# JAVA 8
# FEATURES

ChandraSekhar(CS) Baratam

## Java 8 Features

Lambda Expression

Functional Interfaces

Default Methods

Predicates

Functions

Stream API

## Lambda Expression

Functional Programming

anonymous function or closure

```
public void display(){
        sop(“Sandy”);
  }
```

```
()   ->    sop(“Sandy”);
```

```
public void multiply(int a,int b){
            sop(a*b);
      }
```

```
(int a,int b)->sop(a*b)
```

## Why

Less Code

Easy to implement anonymous inner classes

Parameters to methods

## LAMBDA EXPRESSIONS

LAMBDA EXPRESSION HELPS US TO WRITE OUR CODE IN FUNCTIONAL STYLE.

IT PROVIDES A CLEAR AND CONCISE WAY TO IMPLEMENT SAM INTERFACE(SINGLE ABSTRACT METHOD) BY USING AN EXPRESSION.

IT IS VERY USEFUL IN COLLECTION LIBRARY IN WHICH IT HELPS TO ITERATE, FILTER AND EXTRACT DATA.

**WHY USE LAMBDA EXPRESSION**

- TO PROVIDE THE IMPLEMENTATION OF FUNCTIONAL INTERFACE.
- LESS CODING.

## Functional Interfaces

```
interface MyInterface{

        void myMethod();

    }
```

|  |  |
|---|---|
| Runnable | run |
| Comparator | compareTo |

Default Methods

One Abstract Method

```
@FunctionalInterface
interface MyInterface{

        void myMethod();

    }
```

## CREATE A FUNCTIONAL INTERFACE

- IN THE FUNCTIONAL INTERFACE WHENEVER USING @FUNCTIONALINTERFACE THEN IT TAKES ONLY ONE ABSTRACT CLASS,
- IF YOU TAKE MORE THAN ONE INTERFACE THEN IT WILL SHOW ERROR.

## CREATE A 2 INTERFACE WITH @FUNCTIONAL INTERFACE WITH LIVE

```
@FUNCTIONALINTERFACE
PUBLIC INTERFACE A {
PUBLIC VOID M1();
}

@FUNCTIONALINTERFACE
PUBLIC INTERFACE B EXTENDS A {


PUBLIC VOID M1();
}
```

```
PUBLIC CLASS C IMPLEMENTS A {
@OVERRIDE
PUBLIC VOID M1() {
SYSTEM.OUT.PRINTLN("HELLO M1");
}
}
PUBLIC CLASS TEST {

PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

/*A A = NEW C();
A.M1();*/

A A = ()->SYSTEM.OUT.PRINTLN("HELLO M1");

}
```

```
}
```

## BEFORE LAMBDA EXPRESSION

```
PUBLIC INTERFACE COLLEGE {

PUBLIC VOID STUDENT();

}

PUBLIC CLASS TEST {

PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

COLLEGE C = NEW COLLEGE() {

@OVERRIDE
PUBLIC VOID STUDENT() {
        SYSTEM.OUT.PRINTLN("HELLO WORLD");

}
};
C.STUDENT();
}
}
```

## USING LAMBDA EXPRESSION

```
PUBLIC INTERFACE COLLEGE {
        PUBLIC VOID STUDENT();
}



PUBLIC CLASS TEST1 {
PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

COLLEGE C = ()->SYSTEM.OUT.PRINTLN("HELLO STUDENT");
C.STUDENT();
}
}
```

## NO PARAMETER BUT RETURN

```
PUBLIC INTERFACE COLLEGE {
        PUBLIC STRING STUDENT();
}

PUBLIC CLASS TEST1 {
PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

COLLEGE C = ()->{
RETURN "HELLO STUDENT";
};
SYSTEM.OUT.PRINTLN(C.STUDENT());
}
}
```

## SINGLE PARAMETER

```
PUBLIC INTERFACE COLLEGE {
        PUBLIC STRING STUDENT(STRING NAME);
}
PUBLIC CLASS TEST1 {

PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

COLLEGE C = ()->{
RETURN "NAMEHELLO: "+;
};
SYSTEM.OUT.PRINTLN(C.STUDENT("SANDY"));
}
}
```

