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

METODOLOGÍA DE LA PROGRAMACIÓN **25-OCTUBRE-2024 – 1ª CONVOCATORIA**

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

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 static int cont;

	private R r;

	public A(){
		r = new R(A.cont);
		A.m();
	}

	public B generarB(E e) {
		return new B(r, e);
	}

	static void m() {
		cont++;
	}
}
```

```java
package mepro.exam2;

import mepro.exam4.R;

public class B {

	private E e;
	private R r;

	public B(R r, E e) {
		this.r = r;
		this.e = e;
	}

	public String aTexto() {
		return r.valor() + "->" + e.toString();
	}
}
```

```java
package mepro.exam3;

public class P {
	public static void main(String[] args) {
	    A a1 = new A();                    // línea 1
	    B b1 = new B(new R(0), E.VALOR_X);  // línea 2
	    R r1 = new R(0);                   // línea 3
	}
}
```

```java
package mepro.exam1.exam2;

public enum E {
	VALOR_X,
	VALOR_Y;
}
```

```java
public record R(int valor) {
}
```

a) Indicar solo las líneas imprescindibles a <u>añadir</u> en todas las clases, enumeraciones y registros, para que se produzca el ensamblaje correcto del sistema, SIN modificar las declaraciones de paquetes actuales, salvo en el tipo *record* `R`. (0.25 ptos)

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

```
A.cont = 10;                    // línea 4
System.out.println(A.m());      // línea 5
B b2 = a1.generarB(r1);         // línea 6
E e1 = b1.e;                    // línea 7
r1 = new R();                   // línea 8
```

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

c) Si los anteriores ficheros fuente se compilan en nuestro proyecto, dejando los binarios resultantes en el subdirectorio `.\bin` indicar: (0.15 ptos)

c.1) ¿Cómo ejecutamos en línea de comandos la clase principal `P` desde el directorio actual del proyecto, configurando correctamente la búsqueda de clases por parte de la Máquina Virtual Java?

c.2) ¿Cómo decompilamos los ficheros binarios correspondientes al tipo *record* `R` y al enumerado `E` desde el directorio actual del proyecto, configurando correctamente la búsqueda de clases por parte de la Máquina Virtual Java?

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

a) Clase A

```
import mepro.exam1.exam2.E;
import mepro.exam4.R;
import mepro.exam2.B;
```

Clase B

```
import mepro.exam1.exam2.E;
```

Clase P

```
import mepro.exam1.A;
import mepro.exam1.exam2.E;
import mepro.exam4.R;
import mepro.exam2.B;
```

Enumerado E (nada)

Registro R

```
package mepro.exam4;
```

b)

```
A.cont = 10;                    // acceso incorrecto a atributo estático privado
System.out.println(A.m());      // acceso incorrecto a método amigable y el método
además no retorna nada para imprimir
B b2 = a1.generarB(r1);         // incompatibilidad de tipos en argumento al método
E e1 = b1.e;                    // atributo no accesible
r1 = new R();                   // argumentos incorrectos al constructor
```

c)

c.1) `java -cp .\bin mepro.exam3.P`

c.2)

```
javap -cp .\bin mepro.exam4.R
javap -cp .\bin mepro.exam1.exam2.E
```

2. **A partir del siguiente código que utiliza las clases `A`, `B`, tipo enumerado `E` y tipo registro `R` del Ejercicio 1:**

```
// se omiten la declaración de paquete e importaciones, NO hay que añadirlas
public class Principal {

