll find warning without any messages.

[Video description begins] *He executes the following command: jdeprscan --class-path . --release 6 sample.* [Video description ends]

Let's go ahead and change it to 11 and press Enter. Once you will change it to release 11 you'll find that it shows that there are certain classes and methods which are deprecated.

[Video description begins] *He executes the following command: jdeprscan --class-path . --release 11 sample.* [Video description ends]

In short, if you want to check whether you are using the right API without any deprecated classes and methods, you can use JDEPRScan tool provided to you by Java.

**Working with Unmodifiable Collections**

[Video description begins] *Topic title: Working with Unmodifiable Collections. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to create unmodifiable collections including list, sets, and maps. Java collection provides us privilege of working with variety of data that can be stored in collections. You'll have one variable, using the same variable you can reference data of various types. There are various use cases when you don't want your data to be modified. And in order to avoid accidental modification of data you need to ensure that your collections themselves are unmodifiable. There are certain scenarios in which unmodifiable collection is recommended. For example, sometimes you would like to have collections that are initialized from constants that are known when the program is written. Second scenario is sometimes collections that are initialized at the beginning of the program from the data which was computed or which it has read, need not be changed. In those scenarios, it is advisable to go with unmodifiable collections. Now, let's go ahead and see how we will be working with unmodifiable collections.

[Video description begins] *He opens the java - Eclipse IDE.* [Video description ends]

To do that we'll be creating a project, we'll give a name to the project.

[Video description begins] *He points to the it\_sdjsep\_06\_enus\_07 root folder displayed in the Package Explorer pane. The it\_sdjsep\_06\_enus\_07 root folder includes a src folder.* [Video description ends]

As soon as you will create the project in Eclipse it will provide you that project in Package Explorer. You can go to Package Explorer, select src, right-click, select New from the context menu and then select Class.

[Video description begins] *He right-clicks on the src folder. A flyout opens including New and Show in options. He selects the New option and another flyout opens including Package and Class options. He clicks the Class option and the New Java Class dialog box opens.* [Video description ends]

Specify name of the class. For example, we are naming the class as TestCollection. And we'll select public static void main in order to ensure that we are able to execute the program. And we'll click on Finish. Now, we have to see how actually unmodifiable collection has been improvised in various different versions of Java.

[Video description begins] *The TestCollection.java tab opens in the content pane. It displays several code lines, code starts: package it\_sdjsep\_06\_enus\_07; public class TestCollection { public static void main(String[] args) { } }. Code ends.* [Video description ends]

For example, let's declare a list, which can take up string.

[Video description begins] *The following code line adds below the package it\_sdjsep\_06\_enus\_07; code line: import java.util.List;.* [Video description ends]

So we'll declare list with string. And we'll say dataCollection= will take Arrays., and we'll generate a list.

[Video description begins] *The following code line adds below the package it\_sdjsep\_06\_enus\_07; code line: import java.util.Arrays;.* [Video description ends]

We'll pass various different strings in order to generate list. For example, test1, test2, let's take three values, test3. Now, we have a collection of string which is populated in data collection and type of collection that we are using is list. But unfortunately, current list collection is modifiable.

[Video description begins] *He adds the following code line below the public static void main (String[] args) { code line, code starts: List<String> dataCollection=Arrays.asList("test1", "test2", "test3");. Code ends.* [Video description ends]

We want to make the current list as unmodifiable. In Java 8, there was a method that was part of collections, which took responsibility of making list instances unmodifiable. Let's go ahead and use it. So in line number 13, that is immediately after Arrays.asList, we'll be writing dataCollection=, we'll utilize Collections class.

[Video description begins] *The following code line adds below the import java.util.Arrays; code line: import java.util.Collections;.* [Video description ends]

And we'll make call to a method called as unmodifiableList, and we'll pass the list itself. So now we have applied a function that will ensure that current list is not modifiable. This was the approach which was used in Java 8.

[Video description begins] *He adds the following code line, code starts: dataCollection=Collections.unmodifiableList(dataCollection);. Code ends. He adds the aforementioned code line below the following code line, code starts: List<String> dataCollection=Arrays.asList("test1", "test2", "test3");. Code ends.* [Video description ends]

We'll save the program, we'll maximize by double clicking on the title bar of the current file. Now, let's go ahead and see how this was changed in recent versions of Java. To do that, we'll comment the previous code from line number 12 to line number 14. We'll select and we'll press Ctrl+?, in order to comment it.

