Práctica 0: **utilización de memoria dinámica**

Programación II

Febrero 2018 (versión 29/01/2018)

DLSI – Universidad de Alicante

Alicia Garrido Alenda

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 12 de febrero a las 9:00 horas y se **cerrará el viernes 16 de febrero a las 23:59 horas**. No se admitirán entregas fuera de plazo.
- Se debe entregar la práctica en un fichero comprimido de la siguiente manera:
  1. Abrir un terminal.
  2. Situarse en el directorio donde se encuentre el fichero fuente (`.java`) de manera que al ejecutar `ls` se muestre el fichero que hemos creado para la práctica y ningún directorio.
  3. Ejecutar:

     ```
     tar cfvz practica0.tgz Metodos.java
     ```

  4. No se pueden entregar más ficheros con extensión `.java` que los que se piden en la práctica.

Normas generales comunes a todas las prácticas

1. La práctica se debe entregar exclusivamente a través del servidor de prácticas del departamento de Lenguajes y Sistemas Informáticos, al que se puede acceder desde la página principal del departamento (`http://www.dlsi.ua.es`, enlace "Entrega de prácticas") o directamente en `http://pracdlsi.dlsi.ua.es`.
   - **Bajo ningún concepto** se admitirán entregas de prácticas por otros medios (correo electrónico, UACloud, etc.).
   - El usuario y contraseña para entregar prácticas es el mismo que se utiliza en UACloud.
   - La práctica se puede entregar varias veces, pero sólo se corregirá la última entrega.
2. El programa debe poder ser compilado sin errores con el compilador de Java existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector durante su implementación para detectar y corregir errores.

3. La práctica debe ser un trabajo original de la persona que entrega; **en caso de detectarse indicios de copia con una o más prácticas (o compartición de código) la calificación de la práctica será** `0` y se enviará un informe al respecto tanto a la dirección del departamento como de la titulación.

4. Los ficheros fuente deben estar adecuadamente documentados, con comentarios donde se considere necesario. Como mínimo se debe realizar una breve descripción (2 líneas máximo) del funcionamiento de cada método (exceptuando los métodos `getter` y `setter`).

5. La primera corrección de la práctica se realizará de forma automática, por lo que es imprescindible respetar estrictamente los formatos de salida que se indican en este enunciado. Además, al principio de todos los ficheros fuente entregados se debe incluir un comentario con el `DNI` de la persona que entrega la práctica, con el siguiente formato:

   // DNI *tuDNI* Nombre

   donde *tuDNI* es el DNI del alumno correspondiente (sin letras), **tal y como aparece en UACloud**, y Nombre es el nombre completo; por ejemplo, el comentario podría ser:

   `// DNI 2737189 CASIS RECOS, ANTONIA`

6. El cálculo de la nota de la práctica y su influencia en la nota final de la asignatura se detallan en las transparencias de la presentación de la asignatura.

7. Para evitar errores de compilación debido a codificación de caracteres que **descuenta** `0.5` **de la nota de la práctica**, se debe seguir una de estas dos opciones:

   a) Utilizar EXCLUSIVAMENTE caracteres ASCII (el alfabeto inglés) en vuestros ficheros, **incluídos los comentarios**. Es decir, no poner acentos, ni ñ, ni ç, etcétera.

   b) Entrar en el menú de Eclipse `Edit > Set Encoding`, aparecerá una ventana en la que hay seleccionar `UTF-8` como codificación para los caracteres.

8. El tiempo estimado por el corrector para ejecutar cada prueba debe ser suficiente para que finalice su ejecución correctamente, en otro caso se invoca un proceso que obliga a la finalización de la ejecución de esa prueba y se considera que dicha prueba no funciona correctamente.

Descripción del problema

El objetivo de esta práctica es repasar conceptos básicos de programación por un lado, y aprender a utilizar tipos de datos que requieren del uso de memoria dinámica por otro. Se pretende de esta forma que se adquiera el hábito de reservar memoria para variables y de comprobar si se dispone de memoria reservada para una determinada variable a la hora de acceder a sus atributos.

