Sesión 15: Cadena simplemente enlazada (2/2)

Hoja de problemas

Programación 2

Ángel Herranz
aherranz@fi.upm.es
Universidad Politécnica de Madrid

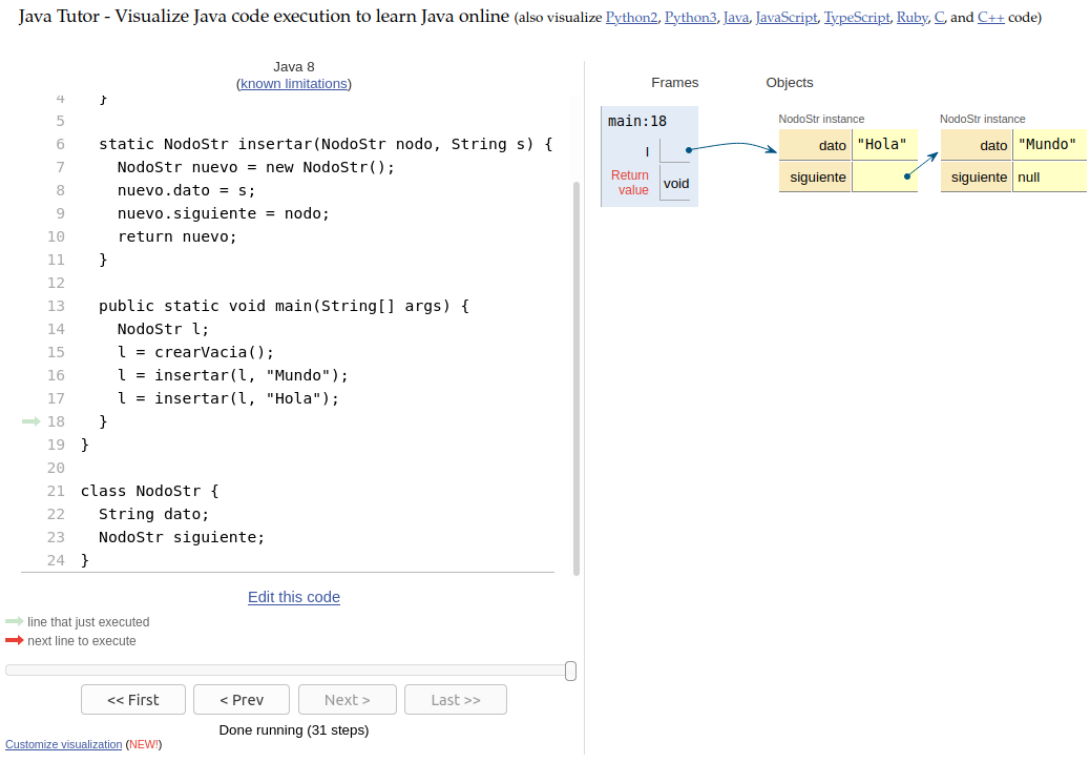
2021-2022

Ejercicio 1. Repasa las transparencias de la sesión 14 sobre cadenas enlazadas. En esta sesión tendrás que implementar todas las *operaciones* sobre la clase `NodoStr` propuestas en dichas transparencias.

Ejercicio 2. El objetivo de estos ejercicios es que entiendas la estructura de datos de las cadenas simplemente enlazadas. Para ello, vas a preparar un único fichero que podrás usar en una herramienta de visualización de ejecución de programas. La herramienta en cuestión es es **JavaVisualizer**[1], una herramienta Web que puedes usar desde

`https://cscircles.cemc.uwaterloo.ca/java_visualize/`

La herramienta te muestra el estado de tu programa paso y podrás ver pantallas como esta, en la que podrás observar cómo se construyen las cadenas y detectar errores en tu código:



[1] Basada en Java Tutor (`http://pythontutor.com/java.html`).

NodoStr **Ejercicio 3.** Antes de empezar vamos a preparar nuestro código tanto para compilarlo localmente como para subirlo a **JavaVisualizer**. Crea un fichero `OperacionesNodo.java` y transcribe el siguiente código:[2]

```
// Clase para representar los nodos de una cadena enlazada
class NodoStr {
  String dato;
  NodoStr siguiente;
}

// Clase con el main y operaciones con cadenas enlazadas
public class OperacionesNodo {
  public static void main(String[] args) {
    NodoStr l = null;
    assert l == null;
    System.out.println("OperacionesNodo.main terminado");
  }
}
```

Compila y ejecuta (no olvides usar el *flag -ea* al ejecutar: `java -ea OperacionesNodo`).

