


<b>Examen Recuperación TRIM1 - Tiempo 30' (Test) + 135' (Ejercicios)</b>	<b>27/01/2020-17:50h</b>
Nombre y Apellidos	DNI/NIE: Firma:
1º Desarrollo de Aplicaciones Web (Vespetino) Módulo: <b>Programación</b> (Aula 28)	 IES Alonso de Avellaneda (Alcalá)

### Cuestiones (2 puntos) - A entregar antes de los ejercicios

1. Cuál es la salida de este programa

```
int balance = 10;
while (true) {
    if (balance < 9)
        break;
    balance = balance - 9;
}
System.out.println("Balance es "+ balance);
```

2. Cuál es el resultado si se llama a la función con **getGrade(59.5)**:

```
public static char getGrade(double score) {
    if (score >= 90.0)
        return 'A';
    else if (score >= 80.0)
        return 'B';
    else if (score >= 70.0)
        return 'C';
    else if (score >= 60.0)
        return 'D';
    else
        return 'F';
}
```

3. Qué falta en (1) y cómo se podría llamar al método getAllNodes en (2) según lo anterior.

```
public class A {
    public ...(1)... double getAllNodes(){ return this.number;}
}
public class B{ ...(2)...}
```

4. Evalúa esta expresión:

```
3 + 4 * 4 > 5 * (4 + 3) - 1 && (4 - 3 > 5)
```

5. Cuál será el resultado de **tax** si income es 10001. ¿Cuál es el valor de tax al finalizar esta sentencia?

```
tax = (income > 10000) ? income * 0.2 : income * 0.17 + 1000;
```

6. Cuál es la salida de el siguiente segmento de código:

```
int count;
for (count = 1; count <= 10; count++)
{
    if (count == 5)
        continue;
    System.out.printf("%d ", count);
}
```

7. Cuál es el valor que imprime este programa:

```
int sum = 0;
int x = 1;
while (x <= 10)
{
    sum += x;
    ++x;
}
System.out.printf("La suma es: %d\n", sum);
```

8. Aplica la Ley de Morgan a esta expresión

```
!(a == b) || !(g != 5)
```

9. Cuál es la impresión de este programa:

```
for (int count = 1; count <= 10; count++)
{
    if (count == 5)
        break;
    System.out.printf("%d ", count);
}
```

10. Supongamos el siguiente código, ¿cuál es el error?:

```
public class ShowErrors {
    public static void main(String[] args) {
        C c = new C(5.0);
        System.out.println(c.value);
    }
}
class C {
    int value = 2;
}
```

**Programas (8 puntos)** (Recogida de los fuentes del programa en repositorio github de cada alumno)

**Ejercicio 1 ( 2.5 puntos)**

La congruencia de **Zeller** es un algoritmo desarrollado por Christian Zeller para calcular el día de la semana. La fórmula es:

$$h = \left( q + \frac{26(m + 1)}{10} + k + \frac{k}{4} + \frac{j}{4} + 5j \right) \% 7$$

donde:

- **h** es el día de la semana (0:Sábado, 1:Domingo, 2:Lunes, 3: Martes, 4:Miércoles, 5: Jueves, 6: Viernes).
- **q** es el día del mes.
- **m** es el mes (3: Marzo, 4: Abril, 5:Mayo, 6:Junio, 7:Julio, 8:Agosto, 9:Septiembre, 10:Octubre, 11:Noviembre, 12:Diciembre). Enero y Febrero se cuentan como 13 y 14 respectivamente del año anterior.
- **j** es el `año / 100`
- **k** es el año del siglo ( `año % 100` )

Todas las divisiones son enteras. Escribe un programa que muestre, introduciendo el año, mes y día del mes, muestre el día de la semana.

Crea una clase **Zeller** que tenga la siguiente función:

```
public static int getZeller (int m, int year, int q)
```

Ejemplo:

```
Año (2012): 2020
Mes (1-12): 1
Day of Month: (1-31)22
El día de la semana es: Miércoles
```

## Ejercicio 2 ( 3 puntos)

Se solicita realizar una aplicación bancaria (clase CuentaCorriente) que gestione cuentas de ahorro con los siguientes métodos y constructores:

- void ingresar(double ingreso) -> **hace un ingreso en cuenta** (0.25)
- void transferencia(Cuenta origen, Cuenta destino, double cantidad) -> **transferencia en cuenta** (0.5)
- void crearCuenta() -> **crea la cuenta con un número aleatorio de 12 dígitos, nombre y apellidos y DNI** (0.5p)
- void retirar(double cantidad) -> **retira dinero en cuenta** (0.25)
- double calcularInteresAnual(double interesmensual) -> **calcula el interés anual dado el parcial mensual** (0.5)
- CuentaCorriente(CuentaCorriente cc) -> **Copia los atributos** (0.25)
- imprimeCuenta() -> imprime "Número de cc 123221 con Saldo 0.00€. Nombre ..... Apellido.... DNI....." (0.25)

**Atributos: nombre, dni, saldo, número cuenta.**

Crea una clase TestCC que realice las siguientes acciones: (0.5)

```
Crea una cc1. Ingreso: 2000€. Nombre: John Dan. DNI: 478733B
Crea una cc2. Ingreso: 7000€. Nombre: Pavel Mitnick. DNI: 120000T
Ingresa 1200 € en cc1.
Transfiere 3000€ a una cuenta cc1.
Copia cc1 en otra cc4
Retira 150€ en cc4
Retira 200 en cc2
```

### Ejercicio 3 (2.5 puntos)

Crear la clase **Empleado** con los siguientes atributos que no puedan ser accedidos fuera de la clase:

**nombre, apellido, numSS, tasaComision, salarioBruto**

Crear los siguientes metodos: (1p)

- **Constructor que inicie todos atributos sólo si cumple la condición, si no, imprime un error y no asigna el atributo:**
  - verifique si el salarioBase < 0 muestra error: "Salario debe ser >= 0"
  - verifique si la tasaComisión está entre 0 y 1. Error: "Tasa de comisión entre 0 y 1"
  - verifique que salarioBase no puede ser negativo. "Error, salario no puede ser negativo"
  - verifique que numSS tiene 10 dígitos.
- **Métodos:**
  - Todos los métodos getters y setters de los atributos aplicando las mismas restricciones en el caso que proceda. (0.25)
  - cálculo de la comisión semestral (**double calcularComision()**) se calcula tasaComision \* salarioBruto pero se añade la cantidad según las tablas siguientes: (0.75p)

SalarioBruto	Cálculo c
=====	=====
0-1000 € ->	5% del salarioBruto
1001-1500 ->	Ídem para el 7%
1501-2000 ->	Idem para el 11%
2001-2500 ->	Ídem para el 15%

- Crear clase **TestEmpleado** que prueba la clase Empleado que pruebe la creación y cálculo de la comisión para 4 empleados según su intervalo de salario y con una comisión de 15%. (0.5)

Rúbricas:

Ejercicio 1,2,3: Los programas deben funcionar correctamente en su totalidad y realizar lo especificado en el ejercicio en cada apartado => 100%. En caso de no funcionamiento, ilegibilidad, no realizado, incompleto, etc. => 0%