Functional Interface & Lambda Expressions

Functional Interface

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

Cybertek

Functional Interface

```
@FunctionalInterface
public interface MyInterface {

    void function(int a);

}
```

Define functional interface

Abstract method

Cybertek

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

Cybertek

Syntax of Lambda Expressions

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

Parameters

Arrow token

Lambda expression's body

(parameters) -> expression
(parameters) -> { statements; }
() -> expression

Cybertek

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

Cybertek

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

Cybertek

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

Cybertek

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

Cybertek

Build in Functional Interfaces

- Predicate
- Consumer
- Function

Cybertek

Functional interface: Predicate

```java
@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

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

Implementation of the abstract method

```java
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

Cybertek

Functional interface: Consumer

```java
@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

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

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

Implementation of the abstract method

Calling the abstract method

Cybertek

Functional interface: Function

```java
@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

```java
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

Cybertek