INTRODUCTION TO JAVA

Java 1.0

STREAMS

Lesson # 15

METHOD REFERENCES

METHOD REFERENCE

- The method references can only be used to replace a single method of the lambda expression
- A code is more clear and short if one uses a lambda expression rather than using an anonymous class, and one can use a method reference rather than using a single function lambda expression to achieve the same

METHOD REFERENCE SYNTAX

Name of class or object

```
Object :: methodName
```

Name of method

METHOD REFERENCE EXAMPLE

```
List<String> fruits = List.of("Apple", "Banana", "Orange");

//Using lambda expression
fruits.forEach(fruit -> System.out.println(fruit));

//Using method reference
fruits.forEach(System.out::println);
```

OPTIONAL CLASS

OPTIONAL CLASS

- Every Java Programmer is familiar with NullPointerException - it can crash your code, and it is tough to avoid it without using too many null checks
- Java 8 has introduced a new class, Optional in java.util package
- Optional is a container object which is used to contain not-null objects
- By using Optional, we can specify alternate values to return or alternate code to run

OPTIONAL INITIALIZATION

A factory method that creates an option from a non-null value

```
Optional<String> optionalValue = Optional.of("Apple");
```

Optional interface

OPTIONAL INITIALIZATION – NULL VALUE

A factory method that creates an option from a nullable value

```
String value = null;
Optional<String> optionValue = Optional.ofNullable(value);
```

OPTIONAL CORE METHODS

Method	Purpose
public boolean isPresent()	Return true if there is a value present, otherwise false.
public void ifPresent(Consumer<T> consumer)	If a value is present, invoke the specified consumer with the value, otherwise, do nothing.
public T get()	If a value is present in this Optional, returns the value, otherwise throws NoSuchElementException.
public T orElse(T other)	Return the value if present, otherwise, return other.
public T orElseGet(Supplier<T> other)	Return the value if present, otherwise, invoke other and return the result of that invocation.

OPTIONAL CORE METHODS

Method	Purpose
public <U> Optional<U> map(Function<T, U> mapper)	If a value is present, apply the provided mapping function to it, and if the result is non-null, return an Optional describing the result. Otherwise return an empty Optional.
public <X> T orElseThrow(Supplier<X> exceptionSupplier)	Return the contained value, if present, otherwise, throw an exception to be created by the provided supplier.

OPTIONAL – IS PRESENT & GET

```
Optional<String> optional = getOptionalValue();
if (optional.isPresent()) {
    String value = optional.get();
    System.out.println(value);
}
```

OPTIONAL – IF PRESENT

```
Optional<String> optional = getOptionalValue();
optional.ifPresent(value -> System.out.println(value));
```

OPTIONAL – OR ELSE

```
Optional<String> optional = getOptionalValue();
String defaultValue = optional.orElse("Default value");
System.out.println(defaultValue);
```

OPTIONAL – OR ELSE GET

```
Optional<String> optional = getOptionalValue();
String defaultValue = optional.orElseGet(() -> getDefaultValue());
System.out.println(defaultValue);
```

OPTIONAL – MAP

```
Optional<String> optional = getOptionalValue();
optional.map(value -> value.toUpperCase())
        .ifPresent(value -> System.out.println(value));
```

OPTIONAL – OR ELSE THROW

```
Optional<String> optional = getOptionalValue();
String value = optional.orElseThrow(() -> new NullPointerException());
System.out.println(value);
```

STREAM API

STREAM API

- Introduced in Java 8
- 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
- If we want to represent a group of objects as a single entity, then we should go for the collection
- But if we want to process objects from the collection then we should go for streams.

STREAM INITIALIZATION

Stream source

```
List<String> fruits = List.of("Apple", "Banana", "Pineapple", “Peach");
Stream<String> fruitStream = fruits.stream();
```

