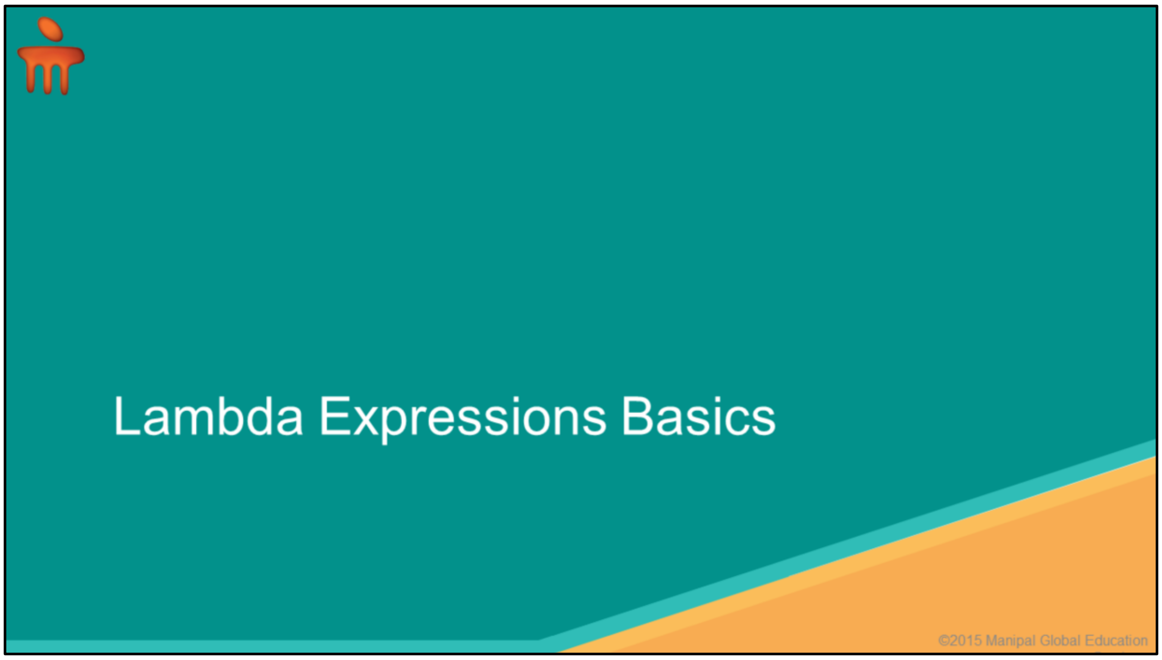


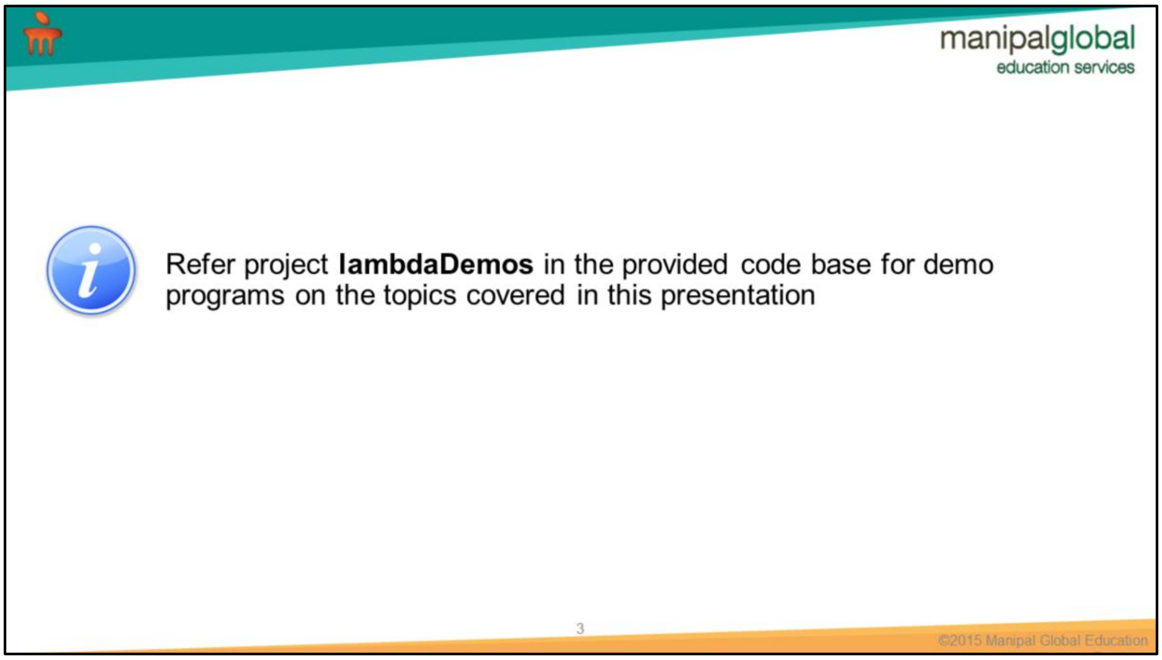
Lambda Expressions Basics

manipalglobal
education services

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Explain the need for Lambda Expressions
- Write and use Lambda Expressions
- Describe Functional Interfaces
- Understand Type Inference in Lambdas
- Implement Functional interfaces



manipalglobal
education services

Refer project **lambdaDemos** in the provided code base for demo programs on the topics covered in this presentation

