Apellidos:

Nombre: D.N.I.:

Introducción a la Programación - E.P. de Ingeniería de Gijón

16 de Enero de 2020

1. (1 p) Declara, con una sola instrucción, un vector de cadenas de caracteres cuyos valores iniciales han de ser "Hola", "ke", "ase".

Solución:

```
String[] v = {"Hola", "ke", "ase"};
```

2. (2 p) Dada la siguiente definición de variables y sus valores iniciales,

```
int x=9, y=2; double f=2.0; char c='a';
```

indica para las siguientes expresiones, si son o no correctas (SI/NO), en caso de resultar incorrectas JUSTIFICA por qué lo son, y en el caso de ser correctas indica el TIPO y el VALOR que producen.

Nota: Los dígitos, al igual que las letras del alfabeto, ocupan posiciones consecutivas en la tabla de códigos.

		Tipo	Valor	Motivo
`y ** 3`	☐Si ☒No			No existe el operador ** en Java
`c < d`	☐Si ☒No			No existe la variable d
`y = x + 1.0;`	☐Si ☒No			La suma resultante es de tipo double e pero y es int
`y != c`	☒Si ☐No	boolean	`true`	
`f + 1.0f`	☐Si ☒No			Falta un operador[1] entre 1.0 y f
`x/2.0 + 1`	☒Si ☐No	double	5.5	

3. (2 p) Dado un valor $N >= 1$, contesta a las siguientes preguntas

   (a) ¿Qué imprime el siguiente código?

```
for (int i = 1, a = 1; i <= N; i++, a += i) {
    System.out.printf("%d %d\n", i, a);
}
```

   (b) ¿Cuánto vale i justo después de salir del bucle?

   (c) ¿Cuánto vale N justo después de salir del bucle?

   (d) El código siguiente ¿produce la misma salida que el del apartado a? ¿por qué?

```
int i = 1, a = 1;
while (i <= N) {
    System.out.printf("%d %d\n", i, a);
    a = a + i;
    i = i + 1;
}
```

   (e) ¿Cuánto vale i justo después de salir del bucle?

Solución:

a) Imprime una columna con los enteros de 1 a N y al lado otra columna con la suma acumulada de los elementos anteriores.
1 1
2 3
3 6
4 10
...

b) No existe i fuera del bucle for

c) N no varía, sigue valiendo lo mismo que al principio

d) No, el bucle anterior incrementa i y luego a en función del nuevo valor de i, y este bucle incrementa primero a y luego i. Ejemplo
1 1
2 2
3 4
4 7
...

e) Al salir del bucle, i vale $N + 1$.

4. (3 p) Implementa un método estático que, dado un número entero N, calcule el resultado de la serie siguiente **sin utilizar `Math.pow()`**:

$$\sum_{i=0}^{N}(-1)^i\frac{1}{2^i} = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \frac{1}{16} - \cdots$$

[1] La intención de esta pregunta es la de detectar que falta un operador. No obstante, en este caso concreto, la expresión es correcta ya que 1.0f es interpretado por Java como que 1.0 es de tipo `float`, con lo que también se acepta dar la expresión por correcta e indicando que el tipo de la expresión sería `double` y el valor sería 3.0 (`(double) 2.0 + (float) 1.0`).

Solución:

```
public static double serie(int N) {
    double s = 0.0;
    int signo = +1, denominador = 1;
    for (int i = 0; i <= N; i++) {
        s += (double) signo/denominador;
        denominador *= 2;
        signo = -signo;
    }
    return s;
}
```

5. (2 p) Escribe en Java las condiciones equivalentes a las siguientes expresiones:

   La matriz M no es cuadrada y su número de filas es par

   Solución: `M[0].length != M.length && M.length%2 == 0`

   El número entero n tiene más de m dígitos

   Solución: `n/Math.pow(10,m) != 0`

   Los números reales x e y son aproximadamente iguales (con un margen de error de 1 milésima)

   Solución: `Math.abs(x - y) <= 0.001`

   Las variables enteras a, b, c y d contienen las longitudes de los lados de un rectángulo o romboide (lados iguales dos a dos) pero NO de un cuadrado ni de un rombo (cuatro lados iguales)

   Solución:
   ```
   (a == b && c == d && a != c)||(a == c && (b == d && a != b)||(a == d && b == c && a != b)
   ```
   o también
   ```
   ((a == b && c == d)||(a == c && b == d)||(a == d && b == c)) && !(a == b && b == c && c == d)
   ```

6. (3 p) Haz un programa completo (incluyendo las librerías necesarias) que pida una cantidad en € y muestre por pantalla el mínimo número de monedas necesarias para alcanzar dicho importe (recuerda que las monedas de curso legal son de 2€, 1€, 50c, 20c, 10c, 5c, 2c y 1c). Se recomienda encarecidamente trabajar con números enteros, pasando los euros a céntimos, para evitar errores de redondeo.

