Academia Xideral Java Monterrey

Daniel Ivan Anaya Alvarez

Stream Activity

In this stream activity the objective is to make two programs that can solve a business problem using streams on Java with a mínimum requirement of using 5 sorting methods. In order to use these Streams we need to be in Java 8 or superior version.

First Program

For my first program the business problem to solve was: There is a national Casino tournament in which takes place in two casinos. These two casinos have their own participants with an unique Id for the tournament. What I did in this program was using an abstract that helped me create the participants and add these participants in their own array depending on which casino they were. After that I used a Stream to paste these two arrays and sort them by Id and then filtering out the ones that has less 3500 points to simulate that they lost and that they are removed from the tournament that's what my program do. At the final print we can see the two arrays made from Casino1's participants and Casino2's participants and at the very bottom we can see the results. At the end the five operations implemented were: creating the stream, use of flatmap, sorted, filter and collect.

TipoDeParticipante.Java

```
package Stream;

//Hacemos una clase abstracta para generar los participantes en los dos casinos
public abstract class TipoDeParticipante {

	//Encapsulación de datos
	private String nombre;
	private int puntos;
	private int id;

	//Constructor de los participantes
		TipoDeParticipante(String nombre, int puntos, int id){
		this.nombre = nombre;
		this.puntos = puntos;
		this.id = id;

	}
		//Generar string
		@Override
		public String toString() {
```

```
                return this.getClass().getSimpleName() + " [id=" + id + ",
nombre=" + nombre + ", puntos=" + puntos + "]";
        }

        //Getters y Setters de cada elemento
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public int getPuntos() {
                return puntos;
        }
        public void setPuntos(int puntos) {
                this.puntos = puntos;
        }
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }


}
```

Casino1.Java

```
package Stream;

public class Casino1 extends TipoDeParticipante {

        Casino1(String nombre, int puntos, int id) {
                super(nombre, puntos, id);
        }


}
```

Casino2.Java

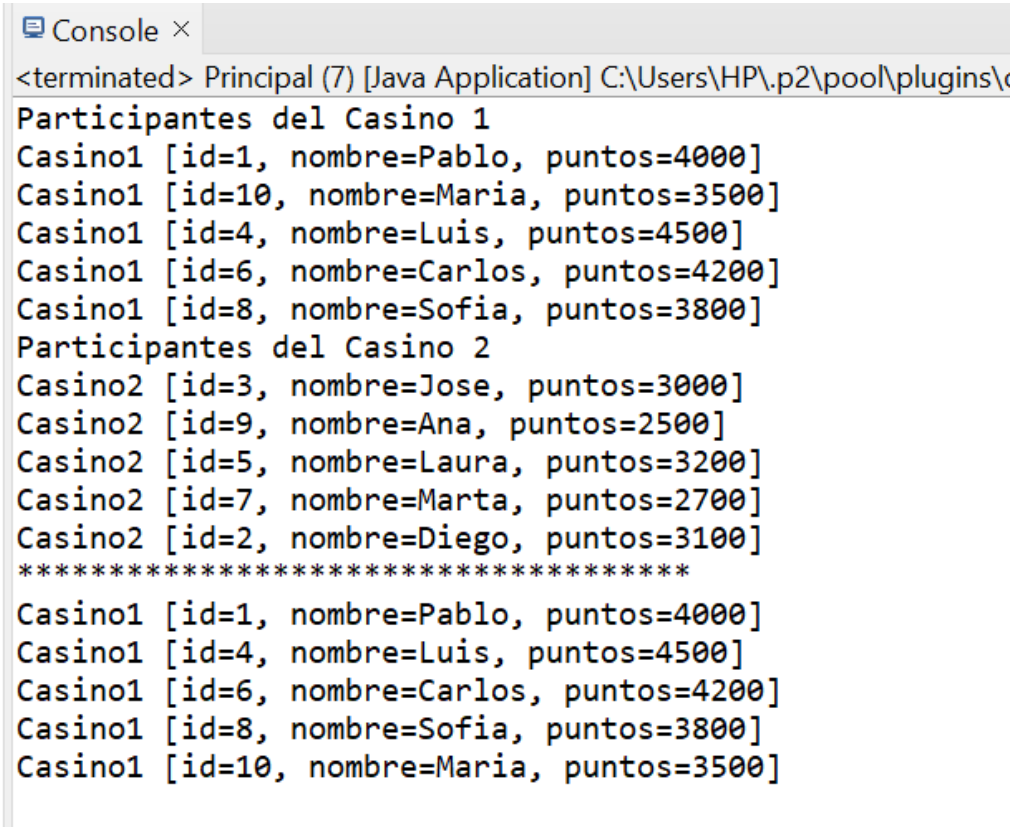
```
package Stream;

public class Casino2 extends TipoDeParticipante {

        Casino2(String nombre, int puntos, int id) {
                super(nombre, puntos, id);
        }

}
```