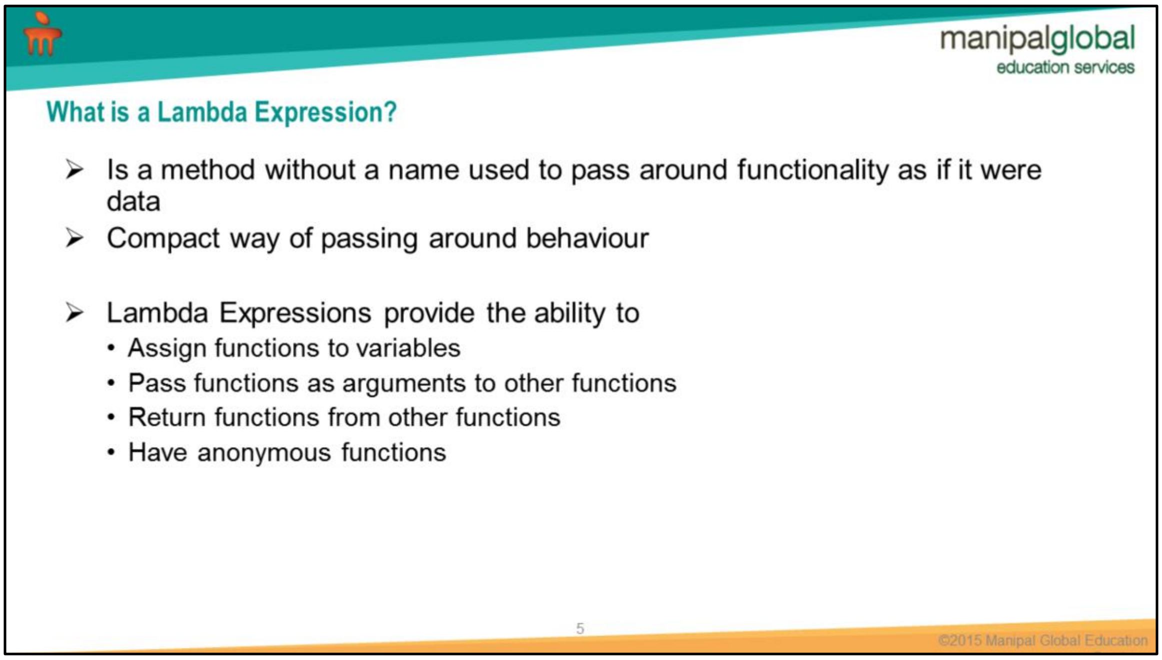
3

manipalglobal
education services

CONCEPT

Basics of Lambda Expressions

4

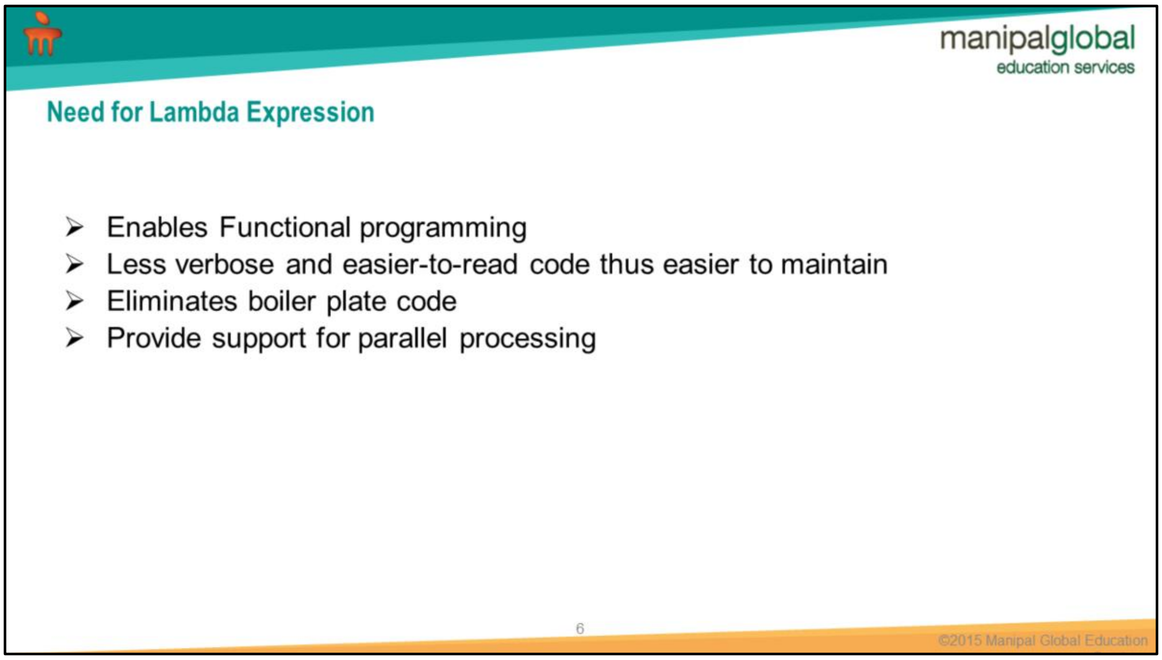


manipalglobal
education services

What is a Lambda Expression?

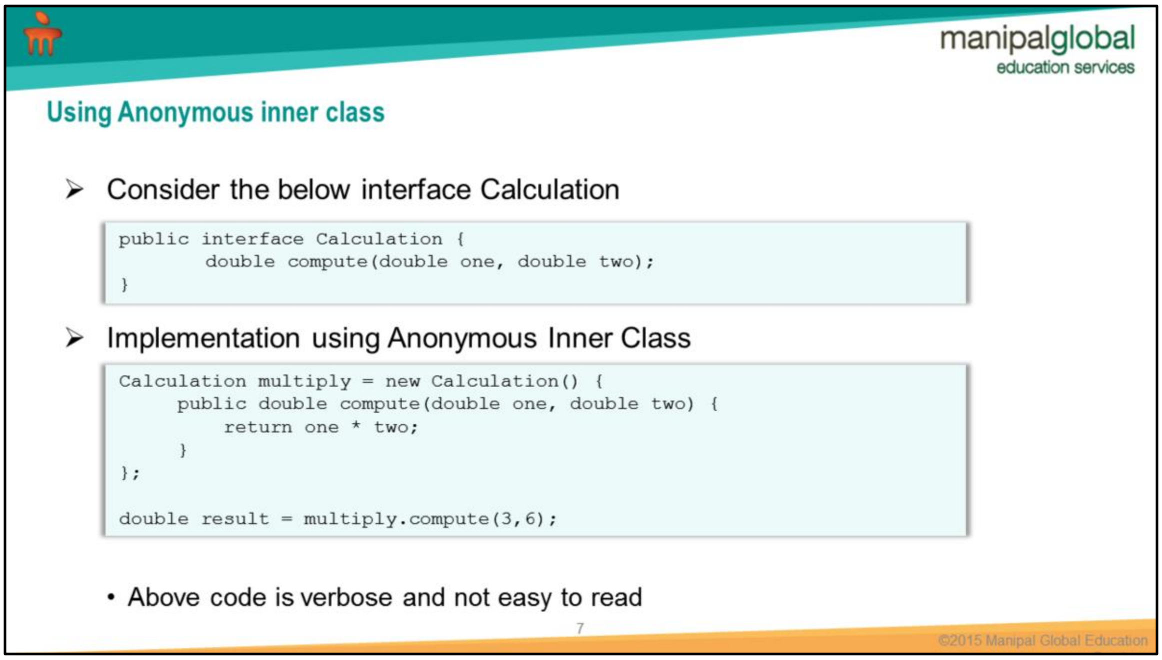
- Is a method without a name used to pass around functionality as if it were data
- Compact way of passing around behaviour

