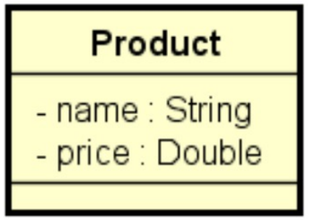
Programación Funcional

Sesión 3

Una experiencia con Comparator

Problema

- Suponga una clase de producto con los atributos de nombre y precio.
- Podemos implementar la comparación de productos mediante implementación de la interfaz Comparable <Product>.
- Sin embargo, de esta manera nuestra clase no está cerrada a cambio: si cambia el criterio de comparación, necesitaremos cambiar la clase de producto.
- A continuación, podemos utilizar el método predeterminado "ordenar" de la interfaz de lista: ordenación vacía por defecto (Comparador <? super E> c)



Comparator

https://docs.oracle.com/javase/10/docs/api/java/util/Comparator.html

Vea el método de clasificación en la interfaz de lista:

https://docs.oracle.com/javase/10/docs/api/java/util/List.html

Resumen de la lección

- Objeto de clase separada comparator (Standard Comparator)
- Objeto de clase anónimo comparator (Anonymous Class)
- Comparator de objetos de expresión lambda con llaves (Lambda 1)
- Comparator de objetos de expresión lambda sin claves (Lambda 2)
- Expresión lambda del comparator "argumento directo”(Lambda 3)

Programación y cálculo funcional lambda

Paradigmas de programación

- Imperativo (C, Pascal, Fortran, Cobol)
- Orientado a objetos (C ++, Object Pascal, Java (<8), C # (<3))
- Funcional (Haskell, Closure, Clean, Erlang)
- Lógico (Prolog)
- Multiparadigma (JavaScript, Java (8+), C # (3+), Ruby, Python, Go)

Transparencia referencial

Una función tiene transparencia referencial si su resultado es siempre el mismo para los mismos datos de entrada. Beneficios: sencillez y previsibilidad. Ejemplo de función que no es referencialmente transparente

```
package application;
import java.util.Arrays;
public class Program {
    public static int globalValue = 3;
    public static void main(String[] args) {
        int[] vect = new int[] {3, 4, 5};
        changeOddValues(vect);
        System.out.println(Arrays.toString(vect));
    }
    public static void changeOddValues(int[] numbers) {
        for (int i=0; i<numbers.length; i++) {
            if (numbers[i] % 2 != 0) {
                numbers[i] += globalValue;
            }
        }
    }
}
```

Inferencia de tipo / tipado dinámico

```
public static void main(String[] args) {
    List<Product> list = new ArrayList<>();

    list.add(new Product("TV", 900.00));
    list.add(new Product("Notebook", 1200.00));
    list.add(new Product("Tablet", 450.00));

    list.sort((p1, p2) -> p1.getPrice().compareTo(p2.getPrice()));

    list.forEach(System.out::println);
}
```

Expresividad / código conciso

```
Integer sum = 0;
for (Integer x : list) {
    sum += x;
}
```

VS.

```
Integer sum = list.stream().reduce(0, Integer::sum);
```

Expresividad / código conciso

```
Integer sum = 0;
for (Integer x : list) {
    sum += x;
}
```

VS.

```
Integer sum = list.stream().reduce(0, Integer::sum);
```

¿Qué son las ”expresiones lambda”?

En programación funcional, la expresión lambda corresponde a un función anónima de primera clase.

```
public class Program {

    public static int compareProducts(Product p1, Product p2) {
        return p1.getPrice().compareTo(p2.getPrice());
    }

    public static void main(String[] args) {

        (...)

        list.sort(Program::compareProducts);

        list.sort((p1, p2) -> p1.getPrice().compareTo(p2.getPrice()));

        (...)
```

Interface funcional

Interface funcional

Es una interfaz que tiene un método abstracto único. Las implementaciones se tratarán como expresiones lambda

```
public class MyComparator implements Comparator<Product> {

    @Override
    public int compare(Product p1, Product p2) {
        return p1.getName().toUpperCase().compareTo(p2.getName().toUpperCase());
    }
}
```

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

    (...)

    list.sort(new MyComparator());
```

Algunas interfaces funcionales comunes

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

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

Consumer
- https://docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html
- Nota: a diferencia de las otras interfaces funcionales, en el caso de Consumer, es esperado que pueda generar efectos secundarios

Predicate (ejemplo de removeif)

Predicate

```
public interface Predicate<T> {

    boolean test(T t);
}
```

Ejemplo

Haga un programa que, de una lista de productos, elimine del enumera solo aquellos cuyo precio mínimo es 100.

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

Ejemplo – Versiones

- Implementación de interfaz
- Método de referencia con método estático
- Método de referencia con método no estático
- Expresión lambda declarada
- Expresión lambda en línea

Consumer (ejemplo con forEach)

Consumer

```
public interface Consumer<T> {

    void accept(T t);
}
```

Ejemplo

Haga un programa que, a partir de una lista de productos, aumente la precio del producto en un 10%.

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

Function (ejemplo con map)

Function

