APRENDERAPROGRAMAR.COM

EJERCICIO Y EJEMPLO RESUELTO: USO DE LA INTERFACE ITERABLE DEL API DE JAVA Y MÉTODO ITERATOR. RECORRER COLECCIONES DE OBJETOS (CU00915C)

Sección: Cursos
Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2029

Resumen: Entrega nº15 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra y José Luis Cuenca

EJERCICIO Y EJEMPLO RESUELTO CON INTERFACE ITERABLE Y MÉTODO ITERATOR

La interface iterable está incluida en el api de Java, en concreto en el paquete **java.lang**. Es una interfaz de uso habitual al igual que el caso de las interfaces Cloneable y Comparable que ya hemos explicado. Implementar **Iterable** tan sólo obliga a sobreescribir un método que es **iterator().** Este método debe devolver un objeto de tipo Iterator. Vamos a explicar la terminología porque puede parecer confusa.

Lo primero es tener claro que hay que distinguir el método **iterator** (en minúsculas) y el tipo definido en el api de Java **Iterator** (con mayúsculas). Iterator con mayúsculas es un tipo definido por la Interface Iterator, igual que List es un tipo definido por la interface List. Por el contrario, iterator() con minúsculas es un método igual que puede ser toString() o cualquier otro. Esto hay que tenerlo claro desde el principio para no llevar a confusiones.

¿Qué es un objeto de tipo Iterator y cómo se implementa el método iterator()? Eso es lo que vamos a explicar a continuación, con detenimiento y a través de un ejemplo para entenderlo mejor.

Lo primero que vamos a recordar es que una interface es un tipo abstracto: no puede ser instanciado porque carece de constructor. Sin embargo, puede definirse un objeto del tipo definido por la interface si se instancia en una clase que implementa la interface. Esto puede parecer complicado pero con un ejemplo lo veremos claramente:

List <Persona> miListaDePersonas = new List<Persona> (); es erróneo ¿Por qué? Porque List es una interface y carece de constructor. En cambio sí sería una escritura correcta definir como de tipo List una colección que creamos instanciándola con una clase que tiene implementada la interface List como es ArrayList: List <Persona> miListaDePersonas = new ArrayList<Persona> ();

De la misma manera que no podemos usar un constructor de List, tampoco podremos usar un constructor para Iterator porque igualmente se trata de una interface sin constructor.

Pasemos a ver ejemplo de código. Vamos a trabajar como en los ejemplos anteriores con una clase **Persona** y una clase **Programa** que hace uso de esta. Para ver la utilidad de la implementación de la interface Iterable vamos a necesitar una colección o conjunto de personas que queremos recorrer. Por ejemplo, imaginemos que tenemos a 15 personas y queremos saber las edades de cada una de ellas. Para ello recorreremos la colección de personas extrayendo la edad cada vez que visitemos un objeto de la colección. El método iterator() nos va a permitir obtener un objeto de tipo Iterator que representa la colección a recorrer, y los métodos disponibles para los objetos de tipo Iterator nos van a permitir operar con cada elemento de la colección.

Para ello vamos a introducir una nueva clase que se va a llamar **ConjuntoPersonas** que va a ser básicamente muy sencilla y se va a componer de un Array de Personas llamado conjuntoPersonas como único atributo o propiedad. Ten en cuenta que el nombre de la clase es un nombre arbitrario: nos

referimos simplemente a un grupo de personas. En este caso, no hay relación ni con los set de Java ni con los conjuntos matemáticos, se tratará simplemente de un array de objetos Persona. Escribe la clase **Persona** que será conforme a esta definición:

```
/* Ejemplo Clase e Interfaz Iterable aprenderaprogramar.com */
public class Persona {        public int dni, edad;
                              public Persona( int d, int e) { this.dni = d;  this.edad = e;  }
}
```

