Examen escrito de Programación 1

Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas

31 de agosto de 2012

- Disponer sobre la mesa en lugar visible un ***documento de identificación*** provisto de fotografía.
- Escribir ***nombre y dos apellidos*** en cada una de las hojas de papel que haya sobre la mesa.
- Comenzar a resolver cada una de las partes del examen ***en una hoja diferente*** para facilitar su corrección por profesores diferentes.
- El tiempo total previsto para realizar el examen es de **dos horas y media**.
- No está permitido consultar libros ni apuntes, excepto los dos documentos facilitados por los profesores de la asignatura: *Un breve resumen del lenguaje Java* y *Documentación de las clases Java utilizadas en la asignatura*.

Problema 1.º (3 puntos)

En un proyecto *software* para el desarrollo de un juego de dominó se hace necesario contar con una clase Java que permita representar fichas de dominó. Cada ficha de dominó está dividida en dos cuadrados y se caracteriza por llevar marcados dos valores, uno en cada cuadrado, que pueden ir desde cero a seis puntos.

Se debe diseñar una clase Java, de nombre `FichaDomino` cuyos objetos permitan representar una ficha del juego del dominó. Para trabajar con objetos de esta clase, se deben diseñar los siguientes métodos públicos con un comportamiento exactamente igual al que se explica a continuación:

- Un método constructor que, cuando se cree un objeto de la clase, permita definir, a través de dos parámetros de tipo **`int`**, los dos valores entre cero y seis que caracterizan la ficha de dominó que se está creando.
- Un método de nombre `valorMenor` que devuelva el valor de tipo **`int`** que corresponde con el menor de los dos valores de la ficha de dominó.
- Un método de nombre `valorMayor` que devuelva el valor de tipo **`int`** que corresponde con el mayor de los dos valores de la ficha de dominó.
- Un método de nombre `esDoble` que devuelva **`true`** si y solo si la ficha es una ficha doble (es decir, sus dos valores son iguales); y que devuelva **`false`** en caso contrario.

- Un método de nombre `puntos` que devuelva un dato de tipo **`int`** que indique el número total de puntos de la ficha (la suma de los dos valores que la componen).
- Un método de nombre `toString` que devuelva una referencia a un objeto de la clase `String` que represente la ficha, conteniendo los dos valores de la ficha separados por un carácter `'|'` y enmarcados también por un par de caracteres `'|'`. Por ejemplo, si esta ficha es el cinco doble, debería devolver una referencia a la cadena `"|5|5|"`. En el caso de la ficha *uno-seis*, es indiferente que devuelva una referencia a `"|1|6|"` o a `"|6|1|"`.
- Un método de nombre `valorExtremoLibre` que tenga como parámetro otro objeto de la clase `FichaDomino`. Suponiendo que la ficha f sobre la que se aplica el método estuviera en uno de los dos extremos de la secuencia de fichas colocadas durante una partida y que la ficha que que se pasa como parámetro (ficha *contigua*) estuviera al lado (es decir, hubiera sido colocada inmediatamente antes), devuelve el valor de la ficha f que queda libre para poder seguir encadenado fichas. Este método tiene que tener la siguiente cabecera:

  ```
  public int valorExtremoLibre(FichaDomino contigua)
  ```

  A modo de ejemplo, si el objeto sobre el que se ejecutara este método representara la ficha *uno-seis*, y si el objeto correspondiente al parámetro `contigua` representara la ficha *dos-uno*, este método debe devolver 6.

Se pide escribir el contenido completo del fichero de código «`FichaDomino.java`», ubicado en el paquete `examen`, cuyos métodos públicos tengan un comportamiento igual al que se ha enunciado. El código debe estar adecuadamente documentado.

Problema 2.º (3 puntos)