```
public interface Function<T, R> {

    R apply(T t);
}
```

Ejemplo

Hacer un programa que, a partir de una lista de productos, genere un nueva lista que contiene los nombres de los productos en mayúsculas

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

Nota sobre la función de map

La función map (que no debe confundirse con la estructura de datos de Map) es una función que aplica una función a todos los elementos de una secuencia.

Conversiones

- List para stream: `.stream()`
- Stream para List: `.collect(Collectors.toList())`

Creando funciones que reciben funciones como parametro

Ejemplo

Haga un programa que, a partir de una lista de productos, calcule el suma de precios solo para productos cuyo nombre comienza con "T"

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

1250.50

Stream

Stream

- Es una secuencia de elementos provenientes de una fuente de datos que admite "operaciones agregadas".
  - Fuente de datos: colección, matriz, función de iteración, recurso de E / S

- Lectura sugerida:
  https://www.oracle.com/br/technical-resources/articles/java-stream-api.html

Caracteristicas

- Stream es una solución para procesar flujos de datos de las siguientes formas:
  - Declarativo (iteración interna: oculta al programador)
  - Compatible con paralelo (inmutable -> seguro para subprocesos)
  - Sin efectos secundarios
  - Bajo demanda (evaluación perezosa)

- Acceso secuencial (sin índices)
- Pipeline: las operaciones en Streams devuelven nuevos Streams. Entonces es posible crear una cadena de operaciones (flujo de procesamiento).

Operaciones intermedias y terminales

- El pipeline consta de cero o más operaciones intermedias y una terminal.

- Operaciones intermedias
  - Produce nuevos flujos (encadenamiento)
  - Se ejecuta solo cuando se invoca una operación de terminal (lazy evaluation)

- Operaciones terminales
  - Produce un objeto que no es una secuencia (colección u otro)
  - Determina el final del procesamiento de la transmisión.

Crear un Stream

- Simplemente llame al método stream () o paraleloStream () desde de cualquier objeto de Colección.
  https://docs.oracle.com/javase/10/docs/api/java/util/Collection.html

- Otras formas de crear un Stream incluyen:
  - Stream.of
  - Stream.ofNullable
  - Stream.iterate

```
List<Integer> list = Arrays.asList(3, 4, 5, 10, 7);
Stream<Integer> st1 = list.stream();
System.out.println(Arrays.toString(st1.toArray()));


Stream<String> st2 = Stream.of("Maria", "Alex", "Bob");
System.out.println(Arrays.toString(st2.toArray()));


Stream<Integer> st3 = Stream.iterate(0, x -> x + 2);
System.out.println(Arrays.toString(st3.limit(10).toArray()));


Stream<Long> st4 = Stream.iterate(new long[]{ 0L, 1L }, p->new long[]{ p[1], p[0]+p[1] }).map(p -> p[0]);
System.out.println(Arrays.toString(st4.limit(10).toArray()));
```

Pipeline (demo)

Demo – pipeline

```
List<Integer> list = Arrays.asList(3, 4, 5, 10, 7);

Stream<Integer> st1 = list.stream().map(x -> x * 10);
System.out.println(Arrays.toString(st1.toArray()));

int sum = list.stream().reduce(0, (x, y) -> x + y);
System.out.println("Sum = " + sum);

List<Integer> newList = list.stream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 10)
    .collect(Collectors.toList());
System.out.println(Arrays.toString(newList.toArray()));
```

Ejercicio resuelto: filter, sorted, map reduce (demo)

Demo

Haga un programa para leer un conjunto de productos de un archivo en formato .csv (suponga que hay al menos un producto). Luego muestre el precio promedio de los productos. Luego, muestra el nombres, en orden descendente, de productos que tienen un precio por debajo del precio medio. Vea el ejemplo en la página siguiente.

Input file:

```
Tv,900.00
Mouse,50.00
Tablet,350.50
HD Case,80.90
Computer,850.00
Monitor,290.00
```

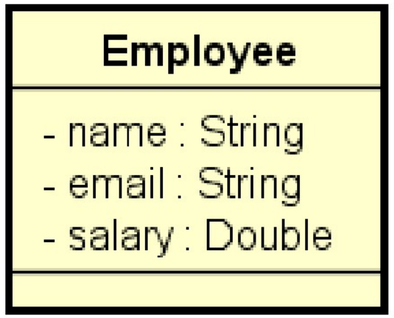
Execution:

```
Enter full file path: c:\temp\in.txt
Average price: 420.23
Tablet
Mouse
Monitor
HD Case
```

Ejercicio

Demo

Haz un programa para leer los datos (nombre, correo electrónico y salario) de empleados de un archivo en formato .csv. Luego muestre, en orden alfabético, el correo electrónico del empleados cuyo salario exceda una cantidad determinada proporcionada por el usuario. También muestre la suma de los salarios de los empleados cuyos El nombre comienza con la letra 'M'. Vea el ejemplo en la página siguiente..



Demo

Input file:

```
Maria,maria@gmail.com,3200.00
Alex,alex@gmail.com,1900.00
Marco,marco@gmail.com,1700.00
Bob,bob@gmail.com,3500.00
Anna,anna@gmail.com,2800.00
```

Execution:

```
Enter full file path: c:\temp\in.txt
Enter salary: 2000.00
Email of people whose salary is more than 2000.00:
anna@gmail.com
bob@gmail.com
maria@gmail.com
Sum of salary of people whose name starts with 'M': 4900.00
```