APRENDERAPROGRAMAR.COM

INTERFACES MAP Y SORTEDMAP DEL API JAVA. CLASES HASHMAP Y TREEMAP. EJEMPLO. DIFERENCIAS ENTRE ELLAS. (CU00922C)

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

Fecha revisión: 2029

Resumen: Entrega nº22 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra

INTERFACES MAP Y SORTEDMAP

Vamos a estudiar dos de las interfaces que en nuestra opinión más utilidad y uso tienen entre la comunidad de programadores Java. Quede claro que indicar qué clases o interfaces son más usadas es algo "opinable" ya que no hay estadísticas sobre esto. Se trata de las interfaces Map y SortedMap. Estas interfaces pertenecen al paquete java.util.

MAP

La interface Map, representa un objeto que sirve para ligar ("hacer un mapeo") un valor clave (Key en inglés) y un valor u objeto (value en inglés). Así podemos pensar que un mapa es una especie de diccionario donde a un objeto clave le asignamos un objeto valor. Como ejemplo podríamos pensar en que una clave puede ser un número de pasaporte de una persona y como valor el objeto Persona que contiene toda la información y métodos de una persona en concreto. Aunque la clave lo más frecuente es que sea algo conciso (como un número) también puede ser otras cosas. Por ejemplo el objeto clave podría ser una imagen, al que le correspondiera por ejemplo otra imagen valor. Por ejemplo podríamos tener un mapa de conversión de imágenes, que dada una imagen clave, le correspondiera una imagen valor que fuera la misma imagen escalada a un determinado tamaño. Esto no es lo más usual, pero sería posible.

Por tanto un mapa está abierto a cualquier tipo de objeto. Eso sí, un mapa no puede tener claves duplicadas. La clave funciona como un identificador único. Para entender esto, considera por ejemplo que esta restricción sería igual que decir que dos personas no pueden tener igual número de pasaporte. También hay que tener en cuenta que un map proporciona tres vistas. Podemos ver un mapa como un conjunto de claves (por ejemplo todos los números de pasaportes en el sistema), colecciones de valores (por ejemplo todas las personas), o un conjunto de pares claves-valor mapeados (todos los números de pasaporte vinculados cada uno a su persona correspondiente).

En principio hay que tener especial cuidado con los métodos equals y hashCode ya que estos en general no están bien definidos o no resultan útiles sobre un map genérico o recién creado, por lo que hay que sobreescribir estos métodos para que tengan un funcionamiento acorde a nuestras necesidades.

SORTEDMAP

Esta interfaz es muy similar a la interface Map. Tan solo se diferencia en que SortedMap permite que los elementos dentro del conjunto de la colección estén ordenados totalmente, facilitando por tanto su acceso en búsquedas y haciendo más rápido su consulta.

Así pues los elementos del sortedmap (mapa ordenado) están ordenados por sus elementos claves. En el ejemplo que estamos usando tendríamos que los números de pasaporte podrían estar ordenados de

menor a mayor. Para que la ordenación de las claves sea posible, estos elementos deben pertenecer a una clase que implemente la interface Comparable o tener un Comparator adecuado. Algunas clases como Integer ya tienen un orden natural definido válido. En otras clases será necesario que nosotros definamos un orden.

¿Cuándo usar la interface Map y cuándo usar la interface SortedMap?

Básicamente cuando vayamos a crear nuestra estructura de datos deberemos pensar en si necesitamos o es ventajoso mantener los datos ordenados de alguna manera para hacer más rápido el acceso. En este caso nos decantaremos por el uso de la interface SortedMap y las clases que la implementan.

Si no resulta ventajoso el mantener los datos ordenados o se trata de mapas donde no hay una gran cantidad de datos, la ordenación puede suponer un consumo de recursos innecesario y podemos optar por usar la interface Map y las clases que la implementan.

HASHMAP

HashMap es la clase que vamos a utilizar para implementar la interfaz Map siendo una de las más usadas para implementar esta interface. Esta clase implementa la interface Map basada en una tabla hash en modo similar a como lo hacíamos con HashSet para la interfaz Set.

TREEMAP

TreeMap es la clase que vamos a utilizar como implementación de la interface SortedMap. Al igual que en el caso de TreeSet implementando la interface SortedSet, TreeMap se basa en una implementación en árbol que permite tener un mapa ordenado implementando así por tanto la interfaz TreeMap.

En este caso además TreeMap es la clase más usada para implementar esta interface, al menos en nuestra opinión.

EJEMPLO DE USO DE MAP Y SORTEDMAP

A continuación vamos a crear un ejemplo basado en la clase Persona que venimos utilizando habitualmente en nuestros ejercicios. En nuestro ejemplo crearemos un mapa (map) y después un mapa ordenado (sortedmap).

Vamos a considerar que una Persona será igual a otra Persona si tienen el mismo id en este caso para facilitar el ejemplo. Y para el caso de orden, vamos a considerar que una Persona es mayor que otra si su id es mayor. Este criterio lo establecemos nosotros a nuestra conveniencia.