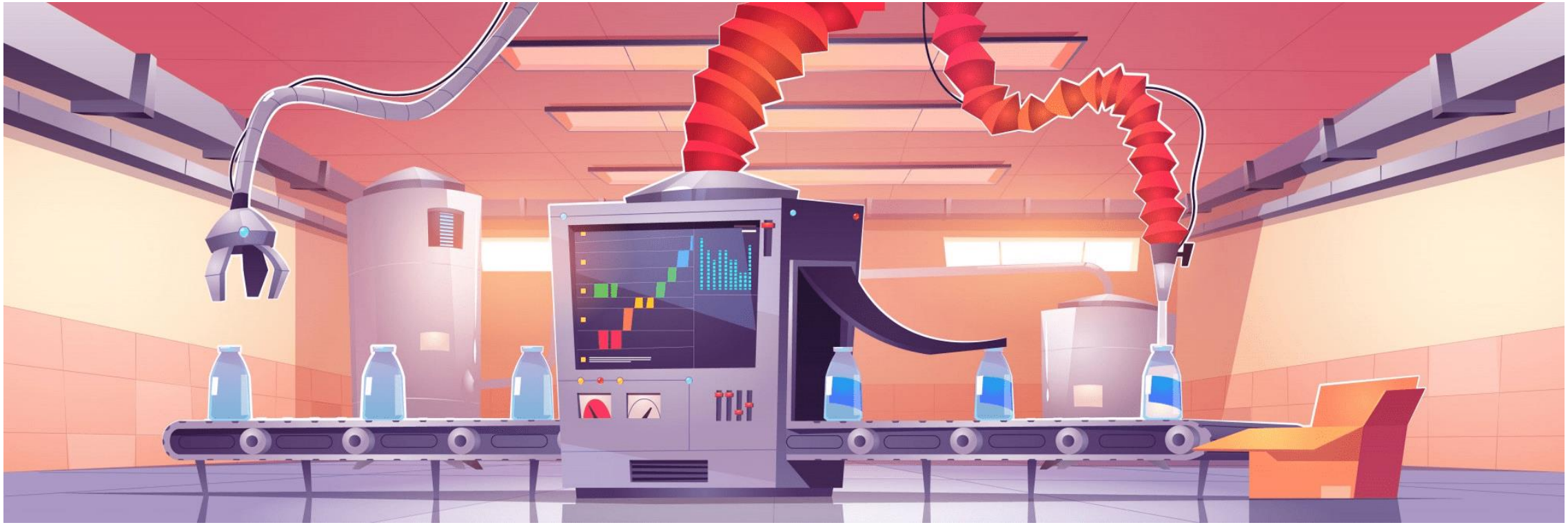
Stream interface

A method that initializes the stream

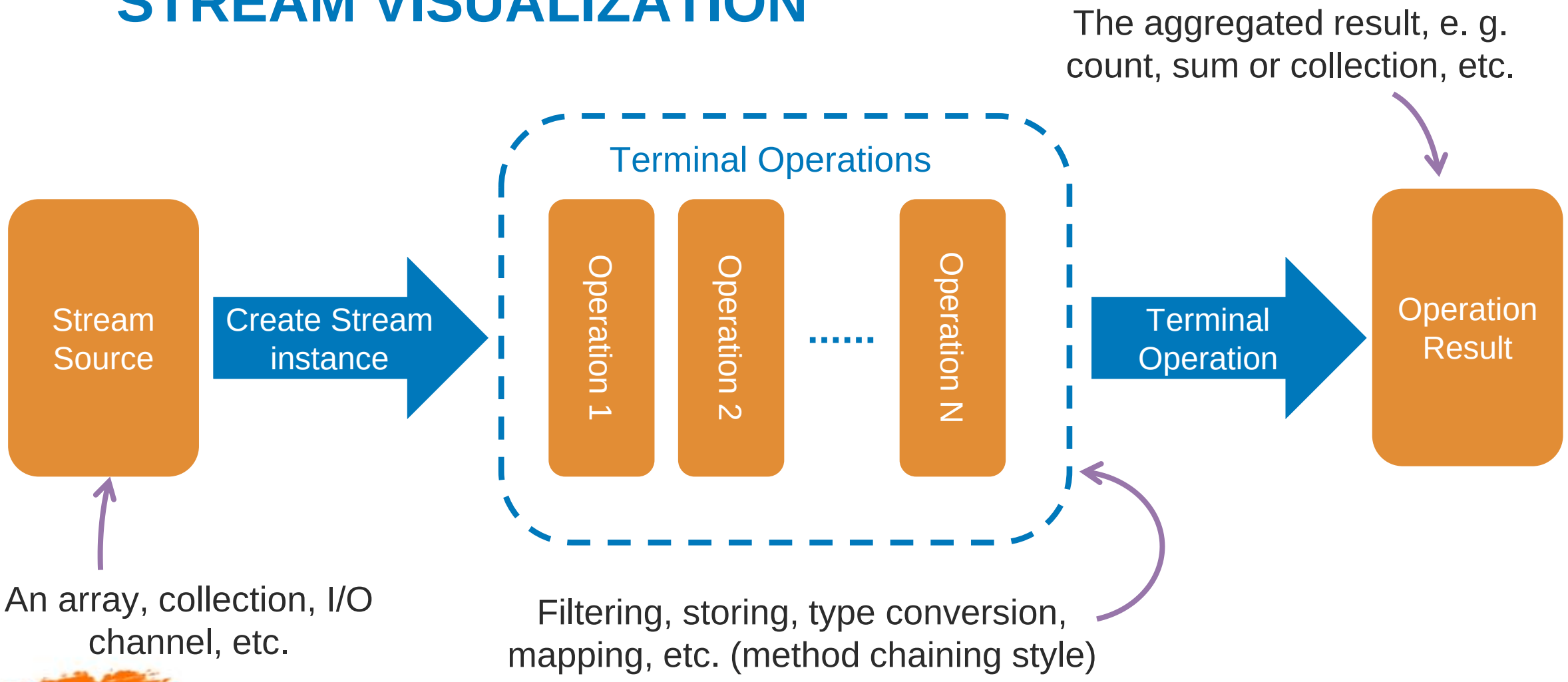
STREAM API

- 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.

JUST LIKE A CONVEYOR BELT



STREAM VISUALIZATION



STREAM EXAMPLE

