OBJECT ORIENTED ANALYSIS & DESIGN
DATA STRUCTURES & ALGORITHMS

Java – (8) Advanced Features

LearnOA
To The Next Level...

Features

Java Lambda Expressions

Lambda as an Object

Functional Interface

Lambda Value Capture

Lambda Parameters

Method References as Lambdas

Features

- Foreach
- Map
- Filter
- Limit
- Sorted
- Parallel processing
- Collectors
- statistics


Java Lambda Expressions

Java Lambda Expressions

- ✔ Java lambda expressions are Java's first step into functional programming
- ✔ It is an anonymous function that doesn't have a name and doesn't belong to any class
- ✔ It provides a clear and concise way to represent a method interface via an expression
- ✔ It provides the implementation of a functional interface & simplifies the software development

Java Lambda Expressions

Syntax

```
parameter -> expression body
```

Arrow Operator is introduced in Java through lambda expressions that divides it into two parts i.e Parameters & Body

Characteristics

```
-> Optional Type Declarations
-> Optional Parentheses Around Parameters
-> Optional Curly Braces
-> Optional return keyword
```

Functional Interface

Functional Interface

✔Functional Interface is an interface that contains exactly one abstract method

✔It can have any number of default or static methods along with object class methods

✔Java provides predefined functional interfaces to deal with functional programming

✔***Runnable***, ***ActionListener***, ***Comparable*** are some of the examples of functional interfaces

Functional Interface

```
@FunctionalInterface
interface displayable{
 void display(String msg);
}

public class Test implements displayable{
 public void display(String msg){
   System.out.println(msg);
 }
public static void main(String[] args) {
 Test dis = new Test();
 dis.display("Welcome to Lambda Tutorial !");
 }
}
```

Lambda Parameters

Lambda Parameters

Lambda Expressions can take parameters just like methods

Zero Parameters

One Parameter

Multiple Parameters

Lambda Parameters

Zero Parameters

One Parameter

Multiple Parameters

Lambda Parameters

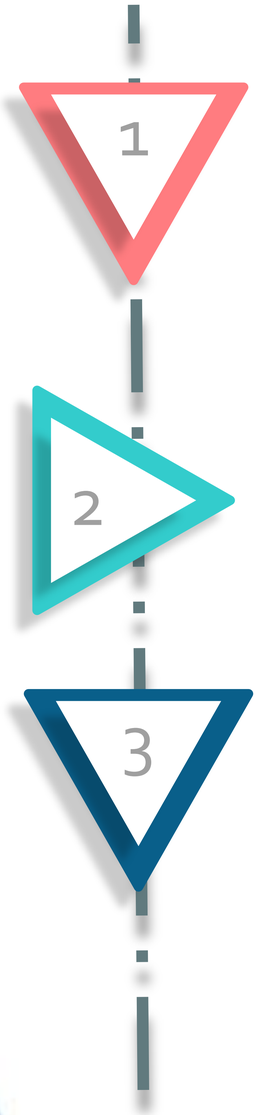
Zero Parameters

```
() -> System.out.println("Zero parameter lambda");
```

One Parameter

Multiple
Parameters

Lambda Parameters



Zero Parameters

```
() -> System.out.println("Zero parameter lambda");
```

One Parameter

```
(param) -> System.out.println("One parameter: " + param);
```

Multiple Parameters

Lambda Parameters

Zero Parameters

```
() -> System.out.println("Zero parameter lambda");
```

One Parameter

```
(param) -> System.out.println("One parameter: " + param);
```

Multiple Parameters

```
(p1, p2) -> System.out.println("Multiple parameters: " +
  p1 + ", " + p2);
```

Lambda As An object

Lambda as an Object

A Java lambda expression is essentially an object that can be assigned to a variable and passed around

Interface

```
public interface LambdaComparator {
public boolean compare(int a1, int a2);
}
```

Implementing class

```
LambdaComparator myComparator = (a1, a2) -> return a1 > a2;

boolean result = myComparator.compare(2, 5);
```

Lambda Variable Capture

Variable Capture

Java lambda expression can access variables that are declared outside the lambda function body under certain circumstances