   Ejemplo:
   ```
   Importe en EUR: 11,47
   5 monedas de 2€
   1 monedas de 1€
   2 monedas de 20c
   1 monedas de 5c
   1 monedas de 2c
   ```

Solución:

```
import java.util.Scanner;
public class CambioMonedas {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        final int[] valores_moneda = {200, 100, 50, 20, 10, 5, 2, 1};

        System.out.print("Introduce el importe en EUR:");
        double importe = s.nextDouble();
        int importe_en_céntimos = (int) (importe * 100);
        int idx = 0;
        while (idx < valores_moneda.length) {
            int cantidad = importe_en_céntimos / valores_moneda[idx];
            if (cantidad > 0)
                System.out.printf("%d monedas de %.2f€\n", cantidad, valores_moneda[idx]/100.0);
            importe_en_céntimos %= valores_moneda[idx];
            idx++;
        }
    }
}
```

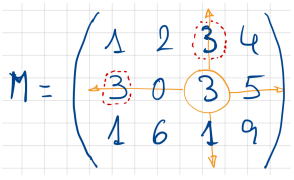
7. (3 p) Implementa el método **`primerPrimo()`** que recibe un vector de enteros como parámetro y retorna el índice más pequeño de la componente que contiene un número primo, o -1 si no hay ninguna. Se recomienda la implementación del método auxiliar **`esPrimo()`** que retorna cierto cuando se le pasa un número primo como parámetro y falso cuando éste no es primo.

Solución:

```
public static boolean esPrimo(int n) {
    int i=2;
    while (i <= n/2 && n%i != 0)
        i++;
    return n != 1 && i > n/2; // el 1 no es primo por convenio
}

public static int primerPrimo(int[] v) {
    int i = 0;
    while (i < v.length && !esPrimo(v[i]))
        i++;
    return (i < v.length ? i : -1);
}
```

8. (3 p) Si definimos como *área de influencia* de un elemento en una matriz a aquellos elementos que están por encima, por debajo, a su izquierda y a su derecha, se pide que implementes un método que, dado un determinado elemento de una matriz de enteros, retorne el número de elemento en su área de influencia que sean iguales a él. Ejemplo: para el elemento de la fila 1, columna 2 de la matriz M el método debería retornar el valor 2.

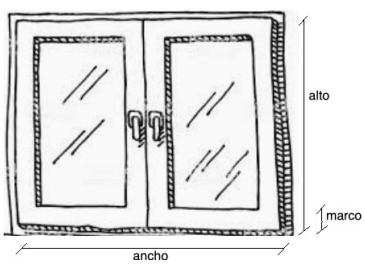


Solución:

```
public static int cuentaIguales(int[][] m, int f, int c) {
    // Se cuenta cuantos elementos hay en la fila f y en la columna c que sean iguales a m[f][c] y se resta 2
    int filas = m.length;
    int columnas = m[0].length;
    int valor = m[f][c];
    int contador = -2; // ya le restamos aquí el exceso
    for (int i = 0; i < filas; i++)
        if (m[i][c] == valor)
            contador++;
    for (int i = 0; i < columnas; i++)
        if (m[f][i] == valor)
            contador++;

    return contador;
}
```

9. (4 p) Se pide hacer la clase **`Ventana`** para representar ventanas de 2 hojas. La clase tiene tres atributos enteros con las dimensiones en cm de la ventana: ancho, alto, marco.



- Constructores: por defecto (con valores para cada atributo de 120, 100, 6), el de copia, con 2 enteros para inicializar ancho y alto (por defecto para el marco: 6 cm), con 3 enteros para poder inicializar los 3 atributos.
- Métodos `get()` y `set()`.
- Método `calculaSuperficieCristales()`, que devuelve el valor de la siguiente fórmula:
$$(ancho - 4 \cdot marco) \cdot (alto - 2 \cdot marco).$$
- Método `tieneMásCristal()` que recibe un objeto Ventana y devuelve cierto si el objeto con el que se llama al método tiene una superficie de cristal mayor que el objeto que se pasa como parámetro y falso en caso contrario.
- Método `toString()`, que devuelve un String con la información del objeto Ventana, por ejemplo, "120 cm. ancho x 100 cm. alto, Marco 6 cm.".

