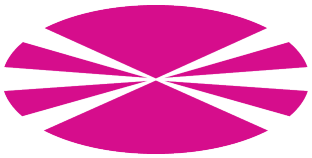
spanish



UNIVERSIDADE DA CORUÑA

Diseño Software
Curso 2018-2019

Boletín de Ejercicios 1

NOTAS:

- **Lee las instrucciones de los ejercicios antes de empezar a realizarlos.**
- **Os proporcionaremos algunos ejemplos de los tests JUnit. Un criterio importante de corrección es que los ejercicios realizados superen sin problemas los tests pasados.**
- **Fecha límite de entrega: jueves 11 de octubre de 2018 (hasta las 23:59).**

1. **Palíndromos**

   Crea una clase `Palindromo` que cumpla la siguiente especificación.

```
public class Palindromo {
    /**
     * Dado un String pasado por parámetro devuelve un valor booleano indicando
     * si dicho String es palíndromo o no. Un palíndromo es una palabra o frase que
     * se lee igual tanto si empiezas por el principio como por el final.
     *
     * En nuestro caso el método no es sensible a las diferencias entre mayúsculas y
     * minúsculas y solo tendrá en cuenta los caracteres ASCII situados entre la 'A'
     * y la 'Z' (o la 'a' y la 'z'). El resto de caracteres como espacios, símbolos
     * de interrogación, etc. serán ignorados.
     *
     * Ejemplos de palíndromos:
     *    * "Radar" (ignora las diferencias entre mayúsculas y minúsculas)
     *    * "Dabale arroz a la zorra el abad" (ignora los espacios)
     *
     * @param texto String a analizar.
     * @return Devuelve true si es palindromo. False si no lo es.
     */
    public static boolean esPalindromo(String texto) { /* ... */  }
}
```

   Criterios:

   - Manejo de bucles.
   - Manejo de Strings con las clases que sean necesarias.

2. **Canciones y métodos de la clase `Object`**

   Crea una clase inmutable `Cancion` que represente a una canción y que incluya los siguientes campos: `titulo`, `autor`, `album` y `estilo`. Definidos todos de tipo `String`.

   La clase deberá incluir (ver código adjunto):

   - Un constructor.
   - Métodos públicos de lectura para las propiedades (que deberán ser privadas). No se incluirán métodos de escritura.
   - Sobrescritura de los métodos `toString`, `equals` y `hashCode` tal y como se indica.

```java
public class Cancion {
    /**
     * Construye una canción con los parámetros pasados
     */
    public Cancion(String titulo, String autor , String album, String estilo)
    {/*...*/}
    // Getters...
    public String getTitulo() { /* ... */ }
    public String getAutor() { /* ... */ }
    public String getAlbum() { /* ... */ }
    public String getEstilo() { /* ... */ }

    /**
     * Devuelve una representación en String que representa a la canción.
     * Se trata de una concatenación de titulo autor y album separados por
     * guiones: "Titulo - Autor - Album".
     * Si cualquiera de los tres campos tiene más de 20 caracteres se truncará
     * a 20 caracteres para evitar Strings largos.
     * @return String representando la canción.
     */
    @Override
    public String toString() { /* ... */ }

    /**
     * Igualdad lógica entre dos canciones. Dos canciones se considerarán iguales
     * si tienen el mismo título y el mismo autor. Para la igualdad lógica se
     * ignorará el álbum (ya que una misma canción se puede incluir en recopilatorios)
     * y el estilo (ya que puede aparecer clasificada bajo diferentes estilos).
     *
     * Para simplificar se supondrá que para que dos canciones sean iguales los
     * Strings que representan al titulo y al autor tienen que estar escritos
     * exáctamente igual (diferencias con mayúsculas y minúsculas son relevantes).
     *
     * @param obj Objeto a comparar con el objeto actual.
     * @return True si son iguales, False en caso contrario
     */
    @Override
    public boolean equals(Object obj) { /* ... */ }

    /**
     * Devuelve un código hash que representa a la canción. Recuerda que canciones
     * que son iguales deben devolver el mismo hashcode.
     * @return Un entero representando el código hash de la canción.
     */
    @Override
    public int hashCode() { /* ... */ }
}
```

Criterios:

- Encapsulación.
- Cumplimiento de los contratos de `toString`, `equals` y `hashCode`.

3. **Tablero de ajedrez**

   En este ejercicio vamos a diseñar e implementar una clase `Tablero` que permita modelar un tablero de ajedrez y jugar una partida. Para ello, en primer lugar, necesitamos crear una clase `Pieza` que represente a las piezas de ajedrez. Cada `Pieza` tendrá una propiedad `tipo` definida como el enumerado `PiezasTipo` con los valores: `REY`, `DAMA`, `ALFIL`, `CABALLO`, `TORRE` o `PEON` y una propiedad `color` definida como el enumerado `PiezasColor` con los valores: `BLANCO` y `NEGRO`.

