PRÁCTICA 07 – Implementación de Mapas

La práctica corresponde al
Tema 4: Tablas y Conjuntos

Peso Total en la nota: ***6%***

Objetivos

Con esta práctica se pretende que el alumno se familiarice y comprenda:

1. Estudio de la jerarquía básica del *Java Collections Framework*
   a) Se estudian brevemente las clases abstractas del Framework
   b) Se estudia la forma de extender la estructura básica del Framework
2. El coste computacional que conlleva a la utilización de una u otra estructura entre las variantes de las estructuras de datos basadas en Mapas.
   a) Identificar en que ocasiones resulta más favorable el uso de una u otra.

Descripción

En la presente práctica se pretende que el alumno implemente una nueva clase que implemente el interfaz `java.util.Map`, tomando como base de trabajo las herramientas que proporciona el propio *Java Collection Framework*.

En este caso, se pretende obtener una clase que funcione como un multi-mapa. En los mapas standard del paquete `java.util`, sólo se puede asociar un valor a una determinada clave. Cuando se vuelve a insertar un valor diferente sobre la misma clave, el resultado es que el 2º valor sobreescribe al 1º.

En el caso de los multi-mapas, el comportamiento esperado es que ambos valores se puedan almacenar asociados a la misma clave. Cada vez que se solicite el valor asociado a la clave, se devolverá alternativamente uno de los valores que tiene asociado, el valor si solo tiene uno asociado o `null`, si no existe la asociación en el mapa.

La llamada al método para eliminar con una determinada clave, eliminará solo el último valor que se devolvió para esa clave antes de la llamada o `IllegalStateException` si para esa clave se dispone de más de un valor y no se ha pedido obtener ninguno anteriormente.

Ejemplo:

```
m [] //mapa vacío
m[] ← añadimos la asociación (1, 'a')
m[ [1, 'a'] ] ← añadimos la asociación (1, 'b')
m [ [1, 'a'], [1, 'b'] ] → solicitamos el valor de la clave 1 → 'a'
m [ [1, 'a'], [1, 'b'] ] → solicitamos el valor de la clave 1 → 'b'
m [ [1, 'a'], [1, 'b'] ] → solicitamos el valor de la clave 1 → 'a'
m [ [1, 'a'], [1, 'b'] ] → solicitamos el valor de la clave 25 → null
m [ [1, 'a'], [1, 'b'] ] ← solicitamos eliminar el valor de la clave 1
m [ [1, 'b'] ]
```

Para mayor funcionalidad de nuestra clase, se deberán añadir varias operaciones auxiliares: En el caso del borrado, se podrá elegir el borrar el último valor asociado que se ha devuelto o eliminar todos los valores asociados a una determinada clave. Se incluirán también operaciones para almacenar varios valores sobre una clave y para obtener todos los valores asociados a una clave. El tamaño que devolverá el `MultiMapa` al consultarlo será el número de asociaciones almacenadas, que difiere del número de claves almacenadas.

Código Fuente

Se solicita al alumno implementar una nueva clase que permita su utilización como el multi-mapa descrito. La clase tendrá como nombre `MultiMapa` y se incluirá en el paquete `es.ubu.inf.pr07`. Para facilitar la implementación de métodos generales, la clase implementará la clase `AbstractMap`. Se valorará de forma positiva que la clase sobrescriba los métodos predeterminados que no modifiquen su comportamiento, siempre que esto signifique una mejora de la complejidad algorítmica sobre el método ya definido.

En la implementación interna de la clase solicitada, se sugiere emplear como atributo alguna variable de tipo `java.util.Map` que permita servir como apoyo para almacenar la información.
Además de implementar los métodos que impone el interfaz `Map`, se pide que la clase incluya los siguientes métodos adicionales:

- `public void putAllMappings(K clave, Collection<V> valores)`

Permite asociar en una sola llamada varios valores a una misma clave. El resultado deberá ser el mismo que si se realizaran varias llamadas al método put, pasando como parámetro la clave y uno de los valores de la colección.

- `public Collection<V> getAllMappings(Object clave)`

Permite obtener en una sola llamada todos los valores asociados a una determinada clave en el mapa o una colección vacía en caso de que no existiera la clave en el mismo.

- `public Collection<V> removeAllMappings(Object clave)`

Permite eliminar en una sola llamada todas las asociaciones que incluyeran como clave aquella que se pasa como parámetro. Como resultado devuelve una colección incluyendo todos los valores que se encontraban asociados a dicha clave.