Cuando se declara una variable de un tipo que requiere el uso de memoria dinámica, dicha memoria no se reserva al declarar la variable, sino que se tiene que hacer de forma explícita, ya que el valor que toman todas estas variables por defecto es `null` (vacío). Si se accede a algún componente de este tipo de variables sin haber realizado una reserva de memoria previamente se produce un error en la ejecución del programa (`NullPointerException`).

En esta práctica se implementarán dentro del fichero *Metodos.java* los métodos descritos a continuación, donde cada método tendrá un nombre, tipo de valor devuelto y tipo y orden de parámetros que deben ser respetados (no se pueden cambiar).

Se pueden implementar métodos adicionales si se considera necesario.

⊙ `public static int[] fusionCR(int[] a, int[] b)`

Implementa un método al que se le pasan por parámetro dos arrays de enteros ordenados. El método debe devolver un array de enteros ordenado resultado de la fusión de los dos arrays. Si hay elementos repetidos hay que mantenerlos.

Ejemplo: si se le pasan por parámetro los arrays

`a={1, 1, 2, 3, 4, 7} y b={1, 3, 5}`

el método debe devolver el siguiente array `{1, 1, 1, 2, 3, 3, 4, 5, 7}`.

⊙ `public static int[] fusionSR(int[] a, int[] b)`

Implementa un método al que se le pasan por parámetro dos arrays de enteros ordenados. El método debe devolver un array de enteros ordenado resultado de la fusión de los dos arrays sin elementos repetidos y del tamaño del número de elementos sin repeticiones.

Ejemplo: con los parámetros del ejemplo anterior este método devolvería un array de tamaño `6` como este `{1, 2, 3, 4, 5, 7}`.

⊙ `public static boolean capicua(String a)`

Implementa un método al que se le pasa por parámetro un String y devuelve cierto si la cadena pasada por parámetro es capicua, y falso en cualquier otro caso.

⊙ `public static int kesimo(int[] v, int k)`

Implementa un método al que se le pasa por parámetro un array v de enteros positivos y un entero k. El método debe devolver el k-ésimo elemento del array más pequeño. Si alguno de los parámetros no es válido se devuelve `-1`.

Ejemplo: si el vector es

`v={2, 8, 3, 21, 2, 25, 13, 10, 4, 6}`

y el entero es un `3`, debe devolver un `4`.

⊙ `public static int kposicion(int [] v, int k)`

Implementa un método al que se le pasa por parámetro un array v de enteros positivos y un entero k. El método debe devolver la posición que ocupa en el array el k-ésimo elemento del array más pequeño. Si alguno de los parámetros no es válido se devuelve `-1`.

Ejemplo: en el array del ejemplo anterior, con el k anterior, el método devolvería un `8`.

⊙ `public static int[][] matriz(int[] v, int[] w)`

Implementa un método al que se le pasa por parámetro dos arrays y devuelve la matriz (de tamaño mxn) resultante de multiplicar el primer array columna (mx1) con el segundo array fila (de tamaño 1xn), siempre que m y n sean mayores que `0`. En otro caso devuelve `null`.

Ejemplo: con los vectores

```
v={1, 2, 3} y w={2, 1, 4, 3}
```

se obtiene la matriz:

```
2  1  4  3
4  2  8  6
6  3  12 9
```

⊙ `public static String[][] procesa(String s)`

Implementa un método al que se le pasa por parámetro una cadena que debe ser procesada para extraer la información contenida y almacenarla en una matriz de `String` que devolverá.

La cadena pasada por parámetro contiene en primer lugar el número de filas y columnas de la matriz[1] que se devuelve, y la información de las filas separada por `#`, y la de las columnas de cada fila separada por espacios en blanco. Si en la cadena pasada por parámetro faltan cadenas para todas las filas y/o columnas después de ser procesada, estas se completan con la cadena "`libre`", si por el contrario hubiera más cadenas, se ignoran las sobrantes.

Ejemplo: a partir de la cadena

```
4 2#Saludos Hola#Despedidas Taluego#Interludio
```

se devolvería la matriz:

Saludos	Hola
Despedidas	Taluego
Interludio	libre
libre	libre

⊙ `public static int borra(String[] v,String s)`

Implementa un método al que se le pasan por parámetro un array de cadenas y una cadena. El método debe borrar del array todas aquellas cadenas que sean menores lexicográficamente, ignorando diferencias entre mayúsculas y minúsculas, que la cadena que se le pasa como segundo parámetro.