## MULTIPLE PARAMETERS

```
PUBLIC INTERFACE COLLEGE {
PUBLIC INT FEES(,INT F2);
}


    PUBLIC CLASS TEST1 {


PUBLIC STATIC VOID MAIN(STRING[] ARGS) {


COLLEGE C = (F1, F2) -> (F1 + F2);
SYSTEM.OUT.PRINTLN(CINT F1.FEES(10, 50));


COLLEGE C1 = (F2) -> (F1 INT F1, INT - F2);
SYSTEM.OUT.PRINTLN(C1.FEES(1000, 500));
}
}
```

## WITH OR WITHOUT RETURN KEYWORD

```
PUBLIC INTERFACE COLLEGE {
PUBLIC INT FEES(INT F1,INT F2);
}

   PUBLIC CLASS TEST1 {

          PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
                  //WITHOUT RETURN
          COLLEGE C = (F1, F2) -> (F1 + F2);
          SYSTEM.OUT.PRINTLN(C.FEES(10, 50));
                   //WITH RETURN
          }; COLLEGE C1 = (INT F1, INT F2) ->{
          RETURN (F1 -F2);

          SYSTEM.OUT.PRINTLN(C1.FEES(1000, 500));
          }
}
```

## FOREACH LOOP

```
PUBLIC CLASS FORLOOPCOLL {
        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
                LIST<STRING> L = NEW ARRAYLIST<>();
                L.ADD("SANDY");
                L.ADD("VIJAY");
                L.ADD("PRAVEEN");
                L.ADD("SANTOSH");

                SYSTEM.OUT.PRINTLN(L);
          L.FOREACH(
          (N)->SYSTEM.OUT.PRINTLN(N)
          );
        }
}
```

OUTPUT

```
[SANDY, VIJAY, PRAVEEN, SANTOSH]
SANDY
VIJAY
PRAVEEN
SANTOSH
```

## LAMBDAS USING RUNNABLE INTERFACE

```
PUBLIC CLASS MYRUNNABLE IMPLEMENTS RUNNABLE {

        @OVERRIDE
        PUBLIC VOID RUN() {
                FOR (INT I = 0; I <10; I++) {
                        SYSTEM.OUT.PRINTLN("CHILD METHOD");
                }
        }
}
```

WITHOUT LAMBDA

```
PUBLIC CLASS TEST2 {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
                MYRUNNABLE R = NEW MYRUNNABLE();
                THREAD T = NEW THREAD(R);
                T.START();

                FOR (INT I = 0; I < 10; I++) {
      SYSTEM.OUT.PRINTLN("PARENT METHOD");
                }
        }
}
```

## USING LAMDA

```
PUBLIC CLASS TEST2 {

          PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
                    RUNNABLE R =()->{
                              FOR (INT I = 0; I < 10; I++) {
                                        SYSTEM.OUT.PRINTLN("CHILD METHOD1");
                              }
                    };
                    THREAD T = NEW THREAD(R);
                    T.START();

                    FOR (INT I = 0; I < 10; I++) {
       SYSTEM.OUT.PRINTLN("PARENT METHOD");
                    }
          }
}
```

## WITHOUT LAMBDAS AND ANONYMOUS CLASSES

```
PUBLIC CLASS TEST2 {


        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
        THREAD T = NEW THREAD(NEW RUNNABLE() {
        PUBLIC VOID RUN() {
        FOR (INT I = 0; I <10; I++) {
        SYSTEM.OUT.PRINTLN("CHILD METHOD");
        }
        }
        });
        T.START();
        FOR (INT I = 0; I < 10; I++) {
      SYSTEM.OUT.PRINTLN("PARENT METHOD");
        }
        }
}
```

