Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

PRACTICA 11:Map y HashMap.

1. OBJETIVO

El objetivo de esta práctica es entender la utilidad de los mapas de tipo `HashMap` en java y la forma de tratar con ellos para las operaciones básicas como son la creación, inserción y borrado de objetos, recuperación de valores, iteración, limpieza y obtención del tamaño.

2. MAPAS

Un mapa es una colección de datos de la forma "clave-valor". En una tabla o matriz, el dato almacenado se referencia mediante un índice que indica la posición dentro de la tabla o la matriz. Por ejemplo, para referirnos al elemento 5 de una tabla se indica el índice dentro de la tabla. Como los índices comienzan en 0 el elemento 5 de la tabla `tab` sería `tab[4]`. Se podría decir que la clave en una tabla es el índice del elemento y que el valor es el propio elemento. En los mapas, para indicar un determinado elemento hacemos referencia a su clave, y conociendo la clave, podemos obtener el valor. Por ejemplo, si queremos almacenar una colección de libros de una biblioteca, la clave podría ser el ISBN del libro y el valor un objeto con todos los datos correspondientes al libro. Otro ejemplo podría ser una colección de usuarios, donde la clave puede ser el DNI y el valor un objeto con todos los datos necesarios del usuario.

La interfaz `Map` de java define operaciones que son soportadas por un conjunto de asociaciones clave-valor en el que las claves son únicas. `HashMap` es una implementación de la interfaz `Map`. Hay otras implementaciones de la interfaz Map que no se van a ver como son `TreeMap` y `LinkedHashMap` (la diferencia entre estas clases es la forma en que se guardan los valores en el mapa).

3. OPERACIONES

Las operaciones que se pueden realizar sobre un `HashMap` se presentan a continuación.

Creación de HashMap

Para la creación de un `HashMap` se especifica el tipo de la clave y el tipo del valor. Por ejemplo, en:

```
HashMap<String,Double> productos = new HashMap<String,Double >();
```

se está declarando la variable productos como un `HashMap` (una colección de productos), donde la clave será un objeto de tipo `String` que será el código del producto y el valor un objeto de tipo `Double` que será el precio del producto. Además se está creando el `HashMap`.

O en:

```
HashMap<Dni, Usuario> usuarios = new HashMap<Dni, Usuario>();
```

se está declarando la variable usuarios como un `HashMap` (una colección de usuarios), donde la clave será un objeto de tipo `Dni` y el valor un objeto del tipo `Usuario`. Además se está creando el `HashMap`.

Inserción de elementos en el HashMap

Para la inserción de elementos en un HashMap se utiliza el método put en el que se indica la clave y el valor del elemento a insertar. Por ejemplo:

```
productos.put("DISK",new Double(100.5));
```

añade un nuevo producto al `HashMap` productos con el código `"DISK"` y el precio 100.5.

Recuperación de valores del HashMap

La recuperación del valor de un elemento del `HashMap` se realiza con el método get indicando la clave del valor que se quiere recuperar. Por ejemplo:

```
Double c =  productos.get("DISK");
```

recupera el precio del producto cuyo código es `"DISK"`.

Iteración en el HashMap

Otra forma de recuperar los valores del `HashMap` es iterando sobre el `HashMap` completo. Para este propósito se utiliza un iterador (`Iterator`). Para obtener un iterador para las claves, primero es necesario obtener el conjunto de claves mediante el método `KeySet` del `HashMap`. Por ejemplo, con el siguiente código se muestran todos los productos del `HashMap` productos:

```
String clave;

Iterator<String> iteradorProductos = productos.keySet().iterator();

while(iteradorProductos.hasNext()){

    clave = iteradorProductos.next();

    System.out.println(clave + " - " + productos.get(clave));

}
```

Obtención del tamaño del HashMap

El número de elementos almacenados en el `HashMap` se puede saber mediante el método `size` del `HashMap`. Por ejemplo:

```
System.out.println("Productos en el HashMap: "  + productos.size() );
```

imprime el número de productos que hay almacenados en el `HashMap` productos.

Limpieza del HashMap

La limpieza del `HashMap` se realiza con el método `clear()` del `HashMap`. Por ejemplo:

```
productos.clear();
```

elimina todos los elementos del `HashMap` productos.

Comprobando si el HashMap contiene una clave o valor

Para comprobar si el `HashMap` contiene una determinada clave o un determinado valor, se utilizan los métodos `containsKey` y `containsValue` respectivamente. Por ejemplo:

```
String codigo = "DISPLAY";
if (productos.containsKey(codigo)){
   System.out.println("El precio del producto es:" +
                      productos.get(codigo));
}
else{
   System.out.println("No hay ningún producto con ese código.");
}
```

Indica si el producto cuyo código es "`DISPLAY`" se encuentra en el `HashMap` productos.

