**L’interface Comparator**

**https://godbolt.org/z/aYv5GrMTs**

L’ordre d’un élement Collection peut être définie deux manières :

-En recourant à la méthode de compareTo de leur classe. A condition que cette classe implemente l’interface Comparable.

-A l’aide d’un comparateur fourni en algorithme, à un cnstructeur. Depuis Java 8 la methode sort de l’interfcae List. Un tel comparateur est un objet implémentant interface Comparator.ie défnissant sa méthode compare(Objet o1, Objet o2).

**public** **int** compare(Car o1, Car o2) {

**return** o1.getMaxSpeed() - o2.getMaxSpeed();

}

@Override

**public** **int** compare**(**Student s1, Student s2**)** **{**

**return** s1.getAge**()**.compareTo**(**s2.getAge**())**;

**}**

//

On voit qu’il est possible d’utiliser des expressions lambda et des refrences de méthodes pour définir des comparateur.

Java 8 introduit de nouvelle méthodes statique dans l’interface Comparator :

Comparing, reversed, reverseOrder, naturalOrder.

Heureusement, à partir de **Java 8** l’interface **Comparator** intègrera une méthode static **comparing()** permettant de créer un comparateur basé sur un champ, en lui passant une expression lambda permettant d’extraire le champ à comparer.

Comparator<Car> comparator = Comparator.

comparing((Car c) -> c.getModel());

Comparator<Car> comparator = Comparator.comparing(Car::getModel);

Collections.sort(slist, Collections.reverseOrder());

Comparator<Message> comparator = Comparator.comparing(obj -> obj.getId());

Supplier<Integer> randomizer = () -> **new** Random().nextInt(1000);

Stream.generate(randomizer)

.limit(10)

.sorted(Comparator.naturalOrder())

.forEach(e -> System.out.println(e));

class Main{

    public static void main(String []args){

        System.out.println("hello !");

        Integer x1=2;

        Integer x2=4;

        int comp=x1.compareTo(x2);

         System.out.println("x1.compare(x1) = "+ comp);

        int comp1=Integer.compare(x1,x2);

        System.out.println("Integer.compare(x1,x2) = "+ comp1);

 System.out.println("\n ");

  List<Integer> list=Arrays.asList(3,53,6,90,32,2);

   System.out.print("initial list : ");// initial list : 3 53 6 90 32 2

  list.forEach(e-> System.out.print(e +" "));

   System.out.println("\n ");

   System.out.print("naturalOrder sorted list : ");

   list.sort(Comparator.naturalOrder());//naturalOrder sorted list : 2 3 6 32 53 90

   list.forEach(e-> System.out.print(e +" "));//

   System.out.println("\n ");

    System.out.print("reverseOrder sorted list : ");

    list.sort(Comparator.reverseOrder());//reverseOrder sorted list : 90 53 32 6 3 2

    list.forEach(e-> System.out.print(e +" "));

    System.out.println("\n ");

    System.out.print("reversed list : ");

    //list.sort(Comparator.reversed());

    //list.reverse();

    list.forEach(e-> System.out.print(e +" "));

        }

    }

Ce code ne va pas s’eexuter. Car on ne peut pas applique un Arrays.asList à remove. Donc on doit le caster.

      List<Integer> listrev=Arrays.asList(4, 5, 1, 2, 8, 9, 6);

      for(int k=0, j=listrev.size()-1;k<j;k++){

        listrev.add(k,listrev.remove(j));

      }

      listrev.forEach(e-> System.out.print(e +" "));

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

 List<Integer> listrev=new ArrayList<>( Arrays.asList(4, 5, 1, 2, 8, 9, 6) );

      for(int k=0, j=listrev.size()-1;k<j;k++){

        listrev.add(k,listrev.remove(j));

      }

      listrev.forEach(e-> System.out.print(e +" "));

Les Streams

La methode forEach s’applique un strem. La methode filter, map … sont dites inermediaires et forEach est dite terminale.

La methode max() de IntStream donne la plus grande valeur. Tandis que la méthode max d’un Stream<Integer> demande uncomparateur en argument.

Creation :

import java.util.stream.Stream;

Applique la methode stream à une collection et retourn un Stream<T>.

