APELLIDOS:	***NOMBRE:***
GRADO:	***FIRMA:***
Duración: 2h 30'	

Ejercicio 1	Ejercicio 2	Ejercicio 3	Ejercicio 4	Ejercicio 5	TOTAL

NOTA: Los ejercicios del 2 al 5 se codificarán en hojas aparte.

Ejercicio 1: Este ejercicio se responderá en esta hoja de examen. [10 puntos]
Definir las constantes y tipos de datos apropiados para:

a) Almacenar en memoria principal un máximo de 100 Clientes. De cada Cliente se quiere almacenar su nombre, domicilio y un campo que indique si es habitual. **[2,5 Puntos]**

```
final int DIRCLI = 100;
class Cliente implements Serializable {

    String Nombre;
    String domicilio;
    boolean habitual;
}
Cliente[] misClientes = new Cliente[DIRCLI];
```

b) Almacenar en memoria la ocupación de un hotel. El hotel tiene 10 plantas y 7 habitaciones por planta. De cada habitación queremos saber el número de personas que la ocupan.
[2,5 Puntos]

```
final int FILA =10;
final int COLUMNA =7;
int[][] hotel = new int[FILA][COLUMNA];
```

c) Almacenar la información de un mensaje de WhatsApp. El mensaje consta de remitente, destinatario, hora de envío y cuerpo del mensaje.
[2,5 Puntos]

```
class whatsapp implements Serializable {
    int hora;
    String remitente;
    String destinatario;
    String cuerpo;
}
whatsapp miwhats = new whatsapp();
```

d) Almacenar la información de los 60 mejores expedientes académicos de la ETSII. Lo único que nos interesa de los alumnos es su número de expediente. **[2,5 Puntos]**

```
final int MAXALUM = 60;
int [] mejores = new int [MAXALUM];
```

APELLIDOS:	***NOMBRE:***
GRADO:	***FIRMA:***
Duración: 2h 30'	

Ejercicio 2: **[20 puntos]**

Escribir un método que reciba dos números enteros positivos y devuelva si son **amigos**. Dos números son amigos si la **suma de sus divisores** propios (distintos de ellos mismos) **son iguales** (se puede crear otro método auxiliar).

Por ejemplo los números (220, 284) sí son amigos.

```
private static boolean amigos(int a, int b) {
    boolean amigos;
    if (a == sumaDivisoresPropios(b) && b == sumaDivisoresPropios(a)) {
        amigos = true;// son amigos
    } else {
        amigos = false; // no son amigos
    }
    return amigos;
}

private static int sumaDivisoresPropios(int num) {
    int suma = 0;
    for (int i = 1; i < num; i++) { // al ser i< num no usamos el propio num
        if (num % i == 0) {// si i divide a num
            suma += i; // acumulamos i
        }
    }
    return suma;
}
```

Ejercicio 3: **[15 puntos]**

Escribir un **método recursivo** que reciba un número entero positivo y permita sumar los dígitos de un número.

Ejemplo: Entrada 1173 Resultado: 12

```
private static int sumar_digREC(int n) {
    if (n == 0) { //caso base
        return n;
    } else {
        return sumar_digREC(n / 10) + (n % 10);
    }

}
```

Ejercicio 4: **[40 puntos]**

Se quiere realizar un programa que realice distintas operaciones sobre arrays uni y bidimensionales.

Se tiene la siguiente declaración de tipos de datos

APELLIDOS:	***NOMBRE:***
GRADO:	***FIRMA:***
Duración: 2h 30'	

```
public static void main(String[] args) throws IOException {
    final int fmatriz =45;
    int[] array = new int[10];
    int[][] matriz = new int[4][4];
    Scanner entrada = new Scanner(System.in);
```

Se pide:

a) MediaArray: Realizar un método que reciba el *array* definido arriba y la clase Scanner y pida tantos datos como el usuario quiera y los vaya introduciendo en el array. El método devolverá la media de los datos introducidos. Para ello después de introducir un dato preguntará al usuario que quiere hacer ("1- Insertar más datos, 2 -Salir"). **[10 Puntos]**

```
private static double MediaArray(int[] array, Scanner scanner) {
        int select = 0;
        int pos = 0;
        int suma = 0;
        double resultado =0;

        while ((select != 2) && (array[pos] <= array.length)) {

            System.out.print("insertar posición[" + pos + "]  ");
            array[pos] = scanner.nextInt();
            suma= suma + array[pos];
            pos = pos + 1;

            System.out.print("1 Insertar más, 2 Salir");
            select = scanner.nextInt();

        }
        resultado = (double)suma/pos;
        return resultado;
    }
```

b) **MaxArray:** Realizar un método reciba el *array* y devuelva el valor máximo que contiene. **[5 Puntos]**

```
private static int maxArray(int[] array) {
    int max = 0;
    for (int i = 0; i < array.length; i++) {
        if (array[i] > max) {
            max = array[i];
        }
    }
    return max;
}
```