- Lambda Expressions provide the ability to
  - Assign functions to variables
  - Pass functions as arguments to other functions
  - Return functions from other functions
  - Have anonymous functions



Need for Lambda Expression

- Enables Functional programming
- Less verbose and easier-to-read code thus easier to maintain
- Eliminates boiler plate code
- Provide support for parallel processing



manipalglobal
education services

Using Anonymous inner class

➢ Consider the below interface Calculation

```
public interface Calculation {
        double compute(double one, double two);
}
```

➢ Implementation using Anonymous Inner Class

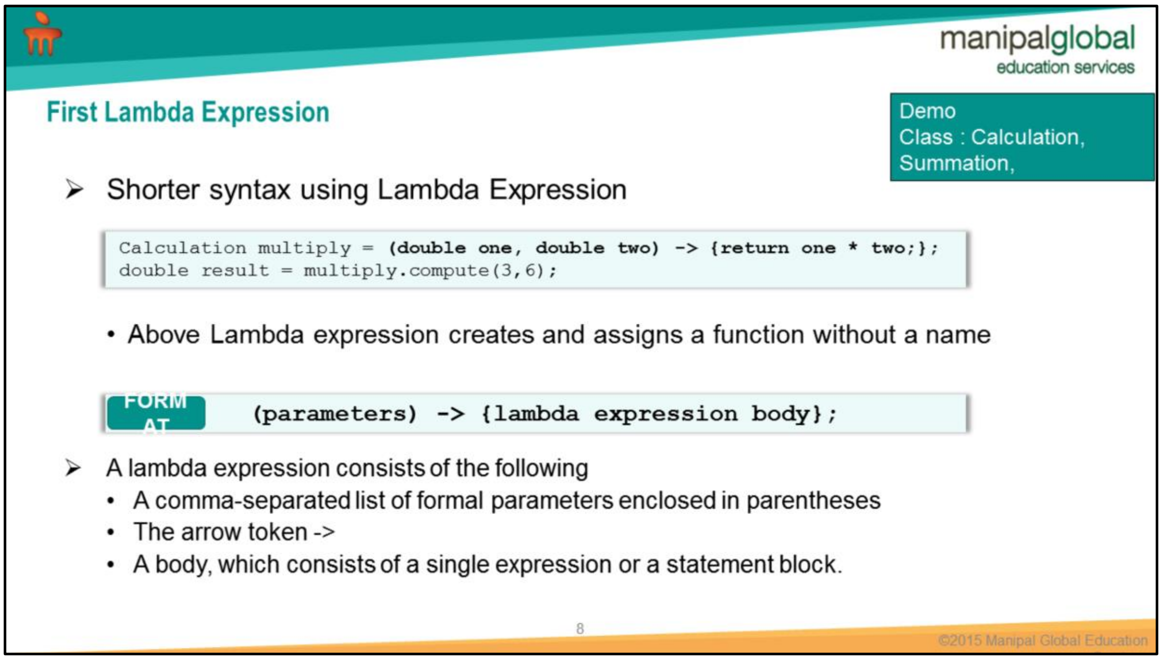
```
Calculation multiply = new Calculation() {
     public double compute(double one, double two) {
         return one * two;
     }
};

double result = multiply.compute(3,6);
```

- Above code is verbose and not easy to read

7

Anonymous inner classes were designed to make it easier for Java programmers to pass around code as data. There are still four lines of boilerplate code required in order to call the single line of important logic in above code. Above code with anonymous inner class is fairly hard to read because it obscures the programmer’s intent.



manipalglobal
education services

First Lambda Expression

Demo
Class : Calculation, Summation,

➢ Shorter syntax using Lambda Expression

```
Calculation multiply = (double one, double two) -> {return one * two;};
double result = multiply.compute(3,6);
```

• Above Lambda expression creates and assigns a function without a name

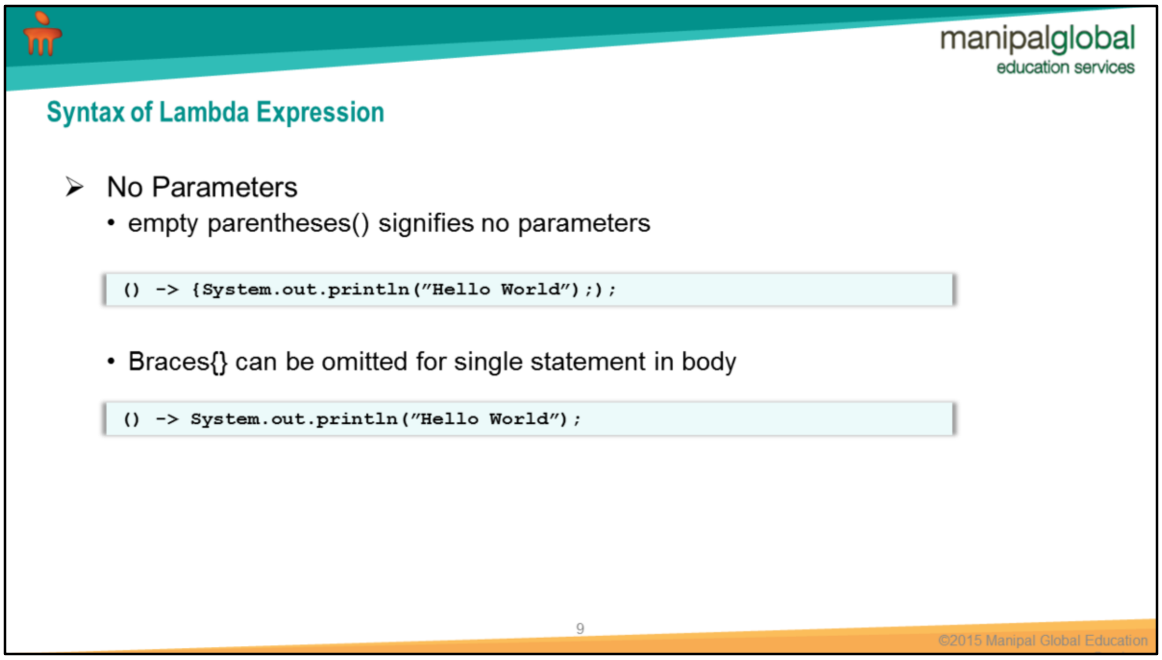
FORMAT

```
(parameters) -> {lambda expression body};
```

➢ A lambda expression consists of the following
- A comma-separated list of formal parameters enclosed in parentheses
- The arrow token ->
- A body, which consists of a single expression or a statement block.

