**Built in functional interfaces in java**

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

**1. Predicate and BiPredicate**

**2. Function and BiFunction**

**3. UnaryOperator and BinaryOperator**

**4. Consumer and BiConsumer**

**5. Supplier**

**java.util.function.Predicate**

This functional interface Represents a predicate (boolean-valued function) of one argument.

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

It has Three default methods:

* default Predicate<T> and(Predicate<? super T> other)
* default Predicate<T> negate()
* default Predicate<T> or(Predicate<? super T> other)

It has one static method:

* static <T> Predicate<T> isEqual(Object targetRef)

One abstract method:

* boolean test(T t)

Evaluates this predicate on the given argument.

To this method we can assign a lambda expression or method reference

PredicateTestCase.java

import java.util.function.Predicate;

public class PredicateTestCase{

public static void main(String[] args){

Predicate<Integer> isEven = (x) -> (x%2) == 0;

Predicate<Integer> isOdd = (x) -> (x%2) == 1;

int a = 5;

int b = 6;

System.out.println( isEven.test( a ) ); //false

System.out.println( isEven.test( b ) ); //true

System.out.println( isOdd.test( a ) ); //true

System.out.println( isOdd.test( b ) ); //false

}

}

PredicateTestCase2.java

import java.util.function.Predicate;

public class PredicateTestCase2{

public static void main(String[] args){

Predicate<Integer> isEven = PredicateTestCase2::isEvenNum;

Predicate<Integer> isOdd = PredicateTestCase2::isOddNum;

int a = 5;

int b = 6;

System.out.println( isEven.test( a ) ); //false

System.out.println( isEven.test( b ) ); //true

System.out.println( isOdd.test( a ) ); //true

System.out.println( isOdd.test( b ) ); //false

}

public static boolean isEvenNum(int n){

if(n%2==0) return true;

return false;

}

public static boolean isOddNum(int n){

return n%2==1;

}

}

PredicateTestCase3.java

**package** com.satya.java8ex.predicate;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.function.Predicate;

**class** Singer{

**private** String singerName;

**private** String language;

**public** Singer() {

}

**public** Singer(String name,String lan) {

**this**.singerName = name;

**this**.language = lan;

}

**public** String getSingerName() {

**return** singerName;

}

**public** String getLanguage() {

**return** language;

}

@Override

**public** String toString() {

**return** "{"+singerName + "," + language + "}";

}

}

**public** **class** PredicateTestCase3 {

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

List<Singer> singers = **new** ArrayList<>();

singers.add(**new** Singer("balu","telugu"));

singers.add(**new** Singer("surya","tamil"));

singers.add(**new** Singer("bindu","telugu"));

singers.add(**new** Singer("chitra","tamil"));

*display*(singers);

System.***out***.println("-----");

*display*(singers,(s)->s.getLanguage().equals("tamil"));

System.***out***.println("-------");

*display*(singers,(s)->s.getLanguage().equals("telugu"));

System.***out***.println("-------");

*display*(singers,(s)-> **true**);

}

**public** **static** **void** display(List<Singer> list) {

**for**(Singer e:list)

System.***out***.println(e);

}

**public** **static** **void** display(List<Singer> list,Predicate<Singer> condition) {

**for**(Singer e:list) {

**if**(condition.test(e))

System.***out***.println(e);

}

}

}

**PredicateTestCase4.java**

**package** com.satya.java8ex.predicate;

**import** java.util.function.Predicate;

**public** **class** PredicateTestCase4 {

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

Predicate<Integer> testEven = PredicateTestCase4::*isEven*;

Predicate<Integer> testPrime = PredicateTestCase4::*isPrime*;

**for**(**int** i=1;i<=100;i++) {

**if**( testEven.negate().and(testPrime).test(i))

System.***out***.println(i);