Utilizando objetos de la clase `FichaDomino` definida en el problema anterior, se pide completar los siguientes dos métodos:

```
/**
 * Pre:  [ficha] es distinta de null; [partida].length >= 2 y cada componente
 *       de [partida] es una referencia no nula a un objeto de la clase
 *       FichaDomino correctamente puesto (de acuerdo con las reglas del
 *       dominó) en una partida: Los elementos referenciados desde [partida]
 *       están ordenados de idéntica forma a como estarían colocadas las
 *       fichas de dominó ya jugadas en una partida real: Las fichas de
 *       índices menor y mayor corresponden con los extremos de la secuencia
 *       de fichas jugadas y, para el resto de los índices i, la ficha
 *       partida[i] está situada entre las fichas partida[i-1] y partida[i+1].
 * Post: Ha devuelto true si y solo si [ficha] puede ponerse en uno de los dos
 *       extremos de la tabla [partida].
 */
public static boolean sePuedePoner(FichaDomino[] partida, FichaDomino ficha)
```

```
/**
 * Pre:  [misFichas] es distinto de null; cada componente de [misFichas] es
 *       distinta de null;
 *       [partida].length >= 2 y cada componente de [partida] es un objeto no
 *       nulo de la clase FichaDomino correctamente puesto (de acuerdo con
 *       las reglas del dominó) en una partida:
 *       Los elementos referenciados desde [partida] están ordenados de
 *       idéntica forma a como estarían colocadas las fichas de dominó ya
 *       jugadas en una partida real: Las fichas de índices menor y mayor
 *       corresponden con los extremos de la secuencia de fichas jugadas y,
 *       para el resto de los índices i, la ficha partida[i] está situada
 *       entre las fichas partida[i-1] y partida[i+1].
 * Post: Si alguna de las fichas de la tabla [misFichas] puede ir en alguno
 *       de los extremos de las fichas de la tabla [partida], ha devuelto la
 *       referencia a una de esas fichas. En caso de que ninguna de las
 *       fichas de la tabla [misFichas] pueda colocarse en ningún extremo de
 *       la tabla [partida], ha devuelto null.
 */
public static FichaDomino puedoPoner(FichaDomino[] partida,
                                     FichaDomino[] misFichas)
```

Observación importante: No se admitirá ninguna solución no estructurada. Ello supone que cualquier bucle debe concluir exclusivamente al dejar de satisfacerse su condición de iteración.

Problema 3.º (4 puntos)

Una forma de representar digitalmente imágenes bidimensionales es considerar que están compuestas por una determinada cantidad (generalmente grande, pero finita) de elementos discretos organizados por filas y columnas. Cada uno de estos elementos que forman la imagen se denomina píxel (del acrónimo inglés de *pix* [plural coloquial de *picture*] y *element*) y tiene como características un grado de brillo y color determinados.

En este problema, se va a trabajar con imágenes cuyos píxeles pueden ser solo blancos o negros y van a estar almacenadas en ficheros **binarios** en los que los dos primeros datos son de tipo **`int`** y representan, respectivamente, el alto (número de filas) y el ancho (número de columnas) de la imagen, seguidos de $ancho \times alto$ datos de tipo **`boolean`** que representan cada uno de los píxeles de la imagen, ordenados primeramente por filas de arriba a abajo y, dentro de cada fila, por columnas de izquierda a derecha. Un valor **`true`** representa un píxel blanco y un valor **`false`**, un píxel negro.

```
<fichero_imagen> ::= <alto><ancho>{<píxel>}
<ancho> ::= int
<alto> ::= int
<píxel> ::= boolean
```

Por ejemplo, la siguiente secuencia correspondiente a un fichero con el formato descrito corresponde con la imagen que se muestra en la figura 1. Se recuerda que el fichero es binario y que, en el ejemplo, se está utilizando una representación decimal para los datos de tipo **int**, textual para los datos de tipo **boolean** y se han añadido espacios en blanco y estructurado el contenido en líneas solo con el propósito de aumentar la legibilidad del mismo.

```
7 9
true  true  true  true  true  true  true  true  true
true  false true  false true  false false false true
true  false true  false true  true  true  false true
true  false true  false true  true  false true  true
true  false true  false true  false true  true  true
true  false false false true  false false false true
true  true  true  true  true  true  true  true  true
```