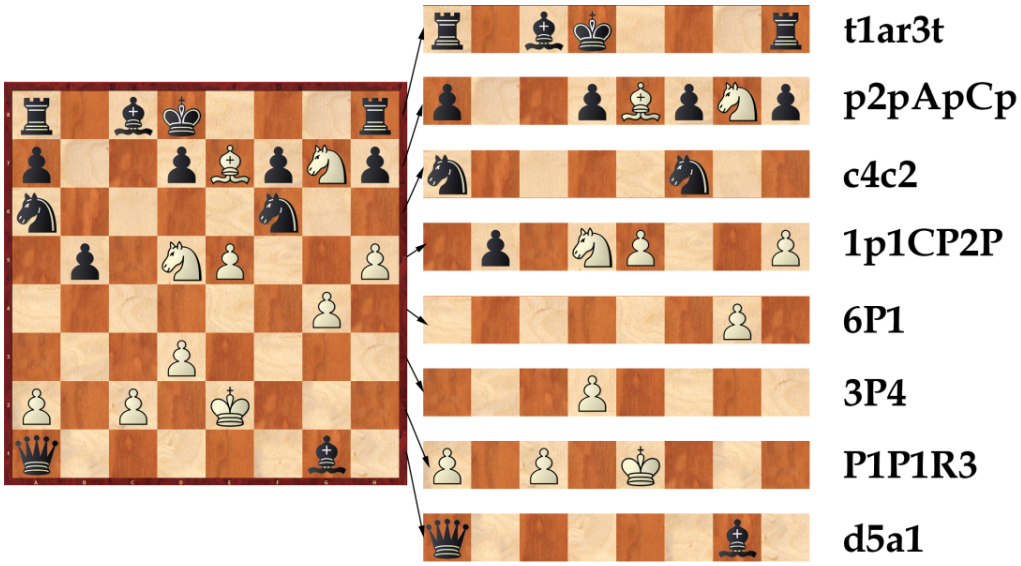
   Antes de definir la clase `Tablero` es necesario aclarar una serie de conceptos. La disposición inicial de un tablero de ajedrez es la siguiente:

   La notación FEN (Forsyth-Edwards Notation) es una notación estándar para describir una posición de tablero particular de una partida de ajedrez. Consiste en lo siguiente:

   - El contenido de cada fila se describe con una cadena de texto y las ocho cadenas se compactan separándolas con barras (/").
   - Se comienza por la fila 8 y se termina por la fila 1 (de arriba abajo desde el punto de vista de las blancas).
   - Dentro de cada fila, el contenido de cada casilla se describe comenzando por la columna .ª"hasta la "h".
   - Cada pieza se identifica por su primera letra: rey es "R", dama es "D", alfil "A", caballo "C", torre "T" y peón "P".
   - Las piezas blancas se designan utilizando letras mayúsculas ("RDACTP") y las piezas negras se designan utilizando letras minúsculas ("rdactp").
   - Las casillas vacías se denotan utilizando un número que indica el número de casillas vacías.

   La cadena FEN de la posición inicial de una partida de ajedrez es:
   tcadract/pppppppp/8/8/8/8/PPPPPPPP/TCADRACT

   La notación FEN de la siguiente figura es:
   t1ar3t/p2pApCp/c4c2/1p1CP2P/6P1/3P4/P1P1R3/d5a1



La clase `Tablero` se definirá por lo tanto de la siguiente manera:

```
/**
 * Representa un tablero de ajedrez
 */
public class Tablero {
    /**
     * Constructor que crea un tablero inicializado a su posición inicial.
     */
    public Tablero() { /* ... */ }

    /**
     * Establece en un tablero ya creado la posición inicial de una partida de
     * ajedrez.
     */
    public void iniciarPartida() { /* ... */ }

    /**
     * Devuelve un String con la representación del tablero en notación FEN
     * @return String representando la notación FEN
     */
    public String toString() { /* ... */ }

    /**
     * Recibe : recibe un tipo de pieza y devuelve una lista de objetos de la
     * clase Posicion se encuentran las Piezas de ese tipo (blancas y negras).
     * @param pieza Pieza de ajedrez
     * @return Lista de objetos de tipo Posicion con las posiciones de la
     *         pieza pasada por parámetro.
     */
    public List buscaPiezas(PiezasTipo pieza) { /* ... */ }

    /**
     * Realiza movimientos de Piezas en el Tablero.
     * Recibe una Posicion de origen y una Posicion de destino.
     * Si en la Posicion de destino hay una Pieza, se hará una captura,
     * colocando en su lugar la pieza que estamos moviendo.
     * Por simplicidad no se harán comprobaciones de si el movimiento es válido o no.
     * De todas formas el método lanzará excepciones si en la posición de origen
     * no hay una pieza o si alguna de las posiciones es errónea.
     * @param origen Posición de origen de la ficha a mover.
     * @param destino Posición de destino de la ficha a mover.
     */
    public void mover(Posicion origen, Posicion destino)  { /* ... */ }
}
```

