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

METODOLOGÍA DE LA PROGRAMACIÓN **25-OCTUBRE-2018 – 1ª CONVOCATORIA**

*Apellidos:*_____ *Nombre:*_____

Estimación del alumno/a de su 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 **NO se solicita documentar el código fuente con comentarios, ni con comentarios *javadoc*.**

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

```
public class A {
        private int valor;
        B b;

        public A(int i){
                valor = i;
                b = new B(i);
        }

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

        int obtener() {
                return valor * b.obtener();
        }
}
```

```
public class B {
        private int j;
        private C c;

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

        public int obtener() {
                return j * c.obtener();
        }
}
```

```
public class C {
        private int k;

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

```
public class M {
        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 clases anteriores, para que se produzca el ensamblaje correcto del sistema, sabiendo que: (0.20 ptos)

- **La clase `A` pertenece al paquete de nombre `paquete1`**
- **La clase `B` pertenece al paquete de nombre `paquete1.paquete2`**
- **La clase `C` pertenece al paquete de nombre `paquete3`**
- **La clase `M` pertenece al paquete de nombre `paquete3`**

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

```
C c2 = c1.generar(0);          // linea 4
int val = a1.obtener();        // linea 5
a1.valor = 0;                  // linea 6
a1.b = new B(0);               // linea 7
```

Indicar, explicando brevemente el motivo, solo de aquellas líneas que generarían errores de compilación. Nota: una línea mal explicada, anula a un error bien indicado. (0.20 ptos)

c) Si las anteriores clases se compilan dejando los binarios resultantes en el directorio `.\bin` y además necesitamos para una correcta ejecución, algunos paquetes y clases contenidos en `.\lib\bib-1.0.jar`, indicar: (Nota: suponemos que el sistema operativo es Windows, pero se puede dar la solución para Mac o GNU/Linux). (0.20 ptos)

c.1) ¿Qué valor tenemos que dar al CLASSPATH para una correcta ejecución?

c.2) ¿Cómo ejecutamos en línea de comandos, desde el directorio actual, la clase principal `M`?

a)

En A.java:

```
package paquete1;
import paquete1.paquete2.B;
```

En B.java:

```
package paquete1.paquete2;
import paquete3.C;
```

En C.java:

```
package paquete3;
```

En M.java:

```
package paquete3;
import paquete1.A;
import paquete1.paquete2.B;
```

b)

```
C c2 = c1.generar(0);    // ERROR: el método generar no existe en el tipo
int val = a1.obtener(); // ERROR: método obtener no es accesible
a1.valor = 0;           // ERROR: atributo valor no es accesible
a1.b = new B(0);        // ERROR: atributo b no accesible
```

c)

c.1) `set CLASSPATH=.\bin;\lib\bib-1.0.jar`

c.2) `java paquete3.M`

2. **A partir del siguiente código que utiliza las clases mostradas en el Ejercicio 1:**

a.1) ¿Explicar de forma razonada cuántos objetos y de qué tipo se han generado al finalizar la ejecución del bucle? Nota: NO es requisito imprescindible dibujar los arryas y sus objetos asociados, pero se puede añadir si se considera oportuno. (0.20 ptos)

a.2) Modifica el código de la clase B del Ejercicio 1 para que se pueda consultar sobre dicha clase el número de instancias generadas en cada momento, a través de un método. (0.30 ptos)

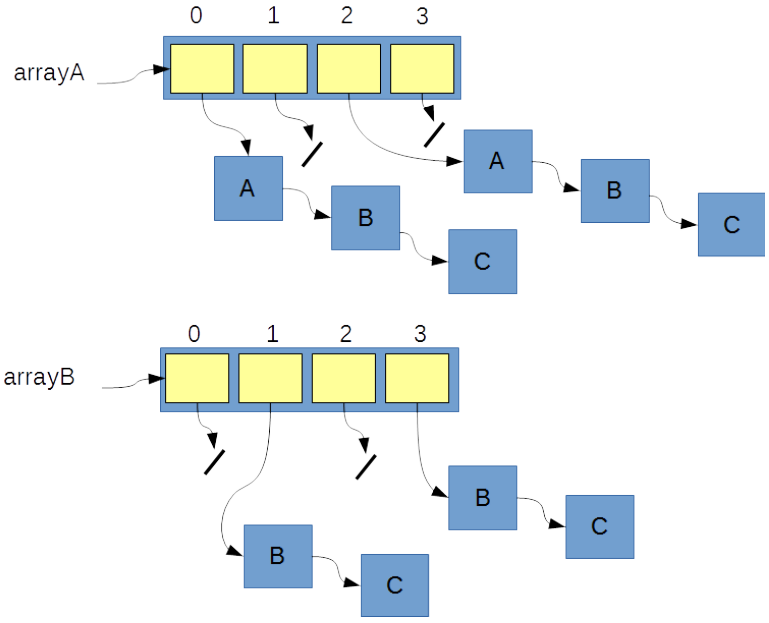
```
public class Main {

		private static A[] arrayA;

		private static B[] arrayB;

		public static void main(String[] args) {
			final int[] valores = {0, 1, 0, 1};
			arrayA = new A[valores.length];
			arrayB = new B[valores.length];
			int i = 0;
			do {
				if (valores[i] % 2 == 0) {
					arrayA[i] = new A(i);
				}
				else {
					arrayB[i] = new B(i);
				}
				i++;
			} while(i < valores.length);
		}
}
```

a.1)
Antes del bucle:

- **Inicialmente tenemos un objeto *array* de tipos primitivos enteros con 4 valores.**
- **Se instancia un objeto *array* de referenciaas objetos de tipo `A`. El *array* permite almacenar 4 referencias de tipo `A`. Inicialmente todas esas posiciones tienen valor por defecto `null`.**
- **Se instancia un objeto array de referencias a objetos de tipo `B`. El *array* permite almacenar 4 referencias de tipo `B`. Inicialmente todas esas posiciones tienen valor por defecto `null`.**

En el bucle:

- **Se generan 2 objetos de tipo `A` e indirectamente 2 objetos de tipo `B`, que a su vez generan 2 objetos de tipo `C`, que se colocan en las posiciones pares del array `arrayA`.**
- **Se generan 2 objetos de tipo `B` e indirectamente 2 objetos de tipo `C` que se colocan en las posiciones impares del `arrayB`.**
- **El resto de posiciones del array tienen valores `null` como hemos indicado previamente.**
- **Así que finalmente tenemos 2 objetos `A`, 4 objetos `B` y 4 objetos `C` más los arrays iniciales.**



a.2) Sólo es necesario indicar los cambios. Por claridad se muestra el código completo modificado.

```
public class B {
        private int j;
        private C c;
        private static int contador;            // modificación en el ejercicio

        public B(int j) {
                this.j = j;
                c = new C();
                contador++;                     // modificación en el ejercicio
        }

        public int obtener() {
                return j * c.obtener();
        }

        /// modificación en el ejercicio nuevo método
        public static int obteneNumeroInstancias() {
                return contador;
        }
}
```

Adicionalmente se puede llevar un atributo de instancia `id` para cada objeto al que se asigna valor en el constructor a partir del valor estático de contador (si se añade, se considera también correcta la solución, aunque no es obligatorio).

3.  **Se quiere dar una implementación del juego denominado *Simon*. El jugador debe recordar una secuencia ordenada de colores y repetir dicha secuencia correctamente (no tenemos en cuenta el tiempo que se tarda en pulsar). Construir la siguiente enumeración y clase en Java pertenecientes al paquete** `mepro.juego`**: (0.90 Ptos)**

- **Enumeración** `Color` **que contiene valores** `VERDE`, `ROJO`, `AZUL` **y** `AMARILLO`. **(0.10 Ptos)**
- **Clase** `Simon` **con constructor y métodos que permitan: (0.80 ptos)**
  - **Constructor público que recibe un *array* con la secuencia ordenada de colores iniciales a repetir por parte del jugador. Es obligatorio utilizar un *array* unidimensional de Java, NO se permite el uso de `ArrayList`.**
  - **Método** `public void pulsarColor(Color color)`**: permite introducir cada color en orden (de uno en uno, según se invoca cada vez al método).**
    - **Si el color introducido es incorrecto respecto al orden de la secuencia inicial, las siguientes pulsaciones se ignoran, puesto que se ha perdido la partida.**
    - **Si se ha acertado ya la secuencia completa, también se ignoran siguientes pulsaciones, puesto que se ha ganado la partida.**
  - **Método** `public boolean esCorrectaUltimaPulsacion()`**: consulta si la última pulsación ha sido correcta en la secuencia o no.**
  - **Método** `public boolean hasGanado()` **: consulta si la secuencia de pulsaciones ha sido correcta respecto a la secuencia inicial completa.**
  - **Método** `public void reiniciar(Color[] nuevaSecuenciaDeColores)`**: reinicia el juego con un nueva secuencia ordenada de colores a adivinar por el jugador.**

```java
package mepro.juego;

public enum Color {
	ROJO, VERDE, AZUL, AMARILLO;
}
```

```java
package mepro.juego;

public class Simon {

	private Color[] secuenciaColores;

	private int posicionActual;

	private boolean ultimaPulsacion;

	public Simon(Color[] secuenciaColores) {
		reiniciar(secuenciaColores);
	}

	public void pulsarColor(Color color) {
		if (esCorrectaUltimaPulsacion() && !hasGanado()) {
			boolean acertado = secuenciaColores[posicionActual] == color;
			if (!acertado) {
				ultimaPulsacion = false;
			} else {
				posicionActual++;
			}
		}
	}

	public boolean esCorrectaUltimaPulsacion() {
		return ultimaPulsacion;
	}

	public boolean hasGanado() {
		if (secuenciaColores.length == 0) return true; // caso especial
		return posicionActual == secuenciaColores.length;
	}

	public void reiniciar(Color[] secuenciaColores) {
		this.secuenciaColores = secuenciaColores;
		posicionActual = 0;
		ultimaPulsacion = true;
	}
}
```

Algunos alumnos han planteado la variante de utilizar un array recopilando los valores introducidos y comparando con el array incial. Es otra solución igualmente válida.