}

}

**static** **boolean** isEven(**int** a) {

**return** a%2==0;

}

**static** **boolean** isPrime(**int** a) {

**if**(a==0) **return** **false**;

**for**(**int** n=2;n<=a/2;n++) {

**if**(a%n==0) **return** **false**;

}

**return** **true**;

}

}

**BiPredicate**

BiPredicateTestCase1.java

**package** com.satya;

**import** java.util.function.BiPredicate;

**class** Book{

**private** String bookTitle;

**private** **double** bookPrice;

**public** Book() {

}

**public** Book(String bookTitle, **double** bookPrice) {

**super**();

**this**.bookTitle = bookTitle;

**this**.bookPrice = bookPrice;

}

**public** String getBookTitle() {

**return** bookTitle;

}

**public** **void** setBookTitle(String bookTitle) {

**this**.bookTitle = bookTitle;

}

**public** **double** getBookPrice() {

**return** bookPrice;

}

**public** **void** setBookPrice(**double** bookPrice) {

**this**.bookPrice = bookPrice;

}

@Override

**public** String toString() {

**return** "Book [bookTitle=" + bookTitle + ", bookPrice=" + bookPrice + "]";

}

}

**public** **class** BiPredicateTestCase1 {

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

Book b1 = **new** Book("spring security", 300);

Book b2 = **new** Book("spring security", 300);

BiPredicate<Integer, Integer> isEqualInts = (x,y) -> x==y;

BiPredicate<Book,Book> isEqualBooks = (bo1,bo2) -> bo1.getBookTitle().equalsIgnoreCase(bo2.getBookTitle());

System.***out***.println(isEqualInts.test(2,2)); // true

System.***out***.println(isEqualBooks.test(b1,b2)); // true

}

}

**package** com.satya;

**public** **class** Event {

**private** **int** id;

**private** String name;

@Override

**public** String toString() {

**return** "Event [id=" + id + ", name=" + name + "]";

}

**public** Event(**int** id, String name) {

**super**();

**this**.id = id;

**this**.name = name;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**package** com.satya;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.ListIterator;

**import** java.util.function.BiPredicate;

**public** **class** BiPredicateTestCase2 {

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

List<Event> events = **new** ArrayList<Event>();

events.add(**new** Event(4,"decode"));

events.add(**new** Event(7,"encode"));

events.add(**new** Event(9,"encrypt"));

events.add(**new** Event(10,"decrypt"));

Event e = **new** Event(10, "double encrypt");

System.***out***.println(events);

*updateEvent*(events, e,(e1,e2)->e1.getId()==e2.getId());

System.***out***.println(events);

}

**public** **static** **void** updateEvent(List<Event> existingEvents, Event newEvent, BiPredicate<Event, Event> isIdEqualEvent) {

ListIterator<Event> ite = existingEvents.listIterator();

**while**(ite.hasNext()) {

Event e = ite.next();

**if**(isIdEqualEvent.test(e, newEvent)) {

ite.remove();

ite.add(newEvent);

//events.add(event); concurrent modification exception

}

}

}

}

**java.util.function.Function**

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

R apply(T t)

Applies this function to the given argument.

**package** com.satya;

**import** java.util.function.Function;

**public** **class** FunctionalTestCase{

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

Function <Integer, Integer> incrementer = (v) -> ++v;

Integer i = incrementer.apply(5);

System.***out***.println(i);

Function<Integer, Integer> decrementer = FunctionalTestCase::*decrement*;

**int** x = decrementer.apply(5);

System.***out***.println(x);

*doItAndPrint*(incrementer,10);

*doItAndPrint*(decrementer,10);

}

**static** **int** decrement(**int** a){

a--;

**return** a;

}

**public** **static** **void** doItAndPrint(Function<Integer,Integer> something,**int** a){

System.***out***.println(something.apply(a));

}

}

**package** com.satya;

**import** java.util.function.Function;

**public** **class** FunctionalTestCase1 {

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

Function<Integer, Integer> incrementer = (v) -> {

System.***out***.println("lambda expression inc executed");

**return** ++v;

};

Integer i = incrementer.apply(5);

System.***out***.println(i);

Function<Integer, Integer> decrementer = FunctionalTestCase1::*decrement*;

**int** x = decrementer.apply(5);

System.***out***.println(x);

*doItAndPrint*(incrementer, 10);

*doItAndPrint*(decrementer, 10);

**int** y = incrementer.compose(decrementer).apply(10);// first dec, then inc

System.***out***.println("y=" + y);

**int** z = incrementer.andThen(decrementer).apply(22); // first inc, then dec

System.***out***.println(z);

}

**static** **int** decrement(**int** a) {

System.***out***.println("decrement executed");

a--;