## LAMBDAS AND ANONYMOUS CLASSES

```
PUBLIC CLASS TEST2 {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
                THREAD T = NEW THREAD(()->{
                        FOR (INT I = 0; I <10; I++) {
                                SYSTEM.OUT.PRINTLN("CHILD METHOD")
                        }
                });
                T.START();
                FOR (INT I = 0; I < 10; I++) {
      SYSTEM.OUT.PRINTLN("PARENT METHOD");
                }
        }
}
```

## DIAMOND PROBLEM AND DEFAULT METHODS

Diamond Problem

$$\begin{tikzcd}
& A & \\
B \arrow[ur] & & C \arrow[ul] \\
& D \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

## DIAMOND PROBLEM AND DEFAULT METHODS

```
PUBLIC INTERFACE A {
DEFAULT VOID M1(){
                    SYSTEM.OUT.PRINTLN("INSIDE INTERFACE A");
          }
}
PUBLIC INTERFACE B {
DEFAULT VOID M1(){
                    SYSTEM.OUT.PRINTLN("INSIDE INTERFACE B")
          }
}
```

## DIAMOND PROBLEM AND DEFAULT METHODS

```
PUBLIC CLASS C IMPLEMENTS A,B {
        PUBLIC VOID M1(){
                SYSTEM.OUT.PRINTLN("INSIDE C");
        }
}
```

//IT WILL SAY AS ERROR WHEN IMPLEMENTS B BUT YOU JUST GO ON, AND PASTE THE METHOD OF M1 AND THE ERROR WILL GO….

## Predicate

Single Argument and returns boolean

```
interface Predicate<T>{
        public boolean test(T t);
}
```

## PREDICATE HANDSON

PREDICATE IS AN FUNCTIONAL INTERFACE.

```
PUBLIC CLASS TEST {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

                PREDICATE<INTEGER> P = I->(I>20);

                SYSTEM.OUT.PRINTLN(P.TEST(10));
                SYSTEM.OUT.PRINTLN(P.TEST(40));
                SYSTEM.OUT.PRINTLN(P.TEST(19));
                SYSTEM.OUT.PRINTLN(P.TEST(21));
        }
}
```

## STRING PREDICATE

```
PUBLIC CLASS STRINGTEST {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

        PREDICATE<STRING> P = S->(S.LENGTH()>10);

        SYSTEM.OUT.PRINTLN(P.TEST("HELLO SANDY"));
        SYSTEM.OUT.PRINTLN(P.TEST("HELLO"));
        SYSTEM.OUT.PRINTLN(P.TEST("HELLO BABYDOLL"));
        SYSTEM.OUT.PRINTLN(P.TEST("HII"));
        }

}
```

OUTPUT

- TRUE
- FALSE
- TRUE
- FALSE

ChandraSekhar(CS) Baratam

## PASSING PREDICATE TO A METHOD

```
PUBLIC CLASS PASSINGMETHODS {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

        INT X[]={5,7,8,9,10,20,30,40,33,55,66,88,77,80};
        PREDICATE<INTEGER> P = I->(I>10);
        SORTING(P,X);
        }

         STATIC VOID SORTING(PREDICATE<INTEGER> P, INT[] X) {
     FOR (INT I : X) {
        IF(P.TEST(I)){
        SYSTEM.OUT.PRINTLN(I);
        }
        }
        }
}
```

## OUTPUT

```
GREATER THAN 10
20
30
40
50
60
80
```

## Predicate Joining

and()

or()

negate()

## PREDICATE JOINS SHOW ONLY EVEN NUMBERS

```
PUBLIC CLASS NEW2 {


        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {


        INT X[]={5,7,8,9,10,20,30,40,33,55,66,88,77,80};
        PREDICATE<INTEGER> P = I->(I>10);
        PREDICATE<INTEGER> P2 = I->(I%2==0);
        SYSTEM.OUT.PRINTLN("GREATER THAN 10");
        SORTING(P,X);
        SYSTEM.OUT.PRINTLN("EVEN NUMBERS");
        SORTING(P2, X);
        }


         STATIC VOID SORTING(PREDICATE<INTEGER> P, INT[] X) {
                                FOR (INT I : X) {
        IF(P.TEST(I)){
        SYSTEM.OUT.PRINTLN(I);
        }
        }
        }
}
```

