JAVA 8

Interfaces and lambdas

INTERFACE

Default method

```
interface Math {
    default int sum(int a, int b) {
        return a + b;
    }
}
SimpleMath implements Math {
    // no override
}
FancyMath implements Math {
    @Override
    public int sum(int a, int b) {
        // with call to interface implem.
        return Math.super.sum(a, b) * 2;
    }
}
int m1 = new SimpleMath().sum(1, 2); // 3
int m2 = new FancyMath().sum(1, 2); // 6
```

Static method

```
interface Math {
    static int mult(int a, int b) {
        return a * b;
    }
}
int m = Math.mult(5, 6); // 30
```

Functional interface

An interface with only one abstract method is called a functional interface. @FunctionalInterface is optional.

```
@FunctionalInterface
interface Math {
    int minus(int a, int b);

    default int sum(int a, int b) {
        return a + b;
    }

    static int mult(int a, int b) {
        return a * b;
    }
}
```

All lambdas implement functional interfaces.

LAMBDA

Expression

```
// Lambda without parameter
() -> "Hello, World!";

// Calculate a mod 10
(int a) -> a % 10;
a -> a % 10;

// Lambda can have multiple statements.
// Use braces and return in that case
(s1, s2) -> {
    String s1Upper = s1.toUpperCase();
    return s1Upper.concat(s2);
}
```

Method references

```
// Instance method (general)
// i -> i.longValue();
Integer::longValue

// Instance method (specific)
// () -> i.longValue();
i::longValue

// Static method
// s -> Integer.valueOf(s);
Integer::valueOf
```

Useful functional interfaces

Interface	Lambda Expression
Function<A,B>	A -> B;
Predicate<A>	A -> Boolean;
UnaryOperator<A>	A -> A;
Consumer<A>	A -> void;
Supplier<B>	() -> A;
Runnable	() -> void;
BiFunction<A,B,C>	(A, B) -> C;

(see java.util.function for more predefined interfaces)