1. Local Variable
2. Instance Variables
3. Static Variables

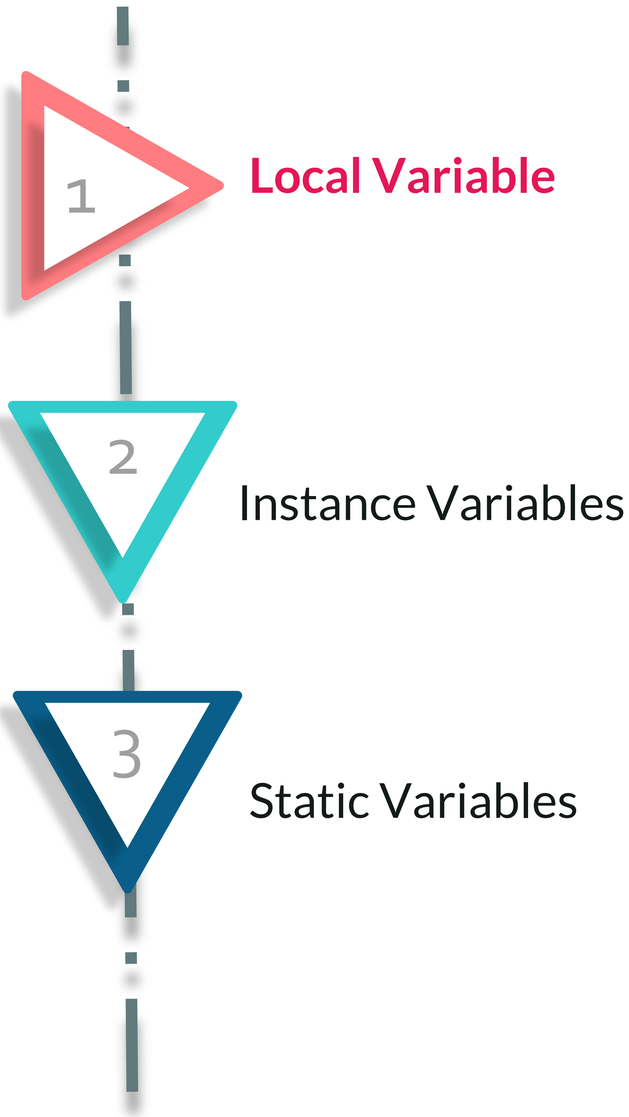
Lambda Parameters

Local Variable

Instance Variables

Static Variables

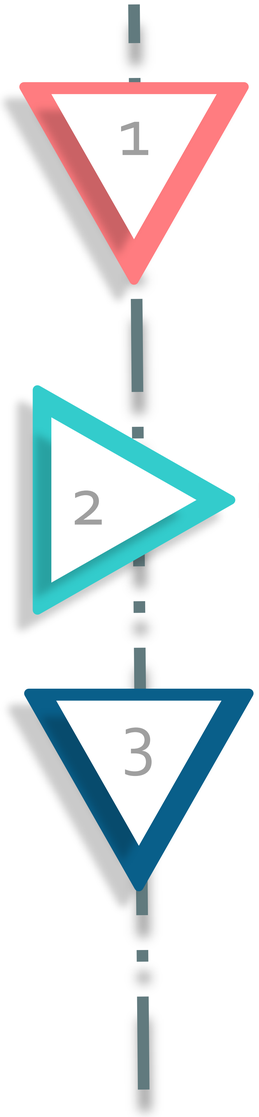
Lambda Parameters




```
String myStr = "Welcome Everyone!";

MyLambda dis = (chars) -> {
      return myStr + ":" + new String(chars);
};
```

Lambda Parameters



Local Variable

Instance Variables

Static Variables

```
public class LambdaStaticConsumerDemo{

  private String str = "Lambda Consumer";

     public void attach(LambdaStaticProducerDemo eventProd){
             eventProd.listen(e -> {
                 System.out.println(this.str);
             });
         }
     }
```

