Sacar los códigos que están mal en el código:

code smell de tipo bloaters

-porque main hace muchas cosas

```
public static void main(String args[])
{
    System.out.println("====================");
    System.out.println("Vamos a refactorizar!");
    System.out.println("====================");

    // creamos un DNI correcto
    DNI dniCorrecto = new DNI(TIPODNI.DNI,  numDNI: "11111111H",  fchValidez: null);
    Boolean esValido = (dniCorrecto.validarDNI() == 1);
    System.out.println( "DNI " + dniCorrecto.numDNI + " es: " + esValido.toString());

    // creamos un DNI incorrecto
    DNI dniIncorrecto01 = new DNI(TIPODNI.DNI,  numDNI: "24324356A",  fchValidez: null);
    Boolean esValido01 = (dniIncorrecto01.validarDNI() == 1);
    System.out.println( "DNI " + dniIncorrecto01.numDNI + " es: " + esValido01.toString());

    // creamos un NIE correcto
    DNI nieCorrecto = new DNI(TIPODNI.NIE,  numDNI: "X0932707B",  fchValidez: null);
    Boolean esValidoNie = (nieCorrecto.validarDNI() == 1);
    System.out.println( "NIE " + nieCorrecto.numDNI + " es: " + esValidoNie.toString());

    // creamos un NIE incorrecto
    DNI nieIncorrecto = new DNI(TIPODNI.NIE,  numDNI: "Z2691139Z",  fchValidez: null);
    Boolean esValidoNieIncorrecto = (nieIncorrecto.validarDNI() == 1);
    System.out.println( "NIE " + nieIncorrecto.numDNI + " es: " + esValidoNieIncorrecto.toString());

    // creamos un CIF correcto
```

-long method hay muchos condicionales

```
TipoUltCaracter tipUltCar;

// si empiezo por P,Q, S, K o W la última letra tiene que ser una LETRA
if (primerCar == 'P' || primerCar == 'Q' || primerCar == 'S' || primerCar == 'K' || primerCar == 'W') {
    tipUltCar = TipoUltCaracter.LETRA;
    if (!(ultimoCar >= 'A' && ultimoCar <= 'Z')) {
        return 0; // no es una letra
    }
// si empiezo por A, B, E o H la última letra tiene que ser un número
} else if (primerCar == 'A' || primerCar == 'B' || primerCar == 'E'
        || primerCar == 'H') {
    tipUltCar = TipoUltCaracter.NUMERO;
    if (!(ultimoCar >= '0' && ultimoCar <= '9')) {
        return 0; // no es un número --> casco!
    }
// en otro caso la última letra puede ser cualquier cosa
} else {
    tipUltCar = TipoUltCaracter.AMBOS;
}


final String digitos = cifUP.substring(1, cifUP.length() - 1);

// sumo los pares
```

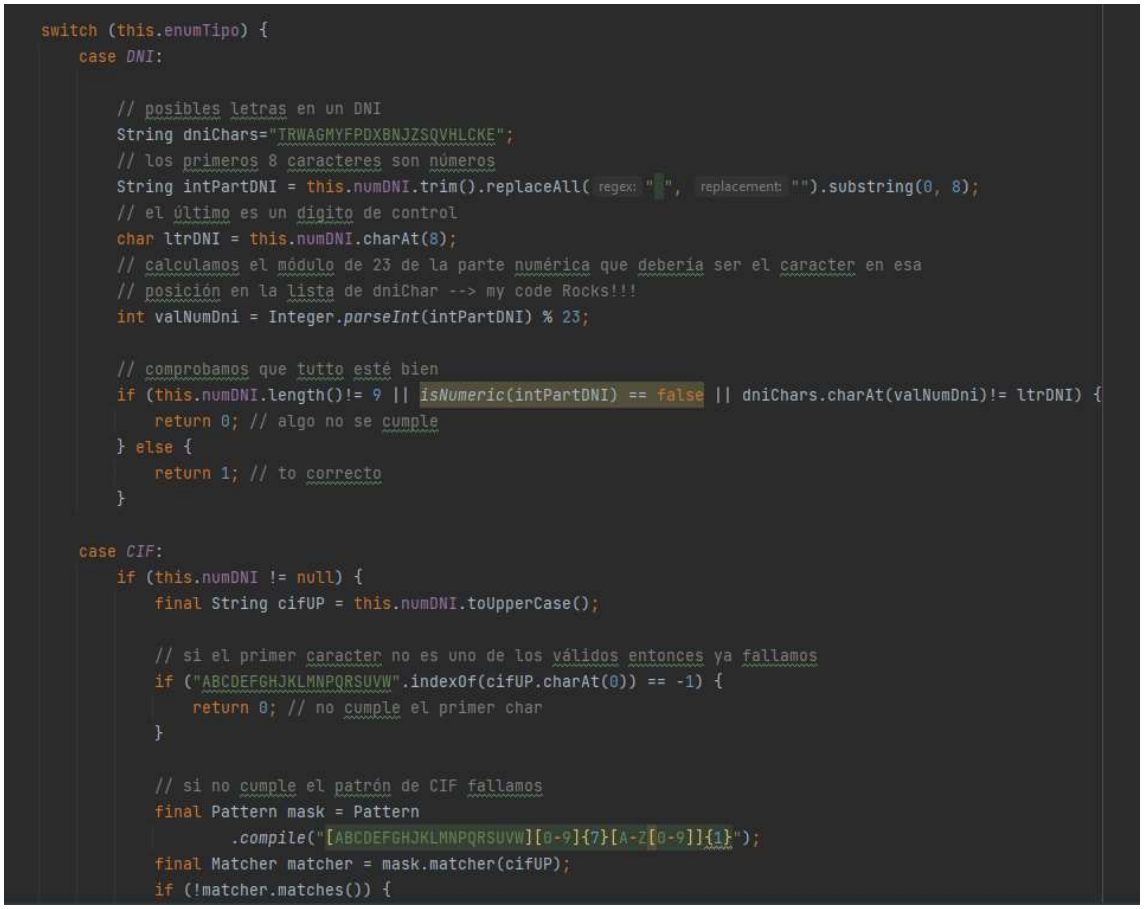
Las funciones son muy largas mas de 20 líneas de código: no debe hacer scroll en una pantalla estándar

