Ingeniería del Software

Java8 Reutilización de Software

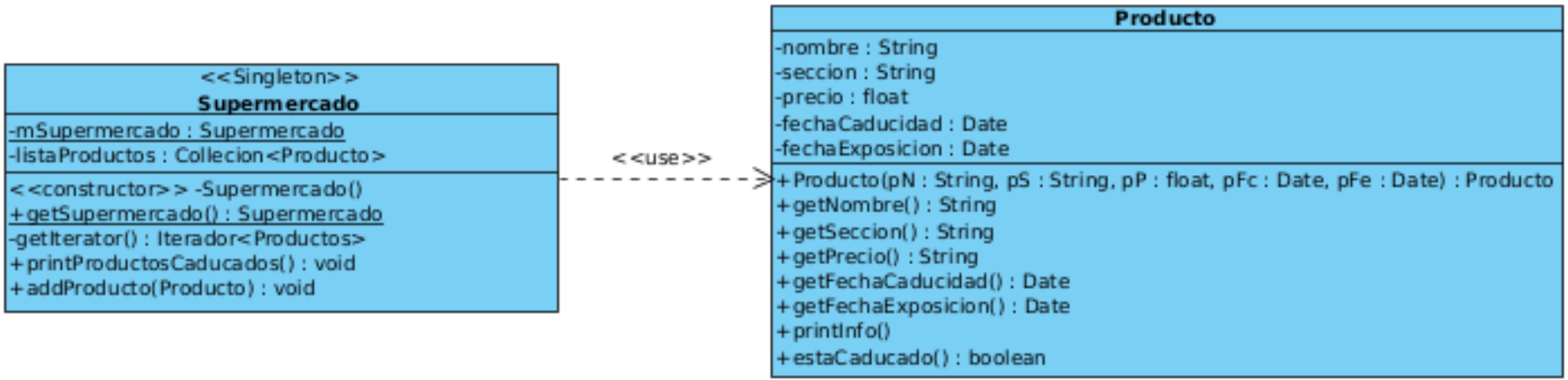
Expresiones Lambda y operaciones de agregación

