FunctionalInterface, default method, static method, lambda expressions, anonymous inner class object

|  |
| --- |
| import java.util.concurrent.\*;  import java.util.\*;  //FunctionalInterface, default method, static method,  //lambda expressions, anonymous inner class object  @FunctionalInterface  interface Intre{      public void m1();      //default must have body      public default void m2(){          System.out.println("m2");      }      //must be called by qualifier 'Intre'      public static void m3(){          System.out.println("m3");      }      //illegal      //public static default void m4(){}    }  class A{      A(){          System.out.println("A constructor");      }      public void printa(){          System.out.println("A");      }      public static void printa1(){          System.out.println("A1");      }  }  public class Test {      public static void fun(){          System.out.println("fun");      }      public static void main(String[] args) throws InterruptedException      {          Intre i1 = new Intre(){              //anonymous inner class object              //can have other delaresion (variable,method)              int x=10;              public void m1(){System.out.println("I1");};          };            i1.m1();          //lambda expression can't have delaresion          Intre i2 = ()->{System.out.println("I2");};          i2.m1();          i2.m2();          //illegal          //i2.m3();          Intre.m3();          //even different function name, still can assign          A a = new A();          Intre i3 = a::printa;          i3.m1();          Intre i4 = A::printa1;//static fun          i4.m1();          Intre i5 = A::new;//assign constructor          i5.m1();      }  } |

stream, pre defined functional interface

|  |
| --- |
| import java.util.\*;  import java.util.stream.\*;  class Demo  {      public static void main(String args[])      {          //input          List<Integer> number = Arrays.asList(6,7,1,2,0,2,3,4,5);          List<String> names = Arrays.asList("Steven","Seven","Collection","Stream");            // Intermediate Operations: map filter sort          // map          // use pre defined functional interface          Function<Integer,Integer> mapper = (x) -> x\*x;          //number = number.stream().map(x -> x\*x).collect(Collectors.toList());          number = number.stream().map(mapper).collect(Collectors.toList());          System.out.println(number);              // filter          names = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());          System.out.println(names);          number = number.stream().filter(n->n>2).collect(Collectors.toList());          System.out.println(number);          // sort          names = names.stream().sorted().collect(Collectors.toList());          System.out.println(names);          number = number.stream().sorted().collect(Collectors.toList());          System.out.println(number);            //Terminal Operations: collect foreach reduce          //foreach          Consumer<Integer> cm=i->System.out.print(i+"-");          //number.stream().map(x->x\*x).forEach(y->System.out.print(y+"-"));          number.stream().map(x->x\*x).forEach(cm);          System.out.println();            // reduce          // Optional: the class may or may not contain a non-null value( get() to get data)          // Optional/Optional<Integer>/Object/var          // Optional ans = number.stream().reduce((x,y)->x+y);          // if reduce don't give initial value, it will return Optional          // Integer/int          // Integer ans = number.stream().reduce((x,y)->x+y).get();          // System.out.println(ans);            BiFunction<Integer,Integer,Integer> reducer = (x,y)->x+y;          // int ans = number.stream().reduce((x,y)->x+y).get();          // reduce(0,...) initial as 0 => int          // int ans = number.stream().reduce(0,(x,y)->x+y);          int ans = number.stream().reduce(0,reducer::apply);          System.out.println(ans);            // how to use reduce          // https://www.geeksforgeeks.org/stream-reduce-java-examples/          // long reduce(int identity, LongBinaryOperator op);          // identity = default or initial value.          // BinaryOperator = functional interface, take two values and produces a new value.      }  } |