ITERADORES FOREACH

En java 8 se añaden muchas características de programación funcional una de ellas es el método foreach que permite utilizando expresiones lambda iterar sobre coleccion.

¡IMPORTANTE ! CONSULTAR API DE JAVA 8(o superior) NO API DE JAVA 7. Recuerda que todo lo relativo a programación funcional es novedad de java 8 y superiores

Realmente hay dos versiones diferentes de foreach:

- la del interface Iterable, que la usara toda la jerarquía que hay bajo Iterable (listas, pilas, colas, …)

forEach

```
default void forEach(Consumer<? super T> action)
```

- la del interface Map para usar con mapas

forEach

```
default void forEach(BiConsumer<? super K,? super V> action)
```

Como ya vimos en otro boletín, la sintaxis de los genéricos puede resultar dura, pero su aplicación práctica como vamos a ver ahora es intuitiva.

EL MÉTODO forEach() de Iterable

forEach

```
default void forEach(Consumer<? super T> action)
```

Observamos en el API que su parámetro debe ser una instancia de la interface funcional Consumer, así que podemos pasar como parámetro una expresión lambda que "encaje" con este interface.
si consultamos API de *Consumer* observamos el método que hay que implementar

Method Detail

accept

```
void accept(T t)
```

Performs this operation on the given argument.

Parameters:

`t - the input argument`

accept() es un método que no devuelve nada pero se supone que hace internamente alguna acción con el valor del parámetro por eso se utiliza para la variable referencia el parámetro el nombre *action*.

Ejemplo:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;


class Camisa {
    private  String Color;
    private String talla;
```

```
    public Camisa(String Color, String talla) {
        this.Color = Color;
        this.talla = talla;
    }

    public String getColor() {
        return Color;
    }

    public void setColor(String Color) {
        this.Color = Color;
    }

    public String getTalla() {
        return talla;
    }

    public void setTalla(String talla) {
        this.talla = talla;
    }

    @Override
    public String toString() {
        return "Camisa{" + "Color=" + Color + ", talla=" + talla + '}';
    }

}


public class App{
     public static List<Camisa> filtrar(List<Camisa> listaCamisas, Predicate<Camisa> cp){
        List<Camisa> camisasFiltradas= new ArrayList<>();
        for(Camisa camisa:  listaCamisas){
            if(cp.test(camisa))
                camisasFiltradas.add(camisa);
        }
        return camisasFiltradas;
    }

    public static void main(String[] args) {
            Camisa[] arrayCamisas={new Camisa("ROJO","M"),new Camisa("ROJO","XL"),new Camisa("BLANCO","L"),new
Camisa("BLANCO","XL")};
        List<Camisa> listaCamisas=Arrays.asList(arrayCamisas);
        List<Camisa> camisasRojas=filtrar(listaCamisas,c->c.getColor().equals("ROJO"));
        System.out.print("\nTallas de camisas rojas con for mejorado  tradicional: ");
        for(Camisa c:camisasRojas){
            System.out.print(c.getTalla()+" ");
        }
        System.out.print("\nTallas de camisas rojas con for each funcional con lambda: ");
        camisasRojas.forEach(c->System.out.print(c.getTalla()+" "));
        System.out.print("\nTallas de camisas rojas con for each funcional con referencia a métodos: ");
        camisasRojas.forEach(System.out::print);
    }


 }
```

Observa el estilo "declarativo" del foreach, simplemente dices lo que quieres hacer para cada elemento de la lista. Elimina de tu cabeza el bucle que recorre secuencialmente una lista, ahora la iteración es automática, tú solo tienes que "declarar" lo que quieres para cada elemento

Ejercicio U8_B5_E1: Observa el siguiente for mejorado tradicional, añade un foreach con lambda equivalente.

```
import java.util.ArrayList;
import java.util.List;

class App{
    public static void main(String[] args){
        List<String> items = new ArrayList<>();
                items.add("A");
                items.add("B");
                items.add("C");
                items.add("D");
                items.add("E");

                for(String item : items){
```

```
                System.out.println(item);
            }

    }
}
```

Ejercicio U8_B5_E2: añade al ejercicio anterior una versión con referencia a métodos.

Ejercicio U8_B5_E3: Reescribe el for "normal" de abajo de dos maneras:

```
import java.util.ArrayList;
import java.util.List;

class App {

    public static void main(String[] args) {
        List<String> items = new ArrayList<>();
        items.add("A");
        items.add("B");
        items.add("B");
        items.add("D");
        items.add("E");

        for (String item : items) {
            if (item.equals("B")) {
                System.out.println(item);
            }
        }
    }
}
```

1. Creando una instancia de Consumer y luego dentro del for invocando expresamente al método accept.

```
for (String item : items) {
        c.accept(item);
 }
```

2. Con foreach()

Borrar elementos de una lista con removeif, no necesita for ni foreach

```
//¡adios iterator!

import java.util.*;
class App {
    public static void main(String[] args) {
        List<Integer> numeros = new ArrayList<>(Arrays.asList(1,2,3,4,5,6,7,8,9));

        numeros.removeIf(n->n%2==0);
        System.out.println(numeros);
    }
}
```

Puedes consultar en el API que este potente método está definido en el interface Collections y por tanto es implementado por todo lo que está debajo de él(listas y sets)

EL MÉTODO forEach() de Map

Ya que los mapas son un poco distintos interna y externamente respecto a las otras colecciones, la interface Map tiene su "propio forEach()". Observa en el api que forEach() espera algo de tipo BiConsumer que tiene dos tipos parametrizados, el primero se corresponde con la clave y el segundo con el valor

```
import java.util.HashMap;
import java.util.Map;

class App{
   public static void main(String[] args){
      Map<String,Integer> m= new HashMap<>();
      m.put("yo",32); m.put("tú",42); m.put("él",62);
      m.forEach((nombre,edad)->System.out.println(nombre+ " "+edad) );
   }
}
```

Observa que no es posible para la última instrucción
`m.forEach(System.out::println);`
ya que el compilador no va encontrar ningún println con dos parámetros de la forma println(Object x, Object y) en el API

Ejercicio U8_B5_E4: Vuelve a escribir el siguiente código

```
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

class App{
   public static void main(String[] args){
      Map<String, Integer> items = new HashMap<>();
         items.put("A", 10);
         items.put("B", 20);
         items.put("C", 30);
         items.put("D", 40);
         items.put("E", 50);
         items.put("F", 60);

         for (Entry<String, Integer> entry : items.entrySet()) {
                  System.out.println("Clave: " + entry.getKey() + " Valor: " + entry.getValue());
         }

   }
}
```

usando un foreach(). Llegarás a la conclusión que el foreach es un "chollazo" porque es ¡muy fácil y natural!. Observa que no tendrás que usar métodos tipo getKey() o getValue() ya que eso lo hace internamente el código que implementa el foreach() del map

Ejercicio U8_B5_E5:
Añadir la capacidad al ejemplo anterior que cuando la clave se "E" se imprima un saludo "HOLA E".

```
clave A con valor 10
clave B con valor 20
clave C con valor 30
clave D con valor 40
clave E con valor 50
HOLA E
clave F con valor 60
```