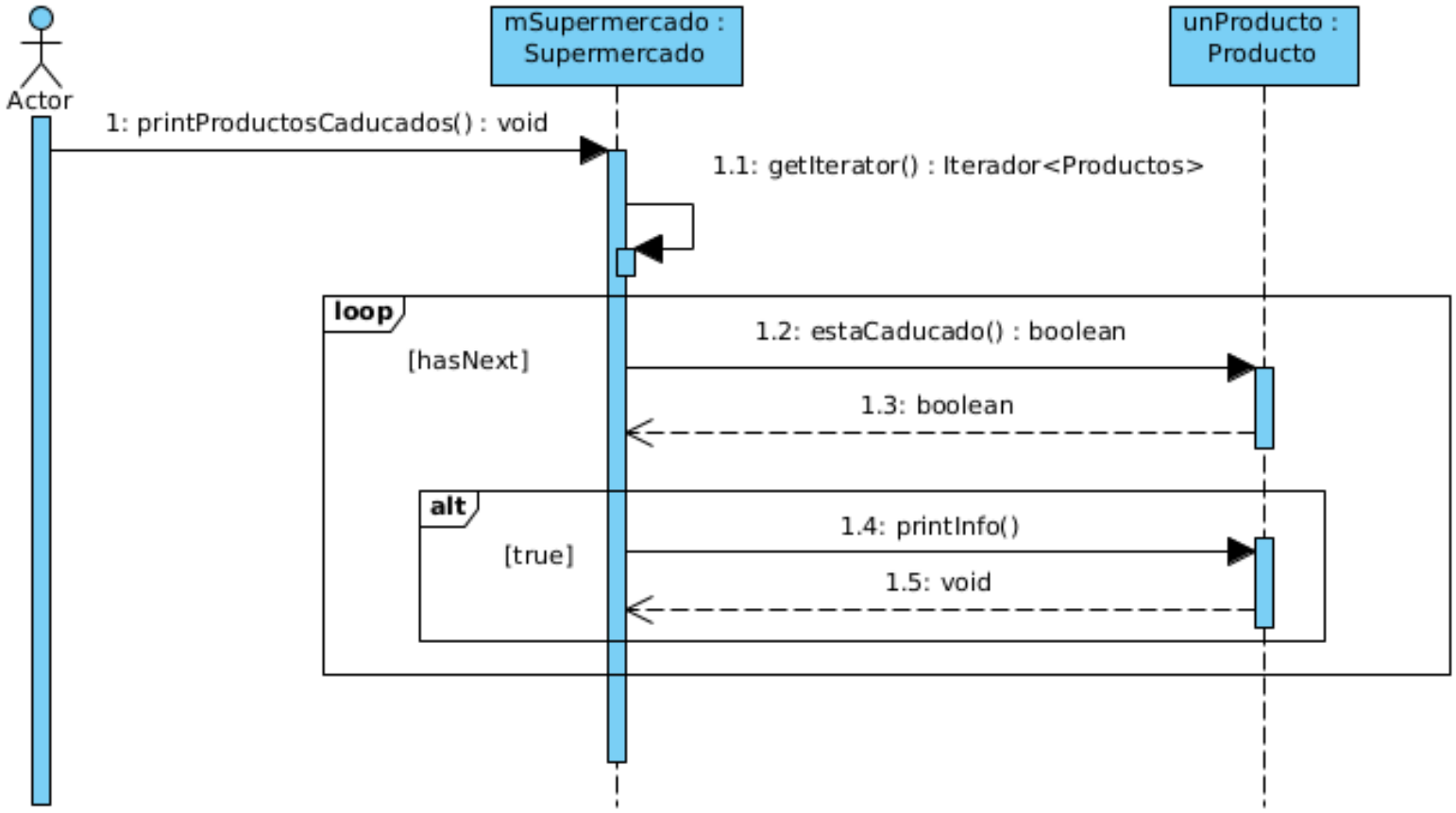
Ingeniería del Software

Índice

Introducción

Dada la definición de la clase Supermercado, que contiene la lista de los productos que tiene a la venta un establecimiento, se quiere implementar un método printProductosCaducados, que visualiza (print) productos cuya fecha de caducidad haya expirado.




Actor
mSupermercado : Supermercado
unProducto : Producto
1: printProductosCaducados() : void
1.1: getIterator() : Iterador<Productos>
loop
[hasNext]
1.2: estaCaducado() : boolean
1.3: boolean
alt
[true]
1.4: printInfo()
1.5: void

Una primera aproximación

```
public void printProductosCaducados() {
   for (Producto producto : listaProductos) {
      if (producto.estaCaducado()){
         producto.printInfo();
      }
   }
}
```

Sin Iterator...

Ahora nos piden… productos en la sección 2

```
public void printProductosSec2() {
    for (Producto producto : listaProductos) {
        if (producto.getSeccion().equals(2)){
            producto.printInfo();
        }
    }
}
```

Ahora nos piden… productos que cuestan más de 12 €

```
public void printProductos12() {
    for (Producto producto : listaProductos) {
        if (producto.getPrecio() > 12){
            producto.printInfo();
        }
    }
}
```

Ahora nos piden...

```
public void printProductosCaducados() {
    for (Producto producto : listaProductos) {
        if (producto.estaCaducado()){
            producto.printInfo();
        }
```

```
public void printProductos12() {
    for (Producto producto : listaProductos) {
        if (producto.getPrecio() > 12){
            producto.printInfo();
        }
    }
}
```

```
public void printProductosSec2() {
    for (Producto producto : listaProductos) {
        if (producto.getSeccion().equals(2)){
            producto.printInfo();
        }
    }
}
```

Ahora nos piden…

```
public void printProductosCaducados() {
    for (Producto producto : listaProductos) {
        if (producto.estaCaducado()){
            producto.printInfo();
        }
```

```
public void printProductos12() {
    for (Producto producto : listaProductos) {
        if (producto
            producto
        }
    }
}
```

¿Cómo podemos dar respuesta a este nuevo cambio en los requisitos **sin repetir** cantidades de código?

```
public void printProductosSec2() {
    for (Producto producto : listaProductos) {
        if (producto.getSeccion().equals(2)){
            producto.printInfo();
        }
    }
}
```