Solución:

```
public class Ventana {
    private int ancho = 120, alto = 100, marco = 6;

    public Ventana() {
        this(120, 100, 6);
    }

    public Ventana(Ventana w) {
        this(w.getAncho(), w.getAlto(), w.getMarco());
    }

    public Ventana(int ancho, int alto) {
        this(ancho, alto, 6);
    }

    public Ventana(int ancho, int alto, int marco) {
        setAncho(ancho);
        setAlto(alto);
        setMarco(marco);
    }

    public int getAlto() {
        return alto;
    }

    public int getAncho() {
        return ancho;
    }

    public int getMarco() {
        return marco;
    }

    public void setAlto(int alto) {
        if (alto > 0)
            this.alto = alto;
    }

    public void setAncho(int ancho) {
        if (ancho > 0)
            this.ancho = ancho;
    }

    public void setMarco(int marco) {
        if (marco > 0)
            this.marco = marco;
    }

    public int calculaSuperficieCristales() {
        return (this.getAncho() - 4 * this.getMarco()) * (this.getAlto() - 2 * this.getMarco());
    }

    public boolean tieneMásCristal(Ventana w) {
        return this.calculaSuperficieCristales() > w.calculaSuperficieCristales();
    }

    @Override
    public String toString() {
        return String.format("%d cm. ancho x %d cm. alto, Marco %d cm.",
                this.getAncho(), this.getAlto(), this.getMarco());
    }

}
```

10. (3 p) Implementa la clase **`PrismaUniforme`** para representar prismas rectos cuyas bases son polígonos regulares y cuyas caras son cuadrados. Un prisma recto requiere, por tanto dos atributos de tipo `PolígonoRegular`:

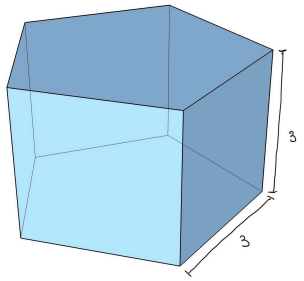
- `base`: representa las bases (como ambas son iguales, basta almacenar una)
- `cara`: representa las caras (como todas son iguales, basta almacenar una)

Deberás implementar el código necesario para que el siguiente programa produzca la salida que se muestra, excepto el código de la clase `PolígonoRegular`, que se presupone ya correctamente implementada. (Nota: Si p es un `PolígonoRegular` de 3 lados de longitud 1 unidad, la instrucción `System.out.println(p)` produce la salida: `Polígono regular de 3 lados de longitud 1,00 (Área: 0,43, Perímetro: 3,00)`)

PolígonoRegular
– n_lados: int – longitud: double
+ PolígonoRegular(int, double) + setNLados(int) + setLongitud(double) + getNLados(): int + getLongitud(): double + perímetro(): double + apotema(): double + área(): double + toString(): String

```
public class Ejemplo {
    public static void main(String[] args) {
        PrismaUniforme r = new PrismaUniforme(5, 3.0);
        System.out.println(r);
    }
}
```

```
Bases(2): Polígono regular de 5 lados de longitud 3,00 (Área: 15,48, Perímetro: 15,00)
Caras(5): Polígono regular de 4 lados de longitud 3,00 (Área: 9,00, Perímetro: 12,00)
Área Total: 75,97
Volumen: 46,45
```



Prisma recto uniforme con base pentagonal (5 lados de longitud 3 unidades)

Solución:

Estrictamente hablando, bastaría con un constructor y el método `toString()`:

```
public class PrismaUniforme {
    private PolígonoRegular base, cara;

    public PrismaUniforme(int n_lados, double longitud) {
        this.base = new PolígonoRegular(n_lados, longitud);
        this.cara = new PolígonoRegular(4, longitud); // son cuadrados, 4 lados
    }

    @Override
    public String toString() {
        double áreaTotal = 2 * this.base.área() + this.base.getNLados() * this.cara.área();
        double volumen = this.base.área() * this.base.getLongitud();

        String s = String.format("Bases(2): %s\n", this.base) +
               String.format("Caras(%d): %s\n", this.base.getNLados(), this.cara) +
               String.format("Área Total: %.2f\nVolumen: %.2f", áreaTotal, volumen);
        return s;
    }
}
```

Aunque algo más *elegante* podría ser:

```
public class PrismaUniforme {
    private PolígonoRegular base, cara;

    public PrismaUniforme(int n_lados, double longitud) {
        this.base = new PolígonoRegular(n_lados, longitud);
        this.cara = new PolígonoRegular(4, longitud); // son cuadrados, 4 lados
    }
    public double áreaBase() { return this.base.área(); }
    public double áreaCara() { return this.cara.área(); }
    public double áreaTotal() { return 2 * this.áreaBase() + this.base.getNLados() * this.áreaCara(); }
    public double altura() { return this.base.getLongitud(); }
    public double volumen() { return this.áreaBase() * this.altura(); }

    @Override
    public String toString() {
        String s = String.format("Bases(2): %s\n", this.base) +
               String.format("Caras(%d): %s\n", this.base.getNLados(), this.cara) +
               String.format("Área Total: %.2f\nVolumen: %.2f", this.áreaTotal(), this.volumen());
        return s;
    }
}
```