## PREDICATES FULL TOPIC

```
PUBLIC CLASS NEW2 {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {

        INT X[]={5,7,8,9,10,20,30,40,33,55,66,88,77,80};
        PREDICATE<INTEGER> P = I->(I>10);
        PREDICATE<INTEGER> P2 = I->(I%2==0);
        SYSTEM.OUT.PRINTLN(" P AND P2 GREATER THAN 10 AND EVEN NUMBERS");
        SORTING(P.AND(P2),X);
        SYSTEM.OUT.PRINTLN("P OR P2 MAY BE EVEN OR ODD");
        SORTING(P.OR(P2), X);
        SYSTEM.OUT.PRINTLN("P NEGATE MEANS LESS THAN 10==>(I>10)");
        SORTING(P.NEGATE(), X);
        }

         STATIC VOID SORTING(PREDICATE<INTEGER> P, INT[] X) {
     FOR (INT I : X) {
        IF(P.TEST(I)){
        SYSTEM.OUT.PRINTLN(I);
        }
        }
        }
```

ChandraSekhar(CS) Baratam

```
}
```

# OUTPUT

```
P AND P2 GREATER THAN 10 AND EVEN NUMBERS
20
30
40
66
88
80
P OR P2 MAY BE EVEN OR ODD
8
10
20
30
40
33
55
66
88
77
80
P NEGATE MEANS LESS THAN 10==>(I>10)
5
7
8
9
10
```