8

In Java 8, we would write this code example as a lambda expression.



manipalglobal
education services

Syntax of Lambda Expression

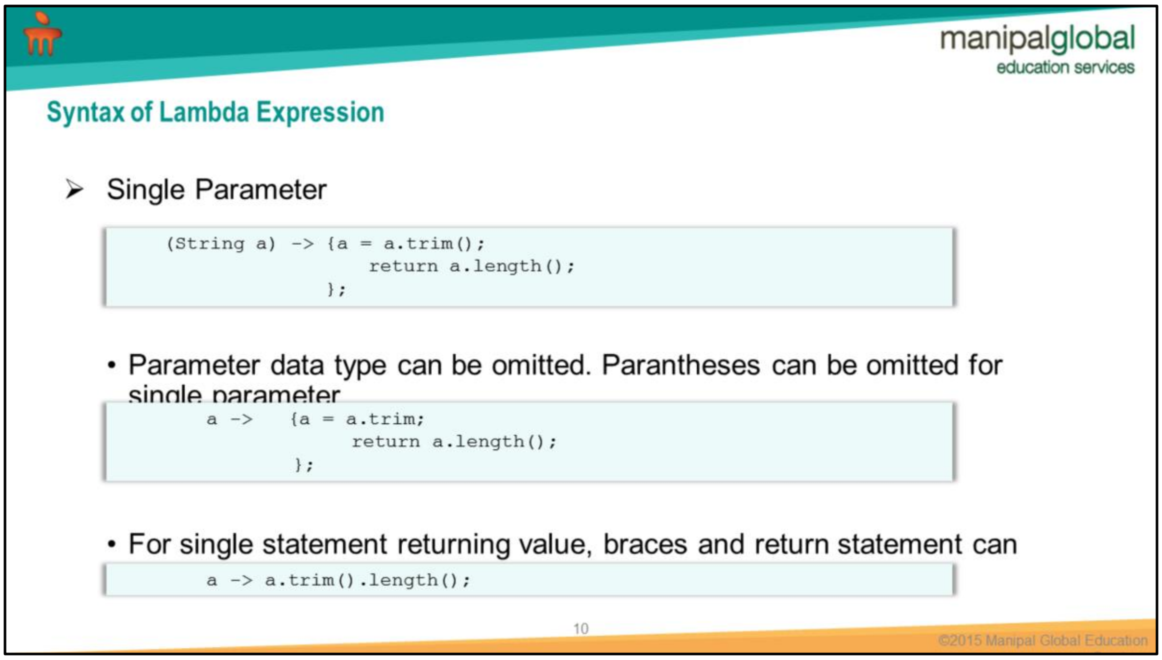
- No Parameters
  - empty parentheses() signifies no parameters

```
() -> {System.out.println("Hello World");};
```

  - Braces{} can be omitted for single statement in body

```
() -> System.out.println("Hello World");
```

9



manipalglobal
education services

Syntax of Lambda Expression

- Single Parameter

```
    (String a) -> {a = a.trim();
                     return a.length();
                 };
```

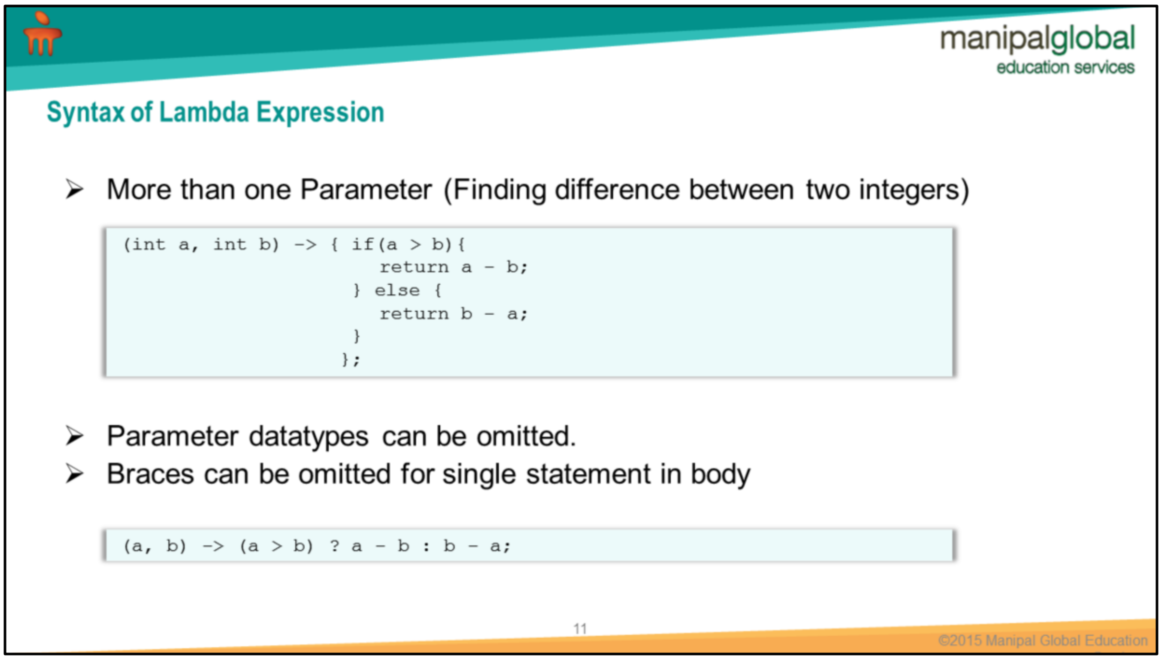
- Parameter data type can be omitted. Parantheses can be omitted for single parameter

```
         a ->    {a = a.trim;
                    return a.length();
              };
```

- For single statement returning value, braces and return statement can

```
         a -> a.trim().length();
```

