Functional interface**

Interface with only one abstract method

abstract - unimplemented

Lambdas are blocks of code used to implement the single abstract method defined by a functional interface

use **@FunctionaInterface** annotation as a convention(Good practice)

```
@FunctionaInterface
public interface Walkable{

• public void walk();

}
```

Main

```
Walakable aBlockOfCode = () -> { //implementation of abstract method walk()
here
                                 };

aBlockOfCode.walk();    // this invokes above defined Lambda  () -> { }
```

`Walkable` type is an interface, so `aBlockOfCode` is variable reference to `Walkable` interface.

Another important point

```
@FunctionalInterface
public interface Calculate{
  public int compute(int arg1, int arg2);

}
```

Main

```
Calculate cal = (a, b) -> a+b;


cal.compute(4, 7);  // invoke the above defined Lambda (a,b) -> a+b;
```

we dont even need a return statement `(a,b) -> return a+b;`

Lambda expression is smart enough to know the return is an `int`

Main

```
Calculate dv = (a,b) -> { if (a == 0) return 0;
                              return a/b;
                        };

dv.compute(10,5);//invoke the above defined Lambda
                //  (a,b) -> { if (a == 0) return 0;
                //               return a/b;
                //            }
```

This Lmbda is also using the Calculate functional interface as it accepts 2 arguments implemented different lambda for same functional interface

There are plenty of built in functional intrefaces we can use in our program provided by java

Can we make functional interfaces Generic? absolutely

```
public interface myGenericInterface<T,T> {
  public T myGenric1ArgMethod(T t1,T t2);
}

Calculate<Integer> dv = (a,b) -> { if (a == 0) return 0;
                              return a/b;
                        };
```

```
@FunctionalInterface
public interface MyGenericInterface<T> {
    public T compute(T t);
}
```

```
public interface MyGenericInterface2<T> {
    public T compute(T arg1, T arg2);
}
```

```
    public static void main(String[] args) {

        MyGenericInterface<Integer> dv = (Integer n) ->{   return n+n;  };
        System.out.println(dv.compute(34));


        MyGenericInterface2<Integer> dv2 =
                                (Integer n1, Integer n2) -> {
                                        return n1 + n2;
                                };
        System.out.println(dv2.compute(23,12));
    }
}
```

Generic Interface

```
public class Car {

    String make,model,color;
    int price;

    public Car(String make,String model,String color,int price){
        this.make = make;
        this.model = model;
        this.color = color;
        this.price = price;
    }

    public void setMake(String make){
        this.make = make;
    }
    public void setModel(String model){
        this.model = model;
    }
    public void setColor(String color){
        this.color = color;
    }
    public void setPrice(int price){
        this.price = price;
    }
    public String getMake(){
        return make;
    }
    public String getModel(){
        return model;
```

```
    }
    public String getColor(){
        return color;
    }
    public int getPrice(){
        return price;
    }

    public void printCar(){
        System.out.print(this);         //this prints the object created
    }

    @Override
    public String toString(){
        return "Car [make=" + make +", model="+ model +", color=" + color +
", price=" + price + "]";
    }
}
```

```
@FunctionalInterface
public interface Condition<T> {
    public boolean test(T t);

}
```

```
    public static void main(String[] args) {

            List<Car> cars = Arrays.asList(
                new Car("Honda", "Accord", "Red", 22300),
                new Car("Toyota","Land Cruiser","White", 17700),
                new Car("Nissan","Sylpy","Blue", 18000)
            );


            System.out.println("\nprinting cars in 17900 - 25000 range...");
            printCars(cars, (c) -> c.getPrice() >= 17900 && c.getPrice() <=
25000);
/*
For singlre line we don't need the return if it was multiple lines.
The c is from test(c) -> c.getPrice() >= 17900 && c.getPrice() <= 25000.
Test(Car c) method is the only method in the functinal interface and it mapps
to this lambda.
*/

            System.out.println("\nprinting Blue cars...");
            printCars(cars, (c) -> c.getColor().equals("Blue"));
```

```java
        }// end of main


//-------------------------------------------------------------
// printcars(cars, condition) enhanced for loop
    public static void printCars(List<Car> cars, Condition<Car> condition){
        for(Car c: cars){
            if(condition.test(c)){
                c.printCar();
            }
        }
    }
```

Using inbuilt JAVA generic predicate method

It has a method test()

```java
App.java    Predicate.class
 * whose functional method is {@link #test(Object)}.
 *
 * @param <T> the type of the input to the predicate
 *
 * @since 1.8
 */
@FunctionalInterface
public interface Predicate<T> {

    /**
     * Evaluates this predicate on the given argument.
     *
     * @param t the input argument
     * @return {@code true} if the input argument matches the predicate,
     * otherwise {@code false}
     */
    boolean test(T t);

    /**
     * Returns a composed predicate that represents a short-circuiting logical
     * AND of this predicate and another.  When evaluating the composed
```

```java
public static void printCars(List<Car> cars, Predicate<Car> condition){
    for(Car c: cars){
        if(condition.test(c)){
            c.printCar();
        }
    }
}
```