[Video description begins] *He comments out the following code lines, code starts: List<String> dataCollection=Arrays.asList("test1", "test2", "test3"); dataCollection=Collections.unmodifiableList(dataCollection);. Code ends.* [Video description ends]

Now, let's go ahead and see how we'll be doing it in recent versions of Java, including Java 11. We'll declare String, we'll name it as dataCollection. And instead of following two different steps in order to create unmodifiable collection, we'll be using one of the functions, which is called as factory function provided by List itself. We'll say List and we'll use .of. Within of, we have to pass the data, which will be the collection. So instead of using Array.asList, we are using List.of, we are able to create an unmodifiable list.

[Video description begins] *He adds the following code line in the public static void main(String[] args) method, code starts: List<String> dataCollection=List.of("test1", "test2", "test3");. Code ends.* [Video description ends]

Now, let's follow the same mechanism in order to create an unmodifiable set. We will be making some changes. For example, now we'll declare a Set. It will say Set will contain string, and we'll specify it as dataCollection2=, we'll take Set.

[Video description begins] *The following code line adds below the import java.util.List; code line: import java.util.Set;.* [Video description ends]

And Set also provides a method where name of the method is .of. We'll select .of and we can pass the data which we want to be unmodifiable once this collection is created. So now, we have created a set, which is again unmodifiable. Finally, we will look into how actually we can go and create an unmodifiable map.

[Video description begins] *He adds the following code line, code starts: Set<String> dataCollection2=Set.of("test1", "test2", "test3");. Code ends. He adds the aforementioned code line below the following code line, code starts: List<String> dataCollection=List.of("test1", "test2", "test3");. Code ends.* [Video description ends]

Map is considered as collection of key value pair. Now, our objective is to define key and value. To do that we'll be using Map. We'll specify key to be Integer and value to be String.

[Video description begins] *The following code line adds below the import java.util.List; code line: import java.util.Map;.* [Video description ends]

We'll name it as mapCollection. And next we have to go and we have to add entries to the map. While adding entries to the map, we need to ensure that it is unmodifiable, and to do that we'll use Map. After specifying map we'll use of function. We'll specify of, and then in sequence we will be specifying the value. For example, first is key, so we'll specify 1, "data1", and 2, we'll specify "data2". Depending on number of key and value that you are aware of you will go and you will specify.

[Video description begins] *He adds the following code line, code starts: Map<Integer, String> mapCollection = Map.of(1, "data1", 2, "data2");. Code ends. He adds the aforementioned code line below the following code line, code starts: Set<String> dataCollection2=Set.of("test1", "test2", "test3");. Code ends.* [Video description ends]

Ensure that sequence is maintained, if you have a specified key to be integer, first value should be integer. If data is string, second value need to be string. Once you have applied and use of factory method of List, Set, and Map it will ensure that data is unmodifiable.

**Process API Capabilities**

[Video description begins] *Topic title: Process API Capabilities. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to recall the essential capabilities of process APIs in Java. In order to write robust applications, we need to identify capabilities of processes. Sometimes we need to invoke various different processes using Java application. Apart from that, we would also like to know information about existing processes which are running on the system. To facilitate us in doing that, Java provides Process API, where process class provides control of native running processes. You can also go ahead and start processes by using processbuilder.Start and runtime.Exec methods. And finally, process class provides us various interesting methods that helps us to interact with various processes. Now let's try to understand the entire life cycle of Process Management. We need to ensure that we have to divide it into two different parts. First is the strategy, analysis, design. Second is implementation, monitoring, process control, and optimization. Process starts with building a right strategy in order to identify, create a process. We need to also analyze relevance of the process, then we need to design. After we have designed, we need to implement processes by using process APIs. We need to always use some monitoring tool in order to monitor the process. And whenever required, we need to have methods in order to intervene to control the process. And finally, we must have a strategy to optimize the processes as well.

[Video description begins] *Process Management Lifecycle.* [Video description ends]

Now let's try to understand some of the methods which are used in order to manage processes.

[Video description begins] *The Process Management Lifecycle is displayed. It starts with the Strategy, then Aanalysis, then Design, then implementation, then Monitoring and process control, then ends at Optimization. After Optimization, the whole process goes again.* [Video description ends]

We'll start up with children. Children method returns you a snapshot of all the processes which are direct child of the current process. Then descendants returns processes descendants. If you want to destroy a process in order to free up memory, you can use destroy. Sometimes some processes might be in use. If you feel like killing it forcefully you'll be using destroy forcibly. Apart from that, if you want to return process exit value, you can use exit value. To work with streams which can be of three different types, output stream, input stream, and error, you have three different methods. First is get error stream, that returns input stream connected to process's error output. Get input stream returns input stream connected to process's normal outputs. And get output stream returns output stream that is connected to the process's normal input. Now let's go ahead and try to understand some of the methods which helps us to manage the process.

