Clase 20: Java 8 (Parte I)

Programación & Laboratorio III

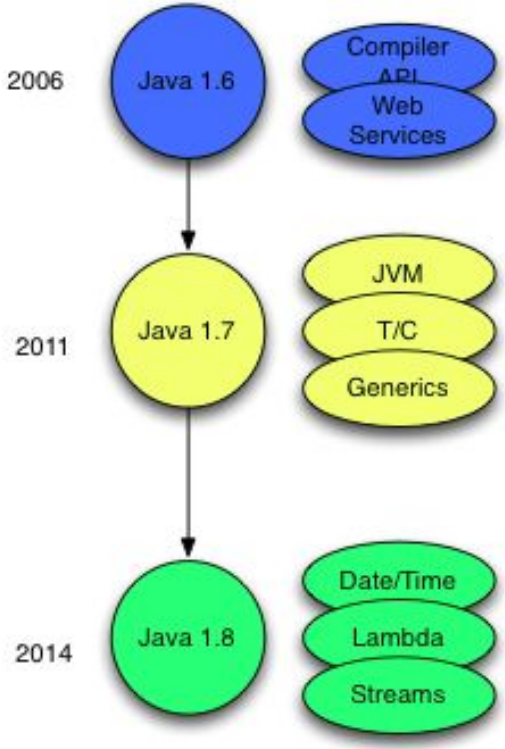
Agenda

- **Introducción**
- Interfaces en Java 7
- Interfaces en Java 8
  - Métodos default
  - Métodos estáticos
  - Interfaces funcionales
- Programación funcional
- Clases anónimas
- Expresiones lambda

Introducción

“En JAVA 8 hemos realizado el mayor cambio desde que nació la plataforma“

Versión 1.8: Se abren las puerta a la programación funcional con el uso de expresiones Lambda y Streams. Se realiza una revisión de APIS y se actualiza de forma importante la gestión de fechas.



Agenda

- ~~Introducción~~
- **Interfaces en Java 7**
- Interfaces en Java 8
  - Métodos default
  - Métodos estáticos
  - Interfaces funcionales
- Programación funcional
- Clases anónimas
- Expresiones lambda

Interfaces en Java 7

- Se utiliza para definir un conjunto de métodos relacionados entre sí.
- Se especifica qué se debe hacer, pero no su implementación.
- Similar a una clase abstracta, pero con **todos** sus métodos sin implementación.
- Ejemplo:

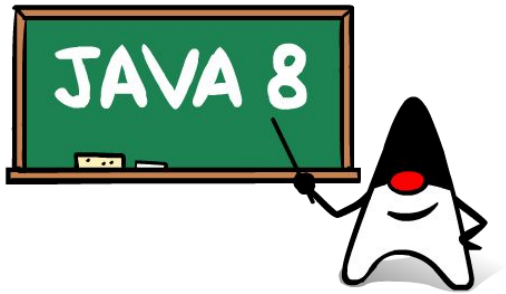
```
public interface List<E> extends Collection<E>{

     int size();
     boolean add(E e);
     boolean remove(Object o);
     ...

}
```

Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- **Interfaces en Java 8**
  - Métodos default
  - Métodos estáticos
  - Interfaces funcionales
- Programación funcional
- Clases anónimas
- Expresiones lambda

Interfaces en Java 8

- Diseñar interfaces siempre ha sido un trabajo difícil, porque si queremos agregar métodos adicionales en las interfaces, se requerirá un cambio en todas las clases de implementación.
- Es posible que la cantidad de clases que la implementa la interfaz crezca hasta tal punto que no sea posible ampliarla.



Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- **Interfaces en Java 8**
  - **Métodos default**
  - Métodos estáticos
  - Interfaces funcionales
- Programación funcional
- Clases anónimas
- Expresiones lambda

Interfaces en Java 8 - Métodos default

- Los métodos default permiten agregar nueva funcionalidad a las interfaces, definiendo su implementación por defecto.
- Ventaja: no es necesario modificar las clases que hagan uso de esa interfaz.
- El método default se hereda por las clases que implementan la interfaz.
- Se puede dejar el default method sin cambios o redefinirlo.

Interfaces en Java 8 - Métodos default (2)

- Ejemplo:

```
public interface List<E> extends Collection<E>{
     int size();
     boolean add(E e);
     ...
     default void sort(Comparator<? super E> c) {
          Object[] a = this.toArray();
          Arrays.sort(a, (Comparator) c);
          ListIterator<E> i = this.listIterator();
          for (Object e : a) {
               i.next();
               i.set((E) e);
          }
     }
}
```

Interfaces en Java 8 - Métodos default (3)

- El modificador **`default`** nos permite crear un nuevo método en la interfaz, pero definiendo su implementación por defecto, de tal manera que la operación no requiera modificar las clases que hagan uso de esa interfaz.

