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

METODOLOGÍA DE LA PROGRAMACIÓN **20-ENERO-2025 – 2ª 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(){
		A.m();
		r = new R(A.cont);
	}

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

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

```java
package mepro.exam1.exam2;

import mepro.exam1.exam3.E;

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.exam1.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
public enum E {
	VALOR_X,
	VALOR_Y;
}
```

```java
package mepro.exam1.exam2;

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 enumerado `E`. (0.20 ptos)

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

```
System.out.println(A.m());          // línea 4
A.cont++;                           // línea 5
B b2 = a1.generarB(E.VALOR_X);      // línea 6
R r2 = new R(r1);                   // línea 7
E e1 = new E(E.VALOR_Y);            // línea 8
```

Indicar si son o no correctas <u>en compilación</u>, <u>explicando siempre brevemente el motivo</u>. (0.15 ptos)

c) Si los anteriores ficheros fuente se compilan en nuestro proyecto, dejando los binarios resultantes en el subdirectorio `.\bin` indicar: (0.25 ptos) **Nota: suponemos que el sistema operativo es GNU/Linux o Mac, pero se puede dar la solución para Windows.**

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) Dibujar la estructura correspondiente de directorios/subdirectorios y ficheros con su extensión en el director `bin`, como resultado de una compilación correcta.

c.3) ¿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?

a) Clase A

```
import mepro.exam1.exam2.B;
import mepro.exam1.exam2.R;
import mepro.exam1.exam3.E;
```

Clase B (nada)

Clase P

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

Enumerado E (nada)

```
package mepro.exam1.exam3;
```

Registro R (nada)

b)

```
System.out.println(A.m());    // línea 4 acceso a método amigable INCORRECTO o bien método no retorna nada.
A.cont++;                     // línea 5 acceso INCORRECTO a atributo estático privado
B b2 = a1.generarB(E.VALOR_X);// línea 6 correcta invocación
r1 = new R(0);                // línea 7 INCORRECTA instanciación de un record con otro record
E e1 = new E(E.VALOR_Y);      // línea 8 INCORRECTA instanciación de un valor tipo enumerado
```

c)

c.1)

```
java -cp .\bin mepro.exam1.exam3.P
```

c.2)

```
./bin
   |--- mepro
          |
          |---exam1
              |
              |---exam2
              |       |---B.class
              |       |---R.class
              |
              |---exam3
              |       |---E.class
              |       |---P.class
              |
              |---A.class
```

c.3)

```
javap -cp .\bin mepro.exam1.exam2.R
javap -cp .\bin mepro.exam1.exam3.E
```

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

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[2][3];
		// 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()[columna % 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[0][1] = array[1][1];
		array[1][0] = array[1][2];
		array[0][0] = null;
		// 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`? (0.35 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 durante la ejecución de esas líneas de código? (0.25 ptos)

a.1)[1] Al llegar a la línea FB0 se tiene 1 array de cadenas vacío, 1 *array* de dos dimensiones en Java que realmente contiene un *array* de 2 elementos de tipo *array* y otros 2 *arrays* de longitud 3 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][0]`, `[0][2]` y `[1][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` en la primera iteración y `E.VALOR_Y` en las dos últimas iteraciones 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.**

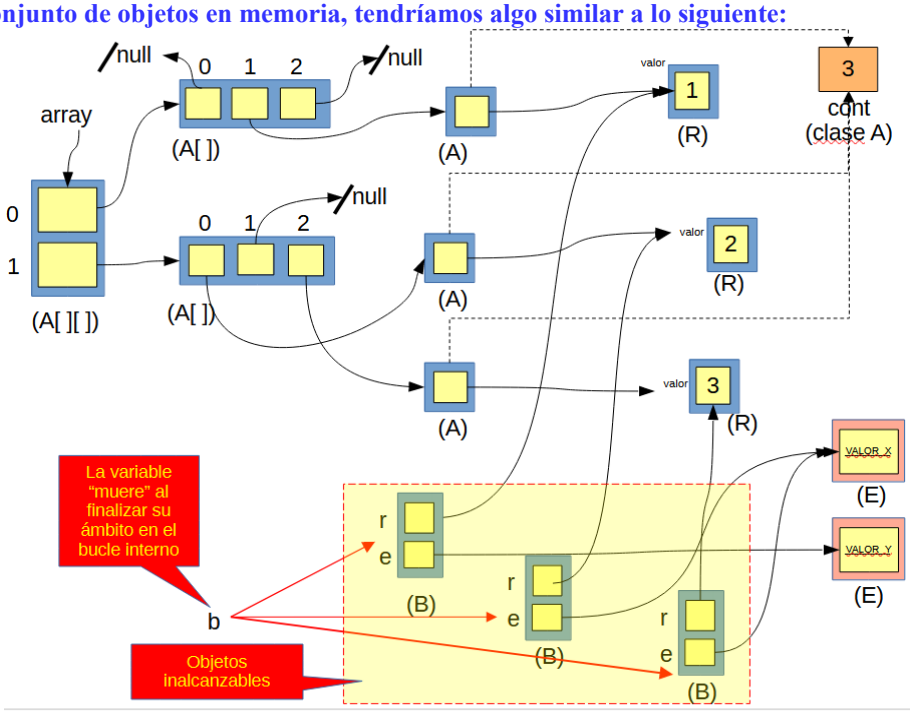
Por lo tanto, tenemos, y sin considerar el argumento `args`, del método `main`:
- **1 *array* con 2 referencias a *arrays* de una dimensión.**
- **2 *arrays* de una dimensión, con tres referencias a objetos `A` cada uno.**
- **3 objetos `A` creados en las distintas iteraciones.**

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

- 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, puesto que las variables no existen en dicho ámbito.

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



a.2)

