UNIVERSIDAD DE BURGOS
Departamento de Ingeniería Civil
Área de Lenguajes y Sistemas Informáticos

METODOLOGÍA DE LA PROGRAMACIÓN **30-OCTUBRE-2017 – 1ª CONVOCATORIA**

Apellidos:_____ ***Nombre:***_____

Estimación de la calificación (sobre 2 puntos): ☐

Total del ejercicio 2 ptos. Nota mínima de corte 0.9 Ptos

Nota: no se corrigen respuestas con tachones o realizadas a lápiz

1. **Construir las siguientes clases y enumeraciones en Java pertenecientes al paquete** `mepro.ejercicio1`**:**
   - **Clase** `CartonBingo` **con constructor y métodos que permitan:** (0.90 ptos)
     - **Constructor público sin argumentos, que inicializa el cartón, sólo reservando memoria para almacenar sus números (3 filas x 5 números). Es obligatorio utilizar un array bidimensional de Java, NO se permite el uso de `ArrayList`.**
     - **Método `public void colocar(int fila, int valor)` : permite añadir un nuevo valor en la fila indicada, en la primera posición libre, de izquierda a derecha. Si el n.º de la fila es incorrecta, la fila está completa, el valor no está en el intervalo [1,90] o el valor ya se había introducido en el cartón (no se permite repetir números en un cartón), simplemente no se hace nada.**
     - **Método `public boolean estaLleno()` : consulta si el cartón está completo. Devuelve `true` si ya se han colocado los 15 valores, `false` en caso contrario.**
     - **Nota: se deben añadir atributos y métodos privados en la clase, evitando métodos con código repetido y excesivamente largos. NO se solicita documentar el código fuente con comentarios javadoc.**

     Ejemplo de un cartón de bingo, bien generado con esta clase, completo y sin repeticiones:

     ```
     20 51  4 79 66
     63 62 44 52  9
     12 80 47 11 42
     ```

   - **Enumeración** `Resultado` **: contiene los dos posibles valores que pueden cantar los clientes:** (0.10 ptos)
     - **`LÍNEA`: cuando un cliente ha completado de marcar una línea del cartón**
     - **`BINGO`: cuando un cliente ha completado de marcar el cartón entero.**

     Nota: NO se solicita documentar el código fuente con comentarios javadoc.

```
package mepro.ejercicio1;

public enum Resultado {
	LINEA, BINGO;
}
```

```java
package mepro.ejercicio1;

public class CartonBingo {

	private static final int COLUMNAS = 5;
	private static final int FILAS = 3;
	private static final int LIMITE_SUP = 90;

	private int[][] numeros;

	private int numerosColocados;

	public CartonBingo() {
		numeros = new int[FILAS][COLUMNAS];
	}

	private boolean contiene(int valor) {
		for (int i = 0; i < numeros.length; i++) {
			for (int j = 0; j < numeros[i].length; j++) {
				if (numeros[i][j] == valor) {
					return true;
				}
			}
		}
		return false;
	}

	private boolean hayHueco(int fila) {
		if (fila >= 0 && fila < FILAS) {
			for (int i = 0; i < numeros[fila].length; i++) {
				if (numeros[fila][i] == 0) {
					return true;
				}
			}
		}
		return false;
	}

	private boolean esValorCorrecto(int valor) {
		return valor >= 1 && valor <= LIMITE_SUP;
	}

	public void colocar(int fila, int valor) {
		if (!contiene(valor) && hayHueco(fila) && esValorCorrecto(valor)) {
			boolean flag = true;
			for (int i = 0; i < numeros[fila].length && flag; i++) {
				if (numeros[fila][i] == 0) {
					numeros[fila][i] = valor;
					numerosColocados++;
					flag = false;
				}
			}
		}
	}

	public boolean estaLleno() {
		return numerosColocados == FILAS * COLUMNAS;
	}
}
```

UNIVERSIDAD DE BURGOS
Departamento de Ingeniería Civil
Área de Lenguajes y Sistemas Informáticos

2. **Dada las siguientes declaraciones de clases en Java:**

```java
public class A {
        private int valor;

        public A(int i){
                valor = i;
        }

        public static A generar(int i){
                return new A(i);
        }

        public int obtener() {
                return valor;
        }
}
```

```java
public class B {
        private int j;

        public B(int j) {
                this.j = j;
        }

        public int obtener() {
                return j;
        }
}
```

```java
public class C {
        private int k;

        int obtener() {
                return k;
        }
}
```

```java
public class M1 {
        public static void main(String[] args) {
                A a1 = new A(0);
                B b1 = new B(0);
                C c1 = new C(); // línea 3
        }
}
```

a) Indicar <u>sólo las líneas imprescindibles</u> a añadir en las cabeceras de las clases anteriores, para que se produzca el ensamblaje correcto del sistema, sabiendo que: (0.30 ptos)

