SENTENCIA IF

Un resumen en
http://manuais.iessanclemente.net/index.php/Elementos_esenciais_da_linguaxe_Java#Control_de_fluxo

que es una traducción de

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

Pero estos principios básicos vienen bien explicados en casi cualquier web, no obstante, recuerda que lo importante es hacer(no leer) ejercicios. No te dediques a leer lo mismo en cientos de webs una y otra vez.

Vista la web oficial, observamos que realmente no es riguroso hablar de "la sentencia if", si no de dos sentencias:

- la sentencia if-then
- la sentencia if-then-else (que realmente no es más que una ampliación de la primera)

Coloquialmente siempre hablamos del "if" o de la "sentencia condicional if" incluso de la "instrucción if". (Aunque el término instrucción se prefiere reservar para las "instrucciones de bajo nivel" del lenguaje máquina).

la sentencia if-then

Ejemplo: Como ya sabemos su funcionamiento básico de una unidad anterior, adelantamos su explicación cuando estudiamos el operador condicional, nos limitamos a observar posibilidades sintácticas

```
class  Unidad3{
   public static void main(String[] args) {

       int x=10;
       //el bloque puede tener muchas intrucciones
       if (x== 10){
          System.out.println("x es mayor que 10 y vale: "+x);
          System.out.println("Si le sume a x pasa a valer: "+x);
      }
      //el bloque puede tener una única instrucción
      if (x== 10){
          System.out.println("x vale exactamente 10");
      }
      //si el bloque sólo tiene una instrucción
      //se pueden omitir llaves aunque no se recomienda
      if (x== 10)
           System.out.println("ciertamente x vale exactamente 10");

  }
}
```

sobre las llaves de bloque en el if

Cuando el bloque de instrucciones que sigue a la condición del IF sólo contiene una instrucción, no es necesario el uso de {} para indicar principio y fin de bloque

Aunque ambas formas son equivalentes, las normas de estilo java recomienda en general la forma 1 (con llaves), pero no es que este siempre mal la forma 2 ya que si consultamos los tutoriales de oracle
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html
vemos que habla del tema

Deciding when to omit the braces is a matter of personal taste. Omitting them can make the code more brittle. If a second statement is later added to the "then" clause, a common mistake would be

forgetting to add the newly required braces. The compiler cannot catch this sort of error; you'll just get the wrong results.

Aunque preferimos usar llaves, en este boletín usaremos muchos ejemplos sin las llaves recomendadas para coger práctica también con esta posibilidad y entender muy la sintaxis del if.

Ejemplo: No usar corchetes puede inducir a errores

Esto no da error de compilación pero al leerlo no tengo claro que quería realmente escribir el programador debido a la indentación del segundo println() sus intenciones me resultan confusas. Suele ocurrir que escribes un if sin corchetes porque sólo tiene una instrucción, pero modificas el programa y quieres añadirle una instrucción al bloque if, pero te olvidas de los corchetes ...

```
class Unidad3 {
   public static void main(String[] args) {
      int x = 9;
      int y = 20;
      if (x == 10)
         System.out.println("ciertamente x vale exactamente 10");
         System.out.println("y vale: " + y);
   }
}
```

OJO: LO ANTERIOR ES EQUIVALENTE A

```
class Unidad3 {
   public static void main(String[] args) {
      int x = 9;
      int y = 20;
      if (x == 10) {
         System.out.println("ciertamente x vale exactamente 10");
      }
      System.out.println("y vale: " + y);
   }
}
```

NO ES EQUIVALENTE A

```
class Unidad3 {
   public static void main(String[] args) {
      int x = 9;
      int y = 20;
      if (x == 10) {
         System.out.println("ciertamente x vale exactamente 10");
         System.out.println("y vale: " + y);
      }

   }
}
```

la sentencia if-then-else

De nuevo, podemos prescindir de los brackets si el bloque contiene una instrucción

```
import java.util.Scanner;
public class Unidad3 {
   public static void main(String[] args) {
      Scanner entrada=new Scanner(System.in);
      int x;
      System.out.println("Teclea número entero");
      x=entrada.nextInt();
      if (x== 10){
        System.out.println("x vale 10");
      }else{
         System.out.println("x NO vale 10");
      }
```

```
   }
}
```

