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

METODOLOGÍA DE LA PROGRAMACIÓN **3-FEBRERO-2022 – 2ª 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:**

```java
package mepro.exam1;

public class A {

	private B b;
	private C c;

	public A(){
		b = new B();
	}

	public static C generar(){
		return new C();
	}
}
```

```java
package mepro.exam1;

public class B {

	static int valor;
	private C c;

	public B() {
		c = new C();
	}

	int obtener() {
		return c.obtener() + valor;
	}
}
```

```java
package mepro.exam2;

public class C {

	static int valor;
	private int k;

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

```java
package mepro.exam2;

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

a) Indicar <u>las líneas imprescindibles a añadir en todas las clases</u>, para que se produzca el ensamblaje correcto del sistema, SIN modificar las declaraciones de paquetes actuales. (0.10 ptos)

b) Realizadas las modificaciones previas, y suponiendo que a continuación de la línea 3 del método `main`, añadimos las siguientes líneas:

```
a1.b = null;                    // linea 4
b1.obtener() = 10;              // linea 5
System.out.println(C.valor);    // linea 6
C.valor = b1.obtener()          // linea 7
c1.k = B.valor;                 // linea 8
```

Indicar si son o no correctas <u>en compilación</u>, <u>explicando siempre brevemente el motivo</u>. (0.20 ptos)

c) Si las anteriores clases se compilan, dejando los binarios resultantes en el directorio `./bin` y además necesitamos para una correcta compilación y ejecución, algunos paquetes y clases contenidos en las bibliotecas `./lib/JColor-5.0.0.jar` y `./lib/kamisado-gui-3.0.0.jar` indicar: (0.40 ptos)

c.1) ¿Cómo ejecutamos en línea de comandos la clase principal `P`, desde el directorio actual, configurando ademas correctamente la opción `-cp` o `-classpath`?

c.2) ¿Qué estructura completa de directorios y ficheros deberíamos tener en el directorio raíz de nuestro proyecto, conteniendo <u>todos los ficheros necesarios</u> mencionados previamente?

Nota: suponemos que el sistema operativo es GNU/Linux o Mac, pero se puede dar la solución para Windows.

a) Se solicita añadir TODAS las importaciones en TODAS las clases.

En la clase A: `import mepro.exam2.C;`

En la clase B: `import mepro.exam2.C;`

En la clase P:

```
import mepro.exam1.A;
import mepro.exam1.B;
```

b)

```
a1.b = null;                    //4 acceso INCORRECTO a atributo privado b no accesible desde otro paquete
b1.obtener() = 10;              //5 INCORRECTA asignación a un método que retorna valor y además amigable
System.out.println(C.valor);    //6 CORRECTO acceso a atributo estático accesible
C.valor = b1.obtener()          //7 acceso INCORRECTO a método amigable en B
c1.k = B.valor;                 //8 acceso INCORRECTO a atributo estático amigable aunque los tipos coincidan y
                                   k es atributo privado
```

c.1)

```
java -cp ./lib/*:./bin mepro.exam2.P
```

o

```
java -cp ./lib/JColor-5.0.0.jar:./lib/kamisado-gui-lib-3.0.0.jar:./bin mepro.exam2.P
```

o
cambiando -cp por -classpath

c2.)

```
./src
   |--- mepro
        |---exam1
        |    |---A.java
        |    |---B.java
        |---exam2
             |---C.java
             |---P.java

./bin
   |--- mepro
        |---exam1
        |    |---A.class
        |    |---B.class
        |---exam2
             |---C.class
             |---P.class
./lib
   |--- JColor-5.0.0.jar
   |--- kamisado-gui-lib-3.0.0.jar
```

2. **A partir del siguiente código que utiliza las clases `A`, `B` y `C` del Ejercicio 1:**

