Stream In Java

16.01.2020

Stream In Java

- Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.
  The features of Java stream are –
- A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
- Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
- Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Streams intermediate operations

- filter() - Returns a stream consisting of the elements of this stream that match the given predicate.
- map() - Returns a stream consisting of the results of applying the given function to the elements of this stream.
- distinct() - Returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.
- sorted() - Returns a stream consisting of the elements of this stream, sorted according to the provided Comparator.

Streams terminal operations

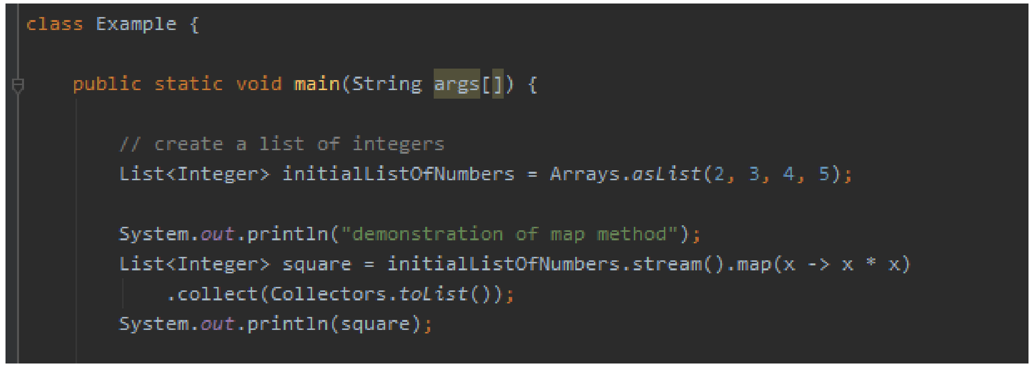
- collect() - Performs a mutable reduction operation on the elements of this stream using a Collector. A Collector encapsulates the functions used as arguments to collect
- count() - Returns the count of elements in this stream.
- forEach() - Performs an action for each element of this stream.
- anyMatch() - Returns whether any elements of this stream match the provided predicate.
- allMatch() - Returns whether all elements of this stream match the provided predicate.
- noneMatch() - Returns whether no elements of this stream match the provided predicate.
- findFirst() - Returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty.

More operations in Stream

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

EXAMPLES

Let’s demonstrate the usage of the mentioned operators starting with map():



```
class Example {

    public static void main(String args[]) {

        // create a list of integers
        List<Integer> initialListOfNumbers = Arrays.asList(2, 3, 4, 5);

        System.out.println("demonstration of map method");
        List<Integer> square = initialListOfNumbers.stream().map(x -> x * x)
            .collect(Collectors.toList());
        System.out.println(square);
```

Explanation: We iterate the list of integers using stream and we calculate the square of each element and collect the results in a list.

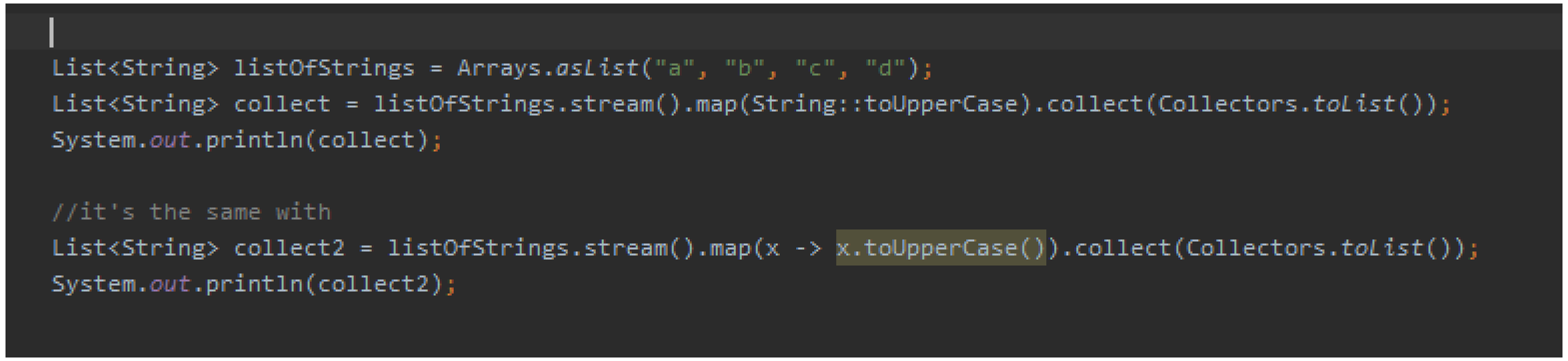
The output will be:

demonstration of map method

[4, 9, 16, 25]