Entonces, ¿cuál es la diferencia entre una clase abstracta y un interfaz que implementa métodos default?

- La diferencia es mínima. Una clase sólo puede extender una clase abstracta pero puede implementar múltiples interfaces. Además, en un interfaz no se pueden declarar variables de instancia, mientras que sí se puede hacer en una clase abstracta.

Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- **Interfaces en Java 8**
  - ~~Métodos default~~
  - **Métodos estáticos**
  - Interfaces funcionales
- Programación funcional
- Clases anónimas
- Expresiones lambda

Interfaces en Java 8 - Métodos estáticos

- Actualmente en el API de Java existen muchas clases de utilidades usadas para dar soporte a los interfaces implementando métodos estáticos que son utilizados por todas las clases que hereden de ese interfaz. Un ejemplo de ello es la clase `Collections` que da soporte a todas las clases que heredan del interfaz Collection.
- Ahora existe la posibilidad de crear estos métodos estáticos directamente en el interfaz, permitiendo tener una mejor organización del código.

Interfaces en Java 8 - Métodos estáticos (2)

- Ejemplo:

```
public interface Map<K,V> {
     int size();
     boolean containsKey(Object key);
     …

     public static <K extends Comparable<? super K>, V>
Comparator<Map.Entry<K,V>> comparingByKey() {
          return (Comparator<Map.Entry<K, V>> & Serializable)
                    (c1, c2) -> c1.getKey().compareTo(c2.getKey());
     }
}
```

Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- **Interfaces en Java 8**
  - ~~Métodos default~~
  - ~~Métodos estáticos~~
  - **Interfaces funcionales**
- Programación funcional
- Clases anónimas
- Expresiones lambda

Interfaces en Java 8 - Interfaces funcionales

- Se conoce como interfaz funcional a toda aquella interfaz que tenga solamente un método abstracto, es decir puede implementar uno o más métodos default, pero deberá tener forzosamente un único método abstracto (un método sin implementación).
- Ejemplo:

```
@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
```

Interfaces en Java (8) - Interfaces funcionales (2)

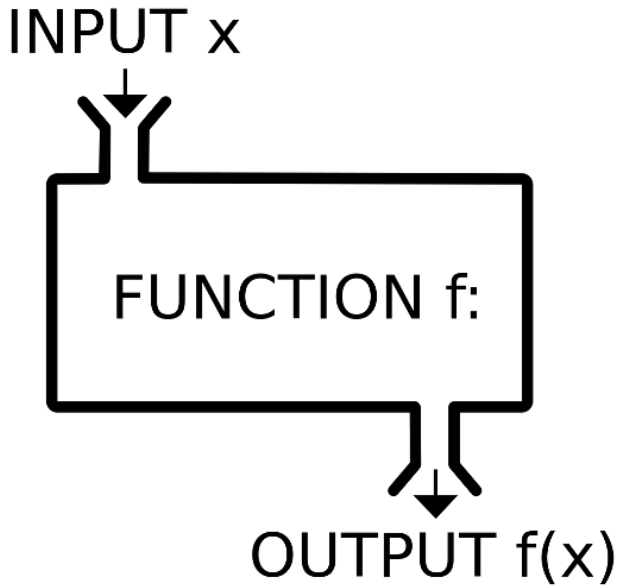
- `@FunctionalInterface` indica la intención de que la interfaz es funcional, produciendo un error de compilación en caso de agregar más métodos abstractos.
- El IDE automáticamente nos arrojará un error si no cumplimos con las reglas de una interfaz funcional.

Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- ~~Interfaces en Java 8~~
  - ~~Métodos default~~
  - ~~Métodos estáticos~~
  - ~~Interfaces funcionales~~
- **Programación funcional**
- Clases anónimas
- Expresiones lambda

Programación funcional

- La programación funcional es un paradigma basado en el uso de funciones matemáticas.



Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- ~~Interfaces en Java 8~~
  - ~~Métodos default~~
  - ~~Métodos estáticos~~
  - ~~Interfaces funcionales~~
- ~~Programación funcional~~
- **Clases anónimas**
- Expresiones lambda

Clases anónimas

- Las clases anónimas representan el caso de clases internas más extraño que se puede presentar.
- Una clase anónima es una clase sin nombre, definida en la misma línea de código donde se crea el objeto de la clase. Esta operación se lleva a cabo en el interior de un método de otra clase, por ello la clase anónima es considerada como una clase interna anidada.