10


manipalglobal
education services

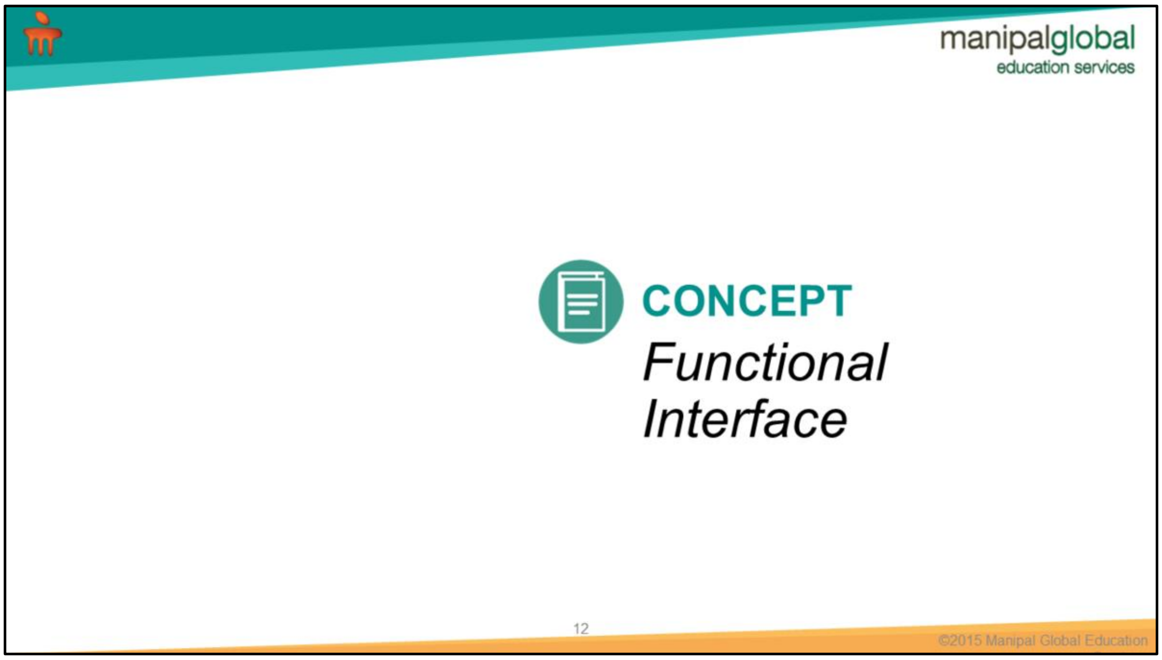
Syntax of Lambda Expression

- More than one Parameter (Finding difference between two integers)

```
(int a, int b) -> { if(a > b){
                        return a - b;
                    } else {
                        return b - a;
                    }
                  };
```

- Parameter datatypes can be omitted.
- Braces can be omitted for single statement in body

```
(a, b) -> (a > b) ? a - b : b - a;
```