Es equivalente a

```
import java.util.Scanner;
public class Unidad3 {
   public static void main(String[] args) {
      Scanner entrada=new Scanner(System.in);
      int x;
      System.out.println("Teclea número entero");
      x=entrada.nextInt();
      if (x== 10)
        System.out.println("x vale 10");
      else
         System.out.println("x NO vale 10");
   }
}
```

Ejercicio U3_B1_E1

Modifica el ejemplo anterior de forma que exista una nueva variable

```
int y;
```

su valor también se pide por teclado y ocurre:

Cuando x vale 10, se imprime (en instrucciones println() diferentes):

- "x vale 10"
- el resultado de la suma x+y

Ejemplo:

```
Teclea número X:
10
Teclea número Y:
5
x vale 10
x +y vale:15
```

cuando x no valga 10 se imprime el mensaje "x NO vale 10",ejemplo:

```
Teclea número X:
4
Teclea número Y:
6
x NO vale 10
```

Ejercicio U3_B1_E2

Modifica el ejercicio anterior de forma que también ocurra que cuando x no valga 10 se imprime el mensaje "x NO vale 10" y el resultado de la operación x-y. Ejemplo:

```
Teclea número X:
4
Teclea número Y:
5
x NO vale 10
x - y vale:-1
```

y si es vale 10 igual que en ejercicio anterior

El ámbito o alcance de una variable local

No es un concepto que tenga que ver estrictamente con if-then-else pero es un buen momento para verlo ya que estamos manejando "bloques" de instrucciones.

Ejecuta el siguiente ejemplo y observa el error que provoca

```
System.out.println("y:" + y);
```

```
class Unidad3 {
```

```
   public static void main(String[] args) {
      int x;
      x=10;
      if (x== 10){
         int y = 20;
         System.out.println("x:" + x + " y: " + y);
         x = y * 2;
         y++;
      }
      System.out.println("x:" + x);
      System.out.println("y:" + y);
   }
}
```

El ámbito o alcance de una variable local está determinado por el bloque en que se define. Si se define una variable en un bloque se puede utilizar en ese bloque y en todos los bloques contenidos en dicho bloque, por ejemplo:

- *x* se puede usar en todo el main(), incluyendo los "sub bloques" que contiene
- *y* sólo es utilizable en el primer bloque del if, fuera de ahí se desconoce.

Ejercicio U3_B1_E3

Ejecuta el siguiente ejemplo y explica por qué da error la instrucción

```
int y=20;
```

```
class Unidad3 {
   public static void main(String[] args) {
      int x,y;
      x=10;
      y=13;
      if (x== 10){
         int y = 20;
         System.out.println("x:" + x + " y: " + y);
         x = y * 2;
         y++;
      }
      System.out.println("x:" + x);
      System.out.println("y:" + y);
   }
}
```

Ejercicio U3_B1_E4 : Busca una solución al ejercicio anterior

Operadores lógicos de cortocircuito.

El operador "and" en java puede formar parte de las siguientes expresiones:

- int & int (el and es a nivel de bit). Y devuelve un int por ejemplo 4&3 vale 0
- boolean & boolean. Devuelve un boolean, por ejemplo true&false vale false
- boolean && boolean. Devuelve un boolean, por ejemplo true&false vale false
- int && int ¡error!. No es posible.

¿Cual es la diferencia entre & y &&?

Java tiene para los operadores lógicos la versión "cortocircuito" (también llamada "perezosa"). Por ejemplo para el "and lógico", tenemos los operadores:

& versión "normal"

&&(versión cortocircuito).

