Universidad Autónoma de Madrid

Escuela Politécnica Superior

Análisis y Diseño de Software 2017-2018

Práctica 1: Introducción a Java

Inicio: Semana del 5 de febrero.

Duración: 1 semana.

Entrega: Semana del 12 de febrero, el día anterior al inicio de la siguiente práctica según grupo, menos los grupos de los lunes que entregarán antes del lunes a las 8:45 de la mañana.

Peso de la práctica: 5%

El objetivo de esta práctica es aprender el funcionamiento de algunas de las herramientas de la Java Development Toolkit (JDK), comprender el esquema de funcionamiento de la máquina virtual de Java y escribir tus primeros programas en Java.

Apartado 1: Hola Mundo

Con un editor de texto, teclea el siguiente programa, y guárdalo con el nombre *HolaMundo.java*

```
public class HolaMundo {

   public static void main(String[] args) {
      System.out.println("Hola mundo!");   // muestra el string por stdout
   }
}
```

En la línea de comandos, ejecuta la siguiente sentencia:

```
javac HolaMundo.java
```

para "compilar" la clase *HolaMundo* y generar el fichero *HolaMundo.class* correspondiente. Ejecuta el fichero *.class* mediantela sentencia:

```
java HolaMundo
```

Nota: El nombre del fichero *.java* tiene que ser el mismo que la clase que contiene respetando mayúsculas y minúsculas. Asegúrate que los programas *javac* y *java* estén en el PATH.

Apartado 2: Generación de Documentación.

El programa *javadoc* permite generar documentación HTML de los distintos programas fuente leyendo los comentarios del código. Por ejemplo, modifica el programa anterior incluyendo los siguientes comentarios, añadiendo tu nombre como autor:

```
/**
 * Esta aplicación muestra el mensaje "Hola mundo!" por pantalla
 *
 * @author Estudiante EPS estudiante.eps@uam.es
 *
*/
```

```
public class HolaMundo {

 /**
  * Punto de entrada a la aplicación.
  *
  * Este método imprime el mensaje "Hola mundo!"
  *
  * @param args Los argumentos de la línea de comando
  */
   public static void main(String[] args) {
      System.out.println("Hola mundo!");   // muestra el string por stdout
   }
}
```

Genera la documentación del programa mediante la sentencia:

```
javadoc –author HolaMundo.java
```

Nota: Es conveniente almacenar los ficheros de documentación en un directorio separado. Por ejemplo, *javadoc –d doc HolaMundo.java* los almacena en el directorio *doc*, creando el directorio si no existe.

Abre la página *index.html* para ver la documentación generada. Tienes más información sobre el formato adecuado para comentarios *JavaDoc* en: http://en.wikipedia.org/wiki/Javadoc