Stream source

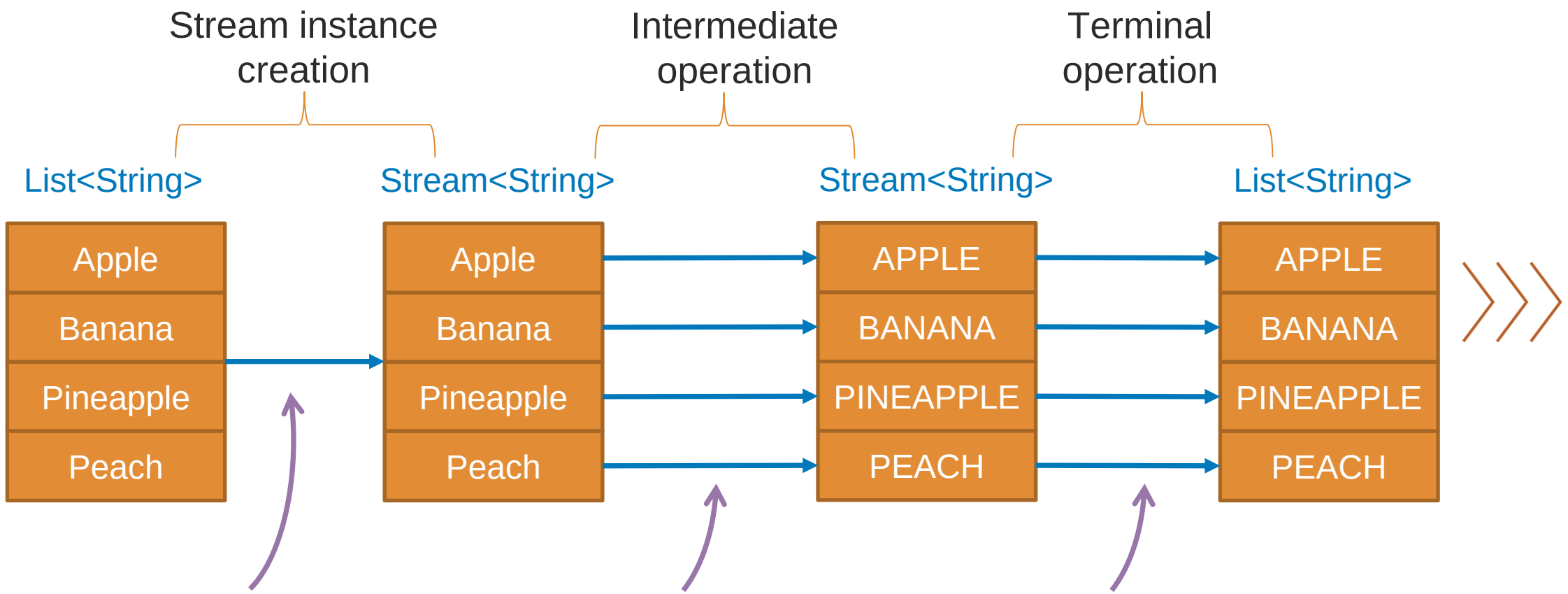
```
List<String> fruits = List.of("Apple", "Banana", "Pineapple", "Peach");
List<String> fruitsInUppercase = fruits.stream()
        .map(fruit -> fruit.toUpperCase())
        .collect(Collectors.toList());
```