1. **La clase `A` pertenece al paquete de nombre `paquete1`**
2. **La clase `B` pertenece al paquete de nombre `paquete3`**
3. **La clase `C` pertenece al paquete de nombre `paquete1.paquete2`**
4. **La clase `M1` pertenece al paquete de nombre `paquete3.paquete4`**

En clase `A` añadir: `package paquete1;`
En clase `B` añadir: `package paquete3;`
En clase `C` añadir: `package paquete1.paquete2;`
En clase `M1` añadir:

```java
package paquete3.paquete4;
import paquete1.A; // o import paquete1.*;
import paquete3.B; // o import paquete3.*;
import paquete1.paquete2.C; // o import paquete1.paquete2.*;
```

b) Suponiendo que a continuación de la línea 3 del método `main`, añadimos las siguientes líneas:

```java
b1 = c1;                    // linea 4
A a2 = A.generar(0);        // linea 5
int val = c1.obtener();     // linea 6
b1.j = a1.obtener();        // linea 7
```

Indicar, explicando brevemente el motivo, sólo aquellas líneas que generarían errores de compilación. Nota: una línea mal explicada, anula a un error bien indicado. (0.15 ptos)

Líneas incorrectas:

- **Línea 4: no hay compatibilidad de tipos entre `B` y `C`.**
- **Línea 6: `obtener()` no es accesible (modificador amigable)**
- **Línea 7: atributo `j` en `B` no accesible, es privado**

3. **A partir del siguiente código, que implementa una nueva clase `Tablero` y que utiliza una clase `Celda` y `Pieza` junto con una enumeración `Color`, equivalentes a la ya utilizadas en prácticas:** (0.55 ptos)

```
public class Tablero {

	private Celda[][] matriz;

	public Tablero(int filas, int columnas) {
		matriz = new Celda[filas][columnas];
		for (int i = 0; i < filas; i++) {
			for (int j = 0; j < columnas; j++) {
				matriz[i][j] = new Celda(i, j);
				if ((i * j) >= 2) {
					matriz[i][j].establecerPieza(new Pieza(Color.NEGRO));
				}
			}
		}
	}

	public void cambiarPiezasEnCiertasCeldas() {
		for (int i = 0; i < matriz.length; i++) {
			for (int j = 0; j < matriz[i].length; j++) {
				if ((i * j) % 2 == 0 && !(matriz[i][j].estaVacia())) { // OJO con !
					matriz[i][j].establecerPieza(new Pieza(Color.BLANCO));
				}
			}
		}
	}

	public static void main(String[] args) {
		Tablero tablero1 = new Tablero(4,4);		// línea 1
		tablero1.cambiarPiezasEnCiertasCeldas();	// línea 2
	}
}
```

a) En el método `main`:

a.1) ¿Explicar de forma razonada cuántos objetos y de qué tipo se han generado tras ejecutar la línea 1?

a.2) ¿Explicar de forma razonada cuántos objetos nuevos y de qué tipo se han generado tras ejecutar la línea 2 y antes de finalizar la ejecución del método `main`?

a.3) ¿Qué objetos, cuántos y de qué tipo pasan a ser inalcanzables, después de ejecutar la línea 2 y antes de finalizar la ejecución del método `main`?

Nota: NO es requisito imprescindible dibujar el tablero y sus objetos asociados, pero se puede añadir si se considera oportuno.

a.1) Se genera **un nuevo objeto `Tablero`**. Para ese tablero se **generan 4x4 celdas (16 nuevos objetos de tipo `Celda`)**. Para aquellas celdas cuya fila*columna sea mayor o igual que 2, se generará una nueva pieza.

Si dibujamos el *array* con sus indices y el resultado de i*j para los 16 casos, tenemos en amarillo señalado dónde se cumple la condición:

	0	*1*	*2*	*3*
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	9

Por lo tanto, se generan **8 nuevos objetos `Pieza`**. Todas las piezas comparten **un único objeto `Color.NEGRO`** dado que está definido en una enumeración.

a.2) Al ejecutar la línea 2, se reemplaza en aquellas posiciones donde `i*j` sea número par y además haya realmente una pieza en la celda correspondiente a `(i,j)`. Estos casos son sólo los coloreados en azul:

	0	*1*	*2*	*3*
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	9

Por lo tanto se reemplazan las piezas negras por **5 nuevos objetos de tipo `Pieza`**, que comparten **un único objeto `Color.BLANCO`** generado por la enumeración.

a.3) Los **5 objetos de tipo `Pieza`** que estaban en las celdas con coordenadas (1,2) (2,1) (2,2) (2,3) y (3,2), y que contenían piezas de `Color.NEGRO`, puesto que han sido sustituidos en la línea 2 por nuevas piezas de `Color.BLANCO`, en dichas posiciones. Esos objetos pasan a ser inalcanzables y deberían ser reclamados por el recolector de basura.