Objetivos Modelización de un problema mediante clases. Utilización de la composición. Creación de objetos y envío de mensajes. Clases de `java.util`

Índice

---

1. Algunas clases del paquete `java.util`

1.1. La clase `StringTokenizer`

Para extraer información de una cadena de texto normalmente hay que realizar la división del texto en partes discretas o *tokens*. La clase `StringTokenizer` sirve para este propósito ya que se puede utilizar para enumerar los tokens individuales contenidos en la cadena. Para poder utilizar esta clase se debe proporcionar la cadena a procesar y otra cadena formada por los delimitadores. La cadena de delimitadores por defecto consiste en: el espacio, la tabulación, nueva línea y retorno de carro.

La clase `StringTokenizer` ofrece 3 constructores:

```
StringTokenizer(String str)

StringTokenizer(String str, String delim)

StringTokenizer(String str, String delim, boolean delimEsToken)
```

donde `str` es el `String` a separar, `delim` es el `String` que contiene los delimitadores y `delimEsToken` indica si queremos o no que los delimitadores sean tokens (los dos primeros constructores no devuelven los delimitadores como tokens).

Los métodos que vamos a utilizar se describen a continuación (el listado de todos los métodos que se pueden utilizar se encuentra en documentación).

Método	*Descripción*
`public boolean hasMoreElements()`	*devuelve* `true` *si quedan en la cadena uno o más tokens y devuelve false si estamos en el último token.*
`public String nextToken()`	*devuelve el siguiente token como* `String` *y avanza al siguiente token.*

El siguiente código muestra un ejemplo de utilización de esta clase.

```
...
String cadena = "System.out.println(\"El valor es\" + v);";
StringTokenizer st = new StringTokenizer(cadena," .,+()\";");
```

```
while (st.hasMoreTokens())
    System.out.println(st.nextToken());

...
```

Como resultado se obtendría lo siguiente:

```
System
out
println
El
valor
es
v
```

1.2. La clase `Hashtable`

La clase `Hashtable` almacena pares del tipo clave/valor en una tabla de dispersión. Al utilizar `Hashtable` se proporciona un objeto que sirve como clave y otro que sirve de valor que se desea vincular a esta clave. Un objeto de este tipo proporciona un modo rápido de inserción y de búsqueda. Una analogía es un listín telefónico o un diccionario, dado un nombre proporcionar un número de teléfono o dada un palabra proporcionar una definición.

En un vector se obtienen los valores mediante un índice entero. Con una tabla de dispersión, no se utiliza un entero directamente sino que se utiliza lo que se denomina como código de dispersión. Los códigos de dispersión asignan un número a un objeto.

El código de dispersión permite buscar un valor en un sólo lugar. Si no se encuentra nada tras convertir la clave al código de dispersión entonces la clave no se encuentra en la tabla.

Cuando más de una clave da el mismo código de dispersión se dice que ocurre una colisión. Para trabajar con tablas de dispersión es necesario que el algoritmo de dispersión tenga un número pequeño de colisiones. Para algunas clase como `String` y `Date` estos algoritmos ya están implementados.

Hay un parámetro conocido como factor de carga que indica lo llena que puede estar la tabla de dispersión antes de que su capacidad sea aumentada. Cuando el número de elementos en la tabla excede el producto del factor de carga por la capacidad, la capacidad es incrementada automáticamente llamando al método `rehash`.

Los constructores disponibles son

```
// Construye una tabla de dispersión vacía con capacidad inicial 11 y factor de carga 0.75
Hashtable()

// Construye una tabla de dispersión vacía con la capacidad dada y factor de carga 0.75
Hashtable(int initialCapacity)


// Construye una tabla de dispersión con la capacidad dada y el factor de carga dado
Hashtable(int initialCapacity, float loadFactor)

//Construye una tabla de dispersión a partir de un objeto Map
Hashtable(Map t)
```

A continuación se proporcionan algunos de los métodos de esta clase (el listado de todos los métodos que se pueden utilizar se encuentra en documentación).

