Estructuras de Datos
Grado en Ingeniería Informática en Sistemas de Información - Curso 2012/2013
Enseñanzas de Prácticas y Desarrollo

EPD-2: ITERADORES DE COLECCIONES

Objetivos

- Introducir el concepto de iterador.
- Crear programas que procesen colecciones de elementos mediante el recorrido de las mismas.
- Familiarizarse con las interfaces que implementan en Java los iteradores de colecciones.

Conceptos

1. Concepto de Iterador

Una de las tareas más básicas que un programa realiza sobre las colecciones es recorrerlas con el objetivo de realizar alguna operación con sus elementos. Los iteradores de colecciones están concebidos precisamente para facilitar esta tarea.

Un iterador es un patrón de software, muy utilizado en desarrollo de programas orientado por patrones de diseño, que define una interfaz que contiene los métodos necesarios para acceder de forma secuencial a los elementos de una colección. Los iteradores son objetos que implementan la citada interfaz. Como objetos poseen un estado que permite controlar qué elementos de la colección se han recorrido y cuáles quedan por recorrer.

En la anterior Actividad Práctica y de Desarrollo, cada vez que surgía la necesidad de recorrer los elementos de una colección se proponía convertir ésta a un vector y posteriormente recorrer los elementos del mismo. Esta forma de acceso a los elementos de una colección que se propuso era temporal hasta que se estudiara el concepto de iterador ya que no es en absoluto óptima. De ahora en adelante siempre que necesite recorrer los elementos de una colección debe usar un iterador.

2. La interfaz Iterator

El anterior concepto de iterador es el que se emplea para el acceso a las colecciones en Java. Para este fin, JCF incluye la interfaz *Iterator*. Los objetos que implementan esta interfaz permiten acceder secuencialmente a los elementos de una colección. Los métodos incluidos en la interfaz *Iterator* son los siguientes:

- *public Object next( )*: Este método devuelve una referencia al siguiente elemento de la colección que se está recorriendo. Si el iterador se acaba de crear, el método devolverá una referencia al primer elemento de la colección. Si ya se han recorrido todos los elementos de la colección y se llama a este método se lanzará una excepción de la clase *NoSuchElementException*, hija de la clase *RuntimeException*. Observe que este método devuelve una referencia de la clase *Object*, por lo que deberá hacer un *casting* para disponer de una referencia a la interfaz de los elementos de la colección.

- *public boolean hasNext( )*: El método indica si hay un elemento posterior al último que se obtuvo llamando al método *next*. Devolverá verdadero mientras queden elementos por recorrer en la colección y falso cuando la colección se haya recorrido entera.

- *public void remove( )*: Este método permite eliminar de la colección el último elemento cuya referencia devolvió el método *next*. Este método es la forma más segura para eliminar elementos de una colección mientras ésta es recorrida, ya que asegura que ésta queda en un estado coherente. Recuerde que este método sólo puede ser llamado una vez por cada llamada al método *next*, es decir, no intente borrar un mismo elemento de la colección varias veces. En caso de que se llame al método varias veces para una sola llamada a *next* o que se llame al método antes de la primera llamada a *next*, se lanzará una excepción de la clase *IllegalEstateException*. Este método es opcional para los iteradores, ya que puede no estar disponible en algunas implementaciones, como por ejemplo en las colecciones que no permitan modificar sus objetos, en cuyo caso se lanzará una excepción de la clase *UnsupportedOperationException*. Ambas clases de excepciones son hijas de la clase *RuntimeException*.

Para obtener un iterador, la colección tiene que invocar el método *iterator* de la interfaz *Collection* que devuelve una referencia de tipo *Iterator*. Por ejemplo:

```
Collection c = new ArrayList();
        ...
Iterator it = c.iterator();
```

Una vez se ha obtenido un iterador para una colección de elementos de tipo *IElemento*, se suele aplicar el siguiente patrón de código para recorrer la misma:

```
while(it.hasNext()){
        IElemento ele = (IElemento) it.next();
        ... // Procesamiento del elemento devuelto por it.next()
}
```

Bibliografía Básica

Documentación de la API de Java.
The Collection Interface: http://java.sun.com/docs/books/tutorial/collections/interfaces/collection.html
Ver sección "Traversing collections", subsección "Iterators".

Especificación de la interfaz *Iterator*: http://download.oracle.com/javase/1.4.2/docs/api/java/util/Iterator.html

Experimentos

E1. Analice el siguiente código. Observe cómo se obtiene un iterador para una colección y cómo se recorren los elementos de ésta haciendo uso de él.

```
import java.util.*;

public class Experimento1 {

    public static void main(String[] args){
        Collection c = new ArrayList();
        Iterator it; // Referencia al iterador

        for(int i=1;i<=5;i++) // Rellenamos la coleccion
            c.add(i);

        it = c.iterator(); // Obtenemos un iterador para la colección
        while(it.hasNext()) // Mientras haya más elementos
            System.out.println((Integer)it.next()); // Imprimimos el siguiente elemento

    }
}
```

E2. a) Analice el siguiente código que imprime la suma de los pares de elementos de una colección. Observe la forma en la que se obtienen los elementos de la colección, usando un *casting*. Ejecute el código para ver el resultado.

```
import java.util.*;

public class Experimento2 {

    public static void main(String[] args){
        Collection c = new ArrayList();
        Iterator it; // Referencia al iterador

        for(int i=1;i<=10;i++) // Rellenamos la coleccion
            c.add(i);

        it = c.iterator(); // Obtenemos un iterador para la colección
        while(it.hasNext()){ // Mientras haya más elementos
            Integer i = (Integer) it.next();
            Integer j = (Integer) it.next();

            System.out.println(i.intValue()+j.intValue()); // Imprimimos la suma
        }
    }
}
```

b) Cambie el límite del bucle *for* para que la colección tenga 11 elementos en lugar de 10. ¿De dónde proviene la excepción que se produce? ¿Qué significa?

c) Modifique el código anterior para que funcione correctamente independientemente del número de elementos de la colección, de tal forma que se ignore a los elementos no pareados.