## Functions  it is a type of predicates

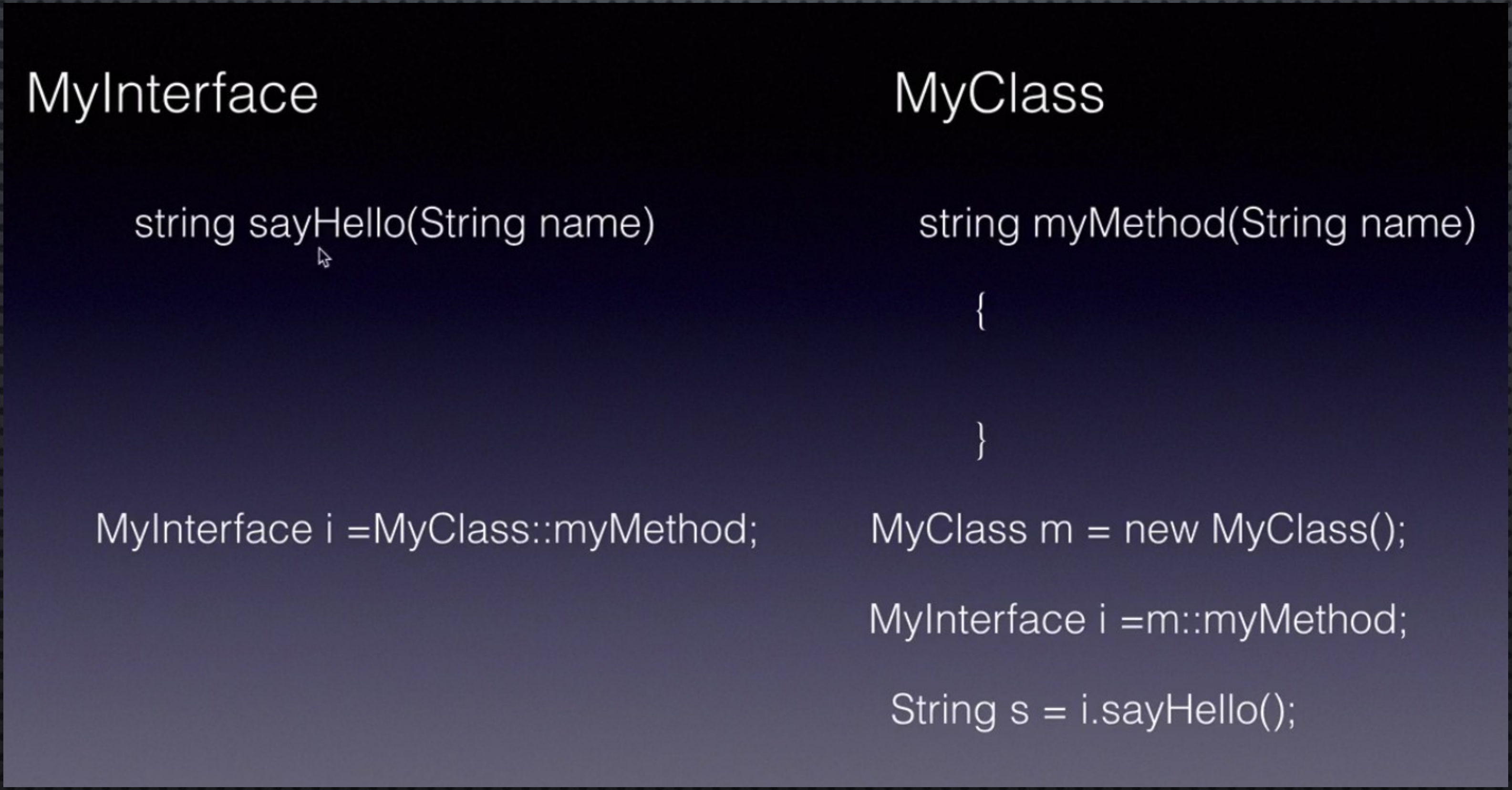
### Function

Return any type

```
interface Function(T,R){

        R apply(T t);

 }
```

```
PUBLIC CLASS FUNCTIONTEST {

      PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
     //FIRST TAKE STRING THEN INTEGER
            FUNCTION<STRING, INTEGER> F = S->S.LENGTH();
            SYSTEM.OUT.PRINTLN(F.APPLY("HELLO SANDY"));
            SYSTEM.OUT.PRINTLN(F.APPLY("AM FINE"));
      }

}
```

# METHOD REFERENCING USING :: OPERATOR DIFFERENT SIGNATURE BUT PASSING ARGUMENTS SHOULD BE SAME



MyInterface

```
string sayHello(String name)

MyInterface i =MyClass::myMethod;
```

MyClass

```
string myMethod(String name)
    {

    }
MyClass m = new MyClass();
MyInterface i =m::myMethod;
String s = i.sayHello();
```

ChandraSekhar(CS) Baratam

## METHOD REFERENCING IN ACTION

```
PUBLIC CLASS FUNCTIONS2 {

        PUBLIC STATIC VOID METHOD32(){
                FOR (INT I = 0; I <10; I++) {
                SYSTEM.OUT.PRINTLN("OUTER CLASSSES");
        }
        }
        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
       //TAKE AS RUNNABLE NOT A OBJECT
                RUNNABLE R = FUNCTIONS2::METHOD32;

                THREAD  T = NEW THREAD(R);
                T.START();

                FOR (INT I = 0; I <10; I++) {
                        SYSTEM.OUT.PRINTLN("INNER CLASSES");
                }
        }
}
```

## REFERENCING AN INSTANCE METHOD AND PASSING ARGUMENT MUST BE SAME

PROGRAM STARTS…….

```
PUBLIC INTERFACE MYINTERFACE {

PUBLIC VOID SANDY(INT I);
}
```

```
UBLIC CLASS MYCLASS {

        PUBLIC VOID SANS(INT I){
        SYSTEM.OUT.PRINTLN(I);
        }
}


        PUBLIC CLASS TEST {


        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {


        MYINTERFACE MI = I->SYSTEM.OUT.PRINTLN(I);
        MI.SANDY(100);


        MYCLASS MY = NEW MYCLASS();
        MYINTERFACE MYI = MY::SANS;
        MYI.SANDY(250);
        }
```

```
}
```

## REFERENCING A CONSTRUCTOR

CREATE A CLASS FOR CONSTRUCTOR

```
PUBLIC CLASS MYCLASS {
PRIVATE STRING S;

PUBLIC MYCLASS(STRING S) {
  THIS.S = S;
  SYSTEM.OUT.PRINTLN("FOR THE CONSTRUCTOR"+ S);
}
}
```