La función validarDNI() ocupa más de 175 líneas y es porque hace varias cosas entre ellas no solo validar el DNI sino el NIE y el CIF esto debería estar separado en varias clases y dentro de esas clases tener varias funciones mas pequeñas que solo hagan una cosa.

el nombre es engañoso porque hace más de lo que dice en el nombre.

Además, esta función tiene un switch que no es nada aconsejable porque incumple el principio SOLID Open/Close

OO Abusers yo he intentado solucionarlo con un Factory pero igual me he equivocado de estrategia.



```
switch (this.enumTipo) {
    case DNI:

        // posibles letras en un DNI
        String dniChars="TRWAGMYFPDXBNJZSQVHLCKE";
        // los primeros 8 caracteres son números
        String intPartDNI = this.numDNI.trim().replaceAll( regex: " ", replacement: "").substring(0, 8);
        // el último es un dígito de control
        char ltrDNI = this.numDNI.charAt(8);
        // calculamos el módulo de 23 de la parte numérica que debería ser el caracter en esa
        // posición en la lista de dniChar --> my code Rocks!!!
        int valNumDni = Integer.parseInt(intPartDNI) % 23;

        // comprobamos que tutto esté bien
        if (this.numDNI.length()!= 9 || isNumeric(intPartDNI) == false || dniChars.charAt(valNumDni)!= ltrDNI) {
            return 0; // algo no se cumple
        } else {
            return 1; // to correcto
        }

    case CIF:
        if (this.numDNI != null) {
            final String cifUP = this.numDNI.toUpperCase();

            // si el primer caracter no es uno de los válidos entonces ya fallamos
            if ("ABCDEFGHJKLMNPQRSUVW".indexOf(cifUP.charAt(0)) == -1) {
                return 0; // no cumple el primer char
            }

            // si no cumple el patrón de CIF fallamos
            final Pattern mask = Pattern
                    .compile("[ABCDEFGHJKLMNPQRSUVW][0-9]{7}[A-Z[0-9]]{1}");
            final Matcher matcher = mask.matcher(cifUP);
            if (!matcher.matches()) {
```

aquí en este trozo de código

```
            }

        default:
            return -99; // se ha producido un error
```

lanza un código de error cuando debería de sacar una excepción.

Comentarios:

los comentarios no deben explicar el código, el código debe explicarse a si mismo.

```
// si empiezo por P,Q, S, K o W la última letra tiene que ser una LETRA
if (primerCar == 'P' || primerCar == 'Q' || primerCar == 'S' || primerCar == 'K' || primerCar == 'W') {
    tipUltCar = TipoUltCaracter.LETRA;
    if (!(ultimoCar >= 'A' && ultimoCar <= 'Z')) {
        return 0; // no es una letra
```

aquí por ejemplo el comentario es el mismo que el nombre de la variable así que no tendría mucho sentido

```
// sumo los pares
Integer sumaPares = 0;
for (int i = 1; i <= digitos.length() - 1; i = i + 2) {
    sumaPares += Integer.parseInt(digitos.substring(i, i + 1));
}

// sumo los impares
Integer sumaImpares = 0;
for (int i = 0; i <= digitos.length() - 1; i = i + 2) {
    Integer cal = Integer.parseInt(digitos.substring(i, i + 1)) * 2;
    if (cal.toString().length() > 1) {
        cal = Integer.parseInt(cal.toString().substring(0, 1))
                + Integer.parseInt(cal.toString().substring(1, 2));
    }
    sumaImpares += cal;
}
```

habría que evitar comentarios que no aclaren cosas

```
char ltrDNI = this.numDNI.charAt(8);
// calculamos el módulo de 23 de la parte numérica que debería ser el caracter en esa
// posición en la lista de dniChar --> my code Rocks!!!
int valNumDni = Integer.parseInt(intPartDNI) % 23;
```

