Lambdas

Functional Interfaces and Lambdas

What is functional interface?

- an interface which has **exactly one** abstract method
- functional interface can be annotated with `@FunctionalInterface`
- Java provides many pre-defined functional interfaces
  - e.g. `Supplier`, `Consumer`, `Predicate`, `Function`, etc.

OCA (1Z0-808)

```
@FunctionalInterface
interface Animal {
  public void speak();
}
class Dog implements Animal {
  public void speak() { System.out.println("Woof!"); }
}
public class MyClass {
  public static void main(String args[]) {
    Dog dog = new Dog();
    dog.speak();
  }
}
```

optional annotation

abstract method

implementation

```
Woof!
```

```
@FunctionalInterface
interface Animal {
  public void speak();
}

public class MyClass {
  public static void main(String args[]) {
    Animal dog = new Animal() {
      public void speak() { System.out.println("Woof!"); }
    };
    dog.speak();
  }
}
```

we have defined a reference of interface `Animal` with a specific implementation of `speak()`

```
Woof!
```

```
// same thing, but using the syntax of lambda expression

@FunctionalInterface
interface Animal {
  public void speak();
}


public class MyClass {
  public static void main(String args[]) {
    Animal dog = () -> System.out.println("Woof!");
    dog.speak();
  }
}
```

```
Animal dog = new Animal() {
  public void speak() { System.out.println("Woof!"); }
};
```

implementation of abstract method

in a functional interface

method parameters

(in this case empty)

implementation

(one line, or block of code { })

```
// different implementations in a single class
@FunctionalInterface
interface Animal {
  public void speak();
}
public class MyClass {
  public static void main(String args[]) {
    Animal dogImplementation = () -> System.out.println("Woof!");
    Animal catImplementation = () -> System.out.println("Meow!");
    Animal cowImplementation = () -> System.out.println("Moo!");
    dogImplementation.speak();
    catImplementation.speak();
    cowImplementation.speak();
  }
}
```

```
// abstract method can have parameters and return type, e.g.
@FunctionalInterface
interface Multiplicable {
  public int multiply (int a, int b);
}


public class MyClass {
  public static void main(String[] args) {
    Multiplicable myImplementation = (a, b) -> a * b;
    int result = myImplementation.multiply(3, 4);
    System.out.println(result);
  }
}
```

abstract method

implementation

Allowed variations in lambda syntax

one parameter

```
n -> 2*n

(n) -> 2*n

(int n) -> 2*n

n -> { return 2*n; }

(n) -> { return 2*n; }

(int n) -> { return 2*n; }
```

more parameters

```
(a, b) -> a*b

(int a, int b) -> a*b

(a, b) -> { return a*b; }

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