PRÁCTICA EN LABORATORIO-2

Kata2

Objetivo

- Dado un array que se rellenará de valores enteros realizar una aplicación de consola para el cálculo del histograma, es decir la frecuencia de cada valor en ese array.

Introducción

Hay determinadas clases que manipulan objetos (ArrayList, HashMap, ...).

Para poder utilizar tipos primitivos con estas clases, Java provee las llamadas **clases envolventes** también llamadas clases contenedoras o ***Wrappers*** (Integer, Double, etc.). Estas clases proporcionan métodos que permiten manipular el tipo de dato primitivo como si fuese un objeto.

Las conversiones entre los tipos primitivos y sus clases envolventes (y viceversa) son automáticas. No es necesario hacer un casting. Para realizarlas Java utiliza el **Autoboxing.**

A lo largo de las versiones de esta Kata2 usaremos un mapa (Array Asociativo) y lo implementaremos como *HashMap*. Inicialmente, tanto la clave como los valores podrán ser objetos de tipo entero (Integer).

Además de los métodos `put()` y `get()` también se tienen estos:

- `containsKey(Object Key)` ➔ comprueba si el mapa contiene la clave `Key`
- `keySet()` ➔ obtiene el conjunto de claves del mapa

ATAJOS EN NETBEANS PARA BUCLES FOR

forc + TAB	for (Iterator *it* = *collection*.iterator();
	it.hasNext();) {
	Object elem = (*Object*) *it*.next();
	}

fore + TAB	for (*Object elem* : iterable) { }
fori + TAB	for (int *i* = 0; i < *arr*.length; *i*++) {}
forl + TAB	for (int i = 0; i < lst.size(); i++) { Object object = lst.get(i); }
forst + TAB	for (StringTokenizer st = new StringTokenizer(""); st.hasMoreTokens();) }
forv + TAB	for (int i = 0; i < vct.size(); i++) { *Object* object = vct.elementAt(i);}

ITERACIÓN SOBRE MAPAS

Hay varias formas de iterar sobre un Mapa en Java. Vamos a repasar los métodos más comunes y revisar sus ventajas y desventajas:

Dado que todos los mapas en Java implementan la interfaz de Mapa, las siguientes técnicas funcionarán para cualquier implementación de un mapa (HashMap, TreeMap, LinkedHashMap, Hashtable, etc.)

Método n°1. Iteración sobre las entradas utilizando un bucle *For-Each*. Este es el método más común y es preferible en la mayoría de los casos. Debería usarse si se necesita en cada iteración tanto las claves como los valores del mapa .

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
    System.out.println("Key = " + entry.getKey() + ", Value = " + entry.getValue());
}
```

Este tipo de bucle lanzará un *NullPointerException* si se intenta iterar sobre un mapa que es nulo, por lo que antes de iterar siempre se debería verificar las referencias nulas.

Método n°2. Iteración sobre claves o valores utilizando un bucle *For-Each*. Si solo se necesita claves o valores del mapa, se puede iterar sobre `keySet` o `values` en lugar de `entrySet`.

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();

//iterar solo sobre claves
for (Integer key : map.keySet()) {
    System.out.println("Key = " + key);
}

//iterar solo sobre valores
for (Integer value : map.values()) {
    System.out.println("Value = " + value);
}
```

Este método proporciona una ligera ventaja de rendimiento sobre la iteración `entrySet` (aproximadamente un 10% más rápido) y es más limpio.

Método nº 3. Iteración usando `Iterator`.

Opción a): usando `Generics`.

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
Iterator<Map.Entry<Integer, Integer>> entries = map.entrySet().iterator();
while (entries.hasNext()) {
    Map.Entry<Integer, Integer> entry = entries.next();
    System.out.println("Key = " + entry.getKey() + ", Value = " + entry.getValue());
}
```

Opción b): sin usar `Generics`.