APELLIDOS:	*NOMBRE:*
GRADO:	*FIRMA:*
Duración: 2h 30'	

c) **DesplazarArray:** Realizar un método que reciba un *array* y modifique el array desplazando sus elementos hacia la izquierda 1 posición. Así el segundo elemento pasaría a la primera posición, el tercer elemento pasaría a la segunda posición y así sucesivamente. El primer elemento pasaría a ocupar la última posición. No se puede usar otro array auxiliar. **[10 Puntos]**
Ejemplo:
Si inicialmente el array = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 cuando finaliza el subprograma el array = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

```
private static void desplazarIzq(int[] array) {
    int aux = array[0];
    int i;
    for (i = 1; i < array.length; i++) {
        array[i - 1] = array[i];
    }
    array[i - 1] = aux;
}
```

d) **Matriz:** Realizar un método que reciba la *matriz* y el valor *fmatriz* ambos del enunciado y rellene sus elementos con los valores que se muestran en el ejemplo (deben usarse bucles y variables para hacerlo; a continuación el método mostrará el contenido de la matriz como en el ejemplo **[10 Puntos]**
Ej:

```
run:
45      42      39      36
33      30      27      24
21      18      15      12
9       6       3       0
```

```
private static void rellenarmatriz(int[][] matriz, int valor) {
    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz[i].length; j++) {
            matriz[i][j] = valor;
            valor = valor - 3;
        }
    }
    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz[i].length; j++) {
            System.out.print(matriz[i][j] + "\t");
        }
        System.out.println("");
    }
}
```

f) Escribe las instrucciones necesarias para realizar las llamadas a los métodos de los apartados a), b) y mostrar el valor que devuelve cada método correctamente al usuario por pantalla. **[5 puntos]**

APELLIDOS:	***NOMBRE:***
GRADO:	***FIRMA:***
Duración: 2h 30'	

```
double media = MediaArray(array, entrada);
System.out.println("La media de los valores es: " + media);

int max = maxArray(array);
System.out.println("El valor ultimo valor máximo es: " + max );
```

Ejercicio 5: [15 puntos]

En la empresa **VideojuegosSL** necesitan almacenar los datos de sus artículos y sus ventas en sendos ficheros. Para ello se desean incorporar dos métodos que almacenen el array de artículos y el array de ventas a su programa de gestión. Las opciones que debe incluir el nuevo programa serán:
Los tipos de datos son los siguientes:

```
final int MAXARTICULOS = 100; //{número de artículos máximo de VideojuegosSL}
final int MAXVENTAS = 200; //{número de ventas máximas diarias de VideojuegosSL}
final double BENEFICIO = 0.1; //{beneficio de cada venta}

class Articulo implements Serializable {

    String codart; // {código de artículo}
    String nombre; // {nombre artículo}
    String caracteristicas; // {características}
    double precio; // {precio del artículo}
    int cantidad; // {cantidad de artículos en stock}
}

Articulo[] misArticulos = new Articulo[MAXARTICULOS];//{array que contiene los artículos}
class Venta implements Serializable {

    String codart; // {código de artículo}
    double precart; // {precio del artículo}
    int cantidad; // {cantidad de artículos vendidos}
    double pvp; // {precio de venta de un artículo al público}
    double costeTotal; // {Coste de la venta realizada (pvp*cantidad)}
}// {Registro de venta de un artículo}

Venta[] misVentas = new Venta[MAXVENTAS]; // {array de las ventas diarias de VideojuegosSL}
```

a) **Guardar las ventas realizadas en fichero de texto**. Realizar un método que reciba el *array de ventas* y escriba en un fichero de texto denominado "*ventas.txt*" una línea por cada venta, donde aparecerá el código de artículo, el precio del artículo (con dos decimales), el coste (precio de venta al público con dos decimales), la cantidad y el coste total como se muestra en el ejemplo. **[7,5 puntos]**
Ejemplo de cómo quedará el fichero de texto:

3455	5.12	5.63	20	112.64
5585	45.10	49.61	1	49.61
5585	7.20	7.92	5	39.60

APELLIDOS:	***NOMBRE:***
GRADO:	***FIRMA:***
Duración: 2h 30'	

```
private static void guardarVentas(Venta[] misVentas) throws IOException {
    try {
        PrintStream fichero;
        fichero = new PrintStream(new FileOutputStream("ventas.txt"));
        for (int i = 0; i < misVentas.length; i++) {
            fichero.println(misVentas[i].codart + "\t" + misVentas[i].precart + "\t");
            fichero.print(misVentas[i].pvp + "\t" + misVentas[i].cantidad);
            fichero.println("\t" + misVentas[i].costeTotal);
        }
        fichero.close();
    } catch (FileNotFoundException e) {
        System.out.println("Fichero no encontrado");
    }

}
```

b) **Guardar el stock de almacén en fichero binario**: Realizar un método que almacene el *array de artículos* de la ferretería en un fichero binario (*articulo.dat*). Se debe almacenar únicamente los datos de stock (solo los artículos existentes en el almacén) **[7,5 puntos]**

```
private static void guardarStock(Articulo[] misArticulos) throws IOException {
    try {
        FileOutputStream out = new FileOutputStream("articulo.dat");
        ObjectOutputStream art = new ObjectOutputStream(out);
        for (int i = 0; i < misArticulos.length; i++) {
            art.writeObject(misArticulos[i]);
        }
        art.close();

    } catch (FileNotFoundException exc) {
        System.out.println("Excepción");
    }
}
```