



Functional Interface & Lambda Expressions

Functional Interface

- Known as SAM (Single Abstract Method) interface
- There is only one abstract method in interface.
- `@FunctionalInterface` is applicable (Optional)
- Effectively acts as a function

Functional Interface

```
@FunctionalInterface  
public interface MyInterface {  
  
    void function(int a);  
  
}
```

Define functional interface

Abstract method

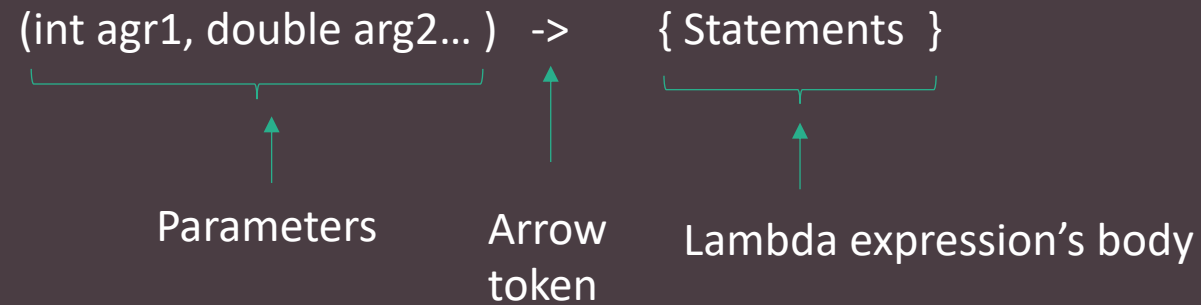
Lambda Expressions

- A function with no name and an identifier
- Can be defined in the place where they are needed
- Expresses the instances of a functional Interface
- Can be assigned to the instance of functional interface

Syntax of Lambda Expressions

`(int agr1, double arg2...) -> { Statements }`

Parameters Arrow token Lambda expression's body



The diagram illustrates the syntax of a lambda expression: `(int agr1, double arg2...) -> { Statements }`. A teal bracket under the parameter list `(int agr1, double arg2...)` is labeled "Parameters" with an upward arrow. A teal arrow points to the `->` token, labeled "Arrow token". Another teal bracket under the body `{ Statements }` is labeled "Lambda expression's body" with an upward arrow.

`(parameters) -> expression`

`(parameters) -> { statements; }`

`() -> expression`

Custom Functional interface

```
@FunctionalInterface
interface TwoStrings {
    String function(String str1, String str2);
}

public class Test{
    public static void main(String[] args) {

        TwoStrings merge = (str1, str2) -> { return str1+str2; };

        String result = merge.function("Cybertek", "School");
    }
}
```

← Abstract method of functional interface

← Implementation of the abstract method in functional interface

Functional interface implementation

```
@FunctionalInterface
interface TwoStrings {
    String function(String str1, String str2);
}

public class Test{
    public static void main(String[] args) {

        TwoStrings longestString = (s1, s2) -> {
            if(s1.length() > s2.length()){
                return s1;
            }
            return s2;
        };

        String result = longestString.function("Java", "Python");
    }
}
```

Abstract method of functional interface

Implementation of the abstract method in functional interface

Custom Functional interface

```
@FunctionalInterface
public interface Data<T> {

    T accept(T t);

}
```

Type can be any type (generic type)

Abstract method of functional interface

```
Data<String> firstThreeChars = (str) -> {
    return str.substring(0, 3);
};
```

Implementation of the abstract method in functional interface

Custom Functional interface

```
@FunctionalInterface
public interface Data<T> {

    T accept(T t);

}
```

Type can be any type (generic type)

Abstract method of functional interface

```
Data<String> reverse = str -> {
    String r = "";
    for(int i = str.length()-1; i>=0; i--)
        r += str.charAt(i);
    return r;
};
```

Implementation of the abstract method in functional interface

Build in Functional Interfaces

- Predicate
- Consumer
- Function

Functional interface: Predicate

```
@FunctionalInterface
public interface Predicate<T> {

    Evaluates this predicate on the given argument.
    Params: t – the input argument
    Returns: true if the input argument matches the predicate,
            false

    boolean test(T t);
```

Abstract method of functional interface

```
Predicate<Integer> oddNumbers = p -> (p % 2 != 0);
```

Implementation of the abstract method

```
Predicate<String> palindrome = p -> {
    String reverse = "";
    for(int i = p.length()-1; i >= 0; i--){
        reverse += p.charAt(i);
    }
    return reverse.equalsIgnoreCase(p);
};
```

Implementation of the abstract method

Functional interface: Consumer

```
@FunctionalInterface
public interface Consumer<T> {

    Performs this operation on the given argument.
    Params: t - the input argument

    void accept(T t);
```

Abstract method of functional interface

```
Consumer<String> printEach = p -> {
    for (int i = 0; i < p.length(); i++) {
        System.out.println(p.charAt(i));
    }
};
```

Implementation of the abstract method

```
printEach.accept(t: "Cybertek");
```

Calling the abstract method

Functional interface: Function

```
@FunctionalInterface  
public interface Function<T, R> {
```

Applies this function to the given argument.

Params: t – the function argument

Returns: the function result

```
R apply(T t);
```

Abstract method of functional interface

```
Function<String, Boolean> isPalindrome = str -> {  
    String reverse = "";  
    for(int i = str.length()-1; i >= 0 ; i--){  
        reverse += str.charAt(i);  
    }  
    return reverse.equalsIgnoreCase(str);  
};
```

Implementation of the abstract method