Ingeniería del Software

Índice

Java8 – Behaviour parametrization

Ahora nos piden…

```
public void printProductosCaducados() {
    for (Producto producto : listaProductos) {
        if (producto.estaCaducado()){
            producto.printInfo();
        }
```

```
public void printProductos12() {
    for (Producto producto : listaProductos) {
        if (producto
            producto
        }
    }
}
```

¿Cómo podemos dar respuesta a este nuevo cambio en los requisitos **sin repetir** cantidades de código?

```
public void printProductosSec2() {
    for (Producto producto : listaProductos) {
        if (producto.getSeccion().equals(2)){
            producto.printInfo();
        }
    }
}
```

Pasamos como parámetro la parte que cambia + Interfaz funcional

Java8 – Behaviour parametrization

```
public interface Predicate{
     boolean test(Producto producto)
}
```

```
public class Productos12 implements Predicate {
     @Override
     public boolean test(Producto producto)
     {
          return producto.getPrecio()>12;
     }
}
```

```
public class ProductosS2 implements Predicate {
     @Override
     public boolean test(Producto producto)
     {
          return producto.getSeccion.equals(2);
     }
}
```

```
public class ProductosCad implements Predicate {
     @Override
     public boolean test(Producto producto)
     {
          return producto.estaCaducado();
     }
}
```

Implementamos cada uno de los comportamientos

+Interfaz

Java8 – Behaviour parametrization

```
public void filtrar(Predicate pPred){
    for (Producto p : listaProductos){
        if(pPred.test(p))
            p.printInfo();
    }
}
```

Supermercado

```
public interface Predicate{
    boolean test(Producto producto)
}
```

```
public class Productos12 implements Predicate {
 public class ProductosS2 implements Predicate {
  public class ProductosCad implements Predicate {
      @Override
      public boolean test(Producto producto)
      {
          return producto.estaCaducado();
```

```
Msupermercado.filtrar(new Productos12());
Msupermercado.filtrar(new ProductosS2());
Msupermercado.filtrar(new ProductosCad());
```

Main

Java8 – Behaviour parametrization

Seguimos teniendo el mismo problema… 1 clase por cada implementación…

Existen las Interfaces funcionales…

¡Expresiones Lambda!

Proporcionan implementaciones de interfaces sin definir clases

```
public class ProductosS2 implements Predicate {
    @Override
    public boolean test(Producto producto)
    {
        return producto.getSeccion.equals(2);
    }
}
```

```
p → p.getSeccion.equals(2)
```

Proporcionan implementaciones de interfaces sin definir clases

```
public class ProductosS2 implements Predicate {
    @Override
    public boolean test(Producto producto)
    {
        return producto.getSeccion.equals(2);
    }
}
```

Parametro de entrada

```
p → p.getSeccion.equals(2)
```

Implementación

Proporcionan implementaciones de interfaces sin definir clases

```
public class ProductosS2 implements Predicate {
    @Override
    public boolean test(Producto producto)
    {
        return producto.getSeccion.equals(2);
    }
}
```

Parametro de entrada

p → p.getSeccion.equals(2)

Implementación

p → p.getPrecio()

p → p.estaCaducado()

Java8 – Expresiones Lambda

Sintaxis: (parametros) → cuerpo

- Los parámetros son la lista de parámetros formales del método abstracto de la interfaz funcional
  - **(Producto p) → p.getPrecio()**
- Se puede omitir el tipo de parámetro
  - **(p) → p.getPrecio()**
- Si sólo hay un parámetro se pueden omitir los paréntesis
  - **p → p.getPrecio()**
- Si el cuerpo consiste en un bloque de instrucciones → añadir llaves
  - **(p,pr) → {p.getPrecio()>pr}**

Java8 – Expresiones Lambda

```
public void filtrar(Predicate<Producto> pPred){
    for (Producto p : listaProductos){
        if(pPred.test(p))
            p.printInfo();
    }
}
```

Supermercado

```
public interface Predicate<P>{
    boolean test(P p)
}
```

```
Msupermercado.filtrar(p → p.estaCaducado());
Msupermercado.filtrar(p → p.getSeccion().equals(2));
Msupermercado.filtrar(p → p.getPrecio()>12);
```

