JAVA 9 Features

- By – Debu Paul

https://www.youtube.com/c/techtalkdebu

https://github.com/admindebu

https://www.linkedin.com/in/debu-paul/

Agenda

- Introduction
- Private Methods in Interface
- Stream API Improvements
- Try with resource Enhancement
- Jshell command promt
- Process API Updates
- Java Module(JPMS)
- Jlink(Java Linker)
- New API : HTTP/2 Client

Introduction

- ✓ Java 9 was introduced on September 21, 2017, by Oracle Corporation. It was a major release of the Java programming language and came with several new features and improvements, including the module system, JShell, and more.

- ✓ Oracle stopped providing free public updates, and technical support for Java 9, including patches and bug fixes, to the general public after this date. Java 9 reached its end of life on March 2018, six months after its release.

- ✓ However, if you have a commercial license with Oracle, you can still receive updates and support for Java 9 beyond its end of life date. It's also worth noting that newer versions of Java, such as Java 11, Java 17, and later versions, provide long-term support for several years after their release.

Private Methods In Interface

Private Methods in Interface: (We know By default all methods in Java Interface is public) Java 9 introduced private methods to interface. It allows interfaces to have private helper methods, reducing code duplication and increasing code reusability. (for same code writing in interface in several method (so, we can create common code in private and then use in side in interface)

Example & Use cases :

```
private static int generateRandomNumber() {
Random random = new Random();
return random.nextInt(50);
}
```

Stream API Improvements

Java 9 introduced a few new methods to the Stream API, such as **takeWhile** and **dropWhile** which allow developers to take or drop elements from a stream based on a specified condition. along with that **offNullable** in stream (it prevents Null Pointer Exception)
takeWhile > it will stop if condition is false for any value, it may or may not take all the values in the collections
dropWhile > it will start if condition is false for any value, it may or may not take all the values in the collections
offNullable > if object null then return empty instead of NullPointerException and if true, return that value

Example & Use cases :

```
Integer[] arr = {2,4,6,8,10};

// take While
List<Integer> takeWhileList = Stream.of(arr).takeWhile(i -> i
<6).collect(Collectors.toList());
System.out.println("takeWhile: " + takeWhileList);
```

Try with Resource Enhancements

Since java 9, resources can de declare out side try and which can be refer inside try with resource. (but from JAVA 7 to JAVA 8 out side try , will give compile time error)

Example & Use cases :

```
BufferedReader br = new BufferedReader(new FileReader("c:\\Debu
Paul\\\\JAVA Version Fetures\\\\JAVA 9\\\\\\helloWorld.txt"));

try (br) {
String line;
while ((line = br.readLine()) != null) {
System.out.println(line);
}
} catch (IOException e) {
e.printStackTrace();
}
```

Jshell :

An interactive shell that allows you to run Java code snippets without having to create a complete Java program. (Used for :writing small program directly command prompt without creating any classes. (but before JShell to write small program, we need to create java class then write the program)
jshell (command)   so, direct write > System.out.print("TechTalk Debu"); // 100 + 100 (you can focus on logic to test it)

Example & Use cases :

```
jshell> int[] arr = {1,2,3,4,5}
arr ==> int[5] { 1, 2, 3, 4, 5 }

jshell> for (int i : arr) System.out.print(i + " ");
1 2 3 4 5
```

Process API Updates :

before JAVA 9 communicate to connect with processor in OS level very difficult and lengthy code, we need to depend on third party api etc (as java is not best choice for system prog but since java 9 it's easy and don't need to rely on third party API and less code few codes.

Use cases :

- ✓ **Get PID of the Process**
- ✓ **Get Process to Handle**
- ✓ **Get Process Information**
- ✓ **Create Process**
- ✓ **Destroy Process**
- ✓ **Wait for process terminate**

JPMS:

JPMS (Java platform Module system) - Java 9 introduced the module system to encapsulate the code and resolve dependency issues. It provides a modularized environment for building large-scale applications. main concept , it's modularity. (before that java development it was jars, but from java 9 it's a module)

Use Case > Suppose : to develop some application we need few classes from particular jars, but if we load then big jar will be loaded, but think for small app like IOT etc. . so from java 9 it's not rt.jar it's only module. from java 9 divide classes in to small module smalls jar.