```
Map map = new HashMap();
Iterator entries = map.entrySet().iterator();
while (entries.hasNext()) {
    Map.Entry entry = (Map.Entry) entries.next();
    Integer key = (Integer)entry.getKey();
    Integer value = (Integer)entry.getValue();
    System.out.println("Key = " + key + ", Value = " + value);
}
```

También se puede usar la misma técnica para iterar sobre `keySet` o `values`.

Este método puede parecer redundante pero tiene sus propias ventajas: 1º) Es la única forma de iterar sobre un mapa en versiones anteriores de Java y 2º) Es el único método que permite eliminar entradas del mapa durante la iteración llamando a `iterator.remove()`. Si se intenta hacer esto durante la iteración For-Each se obtendrá *"resultados impredecibles"* según javadoc.

Desde el punto de vista del rendimiento, este método es igual al *For-Each*.

Método n° 4. iteración sobre claves y búsqueda de valores (ineficaz).

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
for (Integer key : map.keySet()) {
    Integer value = map.get(key);
    System.out.println("Key = " + key + ", Value = " + value);
}
```

Esto podría parecer una alternativa más limpia para el método nº 1 pero en la práctica es bastante lento e ineficaz ya que obtener valores con una clave puede consumir mucho tiempo. Este método debe ser evitado en lo posible.

Conclusión: Si solo necesita claves o valores del mapa, usa el método n° 2. Si está atascado con una versión anterior de Java (menor de Java 5) o planeas eliminar entradas durante la iteración, debe usar el método n° 3. De lo contrario, usa el método n° 1.

Kata2 versión 1

1. Creamos un nuevo proyecto: Kata2

2. En main() creamos un array de enteros que llamaremos *data* con al menos 15 valores que pondremos a nuestro gusto.

3. Instanciamos *histogram* como un mapa con clave y valor enteros e implementado como *HashMap*. Importamos librerías.

4. Recorremos todos los valores del array de enteros *data (fori + TAB).* Si el histograma contiene una clave `data[i]`, guardamos en el mapa para esa clave el valor que tenga + 1:

```
histogram.put(data[i], histogram.get(data[i])+1);
```

Si no contiene esa clave, ponemos para la clave `data[i]` el valor 1:

```
histogram.put(data[i], 1);
```

5. Recorremos todos los elementos del mapa imprimiendo por consola cada clave con su valor correspondiente en el histograma:

```
System.out.println(key + "==>" + histogram.get(key));
```

6. Ejecutamos. La salida será similar a esta (dependiendo de los valores que hayas puesto en `data`):

```
1 ==> 3
2 ==> 2
-4 ==> 1
100 ==> 7
4 ==> 5
5 ==> 6
6 ==> 9
8 ==> 2
```

7. Creamos repositorio local con Git. Añadimos ficheros Hacemos *Commit* (Kata2 versión 1) y subimos a GitHub.

Kata2 versión 2

RECORDATORIO

Operador ternario
La estructura genérica de operador condicional ternario es:

```
resultado = condición ? Valor1 : valor2;
```

Donde a la variable `resultado` recibirá `valor1` en el caso de que la `condición` sea verdadera o bien el `valor2` en el caso de que la `condición` sea falsa.

Por ejemplo, si queremos calcular el mayor de dos números tendremos el siguiente código:

```
mayor = x>y ? x : y;
```

Se usa un operador condicional ternario

1. Modificamos `main()`. Recorremos todos los valores del array de enteros `data` pero reemplazando el primer bucle `for` por un **for mejorado**, *enhanced for*, (fore + TAB):

```
for (int key : data) {
  ...
}
```

Usamos dentro de este bucle `for` la expresión (**op. cond. ternario**):

```
histogram.put(key, histogram.containsKey(key) ? histogram.get(key) + 1 : 1);
```

2. Ejecutamos. La salida debe ser la misma.

3. Hacemos *Commit* (Kata2 versión 2) y subimos a GitHub.

Esta expresión se traduce a lenguaje natural así: "Si el histograma contiene la clave key, rellena el mapa con el valor correspondiente y súmale 1, y si no contiene la clave key, rellena el mapa con el valor 1.

Kata2 versión 3

Se define la clase Histogram

1. En el proyecto Kata2 creamos una clase *Histogram* con un único atributo (privado y final) que llamaremos, *data*, que será un array de enteros.