E3. a) Analice el siguiente código y determine el objetivo del mismo. Ejecútelo para ver el resultado.

```
import java.util.*;
public class Experimento3 {

    public static void main(String[] args){
        Collection c =  new ArrayList();
        Iterator it; // Referencia al iterador

        for(int i=1;i<=5;i++) // Rellenamos la coleccion
            c.add(i);

        it = c.iterator(); // Obtenemos un iterador para la colección
        while(it.hasNext()){ // Mientras haya más elementos
            Integer i = (Integer) it.next();

            if(i.intValue()%2==0)
                it.remove();

            if(i.intValue()%3==0)
                it.remove();
        }

        it = c.iterator(); // Obtenemos un iterador para la colección
        while(it.hasNext()) // Mientras haya más elementos
            System.out.println((Integer)it.next()); // Imprimimos el siguiente elemento
    }
}
```

b) Varíe el límite del bucle *for* de tal forma que la colección tenga 10 elementos. ¿Qué ocurre en este caso? ¿Qué significa la excepción que se produce?

c) Realice los cambios necesarios en el código anterior para que funcione correctamente independientemente de los elementos que contenga la colección.

Ejercicios

EJ1. (15 mins) Modifique el código del ejercicio 1 de la EPD 1 de tal forma que los métodos que recorran las colecciones lo hagan empleando iteradores.

EJ2. (15 mins) Modifique el código del ejercicio 2 de la EPD 1 de tal forma que el método que elimina a los alumnos mayores de 30 haga uso exclusivamente de un iterador, tanto para recorrer la colección como para eliminar elementos.

EJ3. (35 mins) a) Cree una clase, y su interfaz asociada, de tal forma que los objetos de la clase representen los resultados de un partido de fútbol. Estos objetos incluirán en forma de atributos el nombre del equipo local, el nombre del equipo visitante, los goles del equipo local y los goles del equipo visitante.

b) Cree una clase, y su interfaz asociada, de tal forma que ésta incluya los datos relativos a una jornada de liga de fútbol. Estos datos serán el número de jornada, que no puede ser negativo (lance una excepción en dicho caso), y el conjunto de resultados de los partidos de la jornada, en forma de colección de objetos de la clase desarrollada en el apartado anterior. Dote a la clase de métodos para añadir y eliminar resultados, así como para imprimir los mismos por pantalla.

c) Cree unas clases, y sus interfaces asociadas, similares a las desarrolladas en los puntos anteriores, pero que representen los resultados de los partidos en forma de quiniela, usando los caracteres '1', 'X' o '2'.

d) Añada un método en la clase desarrollada en el apartado b, de tal forma que se conviertan los resultados de una jornada de liga a resultados en forma de quiniela.

e) Cree un programa que demuestre el funcionamiento de los elementos desarrollados.

Problemas

P1. (20 mins.) Modifique el método desarrollado en el problema 2 de la EPD 1 usando exclusivamente iteradores.

P2. (20 mins.) Adapte el método desarrollado en el problema 3 de la EPD 1 de tal forma que la iteración y borrado sobre las colecciones se realicen usando exclusivamente iteradores.

P3. (20 mins.) Modifique el método desarrollado en el problema 4 de la EPD 1 usando exclusivamente iteradores.

P4. (50 mins.) Modifique la clase desarrollada en el problema 5 de la EPD 1 de tal forma que sus métodos iteren y manipulen los elementos de la colección empleando sólo iteradores.

P5. (45 mins.) Adapte el código de la clase Diccionario desarrollada en el problema 6 de la EPD 1 de tal forma que sus métodos accedan a las entradas del diccionario usando iteradores.

P6. (25 mins.) Cree un programa similar el propuesto en el problema 7 de la EPD 1 que use iteradores para realizar el acceso a las línea del texto.

P7. (30 mins.) Adapte el código del programa desarrollado en el problema 8 de la EPD 1 de tal forma que realice el acceso a los elementos de la colección usando exclusivamente iteradores.

P8. (30 mins.) a) Cree una clase, y su interfaz asociada, que permita representar los datos de una serie de candidatos a piloto de aviones de combate. Estos datos son el nombre, la altura y el peso.

b) Cree un método estático que reciba una colección de candidatos a piloto de combate y elimine de ésta a aquellos que no cumplen las condiciones mínimas pare poder serlo. Éstas son, no superar una altura de 1.75, y no tener un peso superior a 70 kg.

c) Cree un programa que demuestre el funcionamiento del método anterior.

Ampliación de Bibliografía

Thinking in Java, 3rd Edition. Bruce Eckel. Prentice Hall, 2002.
http://www.mindview.net/Books/TIJ/
Capítulo 11. Seccición “Iterators”.

Aprenda Java como si Estuviera en Primero.
http://mat21.etsii.upm.es/ayudainf/aprendainf/Java/Java2.pdf
Capítulo 4, sección 5, subsección 4.3 Páginas 71 a 73