Práctica 0: **utilización de memoria dinámica**

Programación II

28/01/2019

DLSI – Universidad de Alicante

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 11 de febrero a las 9:00 horas y se **cerrará el viernes 15 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

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 previamente una reserva de memoria 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.

1. `public static int[] nomultiplos( int num, int n1, int n2 )`

   Implementa un método que devuelva un array con todos los números naturales menores o iguales que `num`, que no sean múltiplos ni de `n1` ni de `n2`. En caso de que algún dato sea incorrecto (negativo o cero), devolverá `null`.

   Por ejemplo, al invocar a `nomultiplos( 24, 3, 7 )`, el valor devuelto será un array que contiene: {`1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20, 22, 23`}

2. `public static String[] sufijos( String s )`

   Implementa un método al que se le pase por parámetro una cadena no vacía, y devuelva todos los sufijos de la cadena en un array de String (de menor a mayor tamaño).

   Por ejemplo, si la cadena es ‘‘`aquaman`’’ el array devuelto será:

   {‘‘`n`’’, ‘‘`an`’’, ‘‘`man`’’, ‘‘`aman`’’, ‘‘`uaman`’’, ‘‘`quaman`’’, ‘‘`aquaman`’’}

3. `public static String[] prefijos( String s )`

   Implementa un método al que se le pasa por parámetro una cadena no vacía, y devuelva todos los prefijos de la cadena en un array de String (de menor a mayor tamaño).

   Por ejemplo, si la cadena es ‘‘`aquaman`’’ el array devuelto será:

   {‘‘`a`’’, ‘‘`aq`’’, ‘‘`aqu`’’, ‘‘`aqua`’’, ‘‘`aquam`’’, ‘‘`aquama`’’, ‘‘`aquaman`’’}

4. `public static String IMC( double altura, double peso, int edad )`

   El índice de masa corporal es el cociente entre el peso del individuo en kilos y el cuadrado de su estatura en metros. Además se define la condición de riesgo según la tabla:

	edad $<$ 45	edad $\geq$ 45
IMC $<$ 22.0	bajo	medio
IMC $\geq$ 22.0	medio	alto

Implementa un método al que se le pasa como parámetro la altura, peso y edad de una persona (todos valores positivos), y devuelva una cadena con su condición de riesgo: ‘‘`bajo, medio, alto`’’. En caso de que haya algún dato erróneo (algún parámetro menor o igual a cero) el método devuelve la cadena ‘‘`error en entrada`’’.

5. `public static int[] comunes( int[] v1, int[] v2 )`

   Implementa un método al que se le pasa por parámetro dos arrays de enteros sin repetición [1] y devuelva, en un array ordenado de menor a mayor, sólo los elementos comunes en los dos arrays. Inicialmente los arrays `v1` y `v2` no tienen por qué estar ordenados. Si no hay ninguna salida se devuelve `null`.

   Por ejemplo si `v1 = {5, 1, 3, 4}` y `v2 = {7, 8, 3, 5}`, la salida será `{3, 5}`

6. `public static void secCollatz( int i )`

   La *secuencia Collatz* de un número entero se construye de la siguiente forma:

   - si el número es par, se divide por dos;
   - si es impar, se le multiplica por tres y se le suma uno;
   - la sucesión termina al llegar a uno.

   Implementa un método que muestre por pantalla tantas líneas como enteros hay entre `1` e `i`. Cada línea debe contener el número de línea y, separado por un espacio en blanco, el número de elementos de su *secuencia Collatz*.

   Ejemplo:

   la entrada `i = 5` mostrará:

   ```
   1 1
   2 2
   3 8
   4 3
   5 6
   ```

7. `public static int[] Collatz( int i )`

   Implementa un método al que se le pase por parámetro un entero positivo `i` y devuelva en un array los elementos de su secuencia Collatz desde `i` hasta `1` [2].

   Ejemplos:

[1] No es necesario comprobar que haya repeticiones.

