Apellidos:_____

Nombre: _____

DNI: _____ **UO:** _____

Universidad de Oviedo
Escuela de Ingeniería Informática
Estructura de Datos
Grupo L6
14/12/2016 Evaluación continua Hash

Instrucciones

1. **Incluye tus datos personales en todas las hojas, incluidas las de borrador.**
2. **Está terminantemente prohibido acceder a Internet durante el examen.**
3. **Antes de abandonar este recinto deberás entregar el examen completo (incluso si está en blanco).**
4. **Tras el examen se deberán subir 2 ficheros (por separado) a la tarea de entrega del examen:**
   a. **El PROYECTO exportado a fichero zip al Campus Virtual**
      Nombre del fichero exportado: : EX-Hash-Apellido1Apellido2NombreUOxxxx.zip
   b. **El fichero de la JUnit del examen: ExamenHash_Apellido1Apellido2NombreUOxxxx.java**

Considerando una **colisión** como el paso por una posición de una tabla hash cerrada, que no es la definitiva en la operación de que se trate, bien sea porque está ocupada o borrada, según la operación a realizar.
Implementar una JUnit llamada ***ExamenHash_Apellido1Apellido2NombreUOxxxx*** en la que a un objeto ***ClosedHashTable<Integer>*** creado con la siguiente llamada al constructor:

```
new ClosedHashTable<Integer>(17, 0.9, 0.01, LINEAL); // Siendo LINEAL = 0
```

Se le deben añadir (add), borrar (remove) y buscar (find) elementos de forma que se produzca la siguiente serie de **colisiones** en las **operaciones** indicadas, con el **valor de retorno** indicado y en el **orden** indicado:

Colisiones:

1ª	**false** <- remove( ).Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena Colision.Con.Borrada	7ª	**NOT null** <- find( ).Colision.Con.Borrada Colision.Con.Llena Colision.Con.Llena Colision.Con.Borrada
2ª	**true** <- add( ).Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena	8ª	**true** <- add( ).Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena
3ª	**null** <- find( ).Colision.Con.Borrada Colision.Con.Llena Colision.Con.Llena	9ª	**true** <- add( ) Colision.Con.Llena Colision.Con.Llena
4ª	**true** <- add( ).Colision.Con.Llena Colision.Con.Llena	10ª	**false** <- remove( ).Colision.Con.Borrada Colision.Con.Borrada Colision.Con.Borrada Colision.Con.Borrada Colision.Con.Borrada Colision.Con.Llena Colision.Con.Llena Colision.Con.Llena
5ª	**NOT null** <- find( ).Colision.Con.Borrada Colision.Con.Borrada Colision.Con.Llena Colision.Con.Borrada		
6ª	**true** <- remove( ).Colision.Con.Llena Colision.Con.Llena		

Restricciones:

- Antes de la primera, y entre ellas, se pueden realizar **cualquier** número de operaciones pero que**, EN NINGÚN CASO pueden producir COLISIONES**.

Antes de entregar:

- Rellenar los parámetros que producen las colisiones, y que se pasan a los add, find y remove (en la tabla de arriba)
- En el código de la JUnit:
  Tenéis que rellenar el array con los mismos nodos que producen las colisiones solicitadas:
  Integer[] valoresQueHacenColisionar =new Integer[]{// Estos valores son un ejemplo…
  /* colisiones del 1 al 6 */ 30, 20, 10, 40, 90, 50,
  /* colisiones del 7 al 10 */ 60, 70, 80, 100};

IMPRESCINDIBLE:

- **TODOS** los métodos de las clases HashNode y ClosedHashTable deben ser los indicados por el profesor en las clases prácticas; incluido el método **toString()** que se utilizará para verificar el correcto funcionamiento de la práctica.
- Las invocaciones a los métodos que producen las colisiones **DEBEN** estar dentro de un assert: ***assertTrue()***, ***assertFalse()***, ***assertNull()*** o ***assertNotNull()*** según proceda.
- No **PUEDE** haber invocaciones a métodos propios vuestros.

Duración: 45 minutos