Matching operations are terminal operations that are used to check if elements with certain criteria are present in the stream or not.

There are mainly three matching functions available in `Stream`. These are:

- `anyMatch()`
- `allMatch()`
- `noneMatch()`

We will discuss each one of them with examples.

1) anyMatch()

Here is the syntax of this method:

```
boolean anyMatch(Predicate<? super T> predicate)
```

It takes a predicate as input and returns

- `true` if at least one element matches the criteria.
- `false` if no element matches the criteria.
- `false` if the stream is empty.

In the below example, we have a List of `Person` objects. We need to check if there is any person residing in a particular country.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {

    public static void main(String[] args) {
        List<Person> list = new ArrayList<>();
        list.add(new Person("Dave", 23,"India"));
        list.add(new Person("Joe", 18,"USA"));
        list.add(new Person("Ryan", 54,"Canada"));
        list.add(new Person("Iyan", 5,"India"));
        list.add(new Person("Ray", 63,"China"));

        boolean anyCanadian = list.stream()
                .anyMatch(p -> p.getCountry().equals("Canada"));

        System.out.println("Is there any resident of Canada: " + anyCanadian);

    }
}


class Person {
    String name;
    int age;
    String country;
```

Run Save Reset

2) allMatch()

Here is the syntax of this method:

> **boolean allMatch(Predicate<? super T> predicate)**

It takes a predicate as input and returns

- `true` if all elements match the criteria.
- `true` if the stream is empty.
- `false` if even a single element does not match the criteria.

In the below example, we have a List of `Person` objects. We need to check if all the persons are residents of a particular country.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {

    public static void main(String[] args) {
        List<Person> list = new ArrayList<>();
        list.add(new Person("Dave", 23,"India"));
        list.add(new Person("Joe", 18,"USA"));
        list.add(new Person("Ryan", 54,"Canada"));
        list.add(new Person("Iyan", 5,"India"));
        list.add(new Person("Ray", 63,"China"));

        boolean anyCanadian = list.stream()
                .allMatch(p -> p.getCountry().equals("Canada"));

        System.out.println("Are all persons canadian: " + anyCanadian);

    }
}


class Person {
    String name;
    int age;
    String country;
```

Run Save Reset

3) noneMatch()

Here is the syntax of this method:

> **boolean noneMatch(Predicate<? super T> predicate)**

It takes a predicate as input and returns

- `true` if no elements of the stream match the provided predicate.
- `true` if the stream is empty.
- `false` if even a single element matches the criteria.

In the below example, we have a list of `Person` objects. We need to check if all the persons are residents of a particular country.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {

    public static void main(String[] args) {
        List<Person> list = new ArrayList<>();
        list.add(new Person("Dave", 23,"India"));
        list.add(new Person("Joe", 18,"USA"));
        list.add(new Person("Ryan", 54,"Canada"));
        list.add(new Person("Iyan", 5,"India"));
        list.add(new Person("Ray", 63,"China"));

        boolean anyRussian = list.stream()
                .noneMatch(p -> p.getCountry().equals("Russia"));

        System.out.println(anyRussian);

    }
}


class Person {
    String name;
    int age;
    String country;
```

Run Save Reset