	public static void main(String[] args) {
		A[][] array = new A[3][2];
		// Línea FB0
		for (int fila = 0; fila < array.length; fila++) {
			for (int columna = 0; columna < array[fila].length; columna++) {
				if ((fila + columna) % 2 == 0) {
					array[fila][columna] = new A();
					E e = E.values()[fila % 2];
					B b = array[fila][columna].generarB(e);
					System.out.printf("Texto: %s %n", b.aTexto());
				}
			}
		}
		// Línea FB1
		array[0][0] = array[0][1];
		array[1][0] = array[1][1];
		array[2][0] = array[2][1];
		// Línea FB2
	}
}
```

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 FB0` y posteriormente al llegar a `// Línea FB1`. ¿Qué objetos pasan a ser inalcanzables al llegar a `// Línea FB1`? (0.45 ptos)

a.2) Al llegar a la línea `// Línea FB1` mostrar literalmente la salida a pantalla generada. (0.10 ptos)

a.3) Al llegar a la línea `// Línea FB2` y ANTES de finalizar la ejecución del método `main`, indicar razonadamente, sobre el dibujo previo, qué objetos pasan a ser inalcanzables y el motivo. ¿Se produce algún fenómeno de *aliasing* dinámico? (0.15 ptos)

a.1)[1] **Al llegar a la línea FB0 se tiene 1 *array* de dos dimensiones en Java que realmente contiene un *array* de 3 elementos de tipo *array* y otros 3 *arrays* de longitud 2 para referencias a objetos de tipo `A`. Inicialmente en esas seis posiciones tenemos valores nulos, puesto que solo se ha reservado memoria para el *array*.**

Al llegar a FB1, en dicho *array* las posiciones `[0][1]`, `[1][0]` y `[2][1]` tienen valor nulo. Las otras tres posiciones contienen objetos de tipo `A`. Cada objeto de tipo `A` tiene a su vez una referencia a un objeto `R`. Cada iteracion en la que se crea un objeto `A`, también se crea un nuevo objeto `B` que referencia al valor del tipo registro `R` accesible en esa iteración, junto con un valor de tipo enumerado `E.VALOR_X`, o `E.VALOR_Y` (alternando si estamos en iteración con índice par o impar) compartidos por todos ellos. Dado que solo hay una variable `b` de tipo `B`, en cada iteración la referencia cambia de valor apuntando al nuevo objeto `B` instanciado.

A nivel de clase (estático) tenemos:

- **el valor estático de tipo primitivo, `cont` en `A` que es compartido por todos los objetos.**
- **los dos valores del tipo enumerado `E` que generan dos objetos inmutables con `VALOR_X` y `VALOR_Y` respectivamente que no pueden ser modificados.**

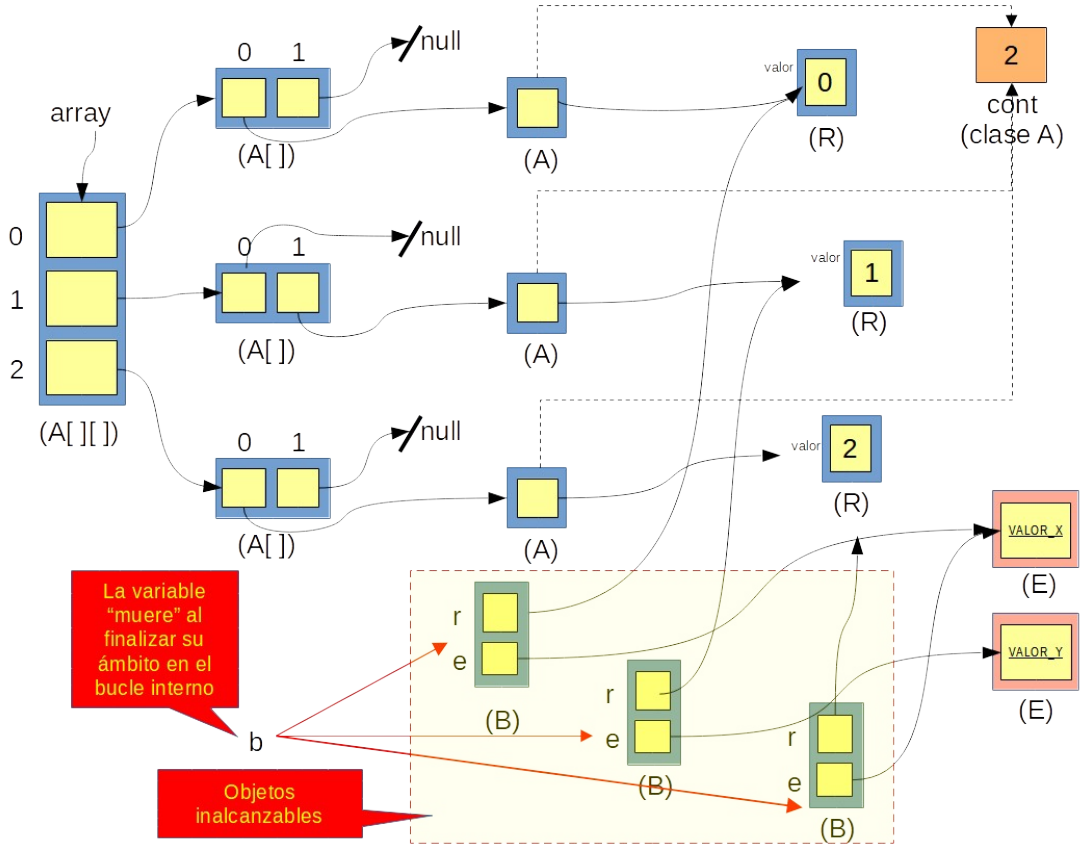
Por lo tanto, tenemos, y sin considerar el argumento `args`, del método `main`:

- **1 *array* con 3 referencias a *arrays* de una dimensión.**
- **3 *arrays* de una dimensión, con dos referencias a objetos `A` cada uno.**
- **3 objetos `A` creados en las distintas iteraciones.**
- **3 objetos `B`, aunque solo se utiliza una variable para asignar dichos objetos.**
- **3 objetos de tipo registro `R`.**
- **A nivel de clase tenemos un valor compartido entero, pero no es un objeto**
- **Dos objetos inmutables compartidos de tipo enumerado `E`.**

Al llegar a FB1 quedan inalcanzables todos los objetos `B` instanciados a través del método generar al no existir la variable en dicho ámbito.

Si dibujamos el conjunto de objetos en memoria, tendríamos algo similar a lo siguiente:

1 La respuesta se da mucho más detallada de lo que se pide por parte del alumnado, para que quede completamente documentado.



a.2)