Como he intentado solucionar el código:

primero intentado que main tenga las menos funciones posibles y los parámetros que se pasan son los mínimos.

```
public static void main(String[] args) {

    printBanner();

    printResultado( documento: "11111111H");
    printResultado( documento: "24324356A");
    printResultado( documento: "X0932707B");
    printResultado( documento: "Z2691139Z");
        //falta implemetar el validador de cif


}
```

partiéndolo en funciones mas pequeñas para hacer el mismo trabajo

```
static void printBanner(){
    System.out.println("======================");
    System.out.println("Vamos a refactorizar!");
    System.out.println("======================");

}

static void printResultado(String documento){

    // faltaria implemetar un printer por tipo
    IValidadorDocumento miValidador = ValidadorFactory.getValidador(documento);

    if(miValidador != null && miValidador.validar() == true) {

        System.out.println("El DNI "+documento+" es: true");
    } else {
        System.out.println("El DNI " +documento +" es: false");
    }
}
```

y después creando una factoria que implementa una interfaz

```java
package amparo.refactor.bien;

interface IValidadorDocumento {
    public boolean validar();
}
```

que es la que instancia los nuevos documentos según sean de un tipo o de otro

```java
        IValidadorDocumento miValidador = null;


        if (isNie(doc)){
            miValidador=new ValidadorNIE(doc);
        }else if (isDni(doc)){
            miValidador = new ValidadorDNI(doc);
        }
         return miValidador;
    }


    private static boolean isDni(String doc){

         if(doc.length() == 9 && Character.isLetter(doc.charAt(8))) {

            return true;
        }
         return false;
    }
```

```java
private static boolean isDni(String doc){

    if(doc.length() == 9 && Character.isLetter(doc.charAt(8))) {

        return true;
    }
    return false;
}

private static boolean isNie(String doc){

    if(doc.length() == 9 && Character.isLetter(doc.charAt(8))
        && (Character.toUpperCase(doc.charAt(0)) == 'X'
        ||  Character.toUpperCase(doc.charAt(0)) == 'Y'
        ||  Character.toUpperCase(doc.charAt(0)) == 'Z')) {

        return true;
    }
    return false;
}
```

y tres clases vallidarDNI,Validar NIE y ValidarCif que serían las que dieran por bueno o no el documento

```java
public ValidadorDNI(String dni) { this.dni = dni; }

public boolean validar() {

    boolean esValido = false;
    int i = 0;
    int caracterASCII = 0;
    char letra = ' ';
    int miDNI = 0;
    int resto = 0;
    char[] asignacionLetra = {'T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X',
                              'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'};

    do {
        caracterASCII = this.dni.codePointAt(i);
        esValido = (caracterASCII > 47 && caracterASCII < 58);
        i++;
    } while(i < this.dni.length() - 1 && esValido);

    if(esValido) {
        letra = Character.toUpperCase(this.dni.charAt(8));
        miDNI = Integer.parseInt(this.dni.substring(0,8));
        resto = miDNI % 23;
        esValido = (letra == asignacionLetra[resto]);
    }

    return esValido;
}
```

```java
public ValidadorNIE(String nie) { this.nie = nie; }

public boolean validar() {

    boolean esValido = false;
    int i = 1;
    int caracterASCII = 0;
    char letra = ' ';
    int miNIE = 0;
    int resto = 0;
    char[] asignacionLetra = {'T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X','B',
                              'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'};


    do {
        caracterASCII = this.nie.codePointAt(i);
        esValido = (caracterASCII > 47 && caracterASCII < 58);
        i++;
    } while(i < this.nie.length() - 1 && esValido);


    if(esValido && this.nie.substring(0,1).toUpperCase().equals("X")) {
        this.nie = "0" + this.nie.substring(1,9);
    } else if(esValido && this.nie.substring(0,1).toUpperCase().equals("Y")) {
        this.nie = "1" + this.nie.substring(1,9);
    } else if(esValido && this.nie.substring(0,1).toUpperCase().equals("Z")) {
        this.nie = "2" + this.nie.substring(1,9);
    }

    if(esValido) {
        letra = Character.toUpperCase(this.nie.charAt(8));
        miNIE = Integer.parseInt(this.nie.substring(1,8));
        resto = miNIE % 23;
        esValido = (letra == asignacionLetra[resto]);
    }

    return esValido;
}
```