CREATE A INTERFACE FOR METHOD

```
PUBLIC INTERFACE MYINTERFCAE {
MYCLASS GET(STRING S);
}
```

ChandraSekhar(CS) Baratam

- MAIN CLASS

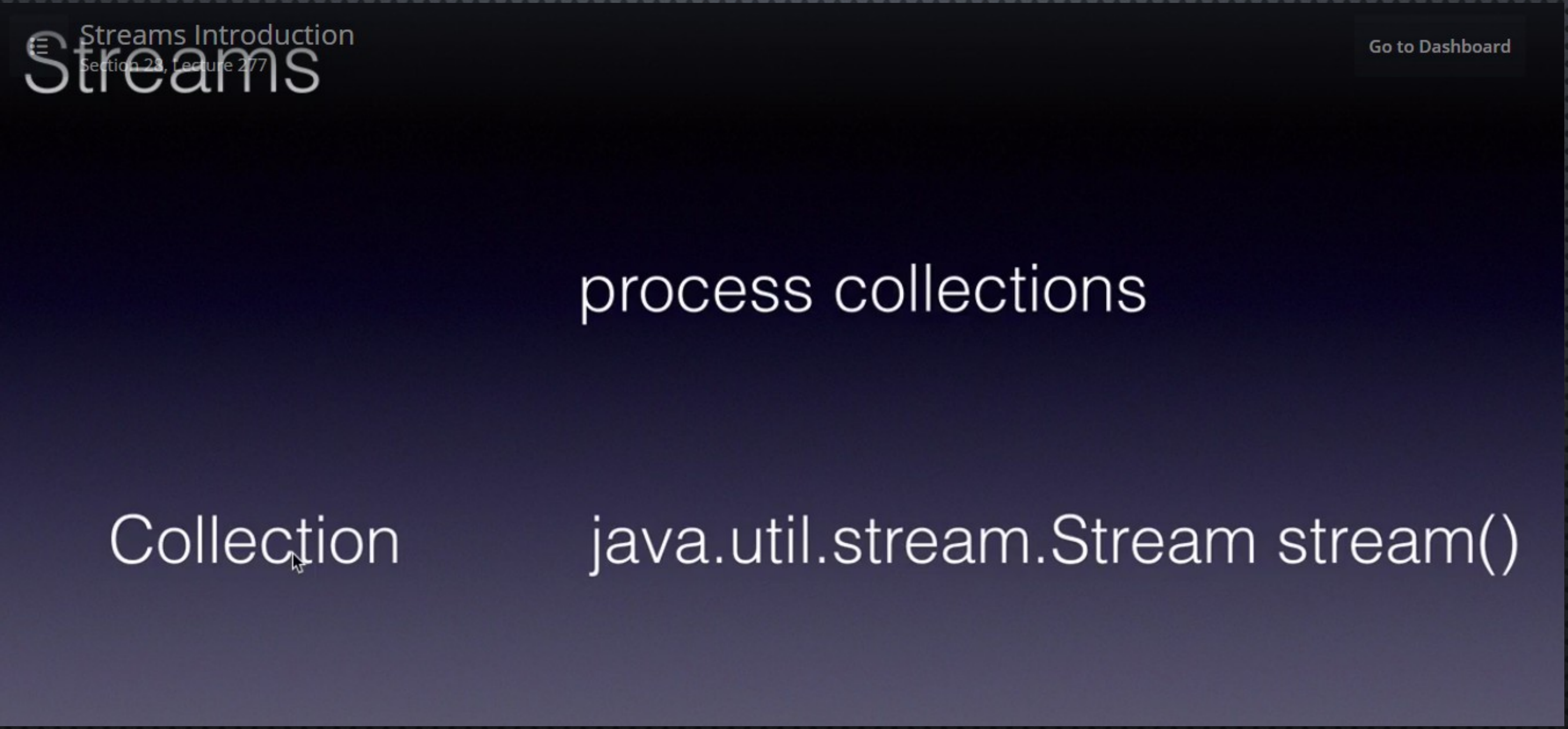
```
PUBLIC CLASS TEST {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
                //USING LAMDA EXPRESSION
                MYINTERFCAE MI = S->NEW MYCLASS(S);
                MI.GET(" YES BY LAMBDA MEHOD");
              //USING OPERATOR
                MYINTERFCAE MI1 = MYCLASS::NEW;
                MI1.GET(" SANDY");
        }

}
```

