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

METODOLOGÍA DE LA PROGRAMACIÓN **28-OCTUBRE-2022 – 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:

```java
package mepro.exam1;

public class A {

	private B b;
	private C c;

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

	void establecer(){
		c = C.VALOR_Y;
	}
}
```

```java
package mepro.exam2;

public class B {

	static int valor;
	private C c;

	public B() {
		c = C.VALOR_X;
		valor++;
	}

	public String obtener() {
		return c.toString() + valor;
	}
}
```

```java
package mepro.exam1.exam3;

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

```java
package mepro.exam3;

public class P {
	public static void main(String[] args) {

		A a1 = new A(); // línea 1
		B b1 = new B(); // línea 2
	}
}
```

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, del código actual. (0.20 ptos)

b) Realizadas las modificaciones previas, y suponiendo que a continuación de la línea 2 del método `main`, añadimos las siguientes líneas:

```
C c = new C();                              // línea 3
a1.b = null;                          // línea 4
B.valor = 10;                               // línea 5
a1.establecer();                            // línea 6
String texto = b1.obtener(C.VALOR_X);       // línea 7
```

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/junit-5.0.0.jar` y `./lib/quantik-gui-1.0.0.jar` indicar: (0.30 ptos)

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

c.2) ¿Qué estructura <u>completa</u> de directorios y ficheros deberíamos tener en el directorio raíz de nuestro proyecto, mencionando <u>todos los ficheros necesarios</u> 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.B;
import mepro.exam1.exam3.C;
```

En la clase B:

```
import mepro.exam1.exam3.C;
```

En la clase P:

```
import mepro.exam1.A;
import mepro.exam2.B;
// import mepro.exam1.exam3.C; // No se pide, solo añadir si se tiene en cuenta el aparta-
do b)
```

b)

```
C c1 = new C();       // no importa C y además no se pueden instanciar tipos enumerados...
a1.b = null;          // acceso INCORRECTO a atributo privado
B.valor = 10;         // acceso INCORRECTO a atributo amigable
a1.establecer();      // acceso INCORRECTO a método amigable desde otro paquete
String texto = b1.obtener(C.VALOR_X); // no importa C y acceso a método público no se puede pasar
                                         argumento porque el método no lo tiene definido
```

c.1)

```
java -cp ./lib/junit-5.0.0.jar:./lib/quantik-gui-1.0.0.jar:./bin mepro.examen3.P
```

c.2)

```
./src
   |--- mepro
          |
         |---exam1
          |    |---A.java
          |      |---exam3
          |      |---C.java
          |
         |---exam2
          |    |---B.java
          |
          |---exam3
              |---P.java

./bin
   |--- mepro
          |
         |---exam1
          |    |---A.class
          |      |---exam3
          |      |---C.class
          |
         |---exam2
          |    |---B.class
          |
          |---exam3
              |---P.class

./lib
   |--- junit-5.0.0.jar
   |--- quantik-gui-lib-1.0.0.jar
```


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` y enumeración `C`, del Ejercicio 1:

```
package mepro.exam2;
// se omiten las importaciones de otras clases y enumeraciones…
public class Principal {
    public static void main(String[] args) {
        A[][] array        = new A[3][2];
        for (int fila = 0; fila < array.length; fila++) {
            for (int columna = 0; columna < array[fila].length; columna++) {
                if (fila + columna > 1) {
                    array[fila][columna] = new A();
                    System.out.println(B.valor);
                }
            }
        }
         // Línea FB1
        array[1][1] = new A();
        array[2][1] = null;
         // Línea FA1
   }
}
```

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`. (0.30 ptos)
a.2) Mostrar y explicar brevemente la salida a pantalla generada al llegar al comentario `// Línea FB1`. (0.10 ptos)
a.3) Al llegar a la línea `// Línea FA1` indicar razonadamente, sobre el dibujo previo, qué objetos pasan a ser inalcanzables y el motivo. (0.30 ptos)