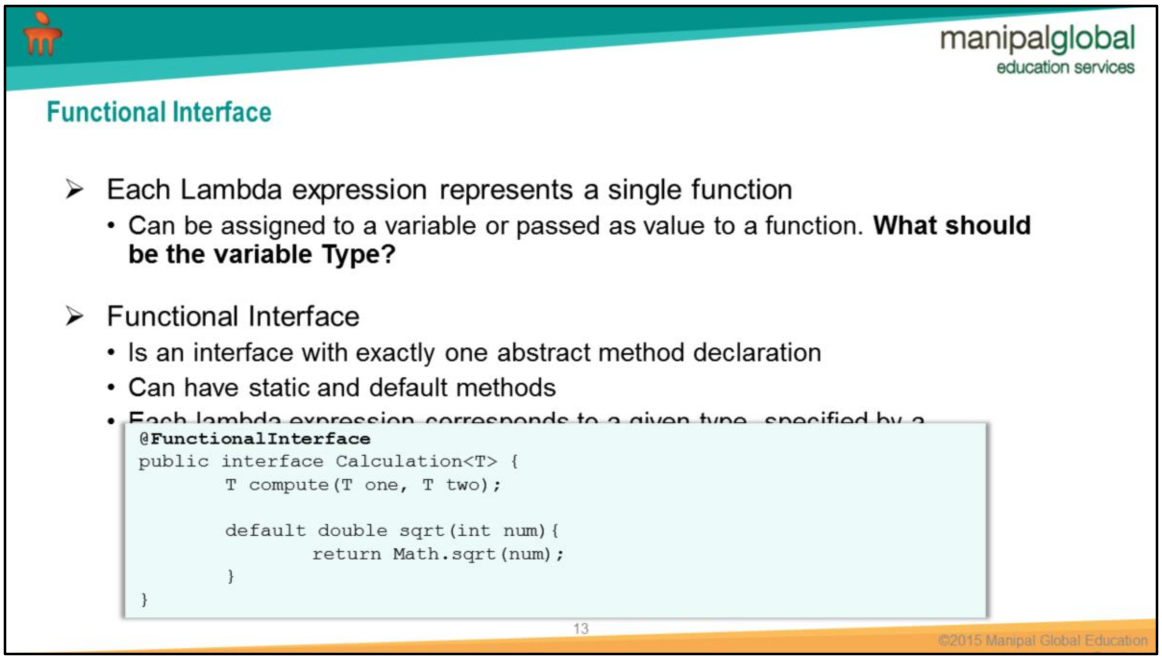


manipalglobal
education services

CONCEPT

Functional Interface

12



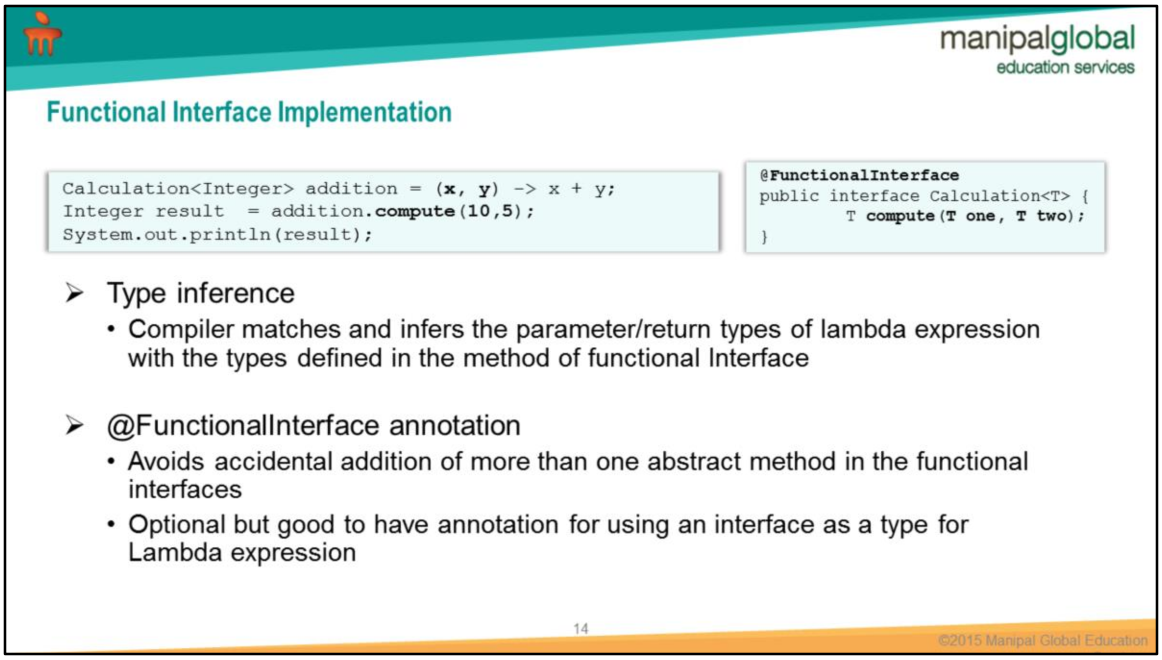
Functional Interface

- Each Lambda expression represents a single function
  - Can be assigned to a variable or passed as value to a function. **What should be the variable Type?**
- Functional Interface
  - Is an interface with exactly one abstract method declaration
  - Can have static and default methods
  - Each lambda expression corresponds to a given type, specified by a

```
@FunctionalInterface
public interface Calculation<T> {
        T compute(T one, T two);

        default double sqrt(int num){
                return Math.sqrt(num);
        }
}
```

13

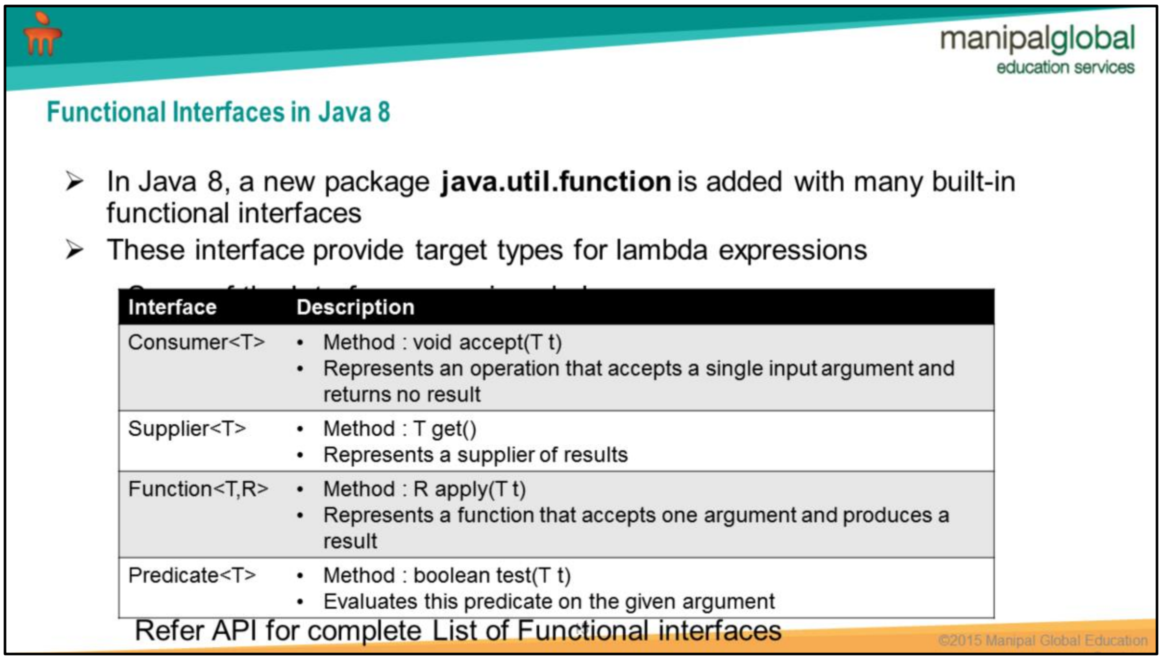

manipalglobal
education services

Functional Interface Implementation

```
Calculation<Integer> addition = (x, y) -> x + y;
Integer result  = addition.compute(10,5);
System.out.println(result);
```

```
@FunctionalInterface
public interface Calculation<T> {
        T compute(T one, T two);
}
```

- Type inference
  - Compiler matches and infers the parameter/return types of lambda expression with the types defined in the method of functional Interface
- @FunctionalInterface annotation
  - Avoids accidental addition of more than one abstract method in the functional interfaces
  - Optional but good to have annotation for using an interface as a type for Lambda expression

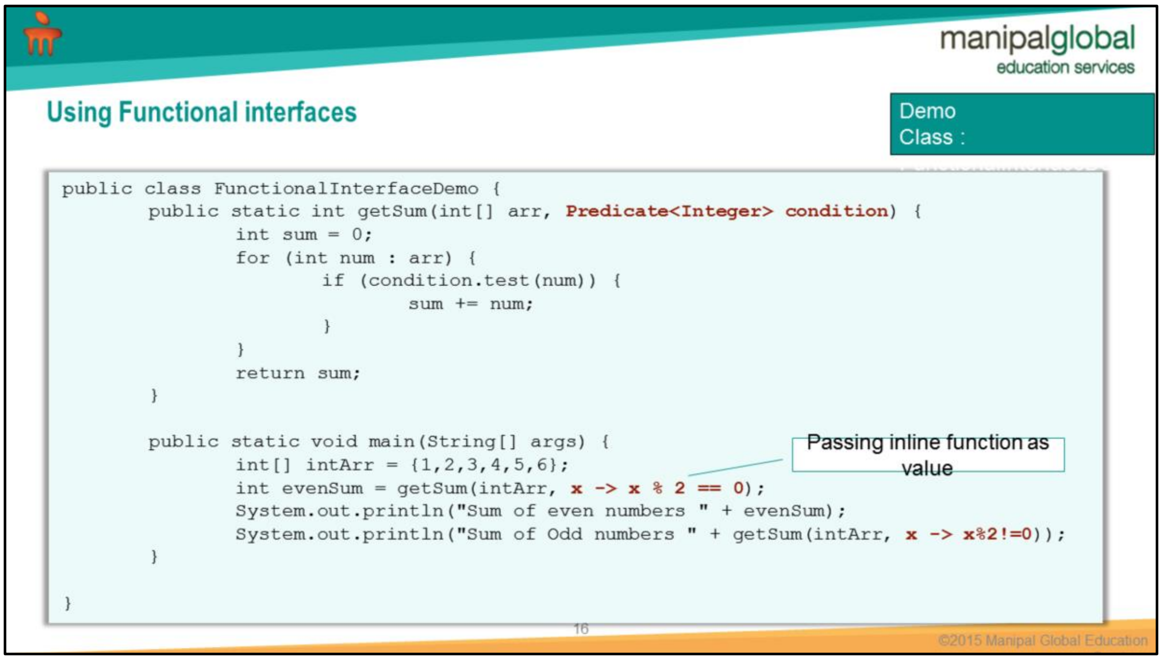


Functional Interfaces in Java 8

- In Java 8, a new package **java.util.function** is added with many built-in functional interfaces
- These interface provide target types for lambda expressions