Principal.Java

```java
package Stream;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Principal {

    public static void main(String[] args) {

        // Creamos los participantes
        Casino1 participantec1_1 = new Casino1("Pablo", 4000, 1);
        Casino2 participantec2_1 = new Casino2("Jose", 3000, 3);
        Casino1 participantec1_2 = new Casino1("Maria", 3500, 10);
        Casino2 participantec2_2 = new Casino2("Ana", 2500, 9);
        Casino1 participantec1_3 = new Casino1("Luis", 4500, 4);
        Casino2 participantec2_3 = new Casino2("Laura", 3200, 5);
        Casino1 participantec1_4 = new Casino1("Carlos", 4200, 6);
        Casino2 participantec2_4 = new Casino2("Marta", 2700, 7);
        Casino1 participantec1_5 = new Casino1("Sofia", 3800, 8);
        Casino2 participantec2_5 = new Casino2("Diego", 3100, 2);

        // Creamos la lista de los participantes
        List<Casino1> Casino1Participantes = new ArrayList<>();
        List<Casino2> Casino2Participantes = new ArrayList<>();

        // Agregamos a los participantes a cada lista
        Casino1Participantes.add(participantec1_1);
        Casino1Participantes.add(participantec1_2);
        Casino1Participantes.add(participantec1_3);
        Casino1Participantes.add(participantec1_4);
        Casino1Participantes.add(participantec1_5);

        Casino2Participantes.add(participantec2_1);
        Casino2Participantes.add(participantec2_2);
        Casino2Participantes.add(participantec2_3);
        Casino2Participantes.add(participantec2_4);
        Casino2Participantes.add(participantec2_5);

        // Imprimimos los arreglos del casino 1
        System.out.println("Participantes del Casino 1");
        for (Casino1 participante : Casino1Participantes) {
            System.out.println(participante);
        }

        // Imprimimos los arreglos del casino 2
        System.out.println("Participantes del Casino 2");
        for (Casino2 participante2 : Casino2Participantes) {
            System.out.println(participante2);
        }
        System.out.println("************************************");
```

```
        // Combinar, ordenar y filtrar los participantes de ambos casinos usando
flatMap
        List<TipoDeParticipante> listanacional = Stream.of(Casino1Participantes,
Casino2Participantes)
                .flatMap(List::stream) // Pega los participantes del Casino 1 y
Casino 2
                .sorted(Comparator.comparingInt(TipoDeParticipante::getId)) //
Ordena por ID ascendente
                .filter(p -> p.getPuntos() >= 3500) // Filtra los que tienen más
de 3000 puntos
                .collect(Collectors.toList()); // Colecta los resultados en una
lista

        // Printeamos todos los participantes
        listanacional.forEach(System.out::println);
    }
}
```

Final Print



Second Program

For this second program I decided to board on an electroncic store inventory, this store has products like: Laptops, smartphones and tablets. Each of these products have a Price, quantity and its own respective name, we create the objects and add them in their own array list, after that we made the list and include on the stream the three arrays we just made, we use the flatmap to paste the three lists and add them a filter to show products above 600 $ pesos and with an ascendent order, we then collect and print the result with that we accomplish the objective of using 5 Stream instructions, additionally I added another Stream to filter in order to show out if a product has more than 50 products available and print it out.

Producto.java

```
package Stream2;

//Clase abstracta que representa un producto en la tienda
public abstract class Producto {

//Encapsulación
 private String nombre;
 private double precio;
 private int cantidad;

 //Constuctor
 public Producto(String nombre, double precio, int cantidad) {
     this.nombre = nombre;
     this.precio = precio;
     this.cantidad = cantidad;
 }
//Getters y Setters
 public String getNombre() {
     return nombre;
 }

 public double getPrecio() {
     return precio;
 }

 public int getCantidad() {
     return cantidad;
 }

 @Override
 public String toString() {
     return this.getClass().getSimpleName() + " [nombre=" + nombre + ", precio="
+ precio + ", cantidad=" + cantidad + "]";
 }
}
```