[Video description begins] *Process Methods.* [Video description ends]

And first is the info. In order to understand what the process is all about, we can use info method. It returns information about the process. If you want to test whether a function is live or not, you can use isAlive. If you want to return a CompletableFuture to ensure that while termination, you're able to get an instance of CompletableFuture, you'll be using onExit. Another interesting method that is provided by process is supportsNormalTermination. It returns true if destroy normally terminates, but if you terminate it forcibly, it returns false. tohandle returns ProcessHandle for processes. waitFor is used in case if thread has to wait until the process represented by a particular process object does not terminates. You can also go and specify timeout and the unit of timeout by using waitFor, passing timeout and Timeunit.

[Video description begins] *Process API Sample Code. A sample code is displayed on the screen.* [Video description ends]

Now let's go ahead and try to understand how to manage processors by using a sample code. We have defined a class called demo in which we have written public static void main, passing string args, which throws exception. Then in order to get the state of the runtime we are using a method called Runtime.getRuntime. Once we get instance of runtime, we would like to see how many processors are available. To do that, we'll make call to a method called as availableProcessors using runtime object. And then we can also print what is the total memory, free memory, and what is the memory occupied by using a formula that is by subtracting free memory from total memory.

[Video description begins] *He points to the following code lines, code starts: public class Demo{ public static void main(String[] args) throws Exception { Runtime r=Runtime.getRuntime(); System.out.println("Total memory:"+r.totalMemory()); System.out.println("Free memory:"+r.freeMemory()); System.out.println("Memory occupied:"+ (r.totalMemory()-r.freeMemory()));. Code ends.* [Video description ends]

Next, we are creating multiple objects by starting a loop. In other words, we are creating 10,000 objects. After creating object we are making call to gc, that is garbage collector.

[Video description begins] *He points to the following code lines, code starts: for(int i=0; i<=1000; i++) { new Object(); } r.gc();. Code ends.* [Video description ends]

And then we are printing the memory statistics in order to see how garbage collector has reacted. And what is the change which has been brought in, in total memory, free memory, and memory occupied.

[Video description begins] *He points to the following code line, code starts: System.out.println(": : Memory status: :"); System.out.println("Total memory:"+r.totalMemory()); System.out.println("Free memory:"+r.freeMemory()); System.out.println("Memory occupied:"+ (r.totalMemory()-r.freeMemory. Code ends.* [Video description ends]

Now let's go through another sample code, which helps us to understand how we can do some useful task. We have again created a class called public class Demo. Then we are writing public static void main, and we are using System.out.println in order to organize what need to be printed. We are creating a process by using ProcessBuilder pb = new ProcessBuilder. And then we are passing what process we want to invoke, we want to invoke cal. We are passing data that you want to pass through the process, so cal will receive 2020. Finally, we have to start the process. For that we're using pb.start. Which gives handle to the process which we have started, and it is stored, and the reference of the process which is started is stored in p.

[Video description begins] *He points to the following code lines, code starts: public class Demo { public static void main(String[] args) { System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*Calendar for year\*\*\*\*\*\*\*\*\*\*\*\*); try{ ProcessBuilder pb=new ProcessBuilder("cal", "2020"); final process p=pb.start();. Code ends.* [Video description ends]

Next, we are creating BufferedReader instance, passing new input stream, and getting the input stream associated with the process by using p.getInputStream. We are declaring a line, and then we are starting a while loop in order to read line which is being generated by input stream of the process, and we are printing it.

[Video description begins] *He points to the following code lines, code starts: BufferedReader br=new BufferedReader( new InputStreamReader( p.getInputStream())); String line; while((line=br.readline())!"null){ System.out.println(line); }. Code ends.* [Video description ends]

Finally, we are handling the exception, and we are just printing System.out.print.ln(ex).

[Video description begins] *He points to the following code lines, code starts: catch (Exception ex) { System.out.println(ex); } System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); } }. Code ends.* [Video description ends]

Final System.out.prinln is just in order to ensure that we are able to separate output for multiple execution of the program.