OUTPUT

- FOR THE CONSTRUCTOR YES BY LAMBDA MEHOD
- FOR THE CONSTRUCTOR SANDY

ChandraSekhar(CS) Baratam

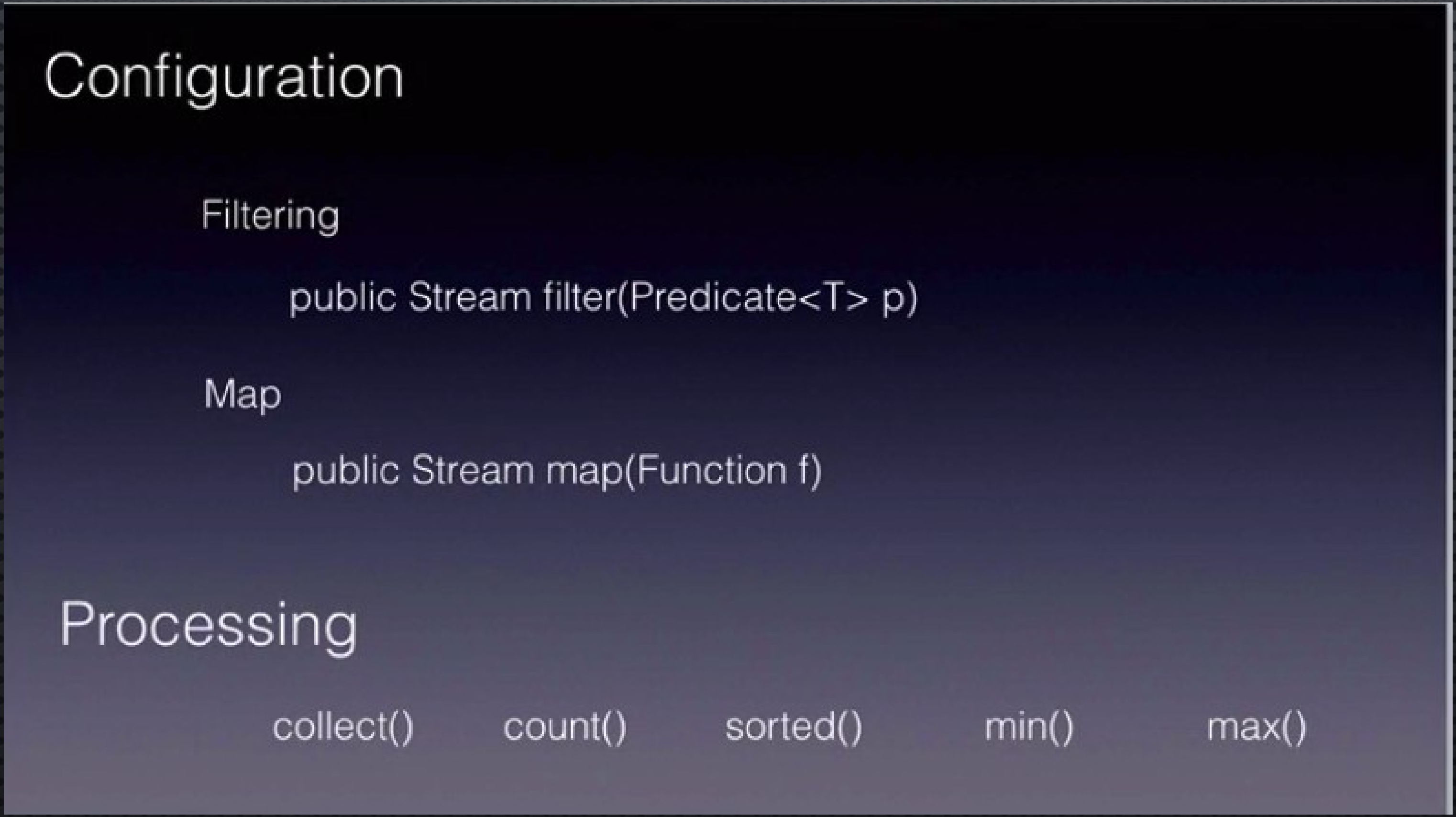
## STREAMS INTRODUCTION



Streams

process collections

Collection java.util.stream.Stream stream()



Configuration

Filtering

public Stream filter(Predicate<T> p)

Map

public Stream map(Function f)

Processing

collect() count() sorted() min() max()

# FILTER EVEN NUMBERS USING STREAMS

```
PUBLIC CLASS STREAMTWONUMBERS {

PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
LIST<INTEGER> L = NEW ARRAYLIST<>();



FOR (INT I = 0; I < 10; I++) {
L.ADD(I);
}
SYSTEM.OUT.PRINTLN(L);

/*LIST<INTEGER> L2 = NEW ARRAYLIST<>();
FOR(INTEGER I : L){
IF(I%2==0){
L2.ADD(I);
}
}
SYSTEM.OUT.PRINTLN(L2); */

LIST<INTEGER> L2 = L.STREAM().FILTER(I->I%2==0).COLLECT(COLLECTORS.TOLIST());
SYSTEM.OUT.PRINTLN(L2);
}
}
```

OUTPUT

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

ChandraSekhar(CS) Baratam

## CONVERT STRINGS TO LOWER CASE USING STREAMS

```
PUBLIC CLASS UPPERTOLOWER {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
        LIST<STRING> L = NEW ARRAYLIST<>();

        L.ADD("SANDY");
        L.ADD("SHRAVAN");
        L.ADD("TARU");

        SYSTEM.OUT.PRINTLN(L);

                            LIST<STRING> L1 = L.STREAM().MAP(S-