Laptop.java

```java
package Stream2;

public class Laptop extends Producto {

    public Laptop(String nombre, double precio, int cantidad) {
        super(nombre, precio, cantidad);
    }
}
```

Tablet.java

```java
package Stream2;

public class Tablet extends Producto {

    public Tablet(String nombre, double precio, int cantidad) {
        super(nombre, precio, cantidad);
    }
}
```

Smarthphone.java

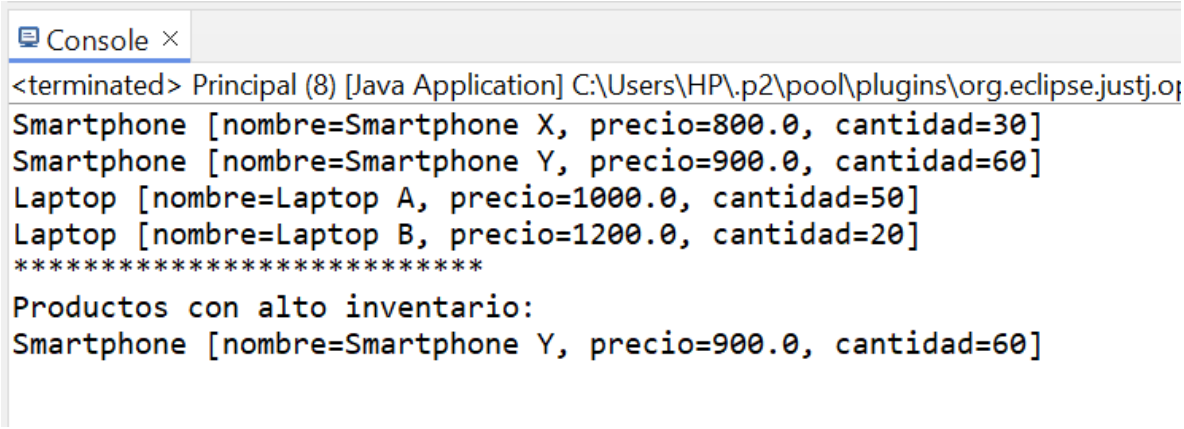
```java
package Stream2;

public class Smartphone extends Producto {

    public Smartphone(String nombre, double precio, int cantidad) {
        super(nombre, precio, cantidad);
    }
}
```

Principal.java

```java
package Stream2;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Principal {

    public static void main(String[] args) {

                    // Creación de productos
                    Laptop laptop1 = new Laptop("Laptop A", 1000.0, 50);
```

```
                    Smartphone smartphone1 = new Smartphone("Smartphone X",
800.0, 30);
                    Tablet tablet1 = new Tablet("Tablet Z", 500.0, 70);
                    Laptop laptop2 = new Laptop("Laptop B", 1200.0, 20);
                    Smartphone smartphone2 = new Smartphone("Smartphone Y",
900.0, 60);
                    Tablet tablet2 = new Tablet("Tablet W", 450.0, 10);

                    // Listas de productos
                    List<Laptop> laptops = new ArrayList<>();
                    List<Smartphone> smartphones = new ArrayList<>();
                    List<Tablet> tablets = new ArrayList<>();

                    // Añadir productos a las listas
                    laptops.add(laptop1);
                    laptops.add(laptop2);
                    smartphones.add(smartphone1);
                    smartphones.add(smartphone2);
                    tablets.add(tablet1);
                    tablets.add(tablet2);

                    // Operaciones de Stream
                    List<Producto> inventario = Stream.of(laptops, smartphones,
tablets)
                            .flatMap(List::stream) // Combina todas las listas
en un solo Stream
                            .filter(p -> p.getPrecio() > 600) // Filtra
productos con un precio superior a 600

.sorted(Comparator.comparingDouble(Producto::getPrecio)) // Ordena por precio
ascendente
                            .collect(Collectors.toList()); // Recolecta los
resultados en una lista

                    // Imprimir los productos filtrados y ordenados
                    inventario.forEach(System.out::println);

                    // Otra operación: Obtener productos con alto inventario
(cantidad > 50)
                    List<Producto> altoInventario = inventario.stream()
                            .filter(p -> p.getCantidad() > 50) // Filtra
productos con alto inventario
                            .collect(Collectors.toList());
                    System.out.println("**************************");

                    // Imprimir productos con alto inventario
                    System.out.println("Productos con alto inventario:");
                    altoInventario.forEach(System.out::println);
                }
            }
```

Print



Conclusion

After working in this activity I can say that Streams are useful when you want to work with different Lists and also want to sort or filter any information.