```
Texto: 0->VALOR_X
Texto: 1->VALOR_Y
Texto: 2->VALOR_X
```

a.3)

El objeto de tipo `A` y el objeto `R` conectado a él, en `[0][0]` queda inalcanzable.

El objeto de tipo `A` y el objeto `R` conectado a él, en `[2][0]` queda inalcanzable.

El objeto de tipo `A` es alcanzable con la referencia en `[1][0]` y `[1][1]` produciéndose un fenómeno de *aliasing* dinámico.

3. **Se quiere implementar una clase simulando el funcionamiento de un oxímetro que almacena solo las últimas 5 mediciones de saturación de oxígeno en sangre. Las mediciones son valores enteros representando porcentajes (e.g. 90%, 100%, etc.). Se debe construir la clase `Oximetro` en Java, perteneciente al paquete `mepro`, que proporcione: (0.70 ptos)**

- **Constructor que inicializa el dispositivo con las 5 mediciones iniciales a valor cero.**
- **Método de consulta con signatura `boolean` `verificarUltimaMedicionSaludable()` que retornará `true` o `false` si la última medición está en el rango [95, 100]. Si no ha habido mediciones devolverá `false`.**
- **Método para añadir una nueva medición con signatura `void` `establecerMedicion(int porcentaje)`. El método añade la nueva medición. Si ya se han realizado 5 mediciones previas, se van sustituyendo las mediciones más antiguas por las nuevas mediciones. Si el valor no estuviese en el rango entre [0, 100], se acota por el valor válido más cercano.**
- **Método de consulta con signatura `int` `obtenerMedia()` que retorna la media aritmética de las mediciones, teniendo en cuenta solo aquellas que sean diferentes de cero.**

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

```java
package mepro;

public class Oximetro {

	private static final int NUMERO_MEDICIONES = 5;

	private int[] niveles;

	private int indice = -1;

	public Oximetro() {
		niveles = new int[NUMERO_MEDICIONES];
	}

	public boolean verificarUltimaMedicionSaludable() {
		if (indice >= 0) {
			return niveles[indice] >= 95 && niveles[indice] <= 100;
		}
		return false;
	}

	public void establecerMedicion(int nivel) {
		if (nivel > 100) {
			nivel = 100;
		}
		else if (nivel < 0) {
			nivel = 0;
		}
		indice++;
		if (indice == niveles.length) {
			// Si nos salimos a un extremo empezamos otra vez
			indice = 0;
		}
		niveles[indice] = nivel;

	}

	public int obtenerMedia() {
		int acumulador = 0;
		int medidas = 0;
		for(int i = 0; i < niveles.length; i++) {
			if (niveles[i] != 0) {
				acumulador += niveles[i];
				medidas++;
			}
		}
		if (medidas != 0) {
			return acumulador/medidas;
		}
		return 0;
	}
}
```