


FINAL EXTRAORDINARIA (TIEMPO 2 HORAS)	16/06/2020 - 16:00
Nombre y Apellidos:	DNI/NIE: Firma:
1º Desarrollo de Aplicaciones Web (Vespertino) Módulo: Entornos de Desarrollo	 IES Alonso de Avellaneda (Alcalá)

La superación de este examen y del módulo será con nota igual o superior a 5. La nota final máxima que se puede obtener en esta prueba es de 7.

Se realizará una entrega parcial y luego una entrega total al repositorio de la prueba

Si el/los ejercicio/s resultan plagio de recursos de Internet o de otras pruebas el valor de la prueba será de 1 sobre 10, no superada

Rúbrica:

- Cuestión correcta en funcionamiento, buen formato, claridad, concisión, rigurosidad, precisión 100%.
- Incorrecto, incompleto, muy deficiente, poca claridad, ilegible, mal formato 0%.
- Si tiene algunas deficiencias que no impiden el funcionamiento, especificación, etc correctos, se tendrá en cuenta hasta un 50%.

Ejercicios (10 puntos) (Recogida de los fuentes del programa en repositorio github de cada alumno)

Todos los ficheros necesarios están en el repositorio de la prueba.

1. Según el código facilitado en el cual se utiliza el método de ordenación burbuja en el que se ordena un array unidimensional. Realizar lo siguiente: (3.5p)

```
public class Burbuja {

    public static void bubbleSort(int[] x) {
        int n = x.length;
        boolean hacerMas = true;
        while (hacerMas) {
            n--;
            hacerMas = false; // assume this is our last pass over the array
            for (int i = 0; i < n; i++) {
                if ( x[i] > x[i + 1] ) {
                    // exchange elements
                    int temp = x[i];
                    x[i] = x[i + 1];
                    x[i + 1] = temp;
                    hacerMas = true; // after an exchange, must look again
                }
            }
        }
    }
}
```

```

} // end method bubbleSort3

public static void main(String[] args) {
    int[] v = { 7, -1, -2, 0, 5, 4 };

    bubbleSort(v);
    for (int i = 0; i < v.length; i++)
        System.out.print(" " + v[i]);
}

}

```

a) Realiza el grafo de complejidad ciclomática completo escribiendo los nodos, flechas y condiciones (2p).

b) Calcula la complejidad ciclomática de las tres formas indicando las fórmulas que se necesitan. (0.5p)

c) Define el conjunto básico de caminos indicando los nodos de cada uno (0.5p)

d) Elabora los casos de prueba para cada camino (0.5p)

La validez de los puntos b, c y d son una consecuencia correcta de a

3. Escribe una clase de pruebas (**TablaEnterosTest.java**) para testear los métodos **sumaTabla**, **mayorTabla**, **posicionTabla** de la clase **TablaEnteros**. En la clase de prueba se debe crear los elementos que sean necesarios para las pruebas (arrays, etc.). Realizar **dos pruebas por cada método**, una que como hipótesis en la que vaya a fallar y otra prueba en la que se acierte. ¿Tiene algún error el programa? Si fuera afirmativo, explícalo con precisión y muestra la posible solución para que no de error en la prueba. (3p)

```

import java.util.NoSuchElementException;

public class TablaEnteros {

    private Integer[] tabla;

    TablaEnteros(Integer[] tabla){
        if (tabla == null || tabla.length == 0)
            throw new IllegalArgumentException("No hay elementos");
        this.tabla = tabla;
    }

    public int sumaTabla() {
        int suma = 0;
        for (int i = 0; i < tabla.length; i++)
            suma = suma + tabla[i];
        return suma;
    }

    public int mayorTabla() {
        int max = -999;
        for (int i = 0; i < tabla.length; i++)
            if (tabla[i] > max )
                max = tabla[i];
        return max;
    }
}

```

```

    public int posicionTabla(int n) {
        for (int i = 0; i < tabla.length; i++)
            if (tabla[i] != n)
                return i;
        throw new NoSuchElementException("No existe: " + n);
    }
}

```

4. Realiza la compilación de las siguientes clases con el compilador de java **en terminal** teniendo la siguiente jerarquía:

```

com/
├─ gestion
│   └─ Gestion.java
├─ it
│   └─ It.java
├─ main
│   └─ Main.java
└─ marketing
    └─ Marketing.java

```

Las clases son las siguientes. Realiza los cambios necesarios para hacer el proceso de compilación por terminal. El programa debe compilar y ejecutar desde la clase `Main.java` y se debe poder ejecutar llamándolo con todo el nombre jerárquico del paquete al que pertenece. Debéis mostrar todo el contenido del paquete mediante consola después de esta operación y adjuntar capturas y las carpetas y ficheros creados. (2p)

```

import java.util.*;
public class Gestion {
    public void print(){
        System.out.println("Departamento Gestion");
    }
}

import java.util.*;
public class It {
    public void print() {
        System.out.println("Departamento IT");
    }
}

import java.util.*;
public class Marketing {
    public void print() {
        System.out.println("Departamento Marketing");
    }
}

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

        It it = new It();
        Gestion gestion = new Gestion();
        Marketing marketing = new Marketing();
        it.print();
    }
}

```

```
        gestion.print();  
        marketing.print();  
    }  
}
```

5. Crea el archivo jar con la información del MANIFEST y ejecútalo en terminal (1p). Lista el contenido del jar (0.25p). Extrae el contenido del jar en la ruta `~/Descargas` (0.25p)