Lambda Parameters

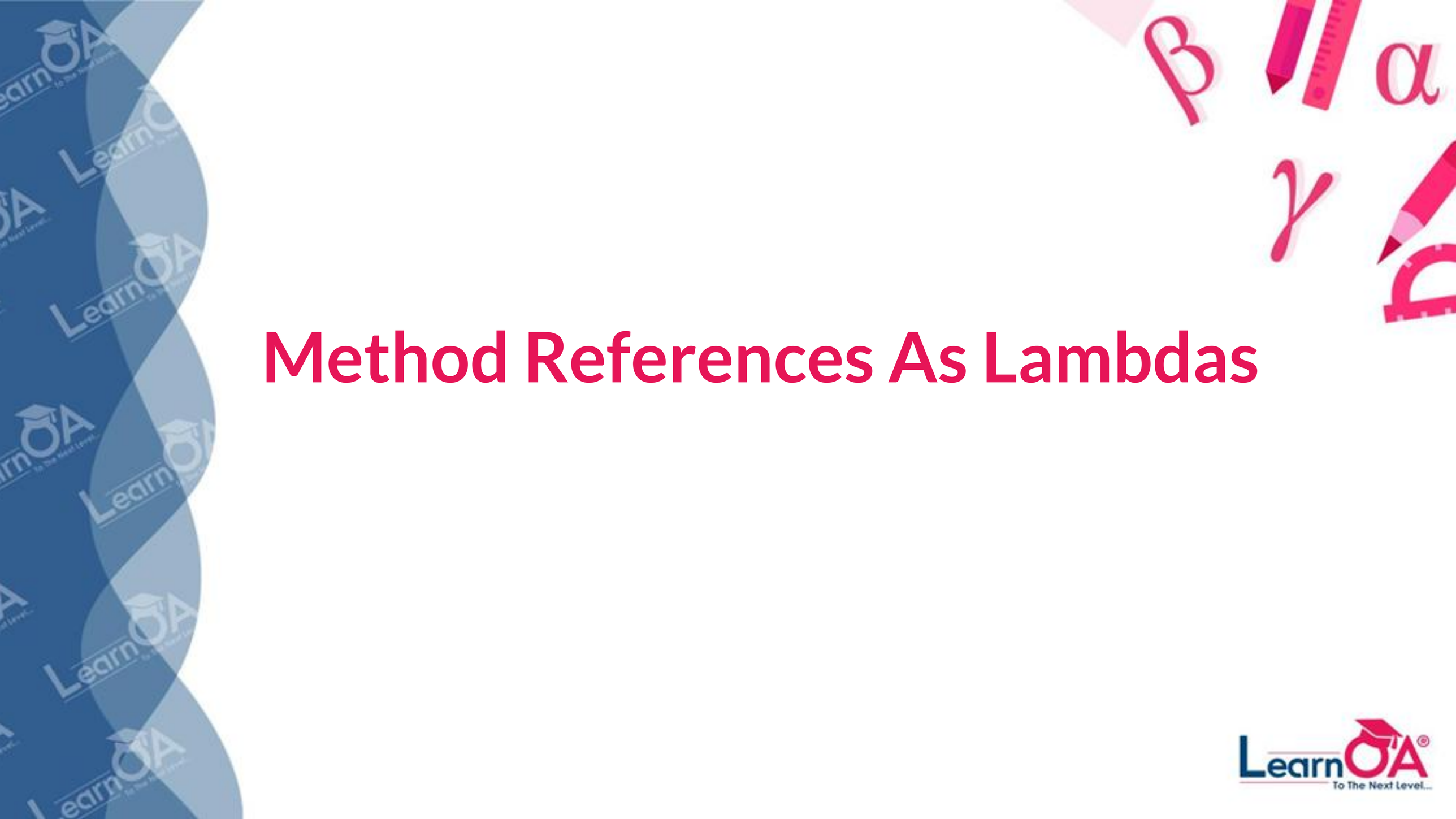
Local Variable

Instance Variables

Static Variables

```
public class LambdaStaticConsumerDemo {
  private static String myStaticVar = ”Café Shop!";

     public void attach(LambdaStaticProducerDemo eventProd){
            eventProd.listen(e -> {
                System.out.println(myStaticVar);
            });
        }
    }
```