Podemos ver que la clase Persona no implementa nada, solo tiene sus 2 atributos dni y edad y un constructor. Será por tanto la nueva clase que vamos a crear, **ConjuntoPersonas,** la que deberá de implementar la interfaz Iterable. Esto es lógico porque los recorridos se hacen sobre grupos de objetos (colecciones, conjuntos, arrays…). Vamos a introducir primero su código completo para después explicarlo. Adelantamos que el código puede resultar complicado de entender. Trata de irlo leyendo y escribiendo el código al mismo tiempo que las explicaciones que exponemos después, si no lo haces así te puede resultar un poco confuso.

```
import java.util.Iterator;
/* Ejemplo interface Iterable aprenderaprogramar.com */
public class ConjuntoPersonas implements Iterable<Persona> {
   public Persona[] conjuntoPersonas; // Atributo de la clase
   public ConjuntoPersonas (Persona [] p) { // Constructor de la clase
    conjuntoPersonas = p;   }

   public Iterator<Persona> iterator() {    Iterator it = new MiIteratorPersona();
                                            return it;       }

protected class MiIteratorPersona implements Iterator<Persona>  {
      protected int posicionarray;

               public MiIteratorPersona()   { posicionarray = 0; }

               public boolean hasNext() {
                  boolean result;
                  if (posicionarray < conjuntoPersonas.length) { result = true; }
                  else { result = false; }
                  return result;
               }

               public Persona next() {
                  posicionarray++;
                  return conjuntoPersonas[posicionarray-1];
               }

               public void remove() {
                  throw new UnsupportedOperationException("No soportado.");
               }
   }
}
```

Vamos a analizar punto por punto el código para que quede claro qué es cada cosa.

Vemos que la clase ConjuntoPersonas tiene un atributo llamado conjuntoPersonas y que este atributo es un array de Personas.

La clase tiene también un constructor y el método que nos obliga a implementar la interfaz Iterable que es **public Iterator<Persona> iterator().**

Analicemos la signatura del método: el método, obligatoriamente, por implementar una interface, ha de ser público (de ahí public). El método, obligatoriamente ha de devolver un objeto de tipo Iterator<tipoQueFormaLaColecciónOGrupo>. Si recuerdas por ejemplo la clase List definida por la interface List, los símbolos < y > nos sirven para definir el tipo de elementos que hay dentro de una colección. Por ejemplo List <Taxi> miColeccionDeTaxis = new ArrayList<Taxi> (); nos permitía crear una colección de taxis. En este caso Iterator<Persona> representa a un objeto de tipo Iterator que contiene objetos de tipo Persona.

Ahora bien como vemos debemos devolver un objeto de la clase Iterator. El tipo Iterator es un tipo definidido por una interface (al igual que List) y que no puede ser instanciado directamente, ya que carece de constructor. Dicho de otra manera, la clase Iterator es una clase abstracta.

Para poder devolver un objeto de tipo Iterator (que es algo a lo que al fin y al cabo nos obliga la interface Iterable) necesitamos instanciar un objeto Iterator y esto no podemos hacerlo directamente. Para resolver este problema, recurrimos a definir una clase interna dentro de la clase Persona denominada MiIteratorPersona, que implementará la interface Iterator, y que por tanto nos permitirá devolver una instancia de Iterator para nuestra clase Persona.

El acceso elegido para crear la clase MiIteratorPersona es protected. ¿Por qué? Porque esta clase no tiene interés que sea visible desde otras clases. Unicamente nos interesa que sea visible desde la clase Persona o subclases de la clase Persona.

A nivel de términos es un poco confuso, pero cuando te acostumbres y realices unos cuantos ejercicios te resultará más fácil. Decimos que es confuso porque parece un trabalenguas: para implementar la interface iterator hemos de sobreescribir el método iterator(), y para ello hemos de poder devolver un objeto Iterator, lo cual logramos creando una clase interna que implementa la interface Iterator. Como decimos, una especie de trabalenguas.