**return** a;

}

**public** **static** **void** doItAndPrint(Function<Integer, Integer> something, **int** a) {

System.***out***.println(something.apply(a));

}

}

**package** com.satya;

**import** java.util.function.Function;

**public** **class** FunctionTestCase2 {

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

Function<Integer, Integer> square = FunctionTestCase2::*findSquare*;

Function<String, Integer> length = FunctionTestCase2::*findLength*;

**int** a = square.apply(3);

System.***out***.println(a);

**int** b = length.apply("Satya");

System.***out***.println(b);

}

**public** **static** **int** findSquare(**int** n) {

**return** n \* n;

}

**public** **static** **int** findLength(String s) {

**return** s.length();

}

}

**package** com.satya;

**import** java.util.function.Function;

**public** **class** FunctionTestCase3 {

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

Function<Integer, Integer> square = x -> x \* x;

Function<String, Integer> length = x -> x.length();

**int** a = square.apply(3);

System.***out***.println(a);

**int** b = length.apply("Satya");

System.***out***.println(b);

}

}

**package** com.satya;

**import** java.util.function.Function;

**public** **class** FunctionTestCase4 {

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

Function<Integer, Integer> square = x -> x \* x;

Function<String, Integer> length = x -> x.length();

**int** b = square.compose(length).apply("Satya");

System.***out***.println(b);

}

}

**package** com.satya;

**import** java.util.function.Function;

**public** **class** FunctionTestCase5 {

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

Function<Integer, Integer> square = x -> x \* x;

Function<String, Integer> length = x -> x.length();

**int** b = length.andThen(square).apply("Satya");

System.***out***.println(b);

}

}

**package** com.satya;

**import** java.util.Arrays;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.function.Function;

**class** Student {

**private** **int** rollNum;

**private** String name;

**private** String course;

Student() {

}

Student(**int** r, String n, String c) {

rollNum = r;

name = n;

course = c;

}

@Override

**public** String toString() {

**return** "Student [rollNum=" + rollNum + ", name=" + name + ", course=" + course + "]";

}

**public** String getCourse() {

**return** course;

}

**public** **int** getRollNum() {

**return** rollNum;

}

**private** String getName() {

**return** name;

}

}

**public** **class** FunctionTestCase6 {

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

Student s1 = **new** Student(2, "satya", "java");

Student s2 = **new** Student(2, "babu", ".net");

Student s3 = **new** Student(2, "vinod", "phython");

List<String> list = Arrays.*asList*("core java", "adv java", "spring boot", "java security");

List<Student> studentList = Arrays.*asList*(s1, s2, s3);

// lambda

Map<String, Integer> map1 = *convertListToMap*(list, x -> x.length());

System.***out***.println(map1);

// method reference

Map<String, Integer> map2 = *convertListToMap*(list, FunctionTestCase6::*getLength*);

System.***out***.println(map2);

Map<Student, String> studentCourseMap = *convertListToMap*(studentList, s -> s.getCourse());

System.***out***.println(studentCourseMap);

Map<Student, Integer> studentRollNumMap = *convertListToMap*(studentList, s -> s.getRollNum());

System.***out***.println(studentRollNumMap);

}

**public** **static** <T, R> Map<T, R> convertListToMap(List<T> list, Function<T, R> func) {

Map<T, R> result = **new** HashMap<>();

**for** (T t : list) {

result.put(t, func.apply(t));

}

**return** result;

}

**public** **static** Integer getLength(String str) {

**return** str.length();

}

}

**package** com.satya;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.function.Function;

**public** **class** FunctionTestCase7 {

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

Function<Integer, Integer> identityFunction = Function.*identity*();

Function<Integer, Integer> intFunction = e -> e;

Function<String, String> idFunction = FunctionTestCase7::*getSame*;

System.***out***.println(identityFunction.apply(10)); // 10

System.***out***.println(intFunction.apply(10)); // 10

System.***out***.println(idFunction.apply("ramu"));// ramu

List<String> names = Arrays.*asList*("raju", "vonod rao", "vinaykumar", "Vijay", "Anand Verma");

System.***out***.println("--------displaying each value");

names.stream().forEach(System.***out***::println);