Interface	Description
Consumer<T>	• Method : void accept(T t) • Represents an operation that accepts a single input argument and returns no result
Supplier<T>	• Method : T get() • Represents a supplier of results
Function<T,R>	• Method : R apply(T t) • Represents a function that accepts one argument and produces a result
Predicate<T>	• Method : boolean test(T t) • Evaluates this predicate on the given argument

Refer API for complete List of Functional interfaces



manipalglobal
education services

Using Functional interfaces

Demo
Class :

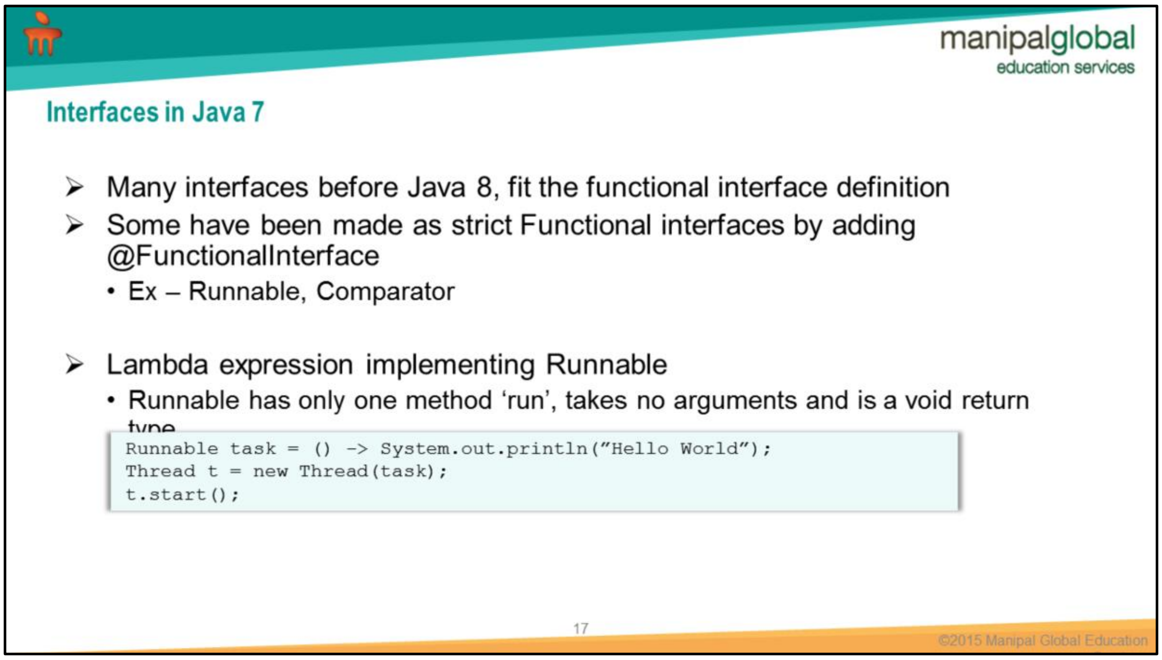
```
public class FunctionalInterfaceDemo {
        public static int getSum(int[] arr, Predicate<Integer> condition) {
                int sum = 0;
                for (int num : arr) {
                        if (condition.test(num)) {
                                sum += num;
                        }
                }
                return sum;
        }

        public static void main(String[] args) {
                int[] intArr = {1,2,3,4,5,6};
                int evenSum = getSum(intArr, x -> x % 2 == 0);
                System.out.println("Sum of even numbers " + evenSum);
                System.out.println("Sum of Odd numbers " + getSum(intArr, x -> x%2!=0));
        }

}
```

Passing inline function as value

16



manipalglobal
education services

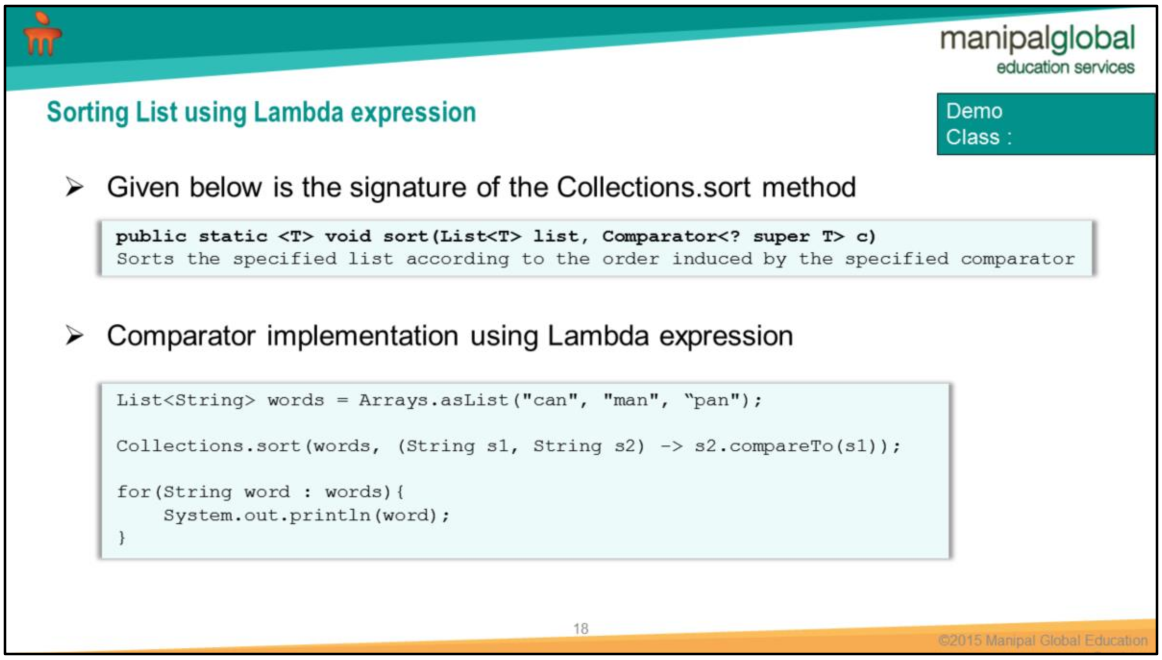
Interfaces in Java 7

- Many interfaces before Java 8, fit the functional interface definition
- Some have been made as strict Functional interfaces by adding @FunctionalInterface
  - Ex – Runnable, Comparator