**Working with Processes**

[Video description begins] *Topic title: Working with Processes. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to demonstrate the approach of creating a process and retrieving information about process. Primary objective of providing process API in Java is to help you creating a process, retrieving information about the process. It also helps you to filter running processes, redirect output. Apart from that, you can also run a series of test, apply clean up leftover processes. In order to help us to build processes, and to get information about processes, there are various classes and interfaces that can be used. Out of which ProcessBuilder is one of the important one. ProcessBuilder provides you capability of building a process. Once you have a process object you can specify what processes you want to invoke. Finally, you can also go and you can get information about process by using ProcessHandle interface. Which comes with various static methods that helps you to get the information.

[Video description begins] *The it\_sdjsep\_06\_enus\_09 project is open in the Eclipse IDE window.* [Video description ends]

We have already created a project using Eclipse.

[Video description begins] *He points to the it\_sdjsep\_06\_enus\_09 root node in the Package Explorer pane. The root node includes a src folder which further contains a it\_sdjsep\_06\_enus\_09 package and hellodata.txt file. The package contains a ProcessTest.java file. In the content pane, the ProcessTest.java tab is open. It displays several code lines.* [Video description ends]

We have specified a name to the project and we have created an empty class called ProcessTest.java under the package. Apart from that, we have created a text file, which we will invoke launching Notepad in Windows.

[Video description begins] *He selects the hellodata.txt file in the Package Explorer pane and the hellodata.txt tab opens in the content pane. It displays a text: hello i am being invoked by process.* [Video description ends]

If you're working on Linux, you can launch appropriate script in form of a process. We'll go back to ProcessTest.java by double clicking in under package inside the package explorer, in the current file, we'll double click to maximize it.

[Video description begins] *He double-clicks the ProcessTest.java file in the Package Explorer pane and the ProcessTest.java tab selects in the content pane.* [Video description ends]

And we'll start writing our code from line number 9 in the current editor. Our first step will always be to create an instance of ProcessBuilder. In order to create instance of ProcessBuilder, we'll be using ProcessBuilder, we'll specify a variable name say builder, and we'll be using the right constructor. For example, we are using constructor where we have to specify the command that we want to invoke as part of the process. We will be invoking Notepad, so we'll write notepad.exe and then we'll specify the name of the file. For example, in our case, it is hellodata.

[Video description begins] *He adds the following code line, code starts: ProcessBuilder builder=new ProcessBuilder("notepad.exe", "hellodata");. Code ends. The aforementioned code line adds below the following code line, code starts: public static void main(String[] args) throws IOException{. Code ends.* [Video description ends]

After creating builder, our next objective is to start the process. To start the process we'll be using start function of builder, which will return an instance of type process. In line number 10, we'll go and we'll specify the entries. For example, we'll write Process process = builder which you have created in the previous step in line number 9 dot start.

[Video description begins] *He adds the following code line: Process process=builder.start();.* [Video description ends]

This should invoke the process. After process is invoked, you would like to get information about the current process. To do that, you can use ProcessHandle interface. We'll specify ProcessHandle dot and will take info, we'll initialize it with process.info. In other words, we're making call to process.info method in order to populate ProcessHandle.Info. We have to go and we have to specify a variable name to ProcessHandle.Info which is info itself.

[Video description begins] *He adds the following code line: ProcessHandle.Info info=process.info();.* [Video description ends]

Now, let's go ahead and print certain general information about the process. To do that, in line number 12, we'll write System.out.println and we'll specify process ID as an argument. For that we'll write process dot and we'll take dot pid. Pid returns the process ID of the current process.

[Video description begins] *He adds the following code line: System.out.println(process.Pid());.* [Video description ends]

Now let's go ahead and print general information, which will be populated in info variable in line number 11. We'll use that in System.out.println in line number 13 as an argument to our System.out.println.

[Video description begins] *He adds the following code line: System.out.println(info);.* [Video description ends]

We'll save it, after saving we'll run it. In order to run, we can go and click on toolbar's Run button or we can being the editor right-click, select Run As from the context menu. And once sub-menus are prompted you can click on Java Application. Now, as a result of which, we'll find that it has launched hellodata.txt in Notepad.

[Video description begins] *The hellodata.txt file opened in the Notepad application displays the following text: hello i am being invoked by process.* [Video description ends]

Apart from that, it has also printed the information where first line is all about the pid and second line is general information which includes user, what is the command.

[Video description begins] *He closes the hellodata.txt file opened in the Notepad application. Then he double-clicks on the Console tab and it opens in the content pane.* [Video description ends]

What is the start time and what is the total time which the current process took in order to complete the execution.