La interface Iterator (del paquete java.util) a su vez nos obliga a implementar al menos 3 métodos que son: ***public boolean hasNext(), public Persona next() y public void remove().***

El primero debe devolver un valor boolean indicando si el iterador tiene un siguiente elemento. El método next(), debe devolver el siguiente elemento del iterador, y remove() debe remover o eliminar el anterior objeto devuelto.

Piensa que un iterador viene siendo “un clon” de la colección a recorrer. Es decir, en vez de operar directamente sobre la colección original operamos sobre una copia.

Veamos ahora qué tenemos en la clase interna:

- Un atributo con acceso protegido al que denominamos posicionarray, de tipo entero. Este atributo nos va a servir como índice para recorrer el array de Personas y nos va a facilitar la implementación de los métodos necesarios.

- Un método hasNext() que devuelve un tipo booleano. Dentro del método tenemos una variable local result de tipo booleano. En el método comprobamos si nuestro índice posicionarray ha llegado al final de la colección verificando si su valor ha alcanzado el número máximo de elementos posible (que es el número de elementos de la colección menos uno, de ahí la comparación de posicionarray con el atributo length del array de personas).

- Un método next() que devuelve el siguiente elemento dentro de la colección.

- Un método remove(). Este último no lo hemos implementado por simplicidad del ejemplo y el código ***throw new UnsupportedOperationException("No soportado.");*** que lo implementa simplemente nos permite salir del paso, ya que no podemos dejar el método sin sobreescribir al ser obligatorio. El significado de la sentencia throw lo explicaremos más adelante cuando veamos la "Gestión de Excepciones". De momento, simplemente podemos pensar que si se invoca el método remove() se devuelve un error.

Ahora pasaremos a implementar nuestra clase Programa, que será muy sencilla. Escribe este código, que comentaremos después:

```
/* Ejemplo Clase e Interfaz Iterable aprenderaprogramar.com */
public class Programa {
   public static void main(String arg[]) {
      Persona p1 = new Persona(74999999,35);
      Persona p2 = new Persona(72759474,30);
      Persona p3 = new Persona(74853735,25);
      Persona[] pp = {p1,p2,p3};
      ConjuntoPersonas cp = new ConjuntoPersonas(pp);
      for (Persona p : cp) // Esto es un for extendido o for-each
      { System.out.println("La persona:"+p.dni+" tiene una edad de:"+p.edad); }
    }
}
```

Creamos 3 personas, para después introducirlas en un array a modo de conjunto de personas. Primero creamos un array, objeto pp, y posteriormente creamos el objeto cp de la clase ConjuntoPersonas que es la que implementa la interfaz Iterable. Finalmente recorremos con un bucle for-each todas las personas del conjunto para imprimir por pantalla sus datos.

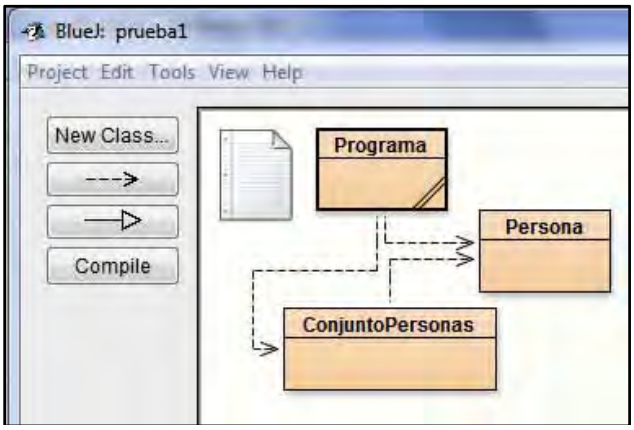
Esto último es lo realmente interesante, ya que gracias a que la clase **ConjuntoPersonas** implementa la interfaz Iterable podemos hacer uso del bucle for-each.