---

Apartado 3: Uso de librerías básicas

El siguiente programa muestra cómo se reciben los parámetros por la línea de comandos y como realizar cálculos matemáticos.

```
/**
 * Esta aplicación calcula el número de combinaciones sin repetición de n elementos
 * tomados de k en k.
 * <p>La implementación es recursiva, basada en c(n, k) = c(n-1, k-1) + c(n-1, k)
 * Los casos base son c(n, 0) = 1 = c(n, n)  y c(n, k) = 0 para todo k mayor que n</p>
 * <p><b>Nota</b>: Esta implementación no es muy eficiente, al hacer muchos cálculos
redundantes.
 * Se aconseja usar valores pequeños de n y k, entre 0 y 30</p>
 *
 * @author Estudiante EPS estudiante.eps@uam.es
 *
 */
public class Combinatoria {

  /*
   * Si la clase tuviera atributos, los declararíamos aquí, como
   * private Tipo1 atributo1;
   * private Tipo2 atributo2;
   * ...
   * También se pueden inicializar al declararlos, por ejemplo
   * private int contador= 0;
   * El valor inicial también se puede asignar en el constructor
   */

  /**
   * Ejemplo de constructor, en esta clase sería innecesario, ya que no tiene argumentos
   * ni inicializa atributos. El compilador crea uno igual si no existe.
   * Es importante que no devuelva nada (tampoco void), y que se llame como la clase.
   * Si fuese privado impediría crear objetos de este tipo desde otras clases.
   */
  public Combinatoria(/* Argumentos para construir el objeto, si los hubiera */) {
         /* Esta clase no tiene atributos, por lo que este constructor vacío lo crearía
          * automáticamente el compilado, y no es necesario
          * Si tenemos un atributo (atributo1) con el mismo nombre que un argumento, podemos
usar
```

```
          * "this.atributo1" para referirnos al atributo, y "atributo1" para el argumento
          * Por ejemplo podemos asignar el valor inicial con:
          * this.atributo1 = atributo1;
          */
  }

  /**
   * Devuelve el número de combinaciones posibles de n elementos tomados de k en k
   * @param n Número de elementos totales
   * @param k Número de elementos, sin repetición, en cada combinación
   * @return valor del coeficiente binomial (n, k)
   */
  public long combinaciones(int n, int k){
    //Primero comprobamos si los argumentos son válidos
    if (n<0 || k <0) throw new IllegalArgumentException("n y k han de ser positivos");
    //Casos base
    else if (k == 0 || n==k ) return 1; //caso base para 1
    else if (k > n) return 0; //caso base para 0
    //caso general
    else return combinaciones(n-1, k-1)+ combinaciones (n-1, k);
  }


  /**
   * Punto de entrada a la aplicación.
   *
   * <p>Este método imprime el valor del coeficiente binomial de los 2 parámetros de
entrada</p>
   *
   * @param args Los argumentos de la línea de comando. Se esperan dos números enteros, como
cadenas
   */
   public static void main(String[] args) {
      if (args.length!=2) {
         System.out.println("Se espera dos número como parámetro, n y k.");
         System.out.println("  n = Número total de elementos ");
         System.out.println("  k = Elementos en cada combinación");
         System.out.println("Devuelve el coeficiente binomial(n, k)");
      }
      else {
         int n  = Integer.parseInt(args[0]);   // convertimos String a int
         int k  = Integer.parseInt(args[1]);   // convertimos String a int
         Combinatoria c = new Combinatoria();  // Creamos un objeto c de tipo Combinatoria
         System.out.println(c.combinaciones(n, k)); // Imprimimos una línea por salida
estándar
         // En java la destrucción de objetos es automática
      }
   }
}
```

Recuerda grabar el programa en un fichero llamado *Combinatoria.java*.

Ejecuta el programa con distintos parámetros (numéricos o no, negativos, etc.), o sin parámetros. ¿Qué sucede?

Apartado 4: Tu primer programa Java (10 puntos)

Crearemos una nueva clase Tartaglia (public class Tartaglia), que tendrá el siguiente constructor:

```
public Tartaglia (Combinatoria c, int n)
```

Este constructor simplemente guardará los argumentos c y n como atributos privados, para su posterior uso. Puedes usar como plantilla para la definición de la clase el código de Combinatoria.
El atributo c es un objeto de tipo Combinatoria, ya creado, y el atributo n será el número de filas del triángulo de tartaglia.

También añadiremos el método:

```
public String toString()
```

Este método ha de devolver como un String el "triángulo de Tartaglia" correspondiente al número de filas indicado al construir el objeto. Para esta tarea utilizaremos la clase Combinatoria del apartado anterior, sabiendo que para la fila n y columna k (ambos con base 0), tendremos el coeficiente binomial (n, k).
Por ejemplo si ejecutamos:
Combinatoria c= new Combinatoria();
System.out.println(new Tartaglia(c, 6)), se imprimirá lo siguiente:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

La estructura del condicional y un bucle while en *Java* es exactamente la misma que en *C*:

```
if ( <expresión lógica> ) {
   <bloque de instrucciones "cierto">
} else {
  <bloque de instrucciones "falso">
}

while (<expresión lógica>) {
   <cuerpo del bucle>
}
```

Además Java permite una estructura "clásica" de bucle *for* (con gestión explícita de índices) como ésta:

```
for ( <var inicialización>; <condición>; <actualización> ) {
   <cuerpo del bucle>
}
```

La concatenación de cadenas la haremos con el operador "+". Podemos usar el operador "+" con cualquier tipo de dato, y este se convertirá a cadena automáticamente si el operando de la izquierda es una cadena. Esta conversión automática también ocurre cuando ejecutamos System.out.println(objeto). En este caso se llama automáticamente al método toString() del objeto.

Como veremos en próximos temas, la liberación de memoria en Java es automática, así que no necesitas liberar memoria ni destruir los objetos al final del programa.