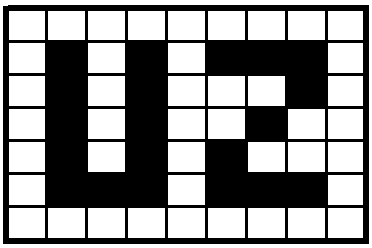

Figura 1: Imagen correspondiente a la secuencia del ejemplo

Se pide completar el código de los métodos constructor y `mostar` de la clase `Imagen` cuyo código fuente, excepto el cuerpo de los métodos solicitados, se muestra a continuación:

```
package examen;

/**
 * Los objetos de esta clase representan imágenes en blanco y negro
 */
public class Imagen {

    /**
     * Anchura en píxeles (número de columnas) de la imagen representada por
     * este objeto
     */
```

```
    private int ancho;

    /**
     * Altura en píxeles (número de filas) de la imagen representada por este
     * objeto
     */
    private int alto;

    /**
     * Tabla bidimensional con los píxeles correspondientes a la imagen
     * representada por este objeto. Un valor true corresponde con un píxel
     * blanco, mientras que un valor false corresponde con un píxel negro.
     */
    private boolean[][] imagen;

    /**
     * Pre: el fichero cuyo nombre representa el parámetro [fichero] tiene un
     * formato como el establecido en el enunciado.
     * Post: Ha inicializado los atributos de este objeto de acuerdo con el
     * contenido del fichero [fichero], a no ser que haya habido algún error
     * en la lectura del mismo, en cuyo caso, ha realizado una inicialización
     * para representar una imagen de 0x0 píxeles.
     */
    public Imagen(File fichero) {
        ...
    }

    /**
     * Post: Ha mostrado esta imagen en la pantalla, utilizando el carácter
     * 'X' para representar los píxeles negros, y el carácter ' ' para los
     * blancos.
     * Por ejemplo, en el caso de la imagen de la figura 1, mostraría en
     * pantalla lo siguiente:
     *
     *  X X XXX
     *  X X   X
     *  X X  X
     *  X X X
     *  XXX XXX
     */
    public void mostar() {
        ...
    }
}
```

Solución al problema 1.º

```java
package examen;

/**
 * Los objetos de esta clase representan fichas para jugar al dominó. Se
 * caracterizan, por tanto, por un par de valores entre 0 y 6 que
 * representan los puntos marcados en cada uno de los cuadrados que las
 * componen
 */
public class FichaDomino {

    /**
     * Menor de los dos valores de esta ficha
     */
    private int valorMenor;

    /**
     * Mayor de los dos valores de esta ficha
     */
    private int valorMayor;

    /**
     * Pre: 0 <= valor1 <= 6 y 0 <= valor2 <= 6
     *
     * Post: Ha inicializado los atributos de esta ficha con los valores
     * de los parámetros [valor1] y [valor2]
     */
    public FichaDomino(int valor1, int valor2) {
        if (valor1 <= valor2) {
            valorMenor = valor1;
            valorMayor = valor2;
        }
        else {
            valorMenor = valor2;
            valorMayor = valor1;
        }
    }

    /**
     * Post: Ha devuelto el menor de los dos valores de esta ficha
     */
    public int valorMenor() {
        return valorMenor;
    }

    /**
     * Post: Ha devuelto el mayor de los dos valores de esta ficha
     */
    public int valorMayor() {
        return valorMayor;
    }
```

```
    /**
     * Post: Ha devuelto true si y solo si esta ficha es una ficha doble
     * (es decir, sus dos valores son iguales)
     */
    public boolean esDoble() {
        return valorMenor == valorMayor;
    }

    /**
     * Post: Ha devuelto el número de puntos de esta ficha (la suma de
     * los dos valores que la componen)
     */
    public int puntos() {
        return valorMenor + valorMayor;
    }

    /**
     * Post: Ha devuelto una cadena de caracteres que representa esta
     * ficha, conteniendo los dos valores de la ficha separados por el
     * carácter '|' y enmarcados también por un par de caracteres '|'.
     *
     * Por ejemplo, si esta ficha es el cinco doble, ha devuelto la
     * cadena "|5|5|"
     */
    public String toString() {
        return "|" + valorMenor + "|" + valorMayor + "|";
    }

    /**
     * Pre: Esta ficha y la ficha respresentada por [contigua] pueden ir
     * juntas en la mesa según las reglas del dominó.
     *
     * Post: Suponiendo que esta ficha está en uno de los dos extremos de
     * la secuencia de fichas puestas en la partida, y que la ficha
     * [contigua] es la ficha que tiene a su lado, ha devuelto el valor
     * del extremo libre de esta ficha.
     */
    public int valorExtremoLibre(FichaDomino contigua) {
        if (valorMenor == contigua.valorMenor
                || valorMenor == contigua.valorMayor) {
            // Esta ficha y la [contigua] comparten el valorMenor de esta
            // ficha
            return valorMayor;
        }
        else {
            // Esta ficha y la [contigua] comparten el valorMayor de esta
            // ficha
            return valorMenor;
        }
    }
}
```