Pero quizás lo más interesante es que podemos crear iteradores para recorrer objetos de tipo ConjuntoPersonas. Escribe el siguiente código:

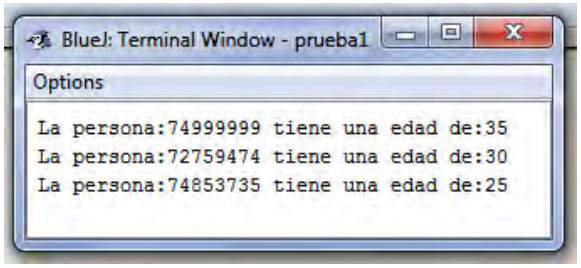
```
import java.util.Iterator;
/* Ejemplo Clase e Interfaz Iterable aprenderaprogramar.com */
public class Programa {
   public static void main(String arg[]) {
      Persona p1 = new Persona(74999999,35); Persona p2 = new Persona(72759474,30);
      Persona p3 = new Persona(74853735,25);  Persona[] pp = {p1,p2,p3};
      ConjuntoPersonas cp = new ConjuntoPersonas(pp);
      Iterator<Persona> it1 = cp.iterator();
      while (it1.hasNext()){
         Persona tmp = it1.next();
         System.out.println("La persona:"+tmp.dni+" tiene una edad de:"+tmp.edad);
      }
   }
}
```

EJECUCIÓN PROGRAMA

Por tanto antes de la ejecución del programa tendremos las siguientes clases definidas en BlueJ:



Y el resultado que obtendremos por pantalla, usemos un for expandido o usemos un objeto Iterator será el siguiente:



VENTAJAS E INCONVENIENTES DE LA INTERFACE ITERABLE

Esta interfaz como puede haberse observado, quizás es un poco más avanzada y compleja que las que hemos visto anteriormente. Pero no por ello deja de ser menos útil y de uso habitual.

Podríamos decir que es quizás una de las interfaces más habitualmente implementadas, ya que en general cuando dispongamos de una clase propia bastante compleja siempre vamos a desear poder hacer recorridos sobre ella, ya sea bien para obtener la información necesaria y poder modificarla, como para poder eliminarla si fuera el caso.

Por tanto su utilidad es altísima, ahora también tiene un punto en contra, y es que implementar esta interfaz como hemos visto tiene cierta complejidad. Aunque el resultado final por ejemplo con el uso del bucle for queda muy elegante, sencillo y efectivo. Y por otro lado, el uso de iteradores permite recorridos seguros y manipulación de los items de una colección de forma segura.

La gran ventaja de trabajar con iteradores es que trabajamos con copias en vez de con las colecciones originales y por otro lado, nos permiten el recorrido de cualquier colección de objetos. Tener en cuenta que no todoas las colecciones de objetos en Java tienen un índice entero asociado a cada objeto, con lo cual no se pueden recorrer basándonos en un índice. En cambio, siempre se puede recorrer una colección usando un iterador.

EJERCICIO

Crea una clase denominada AvesEnZoo con 4 atributos. Uno será de tipo String (tipoDeAve) y los otros tres serán de tipo int (numeroAves, numeroMachos, numeroHembras).

Crea una clase GruposDeAvesZoos que implemente la interface Iterable. Crea varios objetos de tipo AvesEnZoo y añádelos a un objeto GruposDeAvesEnZoos. Utilizando un iterador, muestra los datos de los objetos presentes en el objeto GruposDeAvesZoos. El resultado a conseguir por pantalla deberá ser similar al siguiente:

TIPO	TOTAL	MACHOS	HEMBRAS
Aguilas	35	10	25
Buitres	100	55	45
Halcones	80	25	55

Aunque habría muchas formas de llegar a este resultado, lógicamente lo haremos usando las interfaces y métodos vistos en éste capítulo .
Por ejemplo AvesEnZoo az1 = new AvesEnZoo("Aguilas ",35,10,25) sería un objeto de la clase AvesEnZoo y public AvesEnZoo[] gruposDeAves; sería el atributo de la clase GruposDeAvesZoos en base al cual se realiza la iteración.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00916C

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180