>S.TOUPPERCASE()).COLLECT(COLLECTORS.TOLIST());
        SYSTEM.OUT.PRINTLN(L1);
        }

}
```

OUTPUT

```
• [SANDY, SHRAVAN, TARU]
• [SANDY, SHRAVAN, TARU]
```

ChandraSekhar(CS) Baratam

## OTHER METHODS ON THE STREAM

```
PUBLIC CLASS STREAMTWONUMBERS {

        PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
        LIST<INTEGER> L = NEW ARRAYLIST<>();



        FOR (INT I = 0; I < 10; I++) {
        L.ADD(I);
        }
        SYSTEM.OUT.PRINTLN(L);

        /*LIST<INTEGER> L2 = NEW ARRAYLIST<>();
        FOR(INTEGER I : L){
        IF(I%2==0){
        L2.ADD(I);
        }
        }
        SYSTEM.OUT.PRINTLN(L2); */

        LIST<INTEGER> L2 = L.STREAM().FILTER(I->I%2==0).COLLECT(COLLECTORS.TOLIST());
        SYSTEM.OUT.PRINTLN(L2);
```

ChandraSekhar(CS)Baratam

```
LONG COUNT = L.STREAM().FILTER(I->I%2==0).COUNT();
        SYSTEM.OUT.PRINTLN("THE TOTAL VALUE IS "+ COUNT);


        COMPARATOR<INTEGER> C = (I,I2)->I.COMPARETO(I2);
        LIST<INTEGER> L3 =
L.STREAM().SORTED(C).COLLECT(COLLECTORS.TOLIST());
        SYSTEM.OUT.PRINTLN(L3);


        INTEGER MAX = L.STREAM().MAX(C).GET();
        SYSTEM.OUT.PRINTLN(MAX);
        INTEGER MIN = L.STREAM().MIN(C).GET();
        SYSTEM.OUT.PRINTLN(MIN);
        }
}
```