Consideraciones: Se espera que las implementaciones facilitadas por el alumno sean lo más eficientes posibles desde el punto de vista de la complejidad algorítmica. Este aspecto se tendrá en cuenta a la hora de la evaluación de la práctica.

Informe

Se pide incluir junto con el código fuente, un breve informe que incluya una breve descripción de la implementación de cada método creado por el alumno, así como una justificación razonada de la complejidad algorítmica que el alumno presupone al método.
No es necesario que el alumno realice pruebas empíricas para comprobar la corrección del razonamiento, sin embargo, se valorará positivamente si se decide hacer y se incluye como solución a la práctica.

En caso de que el alumno **decida realizar** (sin ser obligatorio) una comparación entre diferentes formas de implementar un método para servir como apoyo a la clase solicitada, se podrá incluir en un informe en el que se detallen las condiciones de la comparativa y se expliquen de forma razonada las diferencias que puedan observarse.
La inclusión de esta parte adicional del informe en la entrega solo se tendrá en cuenta para mejorar la nota de la presente práctica. La no inclusión no se contabilizará de forma que penalice la nota del alumno.

Recursos Necesarios

Se facilita junto con este enunciado un fichero de test de jUnit (v.4) que permite comprobar que el código generado responde a unos mínimos de corrección incluidos en la descripción de métodos anterior. Estos métodos de pruebas aparecen en el fichero `MultiMapaTest`.

Para obtener información adicional, se sugiere la consulta de los recursos:

- Map (Java SE 21 & JDK 21)
- AbstractMap (Java SE 21 & JDK 21)

Entrega

- La práctica se realizará en grupos de 1 o 2 personas.
- La entrega ser hará SOLAMENTE por medio de la plataforma UBUVirtual. Cada miembro del grupo deberá subir su propia copia de la solución. No se admitirán soluciones fuera de plazo.
- La entrega incluirá la información de los autores de la misma, tanto en los ficheros de código fuente como en los documentos asociados.
- Cada entrega consistirá en un fichero comprimido (formato .zip), con la estructura de nombre Apellidos1Nombre1_Apellidos2_Nombre2
- Se deberá que incluir comprimido:
  - Código fuente de la solución (dentro de la estructura de paquetes necesaria)
  - Informe de complejidad algorítmica teórica. En formato PDF.
  - Documentación generada con javadoc. En formato html.
  - No hace falta entregar ficheros binarios

Criterios de Evaluación

- **Corrección del funcionamiento**. Se valorará como aspecto más importante que el funcionamiento de la estructura de datos implementada cumpla con todas las condiciones solicitadas, tanto en la documentación oficial como en el enunciado. Se facilitan al alumno un subconjunto de los tests que se emplearán en la corrección, pero se aconseja que el alumno realice de forma adicional los que crea conveniente.
- **Complejidad algorítmica**: Como parte fundamental de los conocimientos de la asignatura, se considerará la complejidad algorítmica de las soluciones proporcionadas el segundo aspecto más importante a valorar en los ejercicios. Penalizará en la nota aquellas soluciones

que, a pesar de proporcionar salidas correctas, lo hagan con una complejidad mucho mayor de la estrictamente necesaria.

- **Informe**: se tendrá en cuenta la corrección del informe de complejidad algorítmica de la solución. No es necesario un informe excesivamente extenso, sino que se centre en la comparación de la complejidad algorítmica solicitada. La extensión aproximada es de 5 páginas.
- **Corrección del código**. Ausencia de *warnings* u otros elementos no deseables como variables no utilizadas, código que no se emplea, catch vacíos, ausencia de marcas sobre los métodos (*@Override*, @...) etc.
- **Documentación**. Las explicaciones y razonamientos que se incluyan a modo de comentarios en el código que entrega el alumno. Se solicita que el alumno entregue adicionalmente la documentación generada en formato javadoc, junto con los ficheros de código fuente.

Se valorarán positivamente (siendo opcionales):

- Que el alumno programe algunos **test adicionales** sobre su ejercicio. En este caso, se pide que se incluyan en una clase adicional de nombre: `MultiMapaTestAdicionales`. Se puede incluir en la misma estructura de paquetes que los tests facilitados. Se sugiere copiar la original para conservar la estructura e *imports* y modificarla para incluir sus *tests*.
- Que el alumno programe test de complejidad algorítmica, que permitan comprobar todas o algunas de las complejidades algorítmicas de cada método o permitan comparar varias opciones para la implementación de la clase solicitada.