🔔 **Ejercicio 4.** Recuerda que estamos usando la clase `NodoStr` para representar *cadenas enlazadas de strings* y que conceptualmente dicha estructura de datos vienen a representar listas de *strings*. En este documento, normalmente usaremos la palabra *cadena* para referirnos a las variables e instancias del tipo `NodoStr`.

crearVacia **Ejercicio 5.** En `OperacionesNodo`, implementa la función `crearVacia` que devuelva una cadena vacía y añade un `main` para poder probarlo:

```
// Clase para representar los nodos de una cadena enlazada
class NodoStr {
  String dato;
  NodoStr siguiente;
}

// Clase con el main y operaciones con cadenas enlazadas
public class OperacionesNodo {
  static NodoStr crearVacia() {
    return null;
  }

  public static void main(String[] args) {
    NodoStr l;
    l = crearVacia();
    assert l == null;
    System.out.println("OperacionesNodo.main terminado");
  }
}
```

Compila y ejecuta de nuevo.

[2] Puede que te llame la atención ver dos clases en el mismo fichero fuente. No es lo habitual aunque Java lo admite (siempre y cuando sólo una de las clases sea pública), y va a ser de gran ayuda a la hora de usar JavaVisualizer.

esVacia **Ejercicio 6.** Añade (a `OperacionesNodo`) la función `esVacia` siguiendo esta interfaz:

```
static boolean esVacia(NodoStr cadena)
```

Modifica el `assert` del ejercicio anterior por:

```
assert esVacia(l);
```

insertarP **Ejercicio 7.** Implementa la función `insertarP`. La función toma como argumentos una cadena y un *string* y devuelve una cadena cuyo primero es el *string* recibido y el resto es la cadena recibida. La interfaz que debes respetar es:

```
static NodoStr insertarP(NodoStr cadena, String s)
```

insertarU **Ejercicio 8.** Implementa la función `insertarU`. La función toma como argumentos una cadena y un *string* y devuelve *la cadena* añadiendo el *string* al final de la misma como último elemento. El interfaz que debes respetar es:

```
static NodoStr insertarU(NodoStr cadena, String s)
```

longitud **Ejercicio 9.** Implementa la función `longitud`. La función toma como argumento una cadena y devuelve su longitud. La interfaz que debes respetar es:

```
static int longitud(NodoStr cadena)
```

Asserts **Ejercicio 10.** A estas alturas ya estás en disposición de añadir pruebas muy interesantes como por ejemplo esta:

```
l = crearVacia();
for(i = 0; i < 10; i++) {
  l = insertarU(l, Integer.toString(i))
}
n = longitud(l);
assert n == 10;
System.out.format("La longitud de l es %d\n", n);
```

Compila y ejecuta (no olvides `-ea`).

JavaVisualizer **Ejercicio 11.** Abre la URL `https://cscircles.cemc.uwaterloo.ca/java_visualize/`, copia y pega el fichero `OperacionesNodo.java` y *visualiza la ejecución* (botón *Visualize Execution*). Podrás avanzar en la ejecución de tu programa paso a paso y a la derecha de tu código podrás ver una representación del estado de tu programa: variables, instancias, referencias, etc.

Presta mucha atención a los cambios de estado que provoca la ejecución de cada sentencia e intenta detectar errores en tu código.

longitud **Ejercicio 12.** Implementa la función `longitud`. La función toma como argumento una cadena y devuelve su longitud. La interfaz que debes respetar es:

```
static int longitud(NodoStr cadena)
```

imprimir **Ejercicio 13.** Implementa la función `imprimir`. La función toma como argumento una cadena y la *imprime* en la salida estádar (`System.out`). La interfaz que debes respetar es:

```
static void imprimir(NodoStr cadena)
```

La forma en la que deberá imprimirse la cadena deberá ser compatible con la representación *matemática* de listas vista en clase (corchetes y datos separados por comas). Ejemplos:

[]	lista vacía
["Hola"]	lista con un *string*
["Hola", "mundo"]	lista con dos *strings*

Imprimir **Ejercicio 14.** Prueba tu implementación de la función `imprimir`:

```
for(i = 0; i < 10; i++) {
  l = insertarU(l, Integer.toString(i))
}
imprimir(l);
```