EXAMPLES

Example:



```
List<String> listOfStrings = Arrays.asList("a", "b", "c", "d");
List<String> collect = listOfStrings.stream().map(String::toUpperCase).collect(Collectors.toList());
System.out.println(collect);

//it's the same with
List<String> collect2 = listOfStrings.stream().map(x -> x.toUpperCase()).collect(Collectors.toList());
System.out.println(collect2);
```

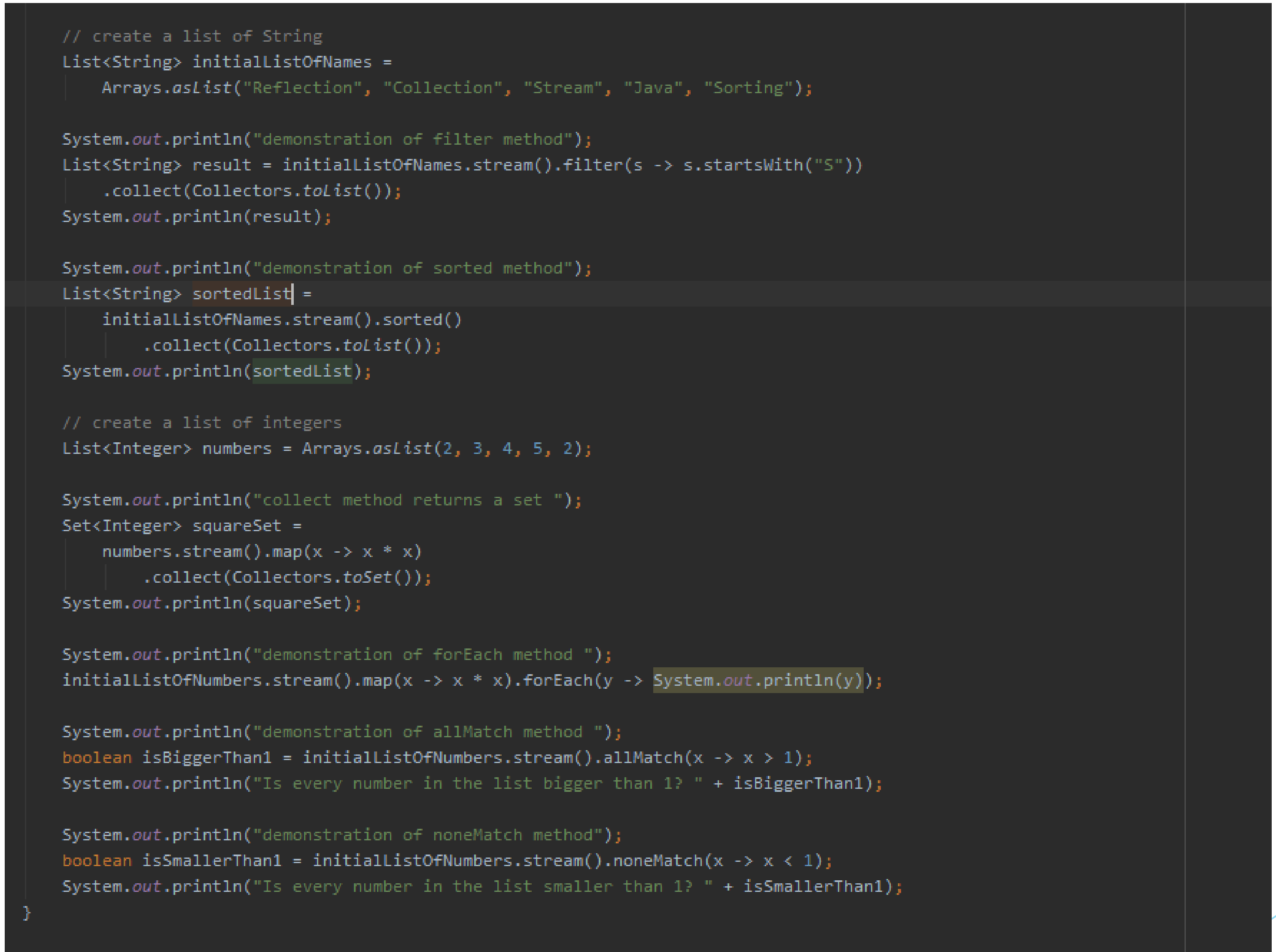
We iterate list of strings and for each element we store the upper case value of the string and collect the result in another list. The both examples will return the same result where the first one is using method reference and the second one lambda expression.

