SESIÓN 06

La práctica corresponde al:
Tema 4: Conjuntos y Tablas

Peso en la nota final:
0%

Objetivos

El objetivo de esta práctica es que el alumno desarrolle habilidades en la manipulación de colecciones en Java, implementando métodos para crear y manipular mapas (diccionarios) utilizando las colecciones de Java.

Se espera que el alumno implemente la clase estática `Mapas` con los métodos especificados a continuación y que estos métodos pasen las pruebas unitarias proporcionadas.

Conceptos Teóricos

Las colecciones en Java son estructuras de datos que permiten almacenar y manipular grupos de objetos. Los mapas (`Map`), en particular, asocian claves únicas con valores, siendo útiles para almacenar pares de datos relacionados.

En este ejercicio práctico se trabajará el manejo de los mapas por defecto que el lenguaje Java incluye en sus librerías estándar *(Java Collection Framework*).

La inversión de mapas implica intercambiar claves y valores, lo cual es útil en diversas aplicaciones. Un multi-mapa extiende este concepto permitiendo que un valor esté asociado con múltiples claves, representando relaciones de uno a muchos. El alumno probará a implementar este concepto en un programa concreto.

Trabajo del Alumno

1. Programa

Se solicita crear una clase estática de nombre `UtilidadesMapas`. Dicha clase no contendrá datos sino que incluirá 3 métodos para la manipulación de datos con estructuras de datos estádard de Java (de tipo `Map`):

1. **creaMapa**:
    - **Descripción**: Este método debe crear un mapa a partir de dos colecciones, una de claves y otra de valores.
    - **Signatura del Método**:

```
public static <K, V> Map<K, V> creaMapa(Collection<K> keys, Collection<V> values)
```

- **Parámetros**:
  - `keys`: Colección de claves.
  - `values`: Colección de valores.
- **Retorno**: Un mapa que asocia cada clave con su correspondiente valor.
- **Consideraciones**: Las colecciones `keys` y `values` deben tener el mismo tamaño. Si no es así, el método debe lanzar una `IllegalArgumentException`.

2. **mapaInverso**:
   - **Descripción**: Este método debe invertir un mapa, es decir, las claves se convierten en valores y los valores en claves.
   - **Signatura del Método**:

```
public static <K, V> Map<V, K> mapaInverso(Map<K, V> map)
```

- **Parámetros**:
  - `map`: Mapa a invertir.
- **Retorno**: Un mapa invertido.

3. **multiMapaInverso**:
   - **Descripción**: Este método debe crear un multi-mapa inverso, donde un valor puede estar asociado con múltiples claves.
   - **Signatura del Método**:

```
public static <K, V> Map<V, Collection<K>> multiMapaInverso(Map<K, V> map)
```

- **Parámetros**:
  - `map`: Mapa a invertir.
- **Retorno**: Un multi-mapa invertido.

2. Informe

Se pide al alumno que, una vez completada la clase a programar, realice el un informe sencillo en el que se incluya un razonamiento de la complejidad algorítmica de cada uno de los métodos que se han implementado. No es necesario realizar un análisis completo y detallado de cada método (es decir, implementar las pruebas de tiempo de ejecución para el análisis de complejidad), pero sí incluir un razonamiento que relacione el código

fuente que se facilita y a la complejidad que se considera que tiene el mismo.

La extensión del informe será de aproximadamente 5 páginas (incluyendo portadas, índices, gráficas...etc.).

Condiciones de Entrega

- La práctica se realizará en grupos de 1 o 2 personas.
- La entrega se 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 o por otros medios de entrega (e-mail, mensajería interna, …).
- La entrega incluirá la información de quienes son 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_Apellidos2Nombre2`"
- Qué deberá incluir el fichero comprimido:
  - Código fuente de la solución (dentro de la estructura de paquetes necesaria)
  - Documentación en formato Javadoc
  - Informe en formato PDF (en aquellas prácticas que se solicite).
- 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.