- Lambda expression implementing Runnable
  - Runnable has only one method 'run', takes no arguments and is a void return type

```
Runnable task = () -> System.out.println("Hello World");
Thread t = new Thread(task);
t.start();
```

17



manipalglobal
education services

Sorting List using Lambda expression

Demo
Class :

- Given below is the signature of the Collections.sort method

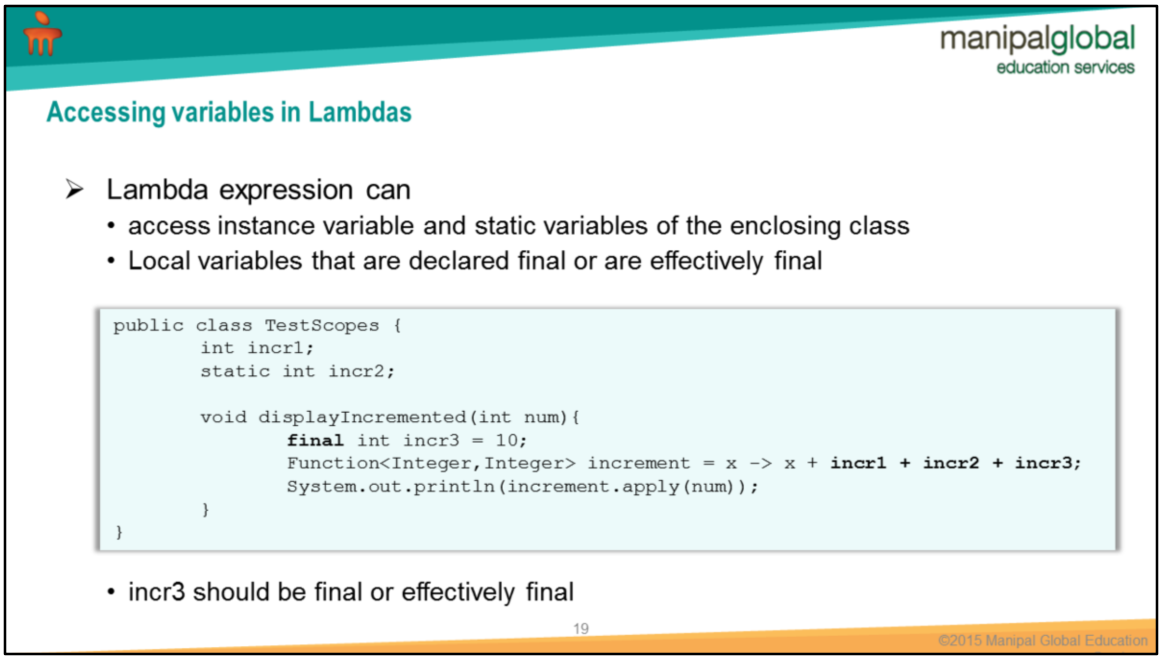
```
public static <T> void sort(List<T> list, Comparator<? super T> c)
Sorts the specified list according to the order induced by the specified comparator
```

- Comparator implementation using Lambda expression

```
List<String> words = Arrays.asList("can", "man", “pan");

Collections.sort(words, (String s1, String s2) -> s2.compareTo(s1));

for(String word : words){
    System.out.println(word);
}
```



manipalglobal
education services

Accessing variables in Lambdas

- Lambda expression can
  - access instance variable and static variables of the enclosing class
  - Local variables that are declared final or are effectively final

```
public class TestScopes {
        int incr1;
        static int incr2;

        void displayIncremented(int num){
                final int incr3 = 10;
                Function<Integer,Integer> increment = x -> x + incr1 + incr2 + incr3;
                System.out.println(increment.apply(num));
        }
}
```

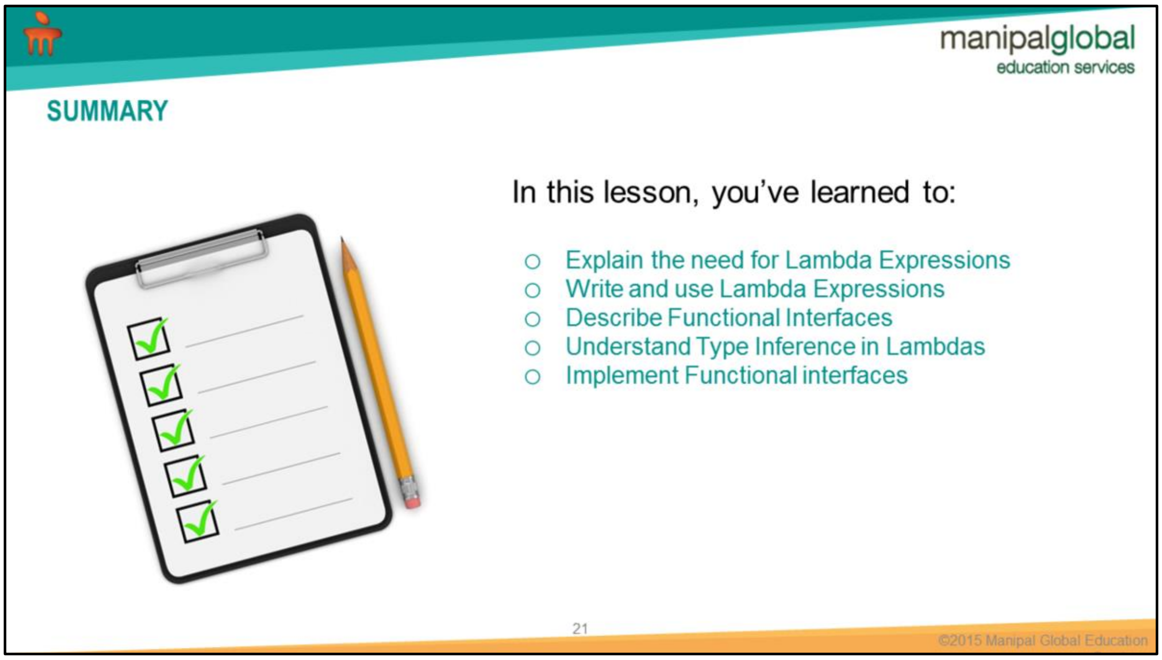
  - incr3 should be final or effectively final

19

manipalglobal
education services

SUMMARY
Lambda Expression Basics

20



manipalglobal
education services

SUMMARY

In this lesson, you've learned to:

- Explain the need for Lambda Expressions
- Write and use Lambda Expressions
- Describe Functional Interfaces
- Understand Type Inference in Lambdas
- Implement Functional interfaces