Solución al problema 2.º

```
/**
 * Pre:  [ficha] es distinta de null;
 *       [partida].length >= 2 y cada componente de [partida] es una
 *       referencia no nula a un objeto de la clase FichaDomino correctamente
 *       puesto (de acuerdo con las reglas del dominó) en una partida:
 *       Los elementos referenciados desde [partida] están ordenados de
 *       idéntica forma a como estarían colocadas las fichas de dominó ya
 *       jugadas en una partida real: Las fichas de índices menor y mayor
 *       corresponden con los extremos de la secuencia de fichas jugadas y,
 *       para el resto de los índices i, la ficha partida[i] está situada
 *       entre las fichas partida[i-1] y partida[i+1].
 *
 * Post: Ha devuelto true si y solo si [ficha] puede ponerse en uno de los dos
 *       extremos de la tabla [partida].
 */
public static boolean sePuedePoner(FichaDomino[] partida,
        FichaDomino ficha) {
    // Se prueba por la izquierda (componente 0)
    int extremo = partida[0].valorExtremoLibre(partida[1]);
    if (extremo == ficha.valorMayor()
            || extremo == ficha.valorMenor()) {
        return true;
    }
    else {
        // Se prueba por la derecha (componente última)
        extremo = partida[partida.length - 1]
                .valorExtremoLibre(partida[partida.length - 2]);
        return extremo == ficha.valorMayor()
                || extremo == ficha.valorMenor();
    }
}
```

```java
/**
 * Pre:  [misFichas] es distinto de null; cada componente de [misFichas] es
 *       distinta de null;
 *       [partida].length >= 2 y cada componente de [partida] es un objeto no
 *       nulo de la clase FichaDomino correctamente puesto (de acuerdo con
 *       las reglas del dominó) en una partida:
 *       Los elementos referenciados desde [partida] están ordenados de
 *       idéntica forma a como estarían colocadas las fichas de dominó ya
 *       jugadas en una partida real: Las fichas de índices menor y mayor
 *       corresponden con los extremos de la secuencia de fichas jugadas y,
 *       para el resto de los índices i, la ficha partida[i] está situada
 *       entre las fichas partida[i-1] y partida[i+1].
 *
 * Post: Si alguna de las fichas de la tabla [misFichas] puede ir en alguno
 *       de los extremos de las fichas de la tabla [partida], ha devuelto la
 *       referencia a una de esas fichas. En caso de que ninguna de las
 *       fichas de la tabla [misFichas] pueda colocarse en ningún extremo de
 *       la tabla [partida], ha devuelto null.
 */
public static FichaDomino puedoPoner(FichaDomino[] partida,
        FichaDomino[] misFichas) {
    // Esquema de búsqueda sin garantía de éxito
    boolean puedoPoner = false;
    int i = 0;
    while (!puedoPoner && i < misFichas.length) {
        // Se intenta poner la i-ésima ficha
        if (sePuedePoner(partida, misFichas[i])) {
            puedoPoner = true;
        }
        else {
            i++;
        }
    }
    // puedoPoner || i == misFichas.length

    if (puedoPoner) {
        return misFichas[i];
    }
    else {
        return null;
    }
}
```

Solución al problema 3.º

```
/**
 * Pre: El fichero cuyo nombre representa el parámetro [fichero]
 * tiene un formato como el establecido en el enunciado.
 *
 * Post: Ha inicializado los atributos de este objeto de acuerdo con
 * el contenido del fichero [fichero], a no ser que haya habido algún
 * error en la lectura del mismo, en cuyo caso, ha realizado una
 * inicialización para representar una imagen de 0x0 píxeles
 */
public Imagen(File fichero) {
    try {
        ObjectInputStream f = new ObjectInputStream(
                new FileInputStream(fichero));

        // Obtención de las dimensiones
        alto = f.readInt();
        ancho = f.readInt();

        // Creación de la tabla para almacenar los píxeles
        imagen = new boolean[alto][ancho];

        // Lectura de los píxeles
        for (int i = 0; i < alto; i++) {
            for (int j = 0; j < ancho; j++) {
                imagen[i][j] = f.readBoolean();
            }
        }
        f.close();
    }
    catch (IOException ex) {
        // En caso de error en la lectura, inicialización para una
        // imagen de 0x0
        alto = 0;
        ancho = 0;
        imagen = new boolean[0][0];
    }
}
```

```java
/**
 * Post: Ha mostrado esta imagen en la pantalla, utilizando el
 * carácter 'X' para representar los píxeles negros, y el carácter
 * ' ' para los blancos.
 *
 * Por ejemplo, en el caso de la imagen de la figura 1, mostraría en
 * pantalla lo siguiente:
 *
 *  X X XXX
 *  X X   X
 *  X X  X
 *  X X X
 *  XXX XXX
 */
public void mostar() {
    // Para cada fila...
    for (int i = 0; i < alto; i++) {
        // ... y para cada columna, ...
        for (int j = 0; j < ancho; j++) {
            // ... se "muestra" el píxel correspondiente
            if (imagen[i][j]) {
                System.out.print('␣');
            }
            else {
                System.out.print('X');
            }
        }
        // Al acabar una fila, se cambia de línea
        System.out.println();
    }
}
```