[2] La conjetura de Collatz afirma que, partiendo desde cualquier número, la secuencia siempre llegará a 1. A pesar de ser una afirmación a simple vista muy simple, no se ha podido demostrar si es cierta o no.

para la entrada `i = 18` se devolverá el array
{18, 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1}.
para la entrada `i = 20` se devolverá el array
{20, 10, 5, 16, 8, 4, 2, 1}.

8. `public static void caballo( int cor1, int cor2 )`

   El caballo es una pieza del ajedrez que se desplaza en forma de L. Su movimiento consiste en avanzar dos casillas en una dirección y luego una en una dirección perpendicular a la primera. Implementa un método que reciba como entrada las coordenadas en las que se encuentra un caballo en un tablero de 8×8, y muestre por la salida estándar la llamada del método con sus parámetros y todas las casillas hacia las cuales el caballo puede desplazarse, teniendo en cuenta que todas las coordenadas mostradas deben estar dentro del tablero [3]. Si alguna de las coordenadas es inválida el programa mostrará el mensaje ``posicion incorrecta''. Algunos ejemplos de salida:

```
caballo(2, 8)
1 6
3 6
4 7

caballo(3, 4)
1 3
1 5
2 2
2 6
4 2
4 6
5 3
5 5

caballo(1, 9)
posicion incorrecta
```

9. `public static int[] multrusa( int ando, int ador )`

   El método de multiplicación rusa consiste en multiplicar sucesivamente por 2 el multiplicando y dividir por 2 el multiplicador hasta que el multiplicador tome el valor 1. Luego, se suman todos los multiplicandos correspondientes a los multiplicadores impares. Dicha suma es el resultado final de la multiplicación.

   Implementa un método que realiza la multiplicación rusa, devolviendo en un array de enteros los multiplicandos que se suman para obtener el resultado final (no hay que devolver el resul-

[3] Las coordenadas se mostrarán en orden creciente de la primera coordenada, y ante valores iguales, en orden creciente de la segunda.

tado de la multiplicación). En caso de que alguno de los dos parámetros sea igual o menor a cero se devolverá `null`.

La siguiente tabla muestra el cálculo realizado para multiplicar 37 por 12, cuyo resultado final es 12 + 48 + 384 = 444.

Multiplicador	Multiplicando	Multiplicador impar	Suma
37	12	sí	12
18	24	no	
9	48	sí	60
4	96	no	
2	192	no	
1	384	sí	444

El array que se devuelve al invocar a `multrusa( 37, 12 )` tendrá como salida: {`12, 48, 384`}

10. `public static String[][] histograma(String s, int i)`

    Implementa un método al que se le pasa por parámetro una cadena no vacía que debe ser procesada para extraer la información contenida y almacenarla en una matriz de String que devolverá. La matriz tendrá dos columnas y un número de filas que coincidirá con el número de palabras diferentes de la cadena de entrada con una frecuencia igual o mayor al entero pasado por parámetro. Las palabras que contiene la cadena de entrada están separadas entre ellas únicamente por un espacio en blanco. La matriz de String devuelta debe presentar, en el mismo orden que aparecieron, sólo las diferentes palabras almacenadas (una por fila) y su frecuencia de aparición, siempre que esta frecuencia sea mayor o igual que el entero i. Si no existe ninguna palabra que deba ser devuelta el método devuelve `null`.

    Por ejemplo, si la cadena de entrada es `‘‘flores en el campo flores en el prado donde las amapolas abundan entre todas las flores’’` y el entero es un `2`, el método devolverá:

    ```
    { {‘‘flores’’, ‘‘3’’},{‘‘en’’, ‘‘2’’}, {‘‘el’’, ‘‘2’’}, {‘‘las’’, ‘‘2’’}}
    ```

    En el caso de que el entero hubiese sido un `3`, el método devolverá:

    ```
    {‘‘flores’’, ‘‘3’’}
    ```

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);
```

- `String substring( int inicio, int fin )`: devuelve una subcadena del String, desde la posición de `inicio` hasta la posición de `fin`. Si sólo se especifica un parámetro, la subcadena contendrá desde la posición de `inicio` hasta el final de la cadena.

```
String s=cadena1.substring(0, 2);
```

  donde si `cadena1` contiene "`marvel`", el método devolverá "`ma`"

- `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`