Utilizando la versión cortocircuito hay en situaciones que sólo es necesario evaluar(calcular) el valor de un operando y se puede prescindir de evaluar el segundo pues el resultado con un único operando ya está determinado

La versión “cortocircuito”, se llama así porque si con el primer operando lógico se determina el resultado de la expresión ya no se evalúa el segundo. Por ejemplo
False AND ? = False
donde ? puede ser True o False indistintamente ya que no influye en el resultado. No es necesario conocer el segundo operando para saber el resultado final
El ? representa en el ejemplo anterior cualquier valor (true o false)

De la misma manera se puede aplicar un comportamiento perezoso en

True OR ? = True

Operando 1	Operando 2	and	or
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Ejemplo:
Ejecuta el siguiente código y observa que provoca un error de división por cero en tiempo de ejecución

```
class Unidad3{
   public static void main(String args[]){
      int numerador=10;
      int denominador=0;
      if (false & numerador/denominador == 5){
         System.out.println("esto no se imprimirá nunca por que el primer operando es false");
      }
      System.out.println("chao");
   }
}
```

Con la versión perezosa demostramos que el segundo operando no se evalúa ya que constatamos que no llega a hacerse la división.

```
class Unidad3{
   public static void main(String args[]){
      int numerador=10;
      int denominador=0;
      if (false && numerador/denominador == 5){
         System.out.println("esto no se imprimirá nunca por que el primer operando es false");
      }
      System.out.println("chao");
```

```
    }
}
```

¿Qué versión de operador lógico usar?
En general, se prefieren los operadores cortocircuito ya que "ahorran el tiempo" de evaluación del segundo operando y se prefieren a los operadores "clásicos", no obstante, se mantiene los dos porque puede haber situaciones en las que quiero que obligatoriamente se evalúen los dos operandos.
por ejemplo si en el segundo operando hago una operación de incremento, normalmente querré que se evalúe.
if(false & i++==15) .....
pero son casos raros ...

Un caso concreto de sentencia if -then-else : else if

No es más que un if-the- else, en el que el else contiene if anidados (uno o varios).
En el siguiente ejemplo, fíjate en la sintaxis e indentaciones:

```
class Unidad3{
    public static void main(String args[]){
        int x=3,y=9,z=7;
        if(x>3)
            System.out.println("x es mayor que 3");
        else
            if(y>4)
                System.out.println("y es mayor que 4");
            else
                if(z>5)
                    System.out.println("z es mayor que 5");
    }
}
```

Al haber if anidados unos dentro de otro voy corriendo el sangrado progresivamente. Esta forma de escribir la situación anterior es totalmente lógica y se ve a menudo, no obstante, este tipo de "escaleras", cuando son grandes a menudo tienen problemas de legibilidad y en los programadores java prefieren escribir

```
class Unidad3{
    public static void main(String args[]){
        int x=3,y=9,z=7;
        if(x>3)
            System.out.println("x es mayor que 3");
        else if(y>4)
             System.out.println("y es mayor que 4");
        else  if(z>5)
             System.out.println("z es mayor que 5");
    }
}
```

Esto se puede hacer SÓLO SÍ OCURRE QUE que después del else viene una única instrucción y está es un if(que a su vez puede tener ifs anidados).

La forma standard de escribir el ejemplo anterior es

```
class Unidad3{
    public static void main(String args[]){
        int x=3,y=9,z=7;
        if(x>3){
            System.out.println("x es mayor que 3");
        }else if(y>4){
            System.out.println("y es mayor que 4");
        }else  if(z>5){
            System.out.println("z es mayor que 5");
        }
    }
}
```

Observa que a la parte del if se le añaden brackets por si queremos añadir más instrucciones, pero como indicamos el else no lleva brackets por legiibilidad y teniendo encuenta que sólo lleva un if(que a su vez puede tener if anidados..)

