Java 11 Features

1. **Running Java File with Single 'java' Command**

One major change is that you don’t need to compile the java source file with javac tool first. You can directly run the file with **java** command and it implicitly compiles.

1. **String API updates**
2. ***isBlank()*** - This instance method returns a boolean value. Empty Strings and Strings with only white spaces are treated as blank.

**Example** –

public class MyClass {

public static void main(String args[]) {

String s = "";

String s1 = " ";

System.out.println(s.isEmpty()); // true

System.out.println(s.isBlank()); // true

System.out.println(s1.isEmpty()); // false

System.out.println(s1.isBlank()); // true

}

}

1. ***lines()* -** This method returns a stream of strings, which is a collection of all substrings split by lines.

**Example** –

import java.util.stream.Collectors;

public class MyClass {

public static void main(String args[]) {

String str = "Java\nJava\nJava";

System.out.println(str);

System.out.println(str.lines().collect(Collectors.toList()));

System.out.println(str.lines().count());

}

}

**Output** –

Java

Java

Java

[Java, Java, Java]

1. ***strip(), stripLeading(), stripTrailing()***

***strip()*** – Removes the white space from both, beginning and the end of string.

**Example** –

public class MyClass {

public static void main(String args[]) {

Character c = '\u2000';

String s = c + "abc" + c;

System.out.println(c); //.

System.out.println(Character.isWhitespace(c)); //true

System.out.println(s.trim()); //.abc.

System.out.println(s.strip()); // abc

}

}

**Example** –

public class Main {

public static void main(String[] args) throws Exception {

String str = " Java ";

System.out.print("Start");

System.out.print(str.strip());

System.out.println("End");

System.out.print("Start");

System.out.print(str.stripLeading());

System.out.println("End");

System.out.print("Start");

System.out.print(str.stripTrailing());

System.out.println("End");

}

}

**Output** –

StartJavaEnd

StartJava End

Start JavaEnd

1. ***repeat(int)*** - The repeat method simply repeats the string that many numbers of times as mentioned in the method in the form of an int.

**Example** –

public class MyClass {

public static void main(String args[]) {

String str = "Java".repeat(4);

System.out.println(str); // JavaJavaJavaJava

}

}

1. **Local Variable Type Inference**

*Java 10 has introduced a new language keyword var which optionally replaces the type information when declaring local variables (local means variable declarations inside method bodies).*

**Example** –

* Prior to Java 10 you would declare variables like this:

String text = "Hello Java 9"; //Java 9

* Now you can replace String with var. The compiler infers the correct type from the assignment of the variable. In this case text is of type String:

var text = "Hello Java 10"; //Java 10

* Variables declared with var are still statically typed. You cannot reassign incompatible types to such variables. This code snippet does not compile:

var text = "Hello Java 11";

text = 23; // Incompatible types

* You can also use final in conjunction with var to forbid reassigning the variable with another value:

final var text = "Banana";

text = "Joe"; // Cannot assign a value to final variable 'text'

* Also var is not allowed when the compiler is incapable of infering the correct type of the variable. All of the following code samples result in compiler errors:

// Cannot infer type:

var a;

var nothing = null;

var lambda = () -> System.out.println("Pity!");

var method = this::someMethod;

* Local variable type inference really shines with generics involved. In the next example current has a rather verbose type of Map<String, List<Integer>> which can be reduced to a single var keyword, saving you from typing a lot of boilerplate:

var myList = new ArrayList<Map<String, List<Integer>>>();

for (var current : myList) {

// current is infered to type: Map<String, List<Integer>>

System.out.println(current);

}

* As of Java 11 the var keyword is also allowed for lambda parameters which enables you to add annotations to those parameters:

Predicate<String> predicate = (@Nullable var a) -> true;

1. **Immutable collections**

*Collections such as List, Set and Map have been extended with new methods. List.of created a new immutable list from the given arguments. List.copyOf creates an immutable copy of the list.*

**Example** –

var list = List.of("A", "B", "C");

var copy = List.copyOf(list);

System.out.println(list == copy); // true

However, if you copy a mutable list, copy is indeed a new instance so it's guaranteed there's no side-effects when mutating the original list:

var list = new ArrayList<String> ();

var copy = List.copyOf(list);

System.out.println(list == copy); // false

When creating immutable maps you don't have to create map entries yourself but instead pass keys and values as alternating arguments:

var map = Map.of("A", 1, "B", 2);

System.out.println(map); // {B=2, A=1}

*Immutable collections in Java 11 still use the same interfaces from the old Collection API. However if you try to modify an immutable collection by adding or removing elements, a java.lang.UnsupportedOperationException is thrown.*

1. **Stream API updates**

The methods dropWhile() and takeWhile() both accept a predicate to determine which elements to abandon from the stream:

**Example** –

public class MyClass {

public static void main(String args[]) {

Stream.of(4, 4, 4, 5, 6, 7, 8, 9, 10)

.dropWhile(number -> (number / 4 == 1))

.forEach(System.out :: println); // [8, 9, 10]

Stream.of(4, 4, 4, 5, 6, 7, 8, 9, 10)

.takeWhile(number -> (number / 4 == 1))

.forEach(System.out :: println); // [4, 4, 4, 5, 6, 7]

}

}

1. **File API Updates**

Makes it easy to read/write Strings to and from a file.

**Example** –

public class MyClass {

public static void main(String args[]) {

Path path = Files.writeString(Files.createTempFile(“Test”, “.txt”), “Java 11 is cool”);

String str = Files.readString(path);

System.out.println(str); // Java 11 is cool

}

}

1. **New method added in Optional class**

**isEmpty() -** This allows for more fluent expression in cases that formerly relied upon negation of Optional.isPresent().

**Example** –

public class MyClass {

public static void main(String args[]) {

Optional<String> str = Optional.empty();

Optional<String> str1 = Optional.of(“test”);

System.out.println(str.isEmpty()); // true

System.out.println(str1.isEmpty()); // false

}

}

1. **Collectors API updates**

**Example** –

public class MyClass {

public static void main(String args[]) {

var list = List.of(15, 17, 21, 25, 30); // Immutable list

List<Integer> mutableNewList = list.stream()

.filter(i -> i % 3 == 0)

.collect(Collectors.toList());

*When we do Collectors.toList() we get mutable list, and hence to convert it back to immutable list we need to add another line of code i.e (Collections.toUnmodifiableList(mutableNewList))*

*Instead now we can directly get the Immutable list from Collectors class.*

List<Integer> immutableNewList = list.stream()

.filter(i -> i % 3 == 0)

.collect(Collectors.toUnmodifiableList());

mutableNewList.add(17); // gets added

immutableNewList.add(17); // throws java.lang.UnsupportedOperationException

}

}