[1] Tanto el número de filas como el de columnas será siempre mayor que `0`.

Para borrar una cadena debe sustituirla por una referencia vacía, es decir, `null`, en el array pasado por parámetro.

El método devuelve el número de cadenas borradas del array.

⊙ `public static int[] elimina(int[] v,int i)`

Implementa un método al que se le pasan por parámetro un array de enteros y un entero. El método tiene que devolver un array de enteros, ordenado de menor a mayor, con todos los enteros del array pasado por parámetro menos el entero pasado como segundo parámetro, que se tiene que eliminar tantas veces como aparezca.

Ejemplo: si se le pasan por parámetro el array

`v={5, 7, 2, 2, 4}`

y un `4`, el método debe devolver un array de tamaño `4` como este {`2, 2, 5, 7`}.

⊙ `public static int organiza(boolean[] v)`

Implementa un método al que se le pasa por parámetro un array de booleanos de manera que modifique dicho array dejando todos los valores `true` juntos en las primeras posiciones del array (empezando en la posición `0`) y todos los valores `false` juntos en las últimas. El método además devuelve cuantas posiciones del array tienen un valor distinto después de la reorganización.

Ejemplo: si se le pasa por parámetro el array

`v={false, false, false, true}`

el método debe devolver un `2` y el array quedaría {`true, false, false, false`}.

Restricciones en la implementación

⊛ Todos los métodos tienen la *`signatura`* concreta indicada en el enunciado que se debe respetar.

⊛ El fichero entregado en esta práctica no debe contener un método `public static void main(String[] args)`.

Métodos de Java

A continuación se comentan una serie de métodos (funciones que se pueden invocar) disponibles en Java que pueden ser útiles/necesarios para la implementación de la práctica. Dichos métodos están asociados al tipo de dato con el que trabajan.

Para trabajar con las cadenas, Java dispone del tipo ***String***, que tiene métodos implementados para realizar operaciones con dicho tipo. Algunos de ellos son:

- `int compareTo(String cadena)`: compara dos cadenas lexicográficamente. Devuelve `0` si son iguales, un número mayor que `0` si el parámetro es menor (precede) a la cadena con la cual se invoca el método y un número negativo en otro caso. Se invoca:

  ```
  String cadena1=new String(‘‘perro’’);
  String cadena2=new String(‘‘mesa’’);
  int n=cadena1.compareTo(cadena2);
  ```

  donde si `n` es mayor que `0` significa que `cadena2` precede lexicográficamente a `cadena1`.

- `int compareToIgnoreCase(String cadena)`: compara dos cadenas lexicográficamente ignorando las diferencias debidas a mayúsculas y minúsculas. Devuelve `0` si son iguales, un número mayor que `0` si el parámetro es menor (precede) a la cadena con la cual se invoca el método y un número negativo en otro caso. Se invoca:

  ```
  String cadena1=new String(‘‘perro’’);
  String cadena2=new String(‘‘mesa’’);
  int n=cadena1.compareToIgnoreCase(cadena2);
  ```

  donde si `n` es mayor que `0` significa que `cadena2` precede lexicográficamente a `cadena1`.

- `boolean equals(Object cadena)`: compara la cadena con la cual se invoca el método con el objeto que se pasa por parámetro, devolviendo cierto si y solo si el parámetro no es `null`, es un `String` y contiene la misma secuencia de caracteres que la cadena con la cual se invocó el método. Se invoca:

  ```
  boolean bool=cadena1.equals(cadena2);
  ```

- `boolean equalsIgnoreCase(Object cadena)`: compara la cadena con la cual se invoca el método con el objeto que se pasa por parámetro, devolviendo cierto si y solo si el parámetro no es `null`, es un `String` y contiene la misma secuencia de carácteres que la cadena con la cual se invocó el método ignorando las diferencias debidas a mayúsculas y minúsculas. Se invoca:

  ```
  boolean bool=cadena1.equalsIgnoreCase(cadena2);
  ```