Método	*Descripción*
`public Object put(Object clave, Object valor)`	*Inserta una clave y un valor en la tabla de dispersión. Devuelve* `null` *si* `clave` *no existe en la tabla o el valor anteriormente asociado con* `clave` *si esta ya existía previamente.*
`public Object get(Object clave)`	*devuelve el objeto asociado con* `clave`*. Si* `clave` *no está en la tabla se devuelve* `null`
`public int size()`	*devuelve en número de pares clave/valor que hay en la tabla.*
`public void rehash()`	*aumenta el tamaño de la tabla de dispersión y redispersiona todas sus claves.*

La siguiente porción de código muestra un ejemplo de utilización de la clase `Hashtable`

```
...
Hashtable ht = new Hashtable(23);

ht.put("uno",new Integer(1));
ht.put("dos",new Integer(2));
ht.put("tres",new Integer(3));
ht.put("cuatro",new Integer(4));


System.out.println( (Integer) ht.get("uno"));
System.out.println( (Integer) ht.get("tres"));
System.out.println( (Integer) ht.get("cinco"));
...
```

Como resultado obtendríamos por pantalla:

```
1
3
null
```

2. Tareas

Se pide modelizar mediante clases un problema. En una red local hay instalados una serie de ordenadores y de impresoras. Cada ordenador e impresora tiene (además de otros atributos) una dirección IP (representada mediante un `String` en este caso) que es única. Los ordenadores y las impresoras se almacenaran en tablas de dispersión (ver 1.2) utilizando como clave su dirección IP.

La descripción de las clases que hay que realizar se indica a continuación:

```
class Ordenador{
   private String ip;
   private String nombre;

   Ordenador(String ip, String nombre){}

   // Devuelve la IP
   public String devuelveIP(){}

   // Devuelve el nombre
   public String devuelveNombre(){}

   // Oculta toString() de Object
```

```
    // mostrando IP y nombre
    public String toString(){}
}
```

```
class Impresora{
    private String ip;
    private String nombre;
    private boolean estado;

    Impresora(String ip, String nombre){}

    // cambia el estado encendida/apagada
    public void establecerEstado(boolean nuevoEstado){}

    // true si la impresora esta encendida false sino
    // si esta encendida debe mostrar el mensaje "Imprimiendo en " + nombre
    public boolean imprimir(){}

    // Devuelve la direccion ip de la impresora
    public String devuelveIP(){}

    // Oculta toString() de Object
    // mostrando IP, nombre y estado
    public String toString(){}
}
```

```
class RedLocal{
    private Hashtable ordenadores;
    private Hashtable impresoras;
    private String nombre;

    RedLocal(String n){}


    public void nuevoOrdenador(Ordenador){}

    public void quitaOrdenador(Ordenador){}

    public void nuevaImpresora(Impresora){}

    // null si no existe
    public Ordenador getOrdenador(String IP){}

    // null si no existe
    public Impresora getImpresora(String IP){}

    // true si la impresora existe y está encendida
    // false si la impresora no existe o está apagada
    public boolean imprimir(String IP){}

    // Devuelve una cadena con informacion sobre
    // todos los ordenadores e impresoras instalados
    // en esta red local
    public String toString(){}

}
```

```
class PruebaRedLocal{

    //Crea un ordenador a partir de la información de la cadena
    //Se debe utilizar la clase StringTokenizer
    private static Ordenador creaOrdenador(String cadena){}
```

```
   //Crea una impresora a partir de la información de la cadena
   //Se debe utilizar la clase StringTokenizer
   private static Impresora creaImpresora(String cadena){}

   public static void main(String[] args){

      //Crea una red local
      RedLocal red = new RedLocal("A");


      //Se obtiene información de ordenadores e impresoras utilizando la clase que se
      //proporciona

      String or;
      Ordenador ord;
      for (int i=1;i<=10;i++){
         or = Utilidad.daOrdenador();
         ord = creaOrdenador(or);
         red.nuevoOrdenador(ord);
      }

      String im;
      Impresora imp;
      for (int i=1;i<=5;i++){
         im = Utilidad.daImpresora();
         imp = creaImpresora(o);
         red.nuevaImpresora(imp);
      }


      //Obtención de una Impresora por su IP
      String IP = "...";
      imp = red.getImpresora(IP);
      imp.establecerEstado(false);

      // Se imprime en una determinada impresora
      red.imprimir(IP);

      // ...


      // Se muestra la red local
      System.out.println(red);
   }
}
```

Se proporciona una clase `Utilidad` que tiene dos métodos públicos y estáticos que se **deben** utilizar para obtener la información necesaria para crear un ordenador o una impresora. El `String` que devuelven estos métodos contiene una dirección IP, una coma y el nombre del ordenador (o de la impresora). Un ejemplo de lo que se puede obtener es:

`127.156.17.53, ant.uv.es`

El siguiente código muestra los métodos públicos de la clase `Utilidad`.

```
public class Utilidad{
   //Da una linea del fichero ordenador.txt cada vez que se le llama
   public static String daOrdenador();

   //Da una linea del fichero impresora.txt cada vez que se le llama
   public static String daImpresora();
}
```