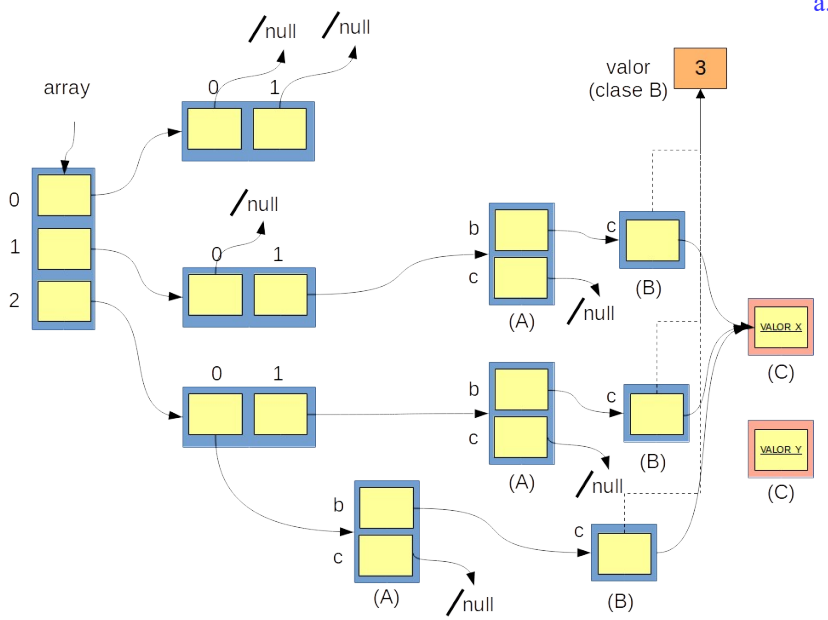
a.1)
1 *array* de dos dimensiones en Java que realmente contiene un *array* de 3 elementos de tipo *array* y otros tres *array* de longitud 2 para referencias a objetos de tipo `A`.

En dicho *array* las posiciones `[0][0]`, `[0][1]` y `[1][0]` tienen valores nulos. Las otras tres posiciones contienen objetos de tipo `A`. Cada objeto de tipo `A` tiene a su vez una referencia a un objeto `B` y una referencia `null` a un tipo `C` que se inicializa con valor por defecto a `null`.

A nivel de clase (estático) tenemos:
- **el valor estático de B que es compartido por todos los objetos**
- **los dos valores del tipo enumerado C que generan dos objetos inmutables con VALOR_X y VALOR_Y respectivamente que no pueden ser modificados.**

Por lo tanto, tenemos:
- **1 *array* con tres referencias a *arrays* de una dimensión**
- **3 *arrays* de una dimensión, con dos referencias a objetos A cada uno**
- **3 objetos `A`**
- **3 objetos `B`**
- **2 objetos de tipo `C`**
- **A nivel de clase tenemos un valor compartido pero no es un objeto**

a.2)

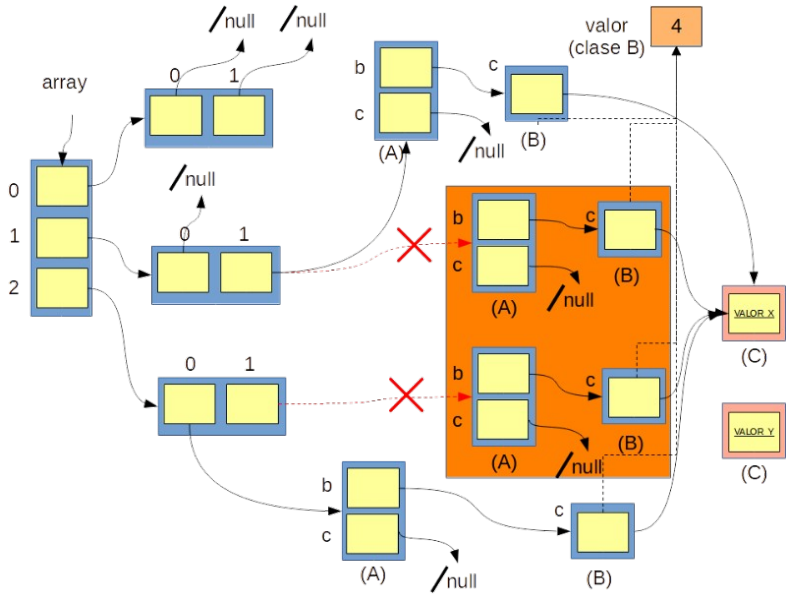


Salida a pantalla:
1
2
3

Explicación: al iterar en los casos que `fila + columna > 1` se intancia un objeto `A` que a su vez provoca la invocación al constructor de la clase `B`, donde se incrementa el valor compartido valor. Como esto ocurre tres veces, se mostraría el incremento de dicho valor en pantalla tres veces.

a.3)
Los objetos a los que se apuntaba desde las posiciones `[1][1]` y `[2][1]` quedan desconectados pasando a ser inalcanzables esos 2 objetos de tipo A. Indirectamente los dos objetos B a los que apuntaban respectivamente, también pasan a ser inalcanzables por lo que tendríamos:

- **2 objetos `A` inalcanzables**
- **2 objetos `B` inalcanzables**



3. En la escritura *Leet* se cambian algunas letras por números.
En concreto el siguiente *array:* `{'O', 'I', 'E', 'A'}` se cambiaría respectivamente, uno a uno, por `{'0', '1', '3', '4'}`.
Se pide implementar la clase *Leet* en el paquete `mepro.examen`, aplicando solo las cuatro conversiones previas, que contenga: (0.70 ptos)