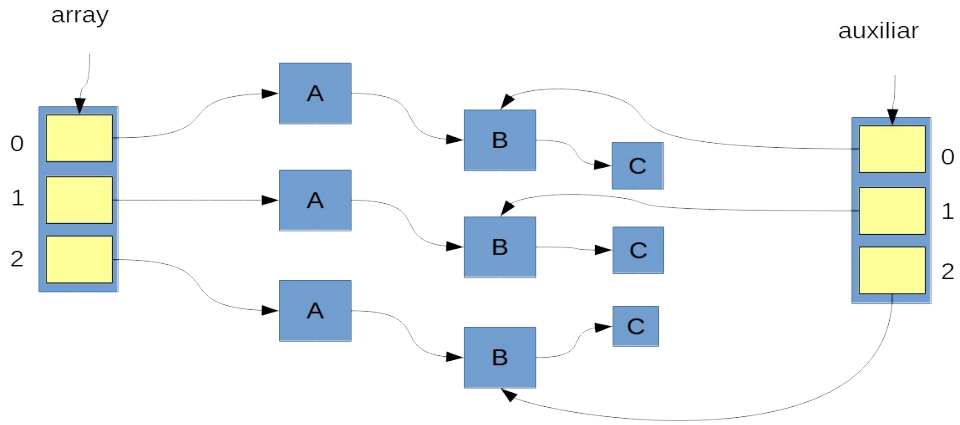
```
// se omiten la declaración de paquete e importaciones
public class Principal {

	public static void main(String[] args) {
		A[] array    = new A[3];
		B[] auxiliar = new B[3];
		for (int i = 0; i < array.length; i++) {
			array[i] = new A();
			auxiliar[i] = array[i].b;
		}
		// Línea FB1
		array[0] = null;
		array[1].b = null;
		auxiliar[1] = null;
		// Línea FA1
		A[] otroArray = auxiliar.clone();
	}
}
```

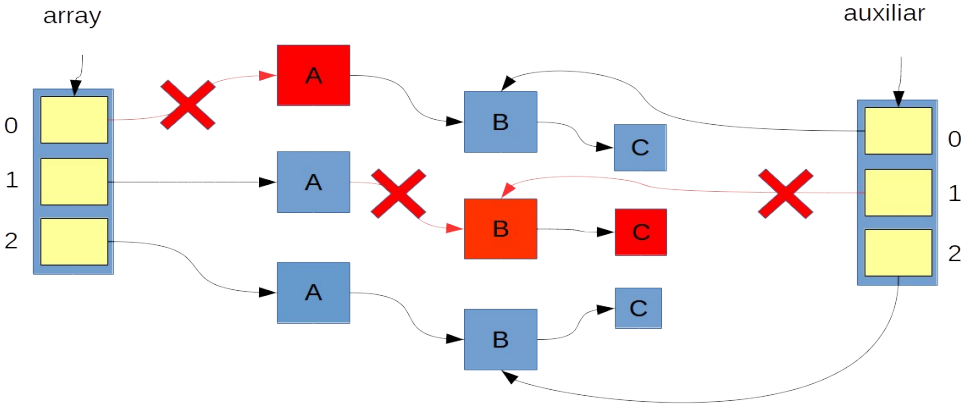
a.1) Explicar de forma razonada e incluyendo un dibujo, cuántos objetos, en qué orden y de qué tipo se han generado, justo al llegar al comentario `// Línea FB1`. (0.20 ptos)
a.2) Al llegar a la línea `// Línea FA1` indicar razonadamente, sobre el dibujo previo, qué objetos pasan a ser inalcanzables y el motivo. (0.20 ptos)
a.3) Suponiendo que el método `clone()`, ejecutado tras todo el código previo, realiza una clonación profunda del *array*, explicar cuántos objetos nuevos y de qué tipo se generan. (0.20 ptos)

[1]**a.1) Se crean dos objetos de tipo array (array y auxiliar) y se reserva la memoria para las referencias a sus elementos. En el bucle for se crean en 3 iteraciones objetos de tipo A que a su vez instancian objetos de tipo B y C. A la par se inicializa en auxiliar con los objetos B creados previamente.**

Por lo tanto tenemos 2 arrays, 3 objetos A, 3 objetos B y 3 objetos C, tal y como se muestra a continuación (se omiten las referencias nulas de objetos de tipo C que nunca se instancian, en objetos A).



a.2) Al llegar a la línea FA1 tenemos la siguiente situación en cuanto a los objetos:



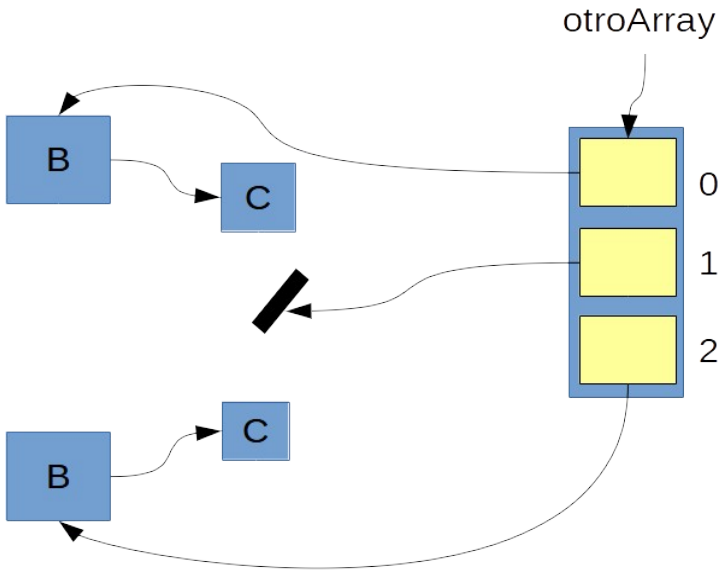
En este ámbito las referencia vivas de inicio, son las de los dos arrays (`array` y `auxiliar`), y los objetos alcanzables se calculan a partir de dichas referencias.

Las referencias con una cruz roja "desparecen" al asignarse valor `null`, y el objeto de tipo A apuntado por `array[0]` pasa ser inalcanzable.
Por otro lado los objetos B y C que eran apuntados por `array[1].b` y `auxiliar[1]` pasan a ser inalcanzables también.

El resto de objetos siguen siendo todos alcanzables.

1 Las explicaciones son más extensas de lo solicitado y no es necesario incluir todos los dibujos. Se incluyen para que la solución sea lo más explicativa posible.

a.3) Al clonar en profundidad *array* tendríamos un nuevo *array* donde en la posición 0 tendríamos un objeto B enlazado a un objeto C, en la posición 1 tendríamos un valor `null` y en la posición un objeto B enlazado a un objeto C. Se crean 2 objetos B y 2 objetos C.



3. **Se quiere dar una implementación de una <u>secuencia ordenada de colores. Por ejemplo:</u>**

Los colores están definidos en una enumeración `mepro.colores.Color` con valores `ROJO`, `AZUL`, `AMARILLO`, ya resuelta. Construir solo la siguiente clase en Java perteneciente al paquete `mepro.colores`:

- **Clase `Secuencia` con constructor y métodos públicos que permitan:** (0.70 ptos)
  - **Constructor público que recibe un *array* de objetos de tipo `Color`, y asigna cada uno de los colores en la posición correspondiente de la secuencia ordenada.**
  - **Método `public int contar(Color color)` que cuenta el número de apariciones del color pasado como argumento, dentro de la secuencia.**
  - **Método `public Secuencia invertir()` que devuelve una <u>nueva secuencia</u> con los colores en orden inverso, a la secuencia actual.**
  - **Método `public static void main(String[] args)` que instancie una secuencia de 6 colores (a libre elección del alumnado los colores y el orden), muestre en pantalla el número de apariciones de cada color, utilizando el método previo `contar`, invierta la secuencia y vuelva a mostrar el número de apariciones de cada color.**

Se pueden incluir métodos privados si se considera adecuado para su resolución, pero no es obligatorio.

```
public class Secuencia {

	private Color[] colores;

	public Secuencia(Color[] colores) {
		this.colores = new Color[colores.length];
		for (int i = 0; i < colores.length; i++) {
			this.colores[i] = colores[i];
		}
	}

	public Secuencia invertir() {
		Color[] nuevosColores = new Color[colores.length];
		for (int i = 0; i < colores.length; i++) {
			nuevosColores[i] = colores[colores.length-1-i];
		}
		return new Secuencia(nuevosColores);
	}


	public int contar(Color colorBuscado) {
		int contador = 0;
		for(Color colorActual : colores) {
			if (colorActual == colorBuscado) {
				contador++;
			}
		}
		return contador;
	}

	public static void main(String[] args) {
		Color[] colores  =
			{ Color.AZUL, Color.ROJO, Color.AMARILLO, Color.AZUL, Color.AZUL, Color.AMARILLO};
		Secuencia secuencia = new Secuencia(colores);
		for(Color color : Color.values()) {
			System.out.printf("Color: %s Número de apariciones: %d%n", color.toString(),
					secuencia.contar(color));
		}
		System.out.println("Invertimos la secuencia y volvemos a contar (deberían obtenerse los mismos
					valores...)");
		secuencia = secuencia.invertir();
		// dado que se repite el código se puede usar un método privado
		for(Color color : Color.values()) {
			System.out.printf("Color: %s Número de apariciones: %d%n", color.toString(),
					secuencia.contar(color));
		}
	}
}
```