Example: A simple example of the module system is creating a module for a specific **package, let's say com.example.**

```
module com.example {
exports com.example;
}
module com.example {
requires com.example;
}
javac --module-source-path -d out -m /project path/package name.classname
```

Jlink:

Introduce Java 9. (used to create custom JRE, but why need although we have default JRE)
Lets assume your code require 5 java classes to run, size of the default JRE almost 200 mb. (so memory wastage), default JRE not recommended for small devices (so java not best choice for small Devices)

Syntax for create Custom JRE ::

jlink --module-path out --add-modules helloModule,java.base --output testjre

now we can run class with custom jre. **Example:** A simple example of the module system is creating a module for a specific **package, let's say com.example.**

```
module com.example {
exports com.example;
}
```

HTTP 2/client:

Java 9 introduced a new HTTP client API that supports HTTP/2 and WebSocket, making it easier to send and receive data from web services.

Example: Here is an example of using the HTTP/2 client API to send a GET request to a web service:

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
.uri(URI.create("https://example.com"))
.build();
HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());

System.out.println(response.body()); // Output: The response body of the HTTP request
```

Code

```
package com.techtalk.debu.java9;

import java.util.Random;
/**
 * @author debup
 *
 */
public interface PrivateInterface {
static void generateNumber() {
System.out.println("Randon Number :" + generateRandomNumber());
}
static void AppendRandomNumber() {
String appendCode = "Tech" + generateRandomNumber();
System.out.println("Randon Number after apending String:" + appendCode);
}
private static int generateRandomNumber() {
Random random = new Random();
return random.nextInt(50);
}
}
```

Code

```
package com.techtalk.debu.java9;

import java.util.Random;
/**
 * @author debup
 *
 */
public interface PrivateInterface {
static void generateNumber() {
System.out.println("Randon Number :" + generateRandomNumber());
}
static void AppendRandomNumber() {
String appendCode = "Tech" + generateRandomNumber();
System.out.println("Randon Number after apending String:" + appendCode);
}
private static int generateRandomNumber() {
Random random = new Random();
return random.nextInt(50);
}
}
```

Code

```
package com.techtalk.debu.java9;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class StreanAPIAdditions {

	public static void main(String[] args) {

		Integer[] arr = {2,4,6,8,10};

		// take While
		List<Integer> takeWhileList = Stream.of(arr).takeWhile(i -> i
<6).collect(Collectors.toList());
		System.out.println("takeWhile: " + takeWhileList);

		// drop While
		List<Integer> dropWhileList = Stream.of(arr).dropWhile(i -> i
<6).collect(Collectors.toList());
		System.out.println("dropWhile: " + dropWhileList);
```

Code

```
// ofNullable

        List<Integer> nullList = null;
        List<Integer> numberList = Arrays.asList(2, 4, 6, 8, 10);
        List tempList = Stream.ofNullable(nullList).collect(Collectors.toList());
        List tempList2 = Stream.ofNullable(numberList).collect(Collectors.toList());
        System.out.println("ofNullable: " + tempList);
        System.out.println("ofNullable tempList2: " + tempList2);

    }

}
```

Code

```
package com.techtalk.debu.java9;
import java.util.Optional;

public class ProcessAPI {

public static void main(String[] args) {
int pid = 866677;// dummy value
int pid2 = 3056;// from Task manager get actual value

Optional<ProcessHandle> processHandle1 = ProcessHandle.of(pid);
Optional<ProcessHandle> processHandle2 = ProcessHandle.of(pid2);
System.out.println(processHandle1.isPresent());// false
System.out.println(processHandle2.isPresent());// true

ProcessHandle processHandle = ProcessHandle.current();
System.out.println(processHandle.info());
}

}
```

Code

```
package com.techtalk.debu.java9;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class TryWithResourceChanges {

        private static void beforeJava9() {

                try (BufferedReader br = new BufferedReader(new FileReader("C:\\Debu Paul\\JAVA
Version Fetures\\JAVA 9\\helloWorld.txt"))) {
                        String line;
                        while ((line = br.readLine()) != null) {
                                System.out.println(line);
                        }

                } catch (IOException e) {
                        e.printStackTrace();
                }

        }
```

Code

```
private static void fromJava9() throws FileNotFoundException {
		BufferedReader br = new BufferedReader(new FileReader("c:\\Debu Paul\\\\JAVA
Version Fetures\\\\JAVA 9\\\\\\helloWorld.txt"));

		try (br) {
			String line;
			while ((line = br.readLine()) != null) {
				System.out.println(line);
			}
		} catch (IOException e) {
			e.printStackTrace();
		}

	}

	public static void main(String[] args) throws FileNotFoundException {
		beforeJava9();
		fromJava9();
	}

}
```

Thank You

Debu Paul | Follow me @