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

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

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

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

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

## **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() <=</pre>
25000);
For singlre line we don't need the return if it was multiple lines.
The c is from test(c) \rightarrow c.getPrice() \Rightarrow 17900 && c.getPrice() \Leftarrow 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"));
```

```
// 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();
        }
    }
}
```

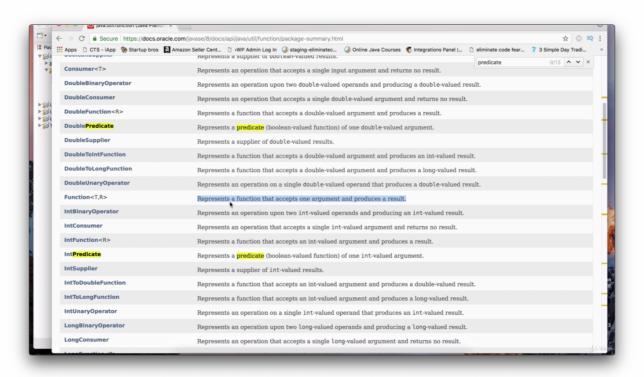
## Using inbuilt JAVA generic predicate method

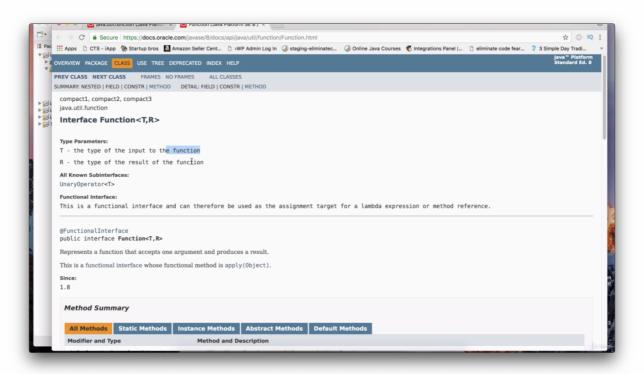
It has a method test()

```
Predicate.class 🖾
App.java
 * whose functional method is {@link #test(Object)}.
 34 *
 35 * @param <T> the type of the input to the predicate
  36
 37 * @since 1.8
 38 */
  39 @FunctionalInterface
 40 public interface Predicate<T> {
 41
420
          * Evaluates this predicate on the given argument.
 43
 44
         * @param t the input argument
 45
 46
         * @return {@code true} if the input argument matches the predicate,
 47
          * otherwise {@code false}
 48
         boolean test(T t);
 49
 50
 51⊖
         * Returns a composed predicate that represents a short-circuiting logical
 52
          * AND of this predicate and another. When evaluatina the composed
```

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

```
src > lambda > practical > 0 App.java > Language Support for Java(TM) by Red Hat > 😭 App > 😚 main(String[])
Q
               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)
                              System.out.println();
         23
24
                              System.out.println("\nprinting cars in 17900 - 25000 range...");
                              printCars(cars, (c) → c.getPrice() >= 17900 && c.getPrice() <= 25000);</pre>
                              // for singlre line we don't need the return if it was multiple lines we need return and {} // the c is from test(c) \rightarrow c.getPrice() >= 17900 \&\& c.getPrice() <= 25000
                              System.out.println("\nprinting Blue cars...");
                              printCars(cars, (c) -> c.getColor().equals("Blue"));
                         // end of main
                    public static void printCars(List<Car> cars, Predicate<Car> condition){
                         for(Car c: cars){
                                 (condition.test(c)){
               Screenshot
                                  c.printCar();
```





```
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();
              };
              Function<Car, String> priceAndColor = (c) -> " price = " + c.getPrice() + " color = "+ c.ge
24
              String stringCar = priceAndColor.apply(cars.get(0));
 25
              System.out.println(stringCar);
 26
<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.