**Filter Processes**

[Video description begins] *Topic title: Filter Processes. The presenter is Niranjan Pandey.* [Video description ends]

Objective of this video is to demonstrate how to use streams in order to filter processes. There are different methods which are provided by process APIs in order to help you to utilize capabilities of streams. We are going to use all processes method of process handle interface that returns a stream of all the processes that are visible to the current process. We can also go and apply various different kind of filters that are applicable on process handle instance. And we do that in similar way as we apply filters on a collection.

[Video description begins] *He opens the it\_sdjsep\_06\_enus\_10 project in the Eclipse IDE.* [Video description ends]

Let's try to create a small program that helps us to understand the concept of applying filters on processes using streams. In order to do that, we have created a project which is visible under Package Explorer.

[Video description begins] *He points to the it\_sdjsep\_06\_enus\_10 project name in the Package Explorer pane. The it\_sdjsep\_06\_enus\_10 is a root name which further includes a src folder. The src folder includes a it\_sdjsep\_06\_enus\_10 package. The it\_sdjsep\_06\_enus\_10 package includes a ProcessTest.java file.* [Video description ends]

Under src, we have a package and inside that package we have written a program called ProcessTest. We have imported the required classes which we are going to use. All the declarations of import are made in line number 3 and 4.

[Video description begins] *He points to the following code lines, code starts: import java.io.IOException; import java.util.Optional;. Code ends.* [Video description ends]

In line number 7, we are creating a class by declaring public class, naming the class as ProcessTest. And then, we are writing public static void main, where we are making call to a another static method called applyFilter that we have written in line number 12. In line number 12, we have defined structure of a method called as applyFilter, which is of type static and returns void.

[Video description begins] *He points to the following code lines, code starts: public class ProcessTest { public static void main(String[] args) throws IOException { applyFilter(); } static void applyFilter(). Code ends.* [Video description ends]

Inside opening and closing braces of the applyFilter method in line number14 we are using ProcessHandle .current().info().user() to get the current user. And we are initializing it in a variable of type Optional, which takes a string, and the name of the variable is currUser.

[Video description begins] *He points to the following code lines, code starts: Optional<String> currUser=ProcessHandle.current().info().user();. Code ends.* [Video description ends]

In line number 15, we are getting the stream of all the processes. And that we are doing by writing ProcessHandle.allProcesses. It returns a stream of all the processes. Next we want to filter and get only those processes which are being executed by the current user. To do that, we are using a lambda expression in line number 16. By using the chain method, which is applied on all processes and name of the method is filter. So we have specified .filter, lambda expression specifies get p1 reference and get.info. It returns info using ProcessHandle instance.

[Video description begins] *He points to the following code line: .filter(p1->p1.info().* [Video description ends]

Next from the info, we are getting the user. And after getting the user we are using .equals in line number 18 in order to compare the user.

[Video description begins] *He points to the following code line: .equals(currUser)).* [Video description ends]

After comparison is done, in line number 19, we are iterating and we are making call to another method called as show process which is part of the current class itself. And that method itself is a static method. This is the reason why we are using class name::showProcess.

[Video description begins] *He points to the following code line: .forEach(ProcessTest: :showProcess);.* [Video description ends]

Show process is written in line number 22, which is again a static method. Return type is void and name of the method is showProcess which takes an object of type ProcessHandle, and finally, prints the complete information about the process.

[Video description begins] *He points to the following code lines, code starts: static void showProcess(ProcessHandle handle) { System.out.println(handle.info()); }. Code ends.* [Video description ends]

After writing applyFilter and the required methods, we are making call to apply filter in line number 10. Now let's go ahead and execute the program. In order to execute, we'll right-click in the editor, select Run As from the context menu and click on Java Application. Once you will run the program, in the Console window, you'll find that all the processes which are currently being executed by the current user are getting filtered. It is not displaying those processes which are invoked by other users. This is how you can apply power of streams in order to filter processes.

**Course Summary**

[Video description begins] *Topic title: Course Summary.* [Video description ends]

So in this course, we have examined the new library features and capabilities that were introduced in Java and how to work with core Java libraries. We did it by exploring the new features core Java libraries and process APIs that were introduced in Java 11. The use of Java serialization filters. How to create pattern based on custom filters? How to create unmodifiable collections like list, set and maps? And how to create processes and retrieve information regarding the processes that we created? In the next course, we will explore some of the advanced concepts of classes in Java 11 and learn how to manage exceptions in Java 11.