Acumuladores booleanos

Un acumulador booleano utiliza las propiedades de la lógica para acumular un valor de verdad (verdadero o falso) durante la vida de un algoritmo.
En general estos algoritmos son una implementación de una función booleana.

Sean a y b dos variables booleanas. Las propiedades lógicas para **and**(&&) y **or**(||) son:

	a	b	A || b	a && b
(1)	f	f	f	f
(2)	f	v	v	f
(3)	v	f	v	f
(4)	v	v	v	v

Son muy útiles cuando se quiere chequear una propiedad booleana en una estructura, por ejemplo, en una lista de valores.

Ejemplo1a:
Queremos ver si toda una lista de números es par:

```
boolean esPar(lista)
for (int i=0, i++, i < lista.size())
     if !(par(lista[i]))
          return false
return true
```

esPar Version1a sin acumuladores

```
boolean esPar(lista)
boolean ret = true
for (int i=0, i++, i < lista.size())
     ret = ret && par(lista[i])
return ret
```

esPar Version2a con acumuladores

Veamos que sucede con la lista [2,3,10] para el ejemplo1a:

i	Sin acumuladores (valor de retorno)	Con acumuladores (valor de ret)	Obs
0	v	v	Ret = v && par(2) Ret = v && v
1	f	f	Ret = v && par(3) Ret = v && f
2	(no llega a este punto)	f	Ret = f && par(10) Ret = f && v

Por ahora, las dos implementaciones son muy parecidas.

Ejemplo1b:
Ahora también queremos devolver la lista con todos sus elementos multiplicados por 2.

```
boolean esPar(lista)
for (int i=0, i++, i < lista.size())
     if !(par(lista[i]))
          return false
     lista[i] = lista[i] * 2
return true
```

esPar Version1b sin acumuladores

Pero esto no siempre funciona. Si no llegamos hasta el final de la lista, no multiplicaremos el resto de la lista por 2.
Veamos como lo podemos arreglar.

```
boolean esPar(lista)
for (int i=0, i++, i < lista.size())
     if !(par(lista[i]))
          return false
for (int i=0, i++, i < lista.size())
     lista[i] = lista[i] * 2
return true
```

esPar Version1c sin acumuladores

Pero esto no siempre funciona. Al multiplicar por 2, todas las listas quedan pares! Pero no es cierto que la lista original sea par. Ej [1,3,5] es impar, mientras que [2,6,10] es par!

```
boolean esPar(lista)
copiaLista = lista.clonar()
for (int i=0, i++, i < lista.size())
     lista[i] = lista[i] * 2
for (int i=0, i++, i < copiaLista.size())
     if !(par(copiaLista[i]))
          return false
return true
```

esPar Version1d sin acumuladores

Finalmente funciono, pero a que costo?

```
boolean esPar(lista)
boolean ret = true
for (int i=0, i++, i < lista.size())
     ret = ret && par(lista[i])
     lista[i] = lista[i] * 2
return ret
```

esPar Version2b con acumuladores

Comparación de codificar con acumuladores, respecto de hacerlo sin:

Sin acumuladores	Con acumuladores
Código mas largo	
	Mas declarativo
Promueve la introducción de errores	
Difícil de modificar/agregar código.	
El código termina antes en algunos casos. Pero demostraremos que ese “ahorro” no tiene un peso significativo en la mayoría de los casos.	Se recorre siempre hasta el final.

Ejemplo2:

Queremos ver si algún número de la lista es par:
En este caso utilizaremos el or para acumular.

```
boolean algunPar(lista)
for (int i=0, i++, i < lista.size())
     if (par(lista[i]))
          return true
return false
```

algunPar Version1 sin acumuladores

```
boolean algunPar(lista)
boolean ret = false
for (int i=0, i++, i < lista.size())
     ret = ret || par(lista[i])
return ret
```

algunPar Version2 con acumuladores

Regla general para acumulación booleana

cuando la hipótesis es:
que la propiedad se cumple, ret comienza valiendo true y la acumulación es con “and”.
cuando la hipótesis es:
que la propiedad no se cumple, ret comienza valiendo false y la acumulación es con “or”.

Otros atajos

En lugar de preguntar por
If (variable == true)
    Return true
}else{
    Return false

Podemos simplemente devolver

return variable

Ejercicio:

Realizar una función que dada una lista de números devuelva verdadero si:

La lista tiene todos los números mayores que 8

La lista tiene algún número menor que 23

Utilizando acumuladores boléanos

```
public class Ppal {

	public static void main(String[] args) {
		Integer [] lista1 = {9,10,11};		//cumple
		Integer [] lista2 = {90,100,110};	//no cumple

	System.out.println(todosMayor8(lista1) && algunoMenor23(lista1));//true
	System.out.println(todosMayor8(lista2) && algunoMenor23(lista2));//false
	}

	static public boolean todosMayor8(Integer [] lista){
		boolean ret = true;
		for (int i = 0; i < lista.length; i++){
			ret = ret && lista[i] > 8;
		}
		return ret;
	}


	static public boolean algunoMenor23(Integer [] lista){
		boolean ret = false;
		for (int i = 0; i < lista.length; i++){
			ret = ret || lista[i] < 23;
		}
		return ret;
	}
```

Nota:

Cuando una propiedad se quiere probar sobre toda la lista, se utiliza “and”
Cuando propiedad se quieren demostrar sobre algún elemento de la lista, se utiliza “or”