En pantalla deberías ver:

```
["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

primero **Ejercicio 15.** Implementa la función `primero`. La función toma como argumento una cadena y devuelve el primer nodo de la cadena. Si la cadena está vacía la función puede fallar. La interfaz que debes respetar es:

```
static String primero(NodoStr cadena)
```

ultimo **Ejercicio 16.** Implementa la función `ultimo`. La función toma como argumento una cadena y devuelve el último nodo de la cadena. Si la cadena está vacía la función puede fallar. La interfaz que debes respetar es:

```
static String ultimo(NodoStr cadena)
```

borrarP **Ejercicio 17.** Implementa la función `borrarP`. La función toma como argumento una cadena y devuelve una cadena eliminando el primer nodo. La interfaz que debes respetar es:

```
static void borrarP(NodoStr cadena)
```

borrarU **Ejercicio 18.** Implementa la función `borrarU`. La función toma como argumento una cadena y devuelve una cadena eliminando el último nodo. La interfaz que debes respetar es:

```
static void borrarU(NodoStr cadena)
```

Más pruebas **Ejercicio 19.** No dejes de añadir más *aserciones* a tu programa principal para probar que tus funciones están correctamente implementadas. De hecho, recuerda que hacer *test driven development* es una muy buena práctica así que... **empieza por los tests**.

Recuerda que las cadenas son una estructura recursiva y que el caso *base* es **`null`**. No dejes de probar las operaciones cuando tienen que trabajar **con listas vacías**.

Ejercicio 20. Cada vez que implementes una función, vuelve al visualizador JavaVisualizer

y asegúrate que entiendes lo que tu código hace. Pide ayuda si no entiendes algo.

obtener

Ejercicio 21. Implementa la función `obtener`. La función toma como argumentos una cadena y un *índice* (un entero entre 0 y la longitud de la cadena menos uno) y devuelve el *string* que ocupa la posición indicada por el índice en la cadena. La interfaz que debes respetar es:

```
static String obtener(NodoStr cadena, int i)
```

cambiar

Ejercicio 22. Implementa la función `cambiar`. La función toma como argumentos una cadena, un índice i y un *string* y cambiar el elemento i-ésimo de la cadena por el *string*. La interfaz que debes respetar es:

```
static void cambiar(NodoStr cadena, int i, String s)
```

insertar

Ejercicio 23. Implementa la función `insertar`. La función toma como argumentos una cadena, un índice i y un *string* y devuelve la cadena tras insertar el *string* en la posición i (dejando la cadena con un elemento más). La interfaz que debes respetar es:

```
static NodoStr insertar(NodoStr cadena, int i, String s)
```

Ejercicio 24. Observa que la interfaz de `cambiar` dice que no devuelve nada (**void**).
¿Podría haber sido `static NodoStr cambiar(NodoStr cadena, int i, String s)`?

Ejercicio 25. Observa que la interfaz de `insertar` dice que devuelve `NodoStr`.
¿Podría haber sido `static void insertar(NodoStr cadena, int i, String s)`?

buscar

Ejercicio 26. Implementa la función `buscar`. La función toma como argumentos una cadena y un *string* y devuelve un booleano: **true** si el string está en la cadena, **false** en otro caso. La interfaz que debes respetar es:

```
static boolean buscar(NodoStr cadena, String s)
```

borrar

Ejercicio 27. Implementa la función `borrar`. La función toma como argumentos una cadena y un índice i y borra el nodo i-ésimo de la cadena. La interfaz que debes respetar es:

```
static NodoStr borrar(NodoStr cadena, int i)
```

Ejercicio 28. Observa que la interfaz de `borrar` dice que devuelve `NodoStr`.
¿Podría haber sido `static void borrar(NodoStr cadena, int i)`?

buscarYborrar

Ejercicio 29. Implementa la función `buscarYborrar`. La función toma como argumentos una cadena y un *string* y devuelve una cadena tras borrar el *string* de la cadena inicial. Si el *string* no está devuelve la misma cadena. La interfaz que debes respetar es:

```
static NodoStr buscarYborrar(NodoStr cadena, String s)
```

insertarO

Ejercicio 30. Implementa la función `insertarO`. La función toma como argumentos una cadena y un *string* y devuelve la cadena tras insertar **en orden** el *string* en la cadena inicial. La interfaz que debes respetar es:

```
static NodoStr insertarO(NodoStr cadena, String s)
```

Más y más pruebas

Ejercicio 31. No dejes de añadir más *aserciones* a tu programa principal para probar que tus funciones están correctamente implementadas.