Clases anónimas (2)

- Ejemplo:

```
public interface EstrategiaMultiplicacion {
    int multiplicar(int i);
}
...
public static void main(Strig[] args) {
    EstrategiaMultiplicacion e = new EstrategiaMultiplicacion() {
        @Override
        int multiplicar(int i) {
            return i * 2;
        }
    }
}
```

Agenda

- ~~Introducción~~
- ~~Interfaces en Java 7~~
- ~~Interfaces en Java 8~~
  - ~~Métodos default~~
  - ~~Métodos estáticos~~
  - ~~Interfaces funcionales~~
- ~~Programación funcional~~
- ~~Clases anónimas~~
- **Expresiones lambda**

Expresiones lambda

- En todos los lugares en los que se espera una interfaz funcional tenemos tres opciones:
  1) Pasar una clase interna anónima.
  2) Pasar una expresión de lambda.
  3) Pasar una referencia a un método en vez de una expresión de lambda, en algunos casos.

Expresiones lambda (2)

- Ejemplo: Nos piden que dupliquemos los valores en una lista de números enteros.
- Solución:

```
public class Duplicador {
    public List<Integer> duplicarValores(List<Integer> lista) {
       List<Integer> resultado = new ArrayList<>();
       for (Integer i : lista) {
           resultado.add(i * 2);
       }
       return resultado;
    }
}
```

Expresiones lambda (3)

- Nos piden una solución para que se puedan triplicar los valores de la lista.

```
public interface EstrategiaMultiplicacion {
    int multiplicar(int i);
}

public class Duplicacion implements EstrategiaMultiplicacion {
   @Override
   public int multiplicar(int i) {
      return i * 2;
   }
}
```

Expresiones lambda (4)

```
public class Triplicacion implements EstrategiaMultiplicacion {
    @Override
    public int multiplicar(int i) {
       return i * 3;
    }
}

public class Multiplicacion {
    public List<Integer> multiplicar(List<Integer> l, EstrategiaMultiplicacion est) {
        List<Integer> resultado = new ArrayList<>();
        for (Integer i : lista) {
            resultado.add(estrateg.multiplicar(i));
        }
        return resultado;
    }
}
```

Expresiones lambda (5)

- O podemos crear clases anónimas:

```
public static void main(String[] args) {
  Multiplicacion m = new Multiplicacion();
  ...
  m.multiplicar(lista, new EstrategiaMultiplicacion() {
      @Override
      public int multiplicar(int i) {
         return i * 2;
      }
  });
}
```

Expresiones lambda (6)

- Solución con expresiones lambda:

```
@FunctionalInterface
public interface EstrategiaMultiplicacion {
    int multiplicar(int i);
}
```

```
public static void main(String[] args) {
    m.multiplicar(lista, new
EstrategiaMultiplicacion() {
        @Override
        public int multiplicar(int i) {
           return i * 2;
        }
    });
  }
```

```
public static void main(String[] args) {
    m.multiplicar(lista, i -> i * 2);
}
```

Expresiones lambda (7)

- Las expresiones lambda son funciones anónimas, es decir, que no necesitan una clase.

(parámetros) -> {cuerpo de lambda}

- El operador lambda (->) separa la declaración de parámetros de la declaración del cuerpo de la función.
- Parámetros: cuando se tiene un solo parámetro no es necesario utilizar los paréntesis.
- Cuerpo de lambda: cuando el cuerpo de la expresión lambda tiene una única línea no es necesario utilizar las llaves.

Expresiones lambda (8)

Java 7	Java 8 con expresiones lambda
`int z = z + 2;`	`z -> z + 2;`

`public void sayHelloWorld() {`
`  System.out.println(”Hello World!“);`
`}`
	`() -> System.out.println(”Hello World!“);`

`public int getArea(int alto, int ancho) {`
`  return alto * ancho;`
`}`
	`(int longitud, int altura) -> { return altura * longitud; }`

Expresiones lambda (9)

- Una expresión lambda es una instancia de una interfaz funcional. Pero no contiene información sobre qué interfaz funcional está implementando.
- Una expresión lambda puede ser compatible con múltiples interfaces funcionales. Esto implica que se puede utilizar la misma lambda expression para diferentes contextos.

Bibliografía oficial

- http://www.oracle.com/technetwork/articles/java/architect-lambdas-part1-2080972.html
- http://www.oracle.com/technetwork/articles/java/architect-lambdas-part2-2081439.html
- https://www.oreilly.com/learning/java-8-functional-interfaces
- https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html