Primero vamos a crear la clase Persona. Escribe este código en el editor para la clase Persona:

```
/* Ejemplo Interface Map y SortedMap, clase HashMap y TreeMap aprenderaprogramar.com */
public class Persona {
   public int idPersona;      public String nombre;     public int altura;

   public Persona(int idPersona, String nombre, int altura)
       { this.idPersona = idPersona; this.nombre = nombre; this.altura=altura;}

   @Override
   public String toString() { return "Persona-> ID: "+idPersona+" Nombre: "+nombre + " Altura: "+altura;  }
   @Override
   public boolean equals(Object obj) {
      if (obj == null) {   return false;  }
      if (getClass() != obj.getClass()) {  return false;   }
      final Persona other = (Persona) obj;
      if (this.idPersona != other.idPersona) {  return false;    }
      return true;
   }
}
```

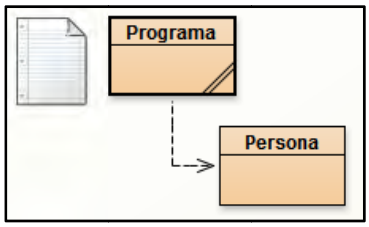
Ahora crearemos un programa que contendrá un mapa al cual añadiremos pares <clave,valor> del tipo <Integer, Persona> en el que usaremos un número como clave para cada Persona.

Escribe el código del Programa en tu editor:

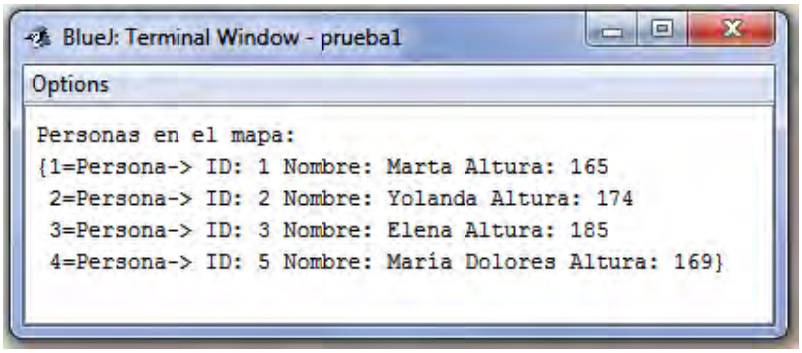
```
/* Ejemplo Interface Map y SortedMap, clase HashMap y TreeMap aprenderaprogramar.com */
import java.util.HashMap;
import java.util.Map;

public class Programa {
   public static void main (String []args)    {

      Map <Integer,Persona> mp = new HashMap<Integer,Persona>();
      Persona p = new Persona(4,"María",167);
      mp.put(4, p); // Añadimos un objeto persona al map
      p = new Persona(1,"Marta",165);
      mp.put(1, p); // Añadimos un objeto persona al map
      p = new Persona(3,"Elena",185);
      mp.put(3, p); // Añadimos un objeto persona al map
      p = new Persona(2,"Yolanda",174);
      mp.put(2, p); // Añadimos un objeto persona al map
      p = new Persona(5,"María Dolores",169);
      mp.put(4, p); // Esto crea una colisión ¡Dos objetos no pueden tener la misma clave!
      System.out.println("Personas en el mapa: \n"+mp.toString().replaceAll(",", "\n"));
   }
}
```

En el código podemos observar como creamos un mapa basado en un HashMap, añadimos a 5 personas y las imprimimos por pantalla. En este caso hemos complicado un poco la instrucción que muestra los resultados en pantalla, para que se mostrara cada elemento del mapa en una línea y por eso hemos decidido sustituir cada "," del mapa por un "\n" o salto de línea. De esta manera cada elemento se muestra en una línea permitiendo que se vea mejor la salida.

El diagrama de clases de BlueJ es el siguiente:



Obtenemos como salida al programa:



Como vemos, a cada valor de clave en el mapa le hemos asignado un objeto Persona.

Ahora bien ¿Por qué sale en distinto orden a como lo hemos introducido? Pues si nos fijamos, hay 4 Personas, es como si una hubiera desaparecido. Pues bien, lo que ha pasado es que para el campo clave ha habido lo que se llama una colisión. Es decir hemos querido insertar en el mapa 2 objetos Persona con el mismo valor clave. En nuestro caso ese valor clave es el Integer "4", donde primero metimos a María con id 4 y altura 167 y posteriormente intodujimos con ese mismo valor clave "4" a la Persona María Dolores con id 5 y altura 169.

Para evitar las colisiones es tan importante no asignar dos claves iguales a distintos objetos. También es importante definir bien los métodos equals y hashCode que utilicemos en los objetos de la clase correspondiente a las claves de los mapas. En nuestro caso al tratarse de claves Integer pues se estarían utilizando los métodos por defecto que trae Integer para equals y hashCode.

Veamos ahora un uso que se puede dar para el caso de SortedMap o mapas ordenados. Vamos a añadir/simular que una Persona tiene una agenda de teléfonos y para ello vamos a tener un SortedMap con la clave como String indicando el tipo de teléfono al que hacemos referencia y como valor un String que representará el número de teléfono a marcar. Cada persona tendrá una agenda y cada agenda será un conjunto de pares <clave, valor>. La agenda será del tipo genérico SortedMap, que implementaremos usando la clase TreeMap.