Method References As Lambdas

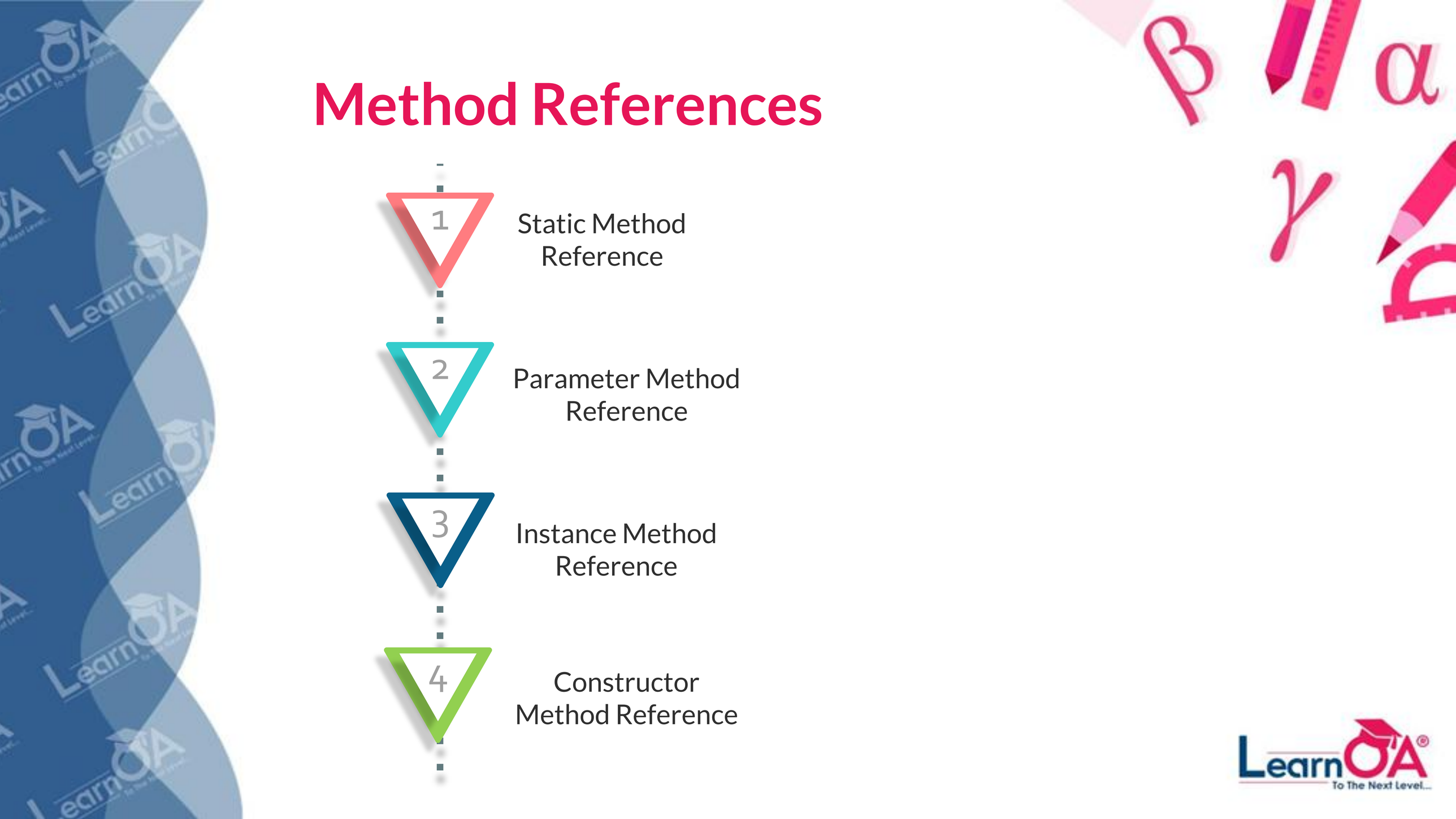
Method References

Java lambda expression can access variables that are declared outside the lambda function body under certain circumstances

1. Static Method Reference
2. Parameter Method Reference
3. Instance Method Reference
4. Constructor Method Reference

Method References

1. Static Method Reference
2. Parameter Method Reference
3. Instance Method Reference
4. Constructor Method Reference



Method References - Static

1. **Static Method Reference**
2. Parameter Method Reference
3. Instance Method Reference
4. Constructor Method Reference

Interface

```
public interface Display {
  public int show(String s1, String s2);
}
```

Class

```
public class Test{
  public static int doShow(String s1, String s2){
        return s1.lastIndexOf(s2);
    }
}
```

Lambda Expression

```
Display disp = Test::doShow;
```

Method References - Parameter

1. Static Method Reference
2. **Parameter Method Reference**
3. Instance Method Reference
4. Constructor Method Reference

Interface

```
public interface Display {
    public int show(String s1, String s2);
}
```

Lambda Expression

```
Display disp = String::indexOf;
```

Method References - Instance

1. Static Method Reference
2. Parameter Method Reference
3. **Instance Method Reference**
4. Constructor Method Reference

Interface

```
public interface Deserializer {
    public int deserialize(String v1);
}
```

Class

```
public class StringConverter {
    public int convertToInt(String v1){
        return Integer.valueOf(v1);
    }
}
```

Lambda Expression

```
StringConverter strConv = new StringConverter();

Deserializer deserializer = strConv::convertToInt;
```

Method References - Constructor

1. Static Method Reference
2. Parameter Method Reference
3. Instance Method Reference
4. **Constructor Method Reference**