    List<Integer> liste=new ArrayList<>( Arrays.asList(4, -5, 1,11,14 ,-2, 8, -9, 6) );

    System.out.print(" liste : ");

liste.stream().forEach(e-> System.out.print(e +" "));

 System.out.println(" \n");

Stream<Integer> lf=liste.stream();

Stream<Integer> filt=lf.filter(e-> e<0 );

System.out.print(" \n liste filter  : ");//liste filter  : -5 -2 -9

 filt.forEach(e-> System.out.print(e +" "));//liste filter  : -5 -2 -9

En appliquant IntStream.of et Sream.of :

 Stream<Integer> str=Stream.of(1,2,3,4,5);

 IntStream str1= IntStream.of(6,7,8,9,10);

 System.out.print(" \n IntSream   : ");

 str.forEach(e-> System.out.print(e +"  "));//1  2  3  4  5

 System.out.println("  ");

  str1.forEach(e-> System.out.print(e +"  "));// 6  7  8  9  10

Creation sur un tableau  :

  int []tab={11,12,13,14,15};

  Stream<Integer> tabs=Stream.of(tab);

Va provoquer une erreur.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Il faut utilizer IntStream :

 IntStream tabs=IntStream.of(tab);

  tabs.forEach(e-> System.out.print(e +"  "));// 11  12  13  14  15

 A [] taba={ new A(), new A(), new A(), new A()};

  Stream<A> tabas=Stream.of(taba);

**Les methides intermédiaires d’un stream**

Filter(), limite, map(), soted(Comparateur pour les élements de types objets ), reduce, collect(Collector.toList())

    List<Integer> l=new ArrayList<>( Arrays.asList(4, -5, 1,11,14 ,-2, 8, -9, 6) );

 //IntStream li=l.stream().map(e->Math.pow(e,2));

System.out.print("\n");

 Stream<Double>  li=l.stream().map(e->Math.pow(e,2));

 List<Double> li1=l.stream().map(e->Math.pow(e,3)).collect(Collectors.toList());

System.out.print("\n");

 li1.forEach(e-> System.out.print(e +" "));

 System.out.print("li1 \n");

  System.out.print(li1);

**Les methodes terminales d’un stream**

forEach,min , max,.

min , max reçoit en argument un comparateur sauf pour les types naturels.

Certaine resulat applique de min max appliqué à Stream<T> renvoie un Optonal<T> qui contient la valeur à envoyer. On trouve des méthode ifPresent, , orElse

**FixCapacity List**

import java.util.Arrays;

interface List<T>{

    public void add(T e);

    public T get(int i);

}

class A {

    private int n;

    public A( int n){

       this.n=n;

    }

    public String toString(){

        return " class A n = "+n;

    }

}

class FixedCapacityList implements List<Object>{

private int capacity;

private int size=0;

private Object [] elements;

public FixedCapacityList(int c){

    this.capacity=c;

this.elements=new Object[c];

}

    public void add(Object e){

        if(this.size<this.capacity){

            this.elements[this.size++]=e;

        }

    }

    public Object get(int i){

        return this.elements[i];

    }

    public String toString(){

        return Arrays.toString(elements);

    }

}

class Point{

    public int x,y;

    public Point(int xx,int yy){

        this.x=xx;

        this.y=yy;

    }

    public String toString(){

        return " Point(" +this.x+" , "+ this.y+" )";

    }

}

class Main{

    public static void main(String [] args){

        System.out.println("Hello world !");

        FixedCapacityList fcl=new FixedCapacityList(5);

        fcl.add("alhas");

        fcl.add(1);

        fcl.add(true);

        fcl.add(new A(3));

        fcl.add(new Point(3,5));

        fcl.add("plein");

        System.out.println(fcl);

    }

}

Iterable

Maintenant si on fait

class FixedCapacityList implements List<Object> , Iterable<Object>{….}

On affaiche une erreur qui dit demande de fournir la methode Iterator<Object> iterato() sauf que :

public Iterator<Object> iterator(){

    return new Iterator<>();// erreur

}

Provoque une erreur car on ne peut peut pas instancié une interface. Pour remédier à ça. On implemente la classe Iterator et on retourn son implemenattion.

https://godbolt.org/z/qx7avq9M7

Donc l’ajout de dans FixCapacityList de cette methode et l’implementatio enleve l’erreur.

public Iterator<Object> iterator(){

    return new AscendingIterator();

}

class AscendingIterator implements Iterator<Object> {

    public boolean hasNext(){

        return true;

    }

    public Object next(){

        return new Object();

    }

}

https://godbolt.org/z/qx7avq9M7