Main

Java8 – Expresiones Lambda

```
public void filtrar(Predicate<Producto> pPred){
    for (Producto p : listaProductos){
        if(pPred.test(p))
            p.printInfo();
    }
}
```

Supermercado

```
public interface Predicate<P>{
    boolean test(P p)
}
```

```
Msupermercado.filtrar(p → p.estaCaducado());
Msupermercado.filtrar(p → p.getSeccion().equals(2));
Msupermercado.filtrar(p → p.getPrecio()>12);
```

Main

- Si una clase dispone de un método con la signatura de una interfaz funcional, se puede pasar como parámetro
  - **Sintaxis:**
    - Clase::metodo
    - Objeto::metodo

`p → p.estaCaducado() = Producto::estaCaducado`

Ingeniería del Software

Índice

Operaciones de agregacion y Streams

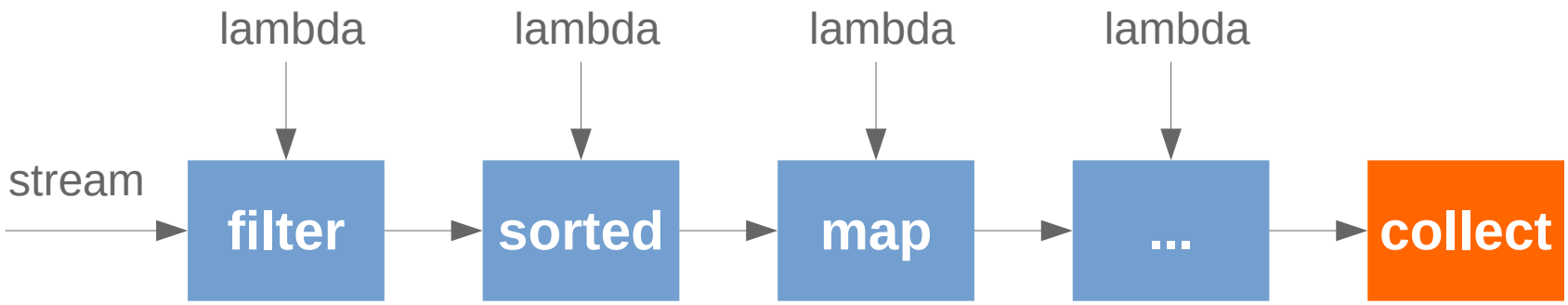
- Implementaciones de los algoritmos mas comunes
  - Filtrado
  - Map
  - Foreach
  - Suma
  - …
- Se utilizan con las expresiones Lambda
- Para iterar? → ¡Streams!

Operaciones de agregacion y stream ¿Como funcionan?

- **Pipelines**, secuencias encadenadas de flujo de datos
  - **Flujos**
    - Stream()
    - parallelStream()
  - Iteran internamente (sin next)
  - **Despues aplicamos operaciones**
    - Intermedias: map, filter, sorted
    - Finales: collect, sum, forEach

Operaciones de agregacion y stream ¿Como funcionan?

- **Pipelines**, secuencias encadenadas de flujo de datos



Operaciones de agregacion y stream ¿Como funcionan?

- **Pipelines**, secuencias encadenadas de flujo de datos

```
public List<String> getNombreProductosCaducados () {
return listaProductos
.stream() //convertir a stream de Productos
.filter(Producto::estaCaducado)// stream de Productos
.map(p->p.getNombre())// stream de Strings
.collect(toList()); //convertir a lista de Strings
}
```

Stream o parallelStream

Stream o parallelStream

Operaciones de agregacion y stream (intermedias)

OP	Argumento	Tipo Devuelto	Propósito
`filter`	`Predicate<T>`	`Stream<T>`	Devuelve un nuevo flujo que consiste en los elementos que satisfacen el predicado.
`map`	`Function<T,R>`	`Stream<R>`	Devuelve un nuevo flujo que consiste en los resultados de aplicar la función a cada elemento del flujo. Existen variantes para tipos primitivos (`mapToInt` o `mapToDouble`).
`sorted`	`Comparator<T>`	`Stream<T>`	Devuelve un nuevo flujo que consiste en los elementos ordenados por el criterio especificado.
`distinct`		`Stream<T>`	Devuelve un nuevo flujo que consiste en elementos no repetidos.