- **Constructor público que recibe un *array* de caracteres con el texto inicial. Supondremos que siempre en mayúsculas, para simplificar el ejercicio, y sin contener números.**
- **Método `public int contarSinCambio()` que cuenta el número de caracteres que no se cambiarían por números en este *leet*. Sí contabilizan los espacios en blanco, tabuladores o saltos de carro.**
- **Método `public int contarCambios()` que devuelve el número de caracteres que se cambian a número, como resultado de las reglas de transformación dadas sobre el *array* inicial.**
- **Método `public String toString()` devuelve el texto correspondiente al *leet* con las transformaciones indicadas previamente.**

Ejemplo de uso en un método `main` de la clase `Leet`:

```
char[] texto = { 'E', 'N', ' ', 'M', 'I', 'L', ' ', 'A', 'Ñ', 'O', 'S' };
Leet l1 = new Leet(texto);
System.out.println("Texto leet: " + l1.toString());           // Texto leet: 3N M1L 4Ñ0S
System.out.println("Nº sin cambio:" + l1.contarSinCambio());  // N.º sin cambio: 7
System.out.println("Nº cambios:" + l1.contarCambios());       // N.º cambios: 4
```

Es decisión del alumnado el uso de atributos y métodos adicionales.
Nota: no se corrigen preguntas con tachones o a lápiz.

```java
// Primera solución sin usar arrays, transformando los datos en el constructor.

package mepro.examen;

public class Leet {

	private char[] elementos;

	public Leet(char[] caracteres) {
		elementos = new char[caracteres.length];
		for (int i = 0; i < elementos.length; i++) {
			elementos[i] = traducir(caracteres[i]);
		}
	}

	private char traducir(char caracter) {
		switch (caracter) {
		case 'O':
			return '0';
		case 'I':
			return '1';
		case 'E':
			return '3';
		case 'A':
			return '4';
		return caracter;
	}

	public String toString() {
		String resultado = "";
		for (int i = 0; i < elementos.length; i++) {
			resultado += elementos[i];
		}
		return resultado;
	}

	public int contarSinCambio() {
		return this.elementos.length - contarCambios();
	}

	private boolean esNumero(char caracter) {
		return  (caracter == '0' || caracter == '1' || caracter=='3' || caracter=='4');
	}

	public int contarCambios() {
		int contador = 0;
		for (int i = 0; i < elementos.length; i++) {
			if (esNumero(elementos[i])) {
				contador++;
			}
		}
		return contador;
	}
}
```


```java
// Otra solución, sin transformar los datos en el constructor y usando arrays con constantes

package mepro.examen;

public class Leet {

	private char[] elementos;

	private static final char[] LETRAS = { 'A', 'E', 'I', 'O' };

	private static final char[] CAMBIOS = { '4', '3', '1', '0' };

	public Leet(char[] caracteres) {
		elementos = new char[caracteres.length];
		for (int i = 0; i < elementos.length; i++) {
			elementos[i] = caracteres[i];
		}
	}

	// con un if...
	private char traducir(char caracter) {
		int posicion = 0;
		for (char c : LETRAS) {
			if (caracter == c) {
				return CAMBIOS[posicion];
			}
			posicion++;
		}
		return caracter;
	}

	public String toString() {
		String resultado = "";
		for (int i = 0; i < elementos.length; i++) {
			resultado += traducir(elementos[i]);
		}
		return resultado;
	}

	public int contarSinCambio() {
		return this.elementos.length - contarCambios();
	}

	// retornando directamente la evaluación
	private boolean esCandidatoACambiar(char caracter) {
		for (char c : LETRAS) {
			if (c == caracter) {
				return true;
			}
		}
		return false;
	}

	public int contarCambios() {
		int contador = 0;
		for (int i = 0; i < elementos.length; i++) {
			if (esCandidatoACambiar(elementos[i])) {
				contador++;
			}
		}
		return contador;
	}
}
```

```java
// Otra solución realizando la transformación y cálculo en el constructor con arrays
// aprovechando que el objeto es inmutable y no hay métodos que cambien su estado

package mepro.examen;

public class Leet {

	private static final char[] LETRAS = { 'A', 'E', 'I', 'O' };

	private static final char[] CAMBIOS = { '4', '3', '1', '0' };

	private char[] elementos;

	private int contadorCambios;

	public Leet(char[] caracteres) {
		elementos = new char[caracteres.length];
		for (int i = 0; i < elementos.length; i++) {
			boolean encontrado = false;
			for (int j = 0; j < LETRAS.length && !encontrado; j++) {
				if (caracteres[i] == LETRAS[j]) {
					elementos[i] = CAMBIOS[j];
					contadorCambios++;
					encontrado = true;
				}
			}
			if (!encontrado) {
				elementos[i] = caracteres[i];
			}
		}
	}

	public String toString() {
		String texto = "";
		for (char c : elementos) {
			texto += c;
		}
		return texto;
	}

	public int contarSinCambio() {
		return this.elementos.length - contarCambios();
	}

	public int contarCambios() {
		return contadorCambios;
	}
}
```