The output will be:

```
demonstration of map method
[A, B, C, D]
[A, B, C, D]
```

EXAMPLES

What would be the output of the following:



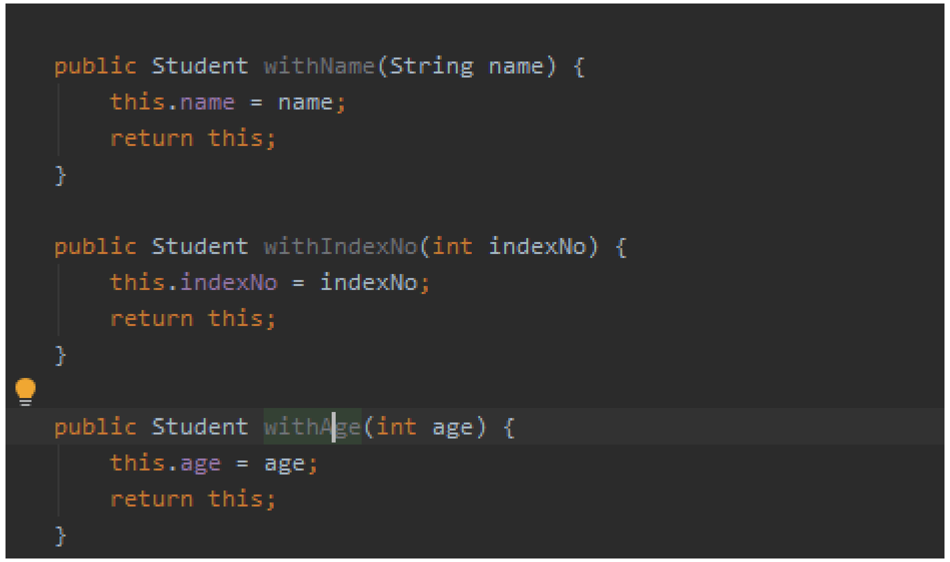
```
    // create a list of String
    List<String> initialListOfNames =
        Arrays.asList("Reflection", "Collection", "Stream", "Java", "Sorting");

    System.out.println("demonstration of filter method");
    List<String> result = initialListOfNames.stream().filter(s -> s.startsWith("S"))
        .collect(Collectors.toList());
    System.out.println(result);

    System.out.println("demonstration of sorted method");
    List<String> sortedList =
        initialListOfNames.stream().sorted()
            .collect(Collectors.toList());
    System.out.println(sortedList);

    // create a list of integers
    List<Integer> numbers = Arrays.asList(2, 3, 4, 5, 2);

    System.out.println("collect method returns a set ");
    Set<Integer> squareSet =
        numbers.stream().map(x -> x * x)
            .collect(Collectors.toSet());
    System.out.println(squareSet);

    System.out.println("demonstration of forEach method ");
    initialListOfNumbers.stream().map(x -> x * x).forEach(y -> System.out.println(y));

    System.out.println("demonstration of allMatch method ");
    boolean isBiggerThan1 = initialListOfNumbers.stream().allMatch(x -> x > 1);
    System.out.println("Is every number in the list bigger than 1? " + isBiggerThan1);

    System.out.println("demonstration of noneMatch method");
    boolean isSmallerThan1 = initialListOfNumbers.stream().noneMatch(x -> x < 1);
    System.out.println("Is every number in the list smaller than 1? " + isSmallerThan1);
}
```