```
Texto: 1->VALOR_Y
Texto: 2->VALOR_X
Texto: 3->VALOR_X
```

a.3)

Al ejecutarse a partir de Línea FB1…

Inicialmente `[0][0]` apunta al mismo objeto que `[0][1]` produciéndose un breve instante de tiempo un *aliasing* dinámico al apuntar ambas referencias al mismo objeto.

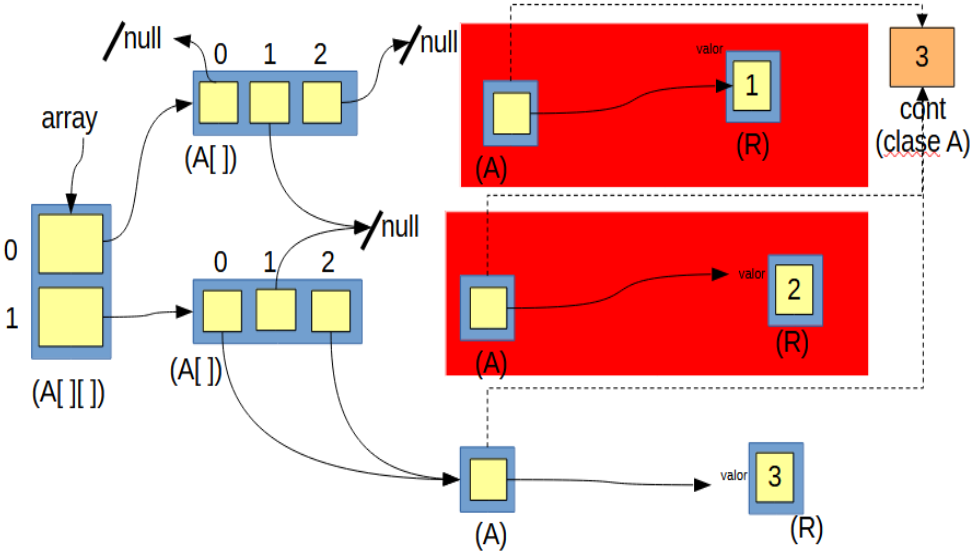
A continuación `[0][1]` pasa a apuntar a `[1][1]` que vale `null`, por lo tanto se deshace el *aliasing*.

A continuación, `[1][0]` apunta a `[1][2]`. En ese momento el objeto de tipo `A` y el objeto `R` conectado a él previamente, en `[1][0]` queda inalcanzable. Y se produce un aliasing entre ambas referencias.

Finalmente, `[0][0]` pasa a valer nulo, dejando inalcanzable al objeto `A` y objeto `R` conectado a él (que inicialmente estaba conectado en `[0][1]`).

La foto final quedaría de la siguiente forma, resaltando sobre fondo rojo los objetos inalcanzables

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



3. **Se quiere implementar una clase simulando el funcionamiento de un podómetro que almacena solo las últimas 30 mediciones del número de pasos andados cada día. Las mediciones son valores enteros (e.g. 5320, 9079, etc.) Se debe construir la clase `Podometro` en Java, perteneciente al paquete `mepro`, que proporcione:** **(0.70 ptos)**

- **Constructor que inicializa el dispositivo con sus 30 mediciones iniciales a cero.**
- **Método para añadir una nueva medición con signatura `void` `añadirPasos(int numeroPasos)`. Si el valor no es positivo se sustituye por un cero. Si ya se han añadido 30 mediciones previas, se va sustituyendo la medición más antigua, por la nueva medición conservando simpre las 30 últimas mediciones más recientes.**
- **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.**
- **Método de consulta con signatura `boolean` `verificarEstadoSaludable()` que retornará `true` o `false` si la media aritmética de los valores distintos de cero, almacenados hasta el momento, es mayor de 8000 pasos.**
- **Método de reinicio con signatura `void` `reiniciar()` que reinicia el podómetro a su estado de fábrica, con todos sus valores a cero.**

Debe resolverse con *arrays* de Java, NO pudiéndose utilizar clases del paquete `java.util`.

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

```java
package mepro;
public class Podometro {

    private static final int NUM_DIAS = 30;
    private static final int RECOMENDACION = 8000;
    private int[] pasos; // Arreglo para almacenar los pasos de los últimos NUM_DIAS
    private int indice;  // Índice que indica la posición actual para sobrescribir
    private int contadorDias; // Contador para saber cuántos días se han registrado

    public Podometro() {
        pasos = new int[NUM_DIAS]; // Inicializamos un array de tamaño fijo para 30 días
    }

    // Método para registrar pasos
    public void añadirPasos(int numeroPasos) {
        if (numeroPasos < 0) {
            numeroPasos = 0;
        }
        this.pasos[indice] = numeroPasos;
        indice = (indice + 1) % NUM_DIAS;
        if (contadorDias < NUM_DIAS) {
            contadorDias++;
        }
    }

    // Método para calcular el promedio de pasos (ignorando días con 0 pasos)
    public double obtenerMedia() {
        int suma = 0;
        int diasValidos = 0;
        for (int i = 0; i < contadorDias; i++) {
            if (pasos[i] > 0) {
                suma += pasos[i];
                diasValidos++;
            }
        }
        return (diasValidos == 0) ? 0 : (double) suma / diasValidos;
    }

    // Método para verificar si la media es saludable
    public boolean verificarEstadoSaludable() {
       return this.obtenerMedia() >= RECOMENDACION;
    }


    // Método para reiniciar el podómetro
    public void reiniciar() {
        for (int i = 0; i < pasos.length; i++) {
            pasos[i] = 0; // Reiniciar todos los registros a 0
        }
        indice = 0;
        contadorDias = 0;
     }
}
```