Iteradores-Java

http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html
http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html

Iteradores

Se utilizan para realizar el recorrido de los elementos de una colección de una forma más intuitiva y rápida que con el bucle for tradicional.

Están basados en

- El interfaz iterable – permite utilizar el bucle foreach
- El interfaz iterator.

Las colecciones implementan el interfaz iterable y tienen un método iterator() que devuelve un objeto de tipo iterator

Se recomienda utilizar iterable en lugar de iterator o de un for normal.

*También tenemos la clase **Enumeration** (parecida a Iterator) que se utiliza, por ejemplo, en el método public Enumeration<V> elements() de Hashtable*

Iterable Formato

- Nos permite utilizar el bucle foreach
- Para recorrido de colecciones (e incluso de arrays) es posible utilizar un bucle for con el formato:

for (TipoDatoContenido variable: Coleccion) {

Utilizar la **variable** para operar con ella

}

Coleccion es la variable con la colección que permite el uso de Iterable (no todas las colecciones permiten este uso)

Variable es un nombre de variable

TipoDatoContenido es el tipo de cada elemento de la colección

- En cada iteracción variable toma el valor de uno de los elementos de la colección. Uno distinto o en cada iteracción hasta procesarlos todos.

Iterable

- Nos permite utilizar el bucle foreach

```
Vector <Alumno> va=new Vector <Alumno>(2);

va.add(new Alumno ("Ana",12));
va.add(new Alumno ("Oscar",13));
va.add(new Alumno ("Pepe",13));

 for (Alumno a: va) {
    System.out.println(a);
  }
```

Similar al for de Shell scripts

```
for (i in `ls`)
  echo $i
```

Iterable

- Otro Ejemplo:

```
System.out.println("Listado2  con foreach");
    List <String> palabras = new LinkedList<String>();
    palabras.add("uno1");
    palabras.add("dos2");
    palabras.add("tres3");
    for(String s: palabras) {
       System.out.println( s );
    }
```

Iterable no utilizar remove

No se deben eliminar los elementos de la colección mientras se está recorriendo.

Por ejemplo

```
Vector <Integer> v= new Vector <Integer>();
for (int i=0; i<10; i++){
  v.add(i);
}
for (Integer i:v){
  System.out.println(i);
  if (i%2==0)
    v.remove(i);
}
```

Provocaría un error del tipo `ConcurrentModificationException` por que se modifica indebidamente el puntero que se usa para el recorrido

Iterable con arrays

Se puede utilizar el bucle for each con arrays, incluso de tipos básicos

Ejemplo:

```
System.out.println("Listado5,  for con iterable de int");
int v[]= {11,12,13};
for (int i: v)
    System.out.println(i);
```

Modificación iterable

No se deben modificar las colecciones cuando se está iterando sobre ellas.

```
static Hashtable <String, Persona> ht= new Hashtable
<String, Persona>();
System.out.println("Recorrido con foreach");
    for (Persona p:ht.values()) {
     System.out.println(p.nombre+":"+p.edad);
     if (p.edad>3)
       ht.put("xxx", new Persona ("xxx",6,33));
```

//java.util.ConcurrentModificationException

No hay problema si se modifican los atributos de los objetos

```
static Hashtable <String, Persona> ht= new Hashtable <String, Persona>();
 System.out.println("Recorrido con foreach");
    for (Persona p: ht.values()) {
        System.out.println(p.nombre+":"+p.edad);
        p.edad++;
    }
```

Método forEach de Iterable

Interface Iterable<T>

default void forEach(Consumer<? super T> acciónIndicada)

Ejecuta la acciónIndicada para cada elemento del objeto Iterable . La acciónIndicada suele ser una expression lambda.

Ejemplos:

```
Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).forEach(n -> System.out.print(n + " "));
Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).forEach(System.out::println);
```

Ejemplo de uso de Consumer

```
Consumer<String> consumidor = (x) -> System.out.println(x);
 consumidor.accept("hola");  // Se imprimirá hola en pantalla.
```

Iterator

Interface Iterator<E>

boolean **hasNext**(); //true si quedan elementos

E **next**() // Devuelve el siguiente elemento

void **remove**() //Borra el último elemento devuelto.

remove es opcional.

Es mejor utilizar Iterator que el interface Enumeration que permite un recorrido similar pero no permite eliminación con remove

Iterator

Utilizar un bucle for con el iterador devuelto por la colección:

```
 System.out.println("Listado con for con iterator");
 for (Iterator <String> it = palabras.iterator();  it.hasNext();)
{
        String objeto = it.next();
        System.out.println(objeto);
//También se puede hacer con System.out.println( it.next());
// en lugar de las 2 líneas anteriores
 }
```

Iterator

Hay que tener cuidado porque el método next() nos devuelve el objeto siguiente una vez.

```
List<String> palabras = new LinkedList<String>();
palabras.add("dos2");
palabras.add("tres3");
palabras.add("cuatro4");
palabras.add("cinco5");
for (Iterator it = palabras.iterator(); it.hasNext();) {
    Object objeto = (Object) it.next();  //Mejor con String
    if (!objeto.equals("kjljklkjlkjl")
       System.out.println(it.next());
}
```

Solo mostrará:

tres3

cinco5

Iterator- modificación

No se debe modificar la colección original una vez asignado el iterador.

Es incorrecto hacer lo siguiente:

```
List<String> palabras = new LinkedList<String>();
 palabras.add("dos2");
 palabras.add("tres3");
 Iterator it = palabras.iterator();
 palabras.add("cuatro4");  //No debes modificar la colección aquí
 for (;it.hasNext();) {
    System.out.println(it.next());
//ERROR: java.util.ConcurrentModificationException :
 }
```

Iterator – remove() 1/3

Si se permiten modificaciones de la colección con remove() en el iterador.

remove(): Elimina el último elemento recuperado con next()

Al borrar un elemento del iterador también se elimina de la lista original:

```
List<String> palabras = new LinkedList<String>();
    palabras.add("uno1");
    palabras.add("dos2");
    palabras.add("tres3");
    palabras.add("cuatro4");
    palabras.add("cinco5");
```

Iterator – remove() 2/3

- Al borrar un elemento del iterador también se elimina de la lista original:

```
for (Iterator it = palabras.iterator(); it.hasNext();) {
    String s= (String) it.next();
    if (s.equals("dos2"))
        it.remove();
    System.out.println(s);
}
```

Mostrará:

uno1
dos2
tres3
cuatro4
cinco5

Iterator – remove() 3/3

- Al borrar un elemento del iterador también se elimina de la lista original:

```
ei.listadoTradicional( palabras);
```

Mostrará:

uno1
tres3
cuatro4
cinco5

Donde

```
void listadoTradicional(List<String> palabras){
 System.out.println("Listado tradicional");
 for (int i=0; i<palabras.size(); i++)
 System.out.println(palabras.get(i));
 }
```

Iterator – remove() con Integer

```
public class PruebaBorradoIterator {
static void mostrar(Vector <Integer>v){
  for (Integer i:v)    System.out.println(i);
}
public static void main (String arg[]){
  Vector <Integer> v= new Vector <Integer>();
  for (int i=0;i<10; i++)
    v.add(i);
  int i=0;
  Integer num;
  for (Iterator <Integer> it=v.iterator();it.hasNext();i++){
    num=it.next();
    if (i%2==0)
      it.remove();  //Elimina el último elemento recuperado con next()
}
mostrar(v);
}
}
```

Mostrará:

1
3
5
7
9