ASSIGNMENT 1

Create a class Student with following attributes:

String name; int indexNo; int age;

Create getters and fluent setters



```
public Student withName(String name) {
    this.name = name;
    return this;
}

public Student withIndexNo(int indexNo) {
    this.indexNo = indexNo;
    return this;
}

public Student withAge(int age) {
    this.age = age;
    return this;
}
```

ASSIGNMENT 1

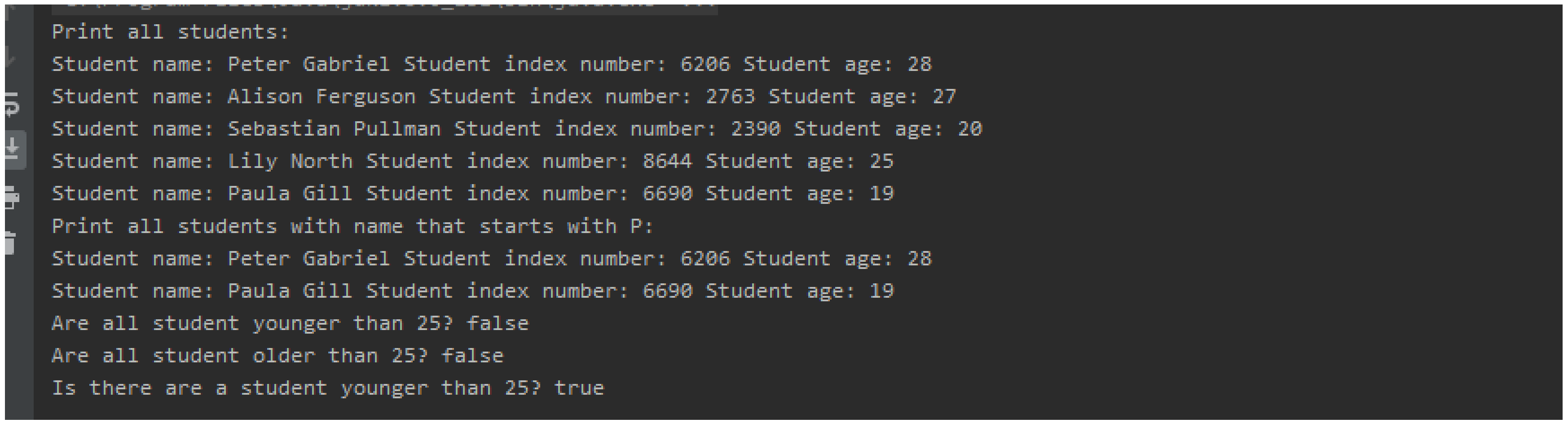
- Create a class Demo
- Create a method Student createStudent(String name, int indexNo, int) which will return object of Student type with the values provided as arguments
- Create a method void printValues(Student s) which will print the values of the student's attributes
- In the main method create a list of strings with 5 names.
- Using stream.map() create Student objects for each name by utilizing createStudent() method (the name of the student will be each element of the names list, the index number will be random number between 1 and 10000 and age will be random number between 18 and 30.
- Once created print the values for each student using printValues() method

ASSIGNMENT 2

- Find all the students whose name starts with P and add them In a new list.
- Print the students from the new list.
- Check if all students are younger than 25 and print the result
- Check if all students are older than 25 and print the result
- Check if at least one of the students is younger than 25 and print the result

ASSIGNMENT 2

The output should look something similar to this:



```
Print all students:
Student name: Peter Gabriel Student index number: 6206 Student age: 28
Student name: Alison Ferguson Student index number: 2763 Student age: 27
Student name: Sebastian Pullman Student index number: 2390 Student age: 20
Student name: Lily North Student index number: 8644 Student age: 25
Student name: Paula Gill Student index number: 6690 Student age: 19
Print all students with name that starts with P:
Student name: Peter Gabriel Student index number: 6206 Student age: 28
Student name: Paula Gill Student index number: 6690 Student age: 19
Are all student younger than 25? false
Are all student older than 25? false
Is there are a student younger than 25? true
```