- `boolean contains(CharSequence s)`: devuelve cierto si el `String` con el cual se invoca contiene la secuencia de caracteres (que puede ser otro `String`) que se pasa por parámetro. Se invoca:

  ```
  String a=new String(‘‘ABC456’’);
  boolean b=a.contains(‘‘6’’); //devuelve cierto
  b=a.contains(‘‘46’’); //devuelve falso
  ```

- `String[] split(String s)`: devuelve un array de `String` resultado de separar el `String` sobre el que se invoca en tantos `String` como veces contenga el `String` pasado por parámetro, es decir, utiliza el `String` pasado por parámetro como separador. Se invoca:

```
String a=new String(‘‘abc:def:ghi:jkl’’);
String[] b=a.split(‘‘:’’); //b es un array de 4 posiciones
// b -> {‘‘abc’’,‘‘def’’,‘‘ghi’’,‘‘jkl’’}
```

- `int length()`: devuelve el número de caracteres de la cadena. Se invoca:

```
int n=cadena1.length();
```

  donde si `cadena1` contiene "`perro`", `n` valdrá `5`.

- `char charAt(int index)`: devuelve el carácter que ocupa la posición indicada en index. El rango de index puede ir desde `0` hasta `length()-1`. Se invoca:

```
char car=cadena1.charAt(0);
```

- `char[] toCharArray()`: devuelve la cadena como un array de caracteres. Se invoca:

```
char[] carr=cadena1.toCharArray();
```

Podéis consultar más información sobre clases y métodos de Java en:

`http://docs.oracle.com/javase/8/docs/api`

Probar la práctica

- En UACloud se publicará un corrector de la práctica con un conjunto mínimo de pruebas (se recomienda realizar pruebas más exhaustivas de la práctica).
- Podéis enviar vuestras pruebas (fichero con extensión `java`, con un método `main` y sin errores de compilación) a los profesores de la asignatura mediante tutorías de UACloud para obtener la salida correcta a esa prueba. En ningún caso se modificará/corregirá el código de las pruebas. Los profesores contestarán a vuestra tutoría adjuntando la salida de vuestro `main`, si no da errores.
- El corrector viene en un archivo comprimido llamado `correctorP0.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar:

  ```
  tar xfvz correctorP0.tgz
  ```

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica0-prueba`: dentro de este directorio están los ficheros con extensión `.java`, programas en Java con un método `main` que realizan una serie de pruebas sobre la práctica, y los ficheros con el mismo nombre pero con extensión `.txt` con la salida correcta para la prueba correspondiente.
- Una vez que tenemos esto, debemos copiar nuestros ficheros fuente (sólo aquellos con extensión `.java`) al mismo directorio donde está el fichero `corrige.sh`.
- Sólo nos queda ejecutar el *shell-script*. Primero debemos darle permisos de ejecución si no los tiene. Para ello ejecutar:

  ```
  chmod +x corrige.sh
  corrige.sh
  ```
- Una vez se ejecuta `corrige.sh` los ficheros que se generarán son:

  - `errores.compilacion`: este fichero sólo se genera si el corrector emite por pantalla el mensaje "`Error de compilacion: 0`" y contiene los errores de compilación resultado de compilar los fuentes de una práctica particular. Para consultar el contenido de este fichero se puede abrir con cualquier editor de textos (gedit, kate, etc.).
  - Fichero con extensión `.tmp.err`: este fichero debe estar vacío por regla general. Sólo contendrá información si el corrector emite el mensaje "`Prueba p01: Error de ejecucion`", por ejemplo para la prueba `p01`, y contendrá los errores de ejecución producidos al ejecutar el fuente `p01` con los ficheros de una práctica particular.
  - Fichero con extensión `.tmp`: fichero de texto con la salida generada por pantalla al ejecutar el fuente correspondiente, por ejemplo `p01.tmp` contendrá la salida generada al ejecutar el fuente `p01` con los ficheros de una práctica particular.

  Si en la prueba no sale el mensaje "`Prueba p01: Ok`", por ejemplo para la prueba `p01`, o algún otro de los comentados anteriormente, significa que hay diferencias en las salidas para esa prueba, por tanto se debe comprobar qué diferencias puede haber entre los ficheros `p01.txt` y `p01.tmp`. Para ello ejecutar en línea de comando, dentro del directorio `practica0-prueba`, la orden: `diff -w p01.txt p01.tmp`