Comprobando si el HashMap está vacío

Para comprobar si el `HashMap` está vacío se utiliza el método `isEmpty`. Por ejemplo:

```
boolean vacio = productos.isEmpty();

if(vacio){

        System.out.println("No hay productos”);

} else {

        System.out.println("Hay productos”);

}
```

Indica si hay elementos en el HashMap productos.

Borrando objetos del HashMap

Para borrar objetos del `HashMap` se usa el método `remove` proporcionando la clave. Por ejemplo:

```
if (productos.containsKey(“DISK”)){

    productos.remove(“DISK”);

}

else{

    System.out.println("No hay ningun producto con ese codigo.");

}
```

elimina el elemento del `HashMap` productos cuya clave es `“DISK”`.

4. Ejemplo de HashMap

En el siguiente ejemplo se ha creado una clase `GestorAlmacen` que contiene el método `main` y es la encargada de gestionar el almacén, añadiendo los productos que introduce el usuario, modificando el precio, mostrando todos los productos o borrando un producto. Para la lectura de datos de la entrada estándar se usa un objeto de la clase `Scanner` que permite al usuario introducir la opción elegida y los datos necesarios para cada opción. La clase `Almacen` es la clase que contiene el `HashMap` con los productos del almacén.

A continuación se muestran tanto la clase `GestorAlmacen` como `Almacen`.

```
/*
 *  Fichero: GestorAlmacen.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 */


package fp2.poo.practica11;

import java.util.Scanner;

/**
 * Descripción: Esta es una clase de ejemplo para presentar
 *              los HashMap.
 *
 * @version version 1.0 Abril 2016
 * @author  Fundamentos de Programacion II
 */
public class GestorAlmacen {

    /**
     *  Este metodo es el método principal de la aplicación
     *  Crea un objeto del tipo Almacen y realiza operaciones sobre él
     */
    public static void main( String args[] ) {


      Almacen almacenDeProductos = new Almacen();
      Scanner sc = new Scanner(System.in);
      int opcionElegida = 0;
      float precio;
      String codigo;

      while (opcionElegida != 5){
          System.out.println("Introduce el numero de la opcion que
quieras:");
          System.out.println("1.- Introducir producto");
          System.out.println("2.- Modificar precio");
          System.out.println("3.- Mostrar todos los productos");
          System.out.println("4.- Eliminar producto");
          System.out.println("5.- Salir");
          opcionElegida = sc.nextInt();

          switch (opcionElegida){
              case 1:
                  System.out.println("Introduce el codigo del producto:");
                  codigo = sc.next();
                  System.out.println("Introduce el precio del producto:");
                  precio = sc.nextFloat();
                  almacenDeProductos.guardaProducto(codigo, precio);
                  break;
              case 2:
                  System.out.println("Introduce el codigo del producto del
que quieres cambiar el precio:");
                  codigo = sc.next();
                  if (almacenDeProductos.existeProducto(codigo)){
                     System.out.println("Introduce el precio del
producto:");
                     precio = sc.nextFloat();
```

```
                    almacenDeProductos.modificaPrecio(codigo, precio);
                }
                else{
                    System.out.println("No hay ningun producto con ese
codigo.");
                }

                break;
            case 3:
                almacenDeProductos.muestraProductos();
                break;
            case 4:
                System.out.println("Introduce el codigo del producto que
quieres eliminar:");
                codigo = sc.next();
                almacenDeProductos.eliminaProducto(codigo);
                break;
            case 5:
                break;   // Si la opcion es 5 no se hace nada
            default:
                System.out.println("Tienes que introducir una opcion
valida");
          }

      }

  }

}
```