App.java × Condition.java

src > lambda > practical > App.java > Language Support for Java(TM) by Red Hat > App > main(String[])

```java
import java.util.function.Predicate;

You, 17 minutes ago | 1 author (You)
public class App {
    Run | Debug
    public static void main(String[] args) {

            List<Car> cars = Arrays.asList(
                new Car("Honda", "Accord", "Red", 22300),
                new Car("Toyota","Land Cruiser","White", 17700),
                new Car("Nissan","Sylpy","Blue", 18000)
            );


            /*version 1.0
            printCarsByColor(cars, "White");
            printCarsPriceRange(cars, 17900, 25000);      You, an hour ago • printCars lambda1
            */

            System.out.println();



            //Lambda version 2.0
            System.out.println("\nprinting cars in 17900 - 25000 range...");
            printCars(cars, (c) -> c.getPrice() >= 17900 && c.getPrice() <= 25000);
            // for singlre line we don't need the return if it was multiple lines we need return and {}
            // the c is from test(c) -> c.getPrice() >= 17900 && c.getPrice() <= 25000
            //test(Car c) method is the only method in the functinal interface and it mapps to this lambda

            //Lambda version 2.0
            System.out.println("\nprinting Blue cars...");
            printCars(cars, (c) -> c.getColor().equals("Blue"));

        }// end of main




//-------------------------------------------



    //Predicate is the inbuilt Generic FunctionalInterface provided by java that has a method returns a boolean
    public static void printCars(List<Car> cars, Predicate<Car> condition){
        for(Car c: cars){
            if(condition.test(c)){
                c.printCar();
            }
```

Screenshot

Secure https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

Apps | CTS - iApp | Startup bros | Amazon Seller Cent... | ›WP Admin Log In | staging-eliminatec... | Online Java Courses | Integrations Panel |... | eliminate code fear... | 3 Simple Day Tradi...

predicate 9/13

Consumer<T>	Represents an operation that accepts a single input argument and returns no result.
DoubleBinaryOperator	Represents an operation upon two double-valued operands and producing a double-valued result.
DoubleConsumer	Represents an operation that accepts a single double-valued argument and returns no result.
DoubleFunction<R>	Represents a function that accepts a double-valued argument and produces a result.
DoublePredicate	Represents a predicate (boolean-valued function) of one double-valued argument.
DoubleSupplier	Represents a supplier of double-valued results.
DoubleToIntFunction	Represents a function that accepts a double-valued argument and produces an int-valued result.
DoubleToLongFunction	Represents a function that accepts a double-valued argument and produces a long-valued result.
DoubleUnaryOperator	Represents an operation on a single double-valued operand that produces a double-valued result.
Function<T,R>	Represents a function that accepts one argument and produces a result.
IntBinaryOperator	Represents an operation upon two int-valued operands and producing an int-valued result.
IntConsumer	Represents an operation that accepts a single int-valued argument and returns no result.
IntFunction<R>	Represents a function that accepts an int-valued argument and produces a result.
IntPredicate	Represents a predicate (boolean-valued function) of one int-valued argument.
IntSupplier	Represents a supplier of int-valued results.
IntToDoubleFunction	Represents a function that accepts an int-valued argument and produces a double-valued result.
IntToLongFunction	Represents a function that accepts an int-valued argument and produces a long-valued result.
IntUnaryOperator	Represents an operation on a single int-valued operand that produces an int-valued result.
LongBinaryOperator	Represents an operation upon two long-valued operands and producing a long-valued result.
LongConsumer	Represents an operation that accepts a single long-valued argument and returns no result.

Secure https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

Apps | CTS - iApp | Startup bros | Amazon Seller Cent... | ›WP Admin Log In | staging-eliminatec... | Online Java Courses | Integrations Panel |... | eliminate code fear... | 3 Simple Day Tradi...

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.util.function

Interface Function<T,R>

Type Parameters:

T - the type of the input to the function

R - the type of the result of the function

All Known Subinterfaces:

UnaryOperator<T>

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

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

Represents a function that accepts one argument and produces a result.

This is a functional interface whose functional method is apply(Object).

Since:

1.8

Method Summary

All Methods | Static Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description

```java
public static void main(String[] args) {

    List<Car> cars = Arrays.asList(
            new Car("Honda", "Accord", "Red", 22300),
            new Car("Honda", "Civic", "Blue", 17700),
            new Car("Toyota", "Land Cruiser", "White", 48500),
            new Car("Toyota", "Corolla", "Black", 16200),
            new Car("Toyota", "Camry", "Blue", 24000),
            new Car("Nissan", "Sentra", "White", 17300),
            new Car("Mitsubishi", "Lancer", "White", 20000),
            new Car("Jeep", "Wrangler", "Red", 24500)
            );

        Function<Car, String> priceAndColor = (c) -> {
            return " price = " + c.getPrice() + " color = "+ c.getColor();
        };
```

```java
        Function<Car, String> priceAndColor = (c) -> " price = " + c.getPrice() + " color = "+ c.ge
        String stringCar = priceAndColor.apply(cars.get(0));
        System.out.println(stringCar);
```

```
Declaration  Console
<terminated> App (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk-10.0.1.jdk/Contents/Home/bin/java (Jul 17, 2018, 4:26:37 PM)
price = 22300 color = Red
```

Udemy - The Complete Java Certification Course (Updated for 2018) Imitiaz Ahmad/18. Lambda Expressions and the Streams API/1. Functional Interfaces and Lambdas.mp4*

Interface definition

1. 1.

   a point where two systems, subjects, organizations, etc. meet and interact.

   "the **interface between** accountancy and the law"

2. 2.

   a device or program enabling a user to communicate with a computer.

   - a device or program for connecting two items of hardware or software so that they can be operated jointly or communicate with each other.