Operaciones de agregacion y stream (intermedias)

```
public Double getPrecioTotalCaducados () {
return listaProductos.stream()
.filter(Producto::estaCaducado) // Filtrar
.mapToDouble(Producto::getPrecio) // Obtener stream de precios
.sum(); // sumar todos los precios
}
```

Operaciones de agregacion y stream (finales)

OP	Argumento	Tipo Devuelto	Propósito
`forEach`	`Consumer<T>`	`void`	Consume cada elemento del flujo aplicando la lambda especificada (método `void`).
`count`		`long`	Devuelven el numero de elementos del flujo.
`collect`	`Collector<T,A,R>`	`(genérico)`	Reduce el flujo para generar una lista mapa o incluso un valor entero en función del método de recolección especificado.
`anyMatch`	`Predicate<T>`	`boolean`	Devuelve true si en el flujo hay algún elemento que cumple con la condición del predicado
`allMatch`	`Predicate<T>`	`boolean`	Devuelve true si todos los elementos del flujo cumplen con la condición del predicado

Operaciones de agregacion y stream (finales) en flujos numéricos (IntStream o DoubleStream)

OP	Arg.	Tipo Devuelto	Propósito
`average`		`OptionalDouble`	Devuelve la media de los elementos del flujo.
`summaryStatistics`		`IntSummaryStatistics o DoubleSummaryStatistics`	Devuelve estadísticas de los elementos del flujo.
`sum`		`int o double`	Devuelve la suma de los elementos de la secuencia.

Operaciones de agregacion y stream (finales) de recolección

```
import static java.util.stream.Collectors.*;
```

OP	Argumento	Tipo Devuelto	Propósito
toList		Collector	Devuelve un colector que agrupa los elementos del flujo en una lista.
partitioningBy	Predicate<T>	Map<Boolean,D>	Devuelve un colector que clasifica los elementos del flujo de acuerdo a un predicado y aplica una función de resumen.
groupingBy	Function<T>	Map<K,D>	Devuelve un colector que agrupa los elementos de acuerdo a una función de clasificación y aplica una función de resumen.

Operaciones de agregacion y stream (finales) de recolección
Ejemplo partitioningBy tipo busqueda

```
public Map<Boolean,Double> getPrecioMedioCaducadosNoCad() {
return listaProductos.stream()
.collect(partitioningBy(Producto::estaCaducado,
averagingDouble(Producto::getPrecio)) );
}
```

¡Devuelve precio medio de caducados y no caducados!
partitioningBy (**True** or **False**) → 2 streams

Operaciones de agregacion y stream (finales) de recolección
Ejemplo groupingBy tipo colección

```
public Map<String,Double> getProductoMasBaratoPorSeccion() {
return listaProductos.stream().collect(groupingBy(
Producto::getSeccion,CollectingAndThen(
minBy(comparingDouble(Producto::getPrecio)),
p->p.get().getPrecio())));
}
```

¡Devuelve el precio del producto mas barato por sección!
groupingBy (porSeccion) **xStreams** (queAtributoYFiltrar, queDevolver)

Ingeniería del Software

Índice

Java8 – Optional

Motivación

- ¿Cuál es la media de una secuencia vacía de números?
- ¿Qué valor se debe devolver en una búsqueda si ningún elemento satisface el criterio de búsqueda?

Optional <T>

- Tipo de datos que encapsula el valor (en caso de que exista). Dispone de métodos para comprobar
  - si esta vacío: isPresent()
  - obtener el valor: get()
  - obtener un valor por defecto si esta vacío: orElseGet()
  - . . .
- Hay implementaciones específicas para tipos primitivos (OptionalDouble,. . .)

Java8 – Optional

Interfaces

- En Java 8, se ha incorporado la posibilidad de proporcionar implementación a métodos en la propia interfaz

Ejemplo

- No es necesario implementar estos métodos en las clases
- Se pueden definir métodos estáticos en la interfaz (no se pueden sobreescribir en las clases)

```
public interface DoIt {
    void doSomething ( int i , double x ) ;
    default boolean didItWork ( int i , double x ) {
        // Implementacion
    }
}
```

Ingeniería del Software

Preguntas