El código de la clase Tartaglia estará en un archivo llamado Tartaglia.java, en la misma carpeta que el archivo Combinatoria.java. Más adelante estudiaremos el concepto de paquete, y como importar clases de distintos paquetes.

Para probar el código se creará un main en la clase Tartaglia, similar al del apartado 4, donde se lea un único argumento, que indicará el número de filas del triángulo. Este main creará los objetos de tipo Combinatoria y Tartaglia, e imprimirá el objeto de tipo Tartaglia, con el número de filas indicado.

Apartado 5: Ejercicio opcional (1 punto)

Para imprimir el triángulo de tartaglia se están haciendo muchos cálculos redundantes. Mejora la clase Combinatoria para usar una caché, de forma que no se realicen cálculos repetidos.
Para esta caché añadiremos un nuevo atributo a Combinatoria, de la siguiente forma:

```
private Map<Integer, Long> cache = new HashMap<>();
```

Los objetos de tipo Map funcionan como una memoria asociativa, de pares clave-valor, donde las claves no están repetidas. En esta caso las claves serán de tipo Integer (objetos con valor int), y los valores de tipo Long (objetos con valor long).
El atributo cache se ha declarado del tipo abstracto Map, aunque realmente es de tipo HashMap. Esto lo haremos así habitualmente, para ocultar detalles de la implementación y hacer más sencillo el mantenimiento del código.

Para que podamos usar estas clases sin poner delante el nombre del paquete (java.util), hemos de importarlas. Añade la siguiente línea al comienzo del archivo para importar todas las clases del paquete java.util:

```
import java.util.*; //importamos todas las clases del paquete java.util
```

También añadiremos a la clase el siguiente método privado:

```
 private int posicion(int n, int k){ /* ... implementación ... */ }
```

Este método calculará la posición que ocuparía en el triángulo de tartaglia (comenzando en 0), el coeficiente binomial (n, k). Para ello usaremos la siguiente fórmula, para todo n>=0, y k entre 0 y n

```
posicion (n, k) = n*(n+1)/2 + k
```

Esta fórmula simplemente está calculando cuántos elementos hay en las filas anteriores del triángulo, como el sumatorio de 1..n, y le suma k.

Gracias al método posición, podemos guardar en la cache los coeficientes binomiales usando su posición en el triángulo de tartaglia como clave. De esta forma podemos:

- Añadir un valor a la caché: cache.put(posicion(n, k), valor)
- Obtener el valor guardado: cache.get(posicion(n, k))
- Conocer si el valor está o no: cache.containsKey(posicion(n, k))

Modifica el método combinaciones para evitar cálculos repetidos, comprobando si el valor ya estaba calculado y almacenando los nuevos valores calculados.

Comprueba que funciona correctamente ejecutando de nuevo el apartado anterior con la nueva clase Combinatoria. Para evitar errores por overflow del tipo long, no se deben usar valores de n mayores que 66.

Notas:

- En lugar de Long, se podría usar BigInteger, para poder trabajar con números de cualquier tamaño con precisión.
- La cache está declarada como Map<Integer, Long>, en lugar de int y long respectivamente, ya que solo funciona con objetos. Java convierte automáticamente entre int e Integer, y entre long y Long.
- Otra forma de saber si el elemento existe, es comprobando si cache.get devuelve null. En este caso hay que usar el tipo Long para almacenar el resultado, en lugar de long.
- En las siguientes prácticas, en lugar de modificar clases existentes, en esta caso Combinatoria, será preferible crear subclases mediante herencia.

Normas de entrega:

- Se deberán entregar el apartado 4 (y opcionalmente el 5)
- El nombre de los alumnos debe ir en la cabecera *JavaDoc* de todas las clases entregadas
- La entrega la realizará uno de los alumnos de la pareja a través de Moodle
- Se debe entregar un único fichero ZIP / RAR con todo lo solicitado, que deberá llamarse de la siguiente manera: GR<numero_grupo>_<nombre_estudiantes>.zip. Por ejemplo Marisa y Pedro, del grupo 2261, entregarían el fichero: GR2261_MarisaPedro.zip
- La estructura de los ficheros entregados deberá ser la siguiente:
    - **src**. Ficheros fuente
    - **doc**. Documentación *JavaDoc* generada