# Java8

Java was object oriented program, so it took more lines of code.

2013, 2014 python comes with functional program; it took less lines of code. So java allowed functional programing.

class Test{

p s int sqrt(int n){

return n\*n;

}

p s v main(String arg[]){

sop(“sqr of 2”+sqrt(4));

sop(“sqr of 7”+sqrt(7));

}

}

class Test{

p s v main(String arg[]){

Function<**Integer**,**Integer**> sqrt = **i**->**i**\***i**;

sop(“sqr of 2”+sqrt.apply(4));

}

}

class Test{

p s v main(String arg[]){

Predicate<**Integer** > p = **i**->i/2==0;

sop(“Even:”+p.test(4));

}

}

### Lambda Calculus

Found it in computer perspective

### x

Function

y x+y+5

we only know input and output

Syntax for this λx. λy. x+y+5

(λx. x+1) 5

=6 function body x+1, give 5 as input

Alan Turing’s Turing machine codes based on this language

λx. λy. x 🡺 true yes x,y are inputs it takes x

λx. λy. y 🡺 false

Abstraction (dealing with idea not real world event)

λx.x

λ 🡺 Begin a Lambda abstraction function head

x 🡺 take one argument, bound to the name ‘x’

. 🡺 Separate function and body

x 🡺 function body (what to do with argument) function body

Sometimes we use \ for λ, -> for .

\x -> x

\x.(\y.x y)

F(f(y)){

X

Return y

}

F(x){

}

….

### Lambda expression

To bring benefits of functional programming into java.

It is an anonymous function.

* No name
* No return type
* No modifiers

1, public void m1(){

sop(“hello”);

}

But need to tell it’s lambda expression **->**

()->{ sop(“hello”);}

If it’s one statement we can remove {}

()->sop(“hello”);

2, public void m1(int a, int b){

Sop(a+b);

}

(int a, int b)->sop(a+b);

If compiler can able to guess the type automatically we can remove int also.

(a, b)->sop(a+b);

3, public int sqrareIt(int n){

return n\*n;

}

(n)->n\*n;

But within {} return is necessary

(n)->{ return n\*n };

If only one parameter we can remove ()

n->n\*n;

4, public int m1(String s){

return s.length();

}

s->s.length();

### Functional Interface (FI)

Bello interfaces in java contain only one method.

Runnable 🡺 run(); // multithread

Comparable 🡺 compareTo(); // collections

Comparator 🡺 compare();

ActionListener 🡺 actionPerfermed() ; // awt swing (UI)

Callable 🡺 call(); // multithread

So single abstract method (SAM) is FI

* Should contain exactly one abstract method
* But can contain any number of static/ default method

interface MyInterface{

public static void display(){ sop(“hello”);} //MyInterface.display();

public default void print(){sop(“hi”);} // can override in child

void call();

}

----------------------------------------------------

n->return n\*n;

n->{return n\*n;};

n->{return n\*n};

n->n\*n;

n->{n\*n};

* Without curly braces we cannot use return keyword. Compiler will consider return value automatically.
* Within curly braces to return, we should use return keyword.

@functionalInterface

Help Compiler: hey you told this gonna be functional interface.

@functionalInterface

interface A{

public void m1();

}

@functionalInterface

interface B extends A{ // it is SAM

}

@functionalInterface

interface B extends A{ // it is SAM m1()

public void m1();

}

@functionalInterface

interface B extends A{ // not SAM m1(),m2()

public void m2();

}

### Use lambda expression

@functionalInterface

interface Adder{

public double add**(int num1, double num2);**

}

class Test{

public static void main(String arg[]){

Adder adder = **(n1, n2)**->n1+n2;

Sop(adder.add());

}

}

**So without FI we can’t use lambda expression.**

But we commonly used this combination

java.util.function

This package contains many commonly used FI.

Lambda expression not going to generate any .class file

At the time of compilation it may converted into private method.

(no relation to anemones-inner-classes)

#### In Multithreading

class MyRunnable implements Runnable{

public void run(){

for(i=0 to 10) sop(“childT”);

}

}

class Test{

main(){

MyRunnable mr = new MyRunnable();

//mr.run(); but r not contain start() to a new thread

Thread t = new Thread(mr);

t.start(); // now onward mainT, t is there

for(i=0 to 10) sop(“mainT”);

}

}

Can’t expect the exact output

λ Expression

class Test{

main(){

Runnable r = ()->{

for (int k = 0; k < 10; k++) System.out.println("child");

};

= is assigning operator 🡸

Thread t = new Thread(t);

t.start();

}

}

#### In Collections

class Test{

main(){

ArrayList<Integer> l = new ArrayList<Integer>();

l.add(20);

l.add(30);

l.add(5);

Sop(l);

}

}

Output: [20,30,5]

Comparator

int compare (Object obj1, Object obj2){

returns –ve iff obj1 has to come before obj2

returns +ve iff obj1 has to come after obj2

returns 0 iff obj1 == obj2

} obj1 obj2 obj2 obj2

To ascending Order

10, 23, 43, 32, 12, 43

class MyComparator implements Comparator {

public int compare(int num1, int num2){

if(num1<num2){

return -1; // yap in this case obj1 need to come first

}

if(num1>num2){

return 1; // yap in this case obj1 need to come first

}

else{

return 0;

}

// return (n1<n2)?-1:(n1>n2)?1:0; turnery operator

}

}

class Test{

main(){

ArrayList<Integer> l = new ArrayList<Integer>();

l.add(20);

l.add(30);

l.add(5);

Collections.sort(l, new Mycomarator());

Sop(l);

}

}

Output: [5,20,30]

λ Expression

class Test{

main(){

ArrayList<Integer> l = new ArrayList<Integer>();

l.add(20);

l.add(30);

l.add(5);

Comparator<Integer> c = (n1, n2)-> (n1<n2)?-1:(n1>n2)?1:0;

Collections.sort(l, c);

Sop(l);

}

}

n1, n2 is Integer type because that Comparator is Integer.

#### In Employee

class Employee{

String name;

int eno;

Employee(String name, int eno){

this.name = name;

this.eno = eno;

}

public String toString(){ // sop(emp) will print values

return name +”,”+ eno;

}

}

class Test{

main(){

ArrayList<Employee> l = new ArrayList< Employee>();

l.add(new Employee(“Tharsi”,232312));

l.add(new Employee(“Nive”,142385));

l.add(new Employee(“Sharu”,321243));

l.add(new Employee(“Uyire”,341225));

Collections.sort(l, (e1, e2)->(e1.eno<e2.eno)?-1: (e1.eno<e2.eno)?1:0);

Sop(l);

Collections.sort(employees, (e1, e2)->e1.name.compareTo(e2.name));

// for sort by name

}

}

Comparator obj is for tells e1 or e2 comes first in the order

Collection.short(list l, **Comparator** c);

When pass FI Type argument we can use this... λ Expression

()-> // public int compare(obj,obj);

### Anonymous Inner classes vs. Lambda Expressions

#### Anonymous class

Thread t = new Thread(){

……..

};

We are writing a class that extends Thread class. (Creating child class for Thread class) and for that child class we are creating an Object.

And this class has no name that’s why it is anonymous class.

Runnable r = new Runnable(){

…….

};

We are creating implementation class for the Runnable Interface, and creating an object for class.

This class is also not has name, so anonymous class.