// Just for example

System.***out***.println("----- Function.identity() -----");

names.stream().map(Function.*identity*()).forEach(System.***out***::println);

System.***out***.println("----- Function(e-> e) -----");

names.stream().map(e -> e).forEach(System.***out***::println);

}

**public** **static** String getSame(String s) {

**return** s;

}

}

**BiFunction:**

**package** com.satya;

**import** java.util.function.BiFunction;

**public** **class** BiFunctionTestCase1 {

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

BiFunction<Integer, Integer, Integer> addFun = (x,y) -> x+y;

BiFunction<Integer, Integer,Double> divFun = (x,y) -> (x\*1.0)/(y);

System.***out***.println(addFun.apply(2, 9));

System.***out***.println(divFun.apply(7, 2));

}

}

**package** com.satya;

**import** java.util.function.BiFunction;

**public** **class** BiFunctionTestCase2 {

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

BiFunctionTestCase2 b = **new** BiFunctionTestCase2();

BiFunction<Integer, Integer, Integer> addFun = b::addition;

BiFunction<Integer, Integer, Double> divFun = b::division;

System.***out***.println(addFun.apply(2, 9));

System.***out***.println(divFun.apply(7, 2));

}

**int** addition(**int** a, **int** b) {

**return** a+b;

}

**double** division(**int** a, **int** b) {

**return** (a\*1.0)/b;

}

}

**3. UnaryOperator and BinaryOperator**

**package** com.satya.java8ex.operator;

**import** java.util.function.UnaryOperator;

**public** **class** UnaryOperatorTestCase {

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

UnaryOperator<Integer> preIncrement = (x) -> ++x;

UnaryOperator<Integer> postIncrement = (x) -> x++;

**int** a = 23, b = 23;

System.***out***.println(preIncrement.apply(a));

System.***out***.println(postIncrement.apply(b));

System.***out***.println(a);

System.***out***.println(b);

a = preIncrement.apply(a);

b = postIncrement.apply(b);

System.***out***.println(a);

System.***out***.println(b);

}

}

**package** com.satya.java8ex.operator;

**import** java.util.function.BinaryOperator;

**public** **class** BinaryOperatorTestCase {

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

BinaryOperator<Integer> add = (x, y) -> x + y;

System.***out***.println(add.apply(2, 3));

}

}

**4. Consumer and BiConsumer**

**package** com.satya.java8ex.consumer;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.function.Consumer;

**class** Student {

**private** **int** rollNum;

**private** String name;

**private** **double** avgMarks;

**private** String branch;

**public** String getBranch() {

**return** branch;

}

**public** **double** getAvgMarks() {

**return** avgMarks;

}

**public** **int** getRollNum() {

**return** rollNum;

}

**public** String getName() {

**return** name;

}

**public** Student(**int** rollNum, String name, **double** avgMarks, String branch) {

**super**();

**this**.rollNum = rollNum;

**this**.name = name;

**this**.avgMarks = avgMarks;

**this**.branch = branch;

}

@Override

**public** String toString() {

**return** "Student [rollNum=" + rollNum + ", name=" + name + ", avgMarks=" + avgMarks + ", branch=" + branch + "]";

}

**public** Student() {

}

}

**public** **class** ConsumerTestCase1 {

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

Student s = **new** Student(5, "satya", 56.6, "ece");

List<Student> students = **new** ArrayList<Student>();

students.add(**new** Student(5, "satya", 56.6, "ece"));

students.add(**new** Student(6, "ravi", 67.7, "it"));

students.add(**new** Student(7, "bond", 68.9, "it"));

students.add(**new** Student(8, "ravi", 38.9, "mech"));

students.add(**new** Student(20, "sony", 48.9, "ece"));

students.add(**new** Student(19, "bob", 78.9, "ece"));

students.add(**new** Student(29, "sonu", 78.9, "mech"));

Consumer<Object> printer = (x) -> System.***out***.println(x);

printer.accept("satya");

printer.accept(23);

printer.accept(s);

printer.accept("------");

*iterateAndConsume*(students, (x) -> System.***out***.println(x));

printer.accept("------");

Consumer<Student> onlyFirstClass = ConsumerTestCase1::*printOnlyFirstClas*;

