In the previous lesson, we looked at matching operations. Those operations check whether the elements in the stream match particular criteria, and they return true or false.

However, sometimes we need to get the matched element instead of just verifying if it is present or not. The finding operations are used for this purpose. There are two basic finding operations in streams, i.e., `findFirst()` and `findAny()`.

These operations are typically used with a `filter()` operation, but it is not necessary that they are used only with a `filter()` operation.

Let's discuss each finding operation.

1) findFirst()

Below is the syntax of this operation.

> Optional<T> findFirst()

It returns an `Optional` describing the first element of this stream, or an empty `Optional` if the stream is empty. We already discussed `Optional` in our lambda expression chapter. Please revisit that lesson to learn about Optional.

In the below example we have a list of `Person` objects. We need to get the first person on the list who belongs to a particular country.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

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"));

        Optional<Person> person = list.stream()               // Creating a Stream of person objects.
                .filter(p -> p.getCountry().equals("India")) // Filter to get only persons living in Ind
                .findFirst();                                // Returning the first person encountered.

        if(person.isPresent()){
            System.out.println(person);
        }

    }
}


class Person {
    String name;
```

Run Save Reset

2) findAny()

Below is the syntax of this operation.

```
Optional<T> findAny()
```

It returns an `Optional` describing some element of this stream, or an empty `Optional` if the stream is empty. Now you might be wondering why we need this method if we already have the `findFirst()` operation.

This operation is particularly useful in the case of parallel streams. We have not discussed parallel streams yet but we will discuss them in future lessons. For now, just imagine that we can create a parallel stream so that the intermediate operations can be applied in parallel.

Now if we use the `findFirst()` method in the parallel stream, it can be very slow. Instead, `findAny()` is used if we are not concerned about which element is returned.

Below is an example of `findAny()`.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

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"));

        Optional<Person> person = list.stream()
                .filter(p -> p.getCountry().equals("India"))
                .findAny();

        if(person.isPresent()){
            System.out.println(person);
        }

    }
}


class Person {
    String name;
```

Run Save Reset