Intermediate operation

Terminal operation

Stream instance creation

STREAM EXAMPLE VISUALIZATION




```
fruits.stream().map(fruit -> fruit.toUpperCase()).collect(Collector.toList());
```

CORE INTERMEDIATE OPERATIONS

Method	Purpose
Stream<T> filter(Predicate<T> predicate)	Returns a stream consisting of the elements of this stream that match the given predicate.
Stream<R> map(Function<T, R> mapper)	Returns a stream consisting of the results of applying the given function to the elements of this stream.
Stream<T> sorted()	Returns a stream consisting of the elements of this stream, sorted according to natural order.
Stream<T> sorted(Comparator<T> comparator)	Returns a stream consisting of the elements of this stream, sorted according to the provided Comparator.

CORE TERMINATION OPERATIONS

Method	Purpose
<R,A> R collect(Collector<T,A,R> collector)	Performs a mutable reduction operation on the elements of this stream using a Collector.
Optional<T> reduce(BinaryOperator<T> accumulator)	Performs a reduction on the elements of this stream, using an associative accumulation function, and returns an Optional describing the reduced value, if any.
boolean anyMatch(Predicate<? super T> predicate)	Returns whether any elements of this stream match the provided predicate. May not evaluate the predicate on all elements if not necessary for determining the result.
Optional<T> findFirst()	Returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.

STREAMS VS IMPERATIVE STYLE

```java
List<Integer> list = List.of(3, 2, 12, 5, 6, 11, 13);
int count = 0;
for (Integer i : list) {
    if (i % 2 == 0) {
        count++;
    }
}
System.out.println(count);
```

```java
List<Integer> list = List.of(3, 2, 12, 5, 6, 11, 13);
long count = list.stream()
        .filter(i -> i % 2 == 0)
        .count();
System.out.println(count);
```

STREAMS VS IMPERATIVE STYLE

```
List<Integer> list = List.of(3, 2, 12, 5, 6, 11, 13);
List<Integer> evenList = new ArrayList<>();
for (Integer I : list) {
    if (i % 2 == 0) {
        evenList.add(i);
    }
}
System.out.println(evenList);
```

```
List<Integer> list = List.of(3, 2, 12, 5, 6, 11, 13);
List<Integer> evenList = list.stream()
                .filter(i -> i % 2 == 0)
                .collect(Collectors.toList());
System.out.println(evenList);
```

STREAMS VS IMPERATIVE STYLE

```
List<Integer> list = List.of(3, 2, 12, 5, 6, 11, 13);
boolean b = true;
for (Integer i : list) {
    if (i >= 10) {
        b = false;
        break;
    }
}
System.out.println(b);
```

```
List<Integer> list = List.of(3, 2, 12, 5, 6, 11, 13);
boolean b = list.stream().allMatch(i -> i < 10);
System.out.println(b);
```

REMEMBER!

- Streams are not data structure
- Streams are not storage for data
- Streams are “pipelines” for streams of data (i.e., of objects)
- While in the pipeline, data undergo transformation
- Streams wrap collections (lists, sets, maps)

REFERENCES

REFERENCES

- https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
- https://www.javatpoint.com/java-8-method-reference
- https://dzone.com/articles/java-an-optional-implementation-of-optional
- https://www.javatpoint.com/java-8-optional
- https://www.javatpoint.com/java-8-stream
- https://stackify.com/streams-guide-java-8/

QUESTIONS?

THANK YOU!