Consumer<Student> onlyIt = (x) -> {

**if** (x.getBranch().contentEquals("it"))

System.***out***.println(x);

};

Consumer<Student> onlyEce = (x) -> {

**if** (x.getBranch().contentEquals("ece"))

System.***out***.println(x);

};

*iterateAndConsume*(students, onlyFirstClass);

printer.accept("------");

*iterateAndConsume*(students, onlyIt);

printer.accept("------");

*iterateAndConsume*(students, onlyIt.andThen(onlyEce));

}

**public** **static** <T> **void** iterateAndConsume(List<T> list, Consumer<T> doIt) {

**for** (T s : list) {

doIt.accept(s);

}

}

**public** **static** **void** printOnlyFirstClas(Student s) {

**if** (s.getAvgMarks() >= 60)

System.***out***.println(s);

}

}

BiConsumerTestCase.java

**package** com.satya.java8ex.consumer;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.function.BiConsumer;

**public** **class** BiConsumerTestCase {

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

BiConsumer<Integer, Integer> printSum = (x,y) -> System.***out***.println(x+y);

printSum.accept(2,3);

List<String> list1 = **new** ArrayList<String>();

list1.add("satya"); list1.add("raja"); list1.add("roja");

List<String> list2 = **new** ArrayList<String>();

list2.add("babu"); list2.add("suma");

System.***out***.println(list1);

BiConsumer<List<String>, List<String>> addToMe = BiConsumerTestCase::*addElements*;

addToMe.accept(list1, list2);

System.***out***.println(list1);

}

**public** **static** <T>**void** addElements(List<T> list1, List<T> list2){

**if**(list1!=**null**&&list2!=**null**) {

**for**(T t:list2)

list1.add(t);

}

}

}

**package** com.satya.java8ex.supplier;

**import** java.util.function.Supplier;

**public** **class** SupplierDemo {

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

Supplier<Double> randomNum = Math::*random*;

System.***out***.println(randomNum.get());

System.***out***.println(randomNum.get());

System.***out***.println(randomNum.get());

}

}

**package** com.satya.java8ex.supplier;

**import** java.util.Calendar;

**import** java.util.function.Supplier;

**public** **class** SupplierDemo2 {

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

Supplier<Calendar> calendarSupplier = Calendar::*getInstance*;

Calendar c = calendarSupplier.get();

System.***out***.println(c.getTime());

}

}

**package** com.satya.java8ex.supplier;

**import** java.util.function.Supplier;

**class** Point {

**private** **int** x;

**private** **int** y;

**public** Point() {

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

@Override

**public** String toString() {

**return** "Point [x=" + x + ", y=" + y + "]";

}

}

**public** **class** SupplierDemo3 {

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

Supplier<Point> pointSupplier = () -> **new** Point();

Point p1 = pointSupplier.get();

Point p2 = pointSupplier.get();

p1.setX(2); p1.setY(3);

p2.setX(4); p2.setY(5);

System.***out***.println(p1);

System.***out***.println(p2);

}

}

**package** com.satya.java8ex.supplier;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.util.function.Supplier;

**public** **class** SupplierDemo4 {

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

Supplier<Connection> conSupplier = () -> {

**try** {

**return** DriverManager.*getConnection*("", "", "");

} **catch** (SQLException sql) {

}

**return** **null**;

};

Connection con = conSupplier.get();

}

}

**package** com.satya.java8ex.supplier;

**import** java.io.PrintStream;

**import** java.util.Calendar;

**import** java.util.function.Supplier;

**public** **class** SupplierDemo5 {

**public** **static** **void** main(String[] args)**throws** Exception {

*dispalyTime*(Calendar::*getInstance*);

*printIt*("satya", () -> System.***out***);

*printIt*("supplier example", () -> System.***out***);

PrintStream ps = **new** PrintStream("c:\\temp\\sample.txt");

*printIt*("supplier example", () -> ps);

ps.close();

}

**public** **static** **void** dispalyTime(Supplier<Calendar> calendarSupplier) {

Calendar c = calendarSupplier.get();

System.***out***.println(c.getTime());

}

**public** **static** **void** printIt(String s, Supplier<PrintStream> printSupplier) {

PrintStream ps = printSupplier.get();

ps.println(s);

}

}