La clase `Posicion` se define de la siguiente forma:

```java
/**
 * Clase inmutable que representa una posición de un tablero de ajedrez
 */
public class Posicion {
    /**
     * Crea una posición a partir de dos coordenadas de un tablero de ajedrez
     * codificadas como char.
     * @param columnaChar Columna con valores entre a y h.
     * @param filaChar Fila con valores entre 1 y 8.
     */
    public Posicion(char columnaChar, char filaChar) { /* ... */ }

    /**
     * Devuelve el valor de la fila.
     * @return La fila
     */
    public char getFila() { /* ... */ }

    /**
     * Devuelve el valor de la columna.
     * @return La columna
     */
    public char getColumna() { /* ... */ }
}
```

Criterios:

- Creación de clases, abstracción, encapsulamiento.
- Manejo de enumerados simples.

4. **Juego de billar**

Crearemos una clase enumerada `BolaBillar` que tenga los siguientes valores enumerados con las siguientes características:

Bola	`numero`	`ColorBolas`	`TipoBolas`
BLANCA	0	BLANCO	LISA
BOLA1	1	AMARILLO	LISA
BOLA2	2	AZUL	LISA
BOLA3	3	ROJO	LISA
BOLA4	4	VIOLETA	LISA
BOLA5	5	NARANJA	LISA
BOLA6	6	VERDE	LISA
BOLA7	7	GRANATE	LISA
BOLA8	8	NEGRO	LISA
BOLA9	9	AMARILLO	RAYADA
BOLA10	10	AZUL	RAYADA
BOLA11	11	ROJO	RAYADA
BOLA12	12	VIOLETA	RAYADA
BOLA13	13	NARANJA	RAYADA
BOLA14	14	VERDE	RAYADA
BOLA15	15	GRANATE	RAYADA

Crea una clase `MesaBillar` que simule el funcionamiento del juego del billar de la siguiente forma:

```java
public class MesaBillar {

    /**
     * Crea una mesa de billar con todas las bolas disponibles en el cajetín de
     * la mesa
     */
    public MesaBillar() { /* ... */ }

    /**
     * Inicializará el estado de la mesa indicando que hay una partida en marcha
     * y que todas las bolas están encima de la misma.
     */
    public void iniciaPartida() { /* ... */ }

    /**
     * Si es una bola normal (de 1 a 7 o de 9 a 15) se sacará de encima de la
     * mesa y se pondrá en el cajetín con las otras bolas ya metidas en orden de
     * introducción.
     *
     * Si es la blanca volverá a la mesa (a no ser que se haya acabado la partida)
     *
     * Si es la negra considerará que la partida ha acabado.
     *
     * @param b Bola de billar que se introduce en la tronera
     * @return El estado de la partida después de la introducción de la bola.
     *         True si la partida está iniciada o False si no lo esta.
     *         Solo cambia si la bola introducida es la negra que inmediatamente
     *         termina la partida.
     */
    public boolean meterBola(BolasBillar b) { /* ... */ }

    /**
     * Devuelve un ArrayList con las bolas que hay en juego en la mesa.
     * @return Un ArrayList con objetos de tipo BolasBillar.
     */
    public ArrayList bolasMesa() { /* ... */ }

    /**
     * Devuelve un ArrayList con las bolas que hay en el cajetín la mesa.
     * @return Un ArrayList con objetos de tipo BolasBillar.
     */
    public ArrayList bolasCajetin() { /* ... */ }

    /**
     * Comprueba si la partida está en juego o no
     * @return True si la partida está en juego. False en caso contrario.
     */
    public boolean esPartidaIniciada() { /* ... */ }

    /**
     * Indica qué jugador va ganando (si el de las bolas lisas (bolas de la 1 a
     * la 7) o el de las bolas rayadas (bolas de la 9 a la 15). Básicamente va
     * ganando (o ha ganado si la partida ha terminado) aquel jugador que tenga
     * menos bolas de su tipo encima de la mesa.

     * @return El enumerado TipoBolas.LISA si van ganando las lisas. El
     *         enumerado TipoBolas.RAYADA si van ganando las rayadas. O el
     *         valor null si hay un empate.
     */
    public BolasBillar.TipoBolas obtenerGanador() { /* ... */ }
}
```

Criterios:

- Creación y manejo de objetos enumerados complejos.
- Manejo de clases de colecciones de objetos como `ArrayList`.