2. Generamos constructor y getter.

3. Creamos un método público que devuelva un tipo *Map* (clave y valor enteros) sin parámetros y que llamaremos `getHistogram()`. Dentro del método:

- Instanciamos un objeto que llamaremos *histogram* de tipo *Map* (clave y valores enteros) e implementado como *HashMap*
- Copiamos el bucle para recorrer todos los valores del array de enteros *data* que tenemos en *main*.
- Devolvemos *histogram.*

4. Modificamos *main()*:

- Eliminamos la declaración del objeto *histogram y* el primer bucle. Instanciamos un objeto de tipo *Histogram* que llamaremos *histo* al que se la pasa *data*.

- Declaramos *histogr* de tipo *Map* (clave y valores enteros) invocando al método *getHistogram*() sobre *histo*:

```
Map<Integer,Integer> histogr = histo.getHistogram();
```

- Modificamos el bucle *for* que hemos dejado (el 2º en las versiones anteriores) y reemplazamos *histogram* por *histogr*

5. Ejecutamos. La salida debe ser la misma.

6. Añadimos ficheros (pues hemos creado un fichero nuevo con la clase Histogram), hacemos *Commit (*Kata2 versión 3) y subimos a GitHub.

Kata2 versión 4

RECORDATORIO

Clases y tipos genéricos en Java
En Java, cuando definimos una nueva clase, **debemos conocer el tipo de dato** con el que trabajaremos.

Si queremos realizar una operación específica dentro de esta nueva clase, sea cual sea el tipo de datos que va a recibir, podemos hacer uso de los tipos genéricos. Este tipo genérico asumirá el tipo de dato que realmente le pasaremos a la clase.

En Java se especifica que una clase es de tipo genérico poniendo `<T>` justo detrás del nombre de la clase:

```
public class NombreClase<T> {
    ...
}
```

Se introduce el concepto de clase genérica y tipo genérico

1. En el proyecto Kata2 modificamos la clase *Histogram* usando tipos genéricos: `public class Histogram<T> {`

2. Reemplazamos en el atributo *data*, el array *int[ ]* por *T[ ]*

3. Cambiamos *int[ ]* por *T[ ]* en Constructor y getter.

4. En el método *getHistogram*:

- Cambiamos el primer *Integer* por *T* la firma del método y en la instanciación de *histogram* (Independientemente de la clave de tipo *T* nos devolverá siempre un valor *Integer*, pues es un histograma y se obtienen frecuencias, que son valores enteros).
- En el bucle for cambiamos int por *T*.

5. Modificamos main():

- Declaramos *data* no como un array de tipo primitivo enteros (*int*) sino como de objetos *Integer*: `Integer[] data = {1,100,4,etc.};`

6. Ejecutamos. La salida debe ser la misma.

7. Añadimos ficheros, hacemos *Commit* (Kata2 versión 4) y subimos a GitHub.

Kata2 versión 5

Se cambia el tipo de dato en el array para comprobar la utilidad de definir las clases y tipos genéricos.

1. En *main* cambiamos *data* de *array* tipo *int* a array de tipo *String* y le ponemos unos valores, por ejemplo

```
String[] data = {"Rosa", "Pepe", "María", "Pepe", "Pepe", "Rosa"};
```

2. Cambiamos la clave de *histogr* a tipo *String*:

3. En el bucle `for` cambiamos `Integer key` por `String key`

4. Ejecutamos. La salida debe ser:

```
María ==> 1
Rosa ==> 2
Pepe ==> 3
```

5. Hacemos *Commit* (Kata2 versión 5) y subimos a Github.

Referencias:

http://puntocomnoesunlenguaje.blogspot.com.es/2013/02/java-clases-envolventes.html

https://es.stackoverflow.com/questions/54419/diferencia-entre-int-e-integer

https://www.baeldung.com/java-loops

http://www.sergiy.ca/how-to-iterate-over-a-map-in-java/

http://lineadecodigo.com/java/el-operador-ternario-en-java/

https://picodotdev.github.io/blog-bitix/2016/04/tutorial-sobre-los-tipos-genericos-de-java/