GestorAlmacen.java

```
/*
 *  Fichero: Almacen.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 */


package fp2.poo.practica11;

import java.util.HashMap;
import java.util.Iterator;

/**
 * Descripcion: Esta es una clase de ejemplo de HashMap.
 *              Un almacén contiene una colección de productos
 *              identificados por un código de producto
 *
 *
 * @version version 1.0 Abril 2016
 * @author  Fundamentos de Programacion II
 */
public class Almacen {

    /** Atributo privado donde se almacenan los productos del almacén. */
    private HashMap<String,Float> productos;

    /**
     *  Constructor de la clase Almacén.
     *
     *  Parametros: No hay parámetros.
     */
    public Almacen() {
```

```
        // Crea el HashMap para la lista de productos
        this.productos = new HashMap<String,Float>();;
    }

     /**
     *   Indica si un producto existe en el almacén
     */
    public boolean existeProducto(String codigo){

       boolean resultado = false;
       if (productos.containsKey(codigo)){
           resultado = true;
       }

       return resultado;
    }

    /**
     *  Guarda un producto en el almacén
     */
    public void guardaProducto(String codigo, float precio){

         if (productos.containsKey(codigo)){
            System.out.println("No se puede introducir el producto. El
codigo esta repetido.");
         }
         else{
            productos.put(codigo, precio);
         }

    }

    /**
     *  Modifica el precio de un producto del almacén
     */
    public void modificaPrecio(String codigo, float precio){

         if (productos.containsKey(codigo)){
             productos.put(codigo, precio);
         }
         else{
            System.out.println("No hay ningun producto con ese codigo.");
         }

    }

    /**
     *  Muestra los productos del almacén
     */
    public void muestraProductos(){

        String clave;
        Iterator<String> iteradorProductos
                             = productos.keySet().iterator();

        System.out.println("Hay los siguientes productos en el almacen:");
        while(iteradorProductos.hasNext()){
            clave = iteradorProductos.next();
            System.out.println(clave + " - " + productos.get(clave));
        }

    }

    /**
     *  Elimina un producto del almacén
     */
    public void eliminaProducto(String codigo){
```

```
        if (productos.containsKey(codigo)){
            productos.remove(codigo);
        }
        else{
            System.out.println("No hay ningun producto con ese codigo.");
        }

    }

}
```

Almacen.java

Ejercicios.

1. Descargue el código proporcionado para esta práctica de la plataforma de enseñanza virtual y compile y ejecute con make para ver el funcionamiento del GestorAlmacen (los números decimales se introducen con coma). El Makefile proporcionado es el siguiente:

```
#
#   Makefile ejemplo almacen
#

# Entorno.

DIRSRC = ./src/
DIRBIN = ./bin/
DIRJAR = ./jar/

# Clases.

RUTACLASES = fp2/poo/practica11/

CLASEALMACEN   = Almacen
CLASEMAIN   = GestorAlmacen


PRINCIPAL = fp2.poo.almacen.GestorAlmacen

ejecuta: $(DIRJAR)$(CLASEMAIN).jar
       java -jar $(DIRJAR)$(CLASEMAIN).jar

$(DIRJAR)$(CLASEMAIN).jar:  $(DIRBIN)$(RUTACLASES)$(CLASEALMACEN).class\
                            $(DIRBIN)$(RUTACLASES)$(CLASEMAIN).class

       jar cvfe $(DIRJAR)$(CLASEMAIN).jar $(PRINCIPAL)            \
              -C  $(DIRBIN) $(RUTACLASES)$(CLASEMAIN).class  \
              -C  $(DIRBIN) $(RUTACLASES)$(CLASEALMACEN).class

# MAIN
$(DIRBIN)$(RUTACLASES)$(CLASEMAIN).class:
$(DIRBIN)$(RUTACLASES)$(CLASEALMACEN).class\

$(DIRSRC)$(RUTACLASES)$(CLASEMAIN).java

       javac -g -Xlint -classpath $(DIRBIN) -encoding ISO-8859-1 -d
$(DIRBIN) -classpath $(DIRBIN) $(DIRSRC)$(RUTACLASES)$(CLASEMAIN).java

# ALMACEN
$(DIRBIN)$(RUTACLASES)$(CLASEALMACEN).class:
$(DIRSRC)$(RUTACLASES)$(CLASEALMACEN).java
       javac -g -Xlint -classpath $(DIRBIN) -encoding ISO-8859-1 -d
$(DIRBIN) -classpath $(DIRBIN) $(DIRSRC)$(RUTACLASES)$(CLASEALMACEN).java
```

Makefile

Incluya en el código las opciones correspondientes para “Mostrar el precio de un producto”, “Mostrar el numero de productos” y “Limpiar el almacen”. Para ello debe crear los métodos en la clase Almacen:

```
void mostrarPrecio(String codigo)

int numProductos() y

void limpiarAlmacen()
```

2. Basándose en el ejemplo anterior cree una clase `GestorEmpresa` en el paquete fp2.poo.empresa que permita dar de alta, baja y modificar los datos de los empleados de la empresa entre otras operaciones. Para ello cree la clase `Empresa` que contenga un `HashMap` con los empleados de la empresa. La clave de los elementos del `HashMap` debe ser un `String` que representa el DNI del empleado y el valor debe ser un objeto del tipo `Empleado` que implementa la interfaz `Persona` ya dada en otra práctica. En el código dado para esta práctica ya se proporciona el fichero `Persona.java`, `Empleado.java`, y el `Makefile` necesario para la compilación y ejecución.

Trabajo a Entregar

Comprima el directorio que contiene el código de los ejercicios realizados y entréguelo en la plataforma de enseñanza virtual.