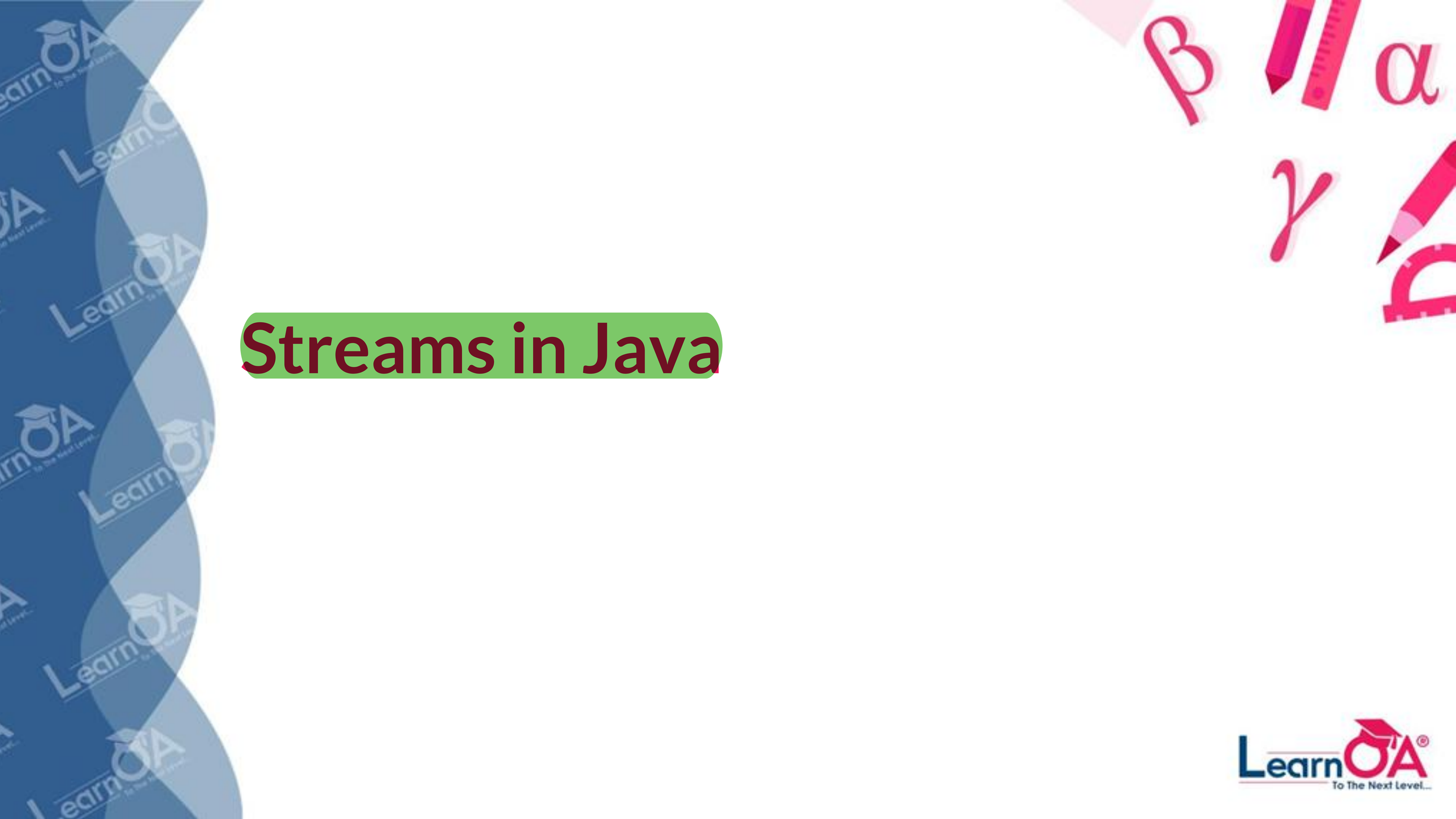
Interface

```
public interface Factory {
    public String create(char[] val);
}
```

Lambda Expression

```
Factory fact = String::new;
```

Streams in Java



Stream represents a sequence of objects from a source, which supports aggregate operations. Following are the characteristics of a Stream –

Sequence of elements – A stream provides a set of elements of specific type in a sequential manner. A stream gets/computes elements on demand. It never stores the elements.

Source – Stream takes Collections, Arrays, or I/O resources as input source.

Aggregate Operations – Stream supports aggregate operations like filter, map, limit, reduce, find, match, and so on.

Pipelining – Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.

Automatic Iterations – Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.

Generating Streams

With Java 8, Collection interface has two methods to generate a Stream.

stream() – Returns a sequential stream considering collection as its source.

parallelStream() – Returns a parallel Stream considering collection as its source.

Thank You!

LearnOA
To The Next Level...