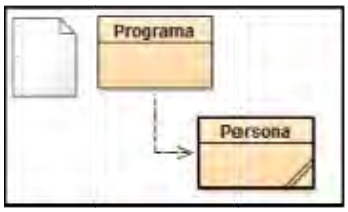
Por tanto en principio nuestra clase Persona quedará como se muestra a continuación. Escribe este código en tu editor:

```
/* Ejemplo Interface Map y SortedMap, clase HashMap y TreeMap aprenderaprogramar.com */
import java.util.SortedMap;
import java.util.TreeMap;

public class Persona {
   public int idPersona; public String nombre; public int altura;public SortedMap<String,String> agendatel;

   public Persona(int idPersona, String nombre, int altura) {
      this.idPersona = idPersona;        this.nombre = nombre;        this.altura = altura;
      this.agendatel = new TreeMap<String,String>();    //inicialmente el mapa está vacío
   }

   @Override
   public String toString() {
      return "Persona-> ID: "+idPersona+" Nombre: "+nombre +" Altura: "+altura+"
\nAgenda:\n"+agendatel.toString().replaceAll(",","\n");
   }

   @Override
   public boolean equals(Object obj) {
      if (obj == null) {   return false;  }
      if (getClass() != obj.getClass()) {  return false;   }
      final Persona other = (Persona) obj;
      if (this.idPersona != other.idPersona) {  return false;   }
      return true;
   }
}
```

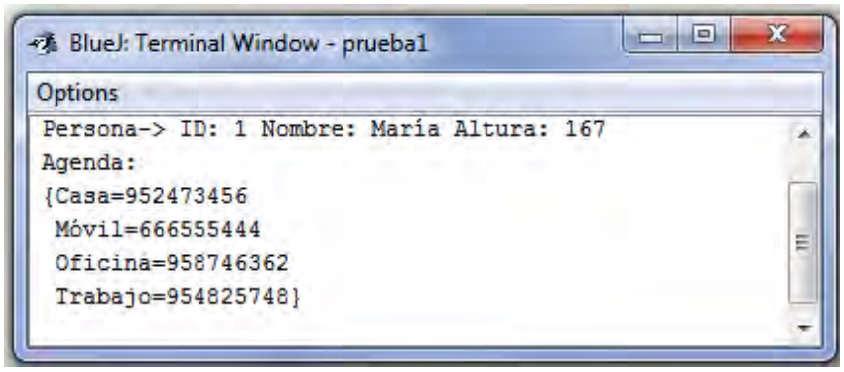
Hemos modificado la clase persona añadiendo un campo agendatel que es un mapa que nos simulará la agenda de teléfonos. La hacemos pública para poder acceder a ella desde nuestra clase Programa. Hemos redefinido el método toString para que se muestre la agenda también. A continuación escribe el código de la clase Programa:

```
/* Ejemplo Interface Map y SortedMap, clase HashMap y TreeMap aprenderaprogramar.com */
public class Programa    {
      public static void main (String []args)      {
         Persona p = new Persona(1,"María",167);
         p.agendatel.put("Trabajo", "954825748");
         p.agendatel.put("Oficina", "958746362");
         p.agendatel.put("Móvil","666555444");
         p.agendatel.put("Casa","952473456");
         System.out.println("Personas en el mapa: \n"+p);
      }
   }
```

El diagrama de clases es al igual que el anterior. Visto en BlueJ es el siguiente:



El resultado de ejecución del programa nos devuelve la siguiente salida:



Como podemos observar en la salida del programa, la persona con id = 1 y nombre María tiene una agenda de teléfonos con 4 teléfonos, pero éstos permanecen ordenados por orden alfabético de las claves (C, M, O, T) a pesar de haberlos introducido desordenados. El que los objetos se mantengan ordenados según el orden de las claves se debe a que hemos utilizado la clase TreeMap, que implementa la interface SortedMap (mapa ordenado). Esta vez el ejemplo es muy sencillo y al utilizar el mapa con tuplas <String,String> no hemos tenido que redefinir los métodos equals y hashcode ya que nos va bien para el ejemplo los de la clase String que estamos utilizando en el mapa como campo clave. El método equals de String implica que los objetos se ordenan por orden alfabético. Si quisiéramos acceder a un objeto de la agenda de teléfonos, por ejemplo al móvil de una persona, bastaría tan solo la instrucción: p.agendatel.get("Móvil"); donde "Móvil" es la clave que nos permite acceder a un objeto.

CONCLUSIONES

Vemos por tanto que las interfaces Map y SortedMap son muy útiles en nuestro día a día y sirven por ejemplo para implementar diccionarios, agendas, etc. Pero hay muchos más situaciones donde se utilizan estas interfaces. Según nuestra experiencia son 2 de las interfaces más usadas por los programadores Java. La implementación basada en TreeMap nos permite tener el mapa ordenado lo que facilita un acceso y búsqueda de datos muy rápido.

Próxima entrega: CU00923C
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