Ejercicio U3_B1_E5

El siguiente código comprueba si el valor de x está entre 1 y 5. Escribe este código al estilo else if de java

```
class Unidad3{
   public static void main(String args[]){
      int x;
      x=3;
      if (x==1)
         System.out.println("x es uno");
      else
         if (x==2)
            System.out.println("x es dos");
         else
            if (x==3)
               System.out.println("x es tres");
            else
               if (x==4)
                  System.out.println("x es cuatro");
               else
                  if (x==5)
                     System.out.println("x es cinco");
                  else
                     System.out.println("x no se encuentra entre uno y cinco");
   }
}
```

DOS ERRORES COMUNES "PELIGROSOS" RELACIONADOS CON IF

Son peligrosos porque aunque "tontos" son difíciles de encontrar para corregir

Error1: Confundir == con =

Ejemplo: prueba y discute

```
class Unidad3{
   public static void main(String args[]){
      int x=3,y=3;
      if(x=y)
         System.out.println("x es igual que y");

    }
}
```

y todavía más peligroso porque no hay error de compilación es el siguiente ejemplo. Imagina que quiero comparar tres variables y escribo erróneamente….

```
class Unidad3{
   public static void main(String args[]){
      int x=5,y=3, z=5;
      if(x==(y=z))
         System.out.println(" las variables x,y,z son iguales ");
      else
         System.out.println(" las variables x,y,z NO son iguales ");
    }
}
```

Y aprovechando el ejemplo anterior, razona por qué son erróneas las siguientes expresiones

x==y==z

x==(y==z)
(x==y)==z

En python una expresión como x==y==z se considera abreviatura de x==y and y==z, pero en java la expresión se evalúa como otra cualquiera. Suponemos x vale 3 y vale 4 y z vale 7:

- Hay dos operadores de misma precedencia (el ==). Luego el primero en evaluarse es el primero a la izquierda
- Como 3==4 es false queda: false==z
- Y sustituir z por su valor queda: false==5 lo que no tiene sentido por que hay mezcla de tipos no permitida

Lo razonamos en JShell

```
jshell> int x=3
x ==> 3

jshell> int y=4
y ==> 4

jshell> int z=7
z ==> 7

jshell> x==y
$4 ==> false

jshell> false==7
|  Error:
|  incomparable types: boolean and int
|  false==7
|  ^------^

jshell>
```

Error2: colocar un ; despues del parentesis del if

```
class Unidad3{
   public static void main(String args[]){
      int x=3;
      if(x==2);
         System.out.println("x es igual a 2");

   }
}
```

si x es igual que 2 ise ejecuta la instrucción vacía!, el println() está fuera del if en el ejemplo anterior.

Observa que si hubiéramos utilizado brackets es más difícil confundirse ya que el error "canta más"

```
class Unidad3{
   public static void main(String args[]){
      int x=3;
      if(x==2){
                  ;
         }
      System.out.println("x es igual a 2");

   }
}
```

Ejercicio U3_B1_E6

Mejora la legibilidad del siguiente programa mejorando los sangrados de las sentencias if.

Escribe una versión sin brackets y otra con brackets

```
import java.util.Scanner;
```

```
class Unidad3 {
   public static void main(String[] args) {
      Scanner teclado= new Scanner(System.in);
      int a,b,s=5;
      System.out.println("a: ");
      a=teclado.nextInt();
      System.out.println("b: ");
      b=teclado.nextInt();

      if(a==0)
      if(b!=0)
      s=s+b;
      else
      s=s+a;
      System.out.println("s: "+s);
    }
}
```

Ejercicio U3_B1_E7. Intenta simplificar el siguiente código. ¿Sobra alguna instrucción?

```
import java.util.Scanner;
class Unidad3{
   public static void main(String[] args) {
      Scanner teclado= new Scanner(System.in);
      int a,b,s=5;
      System.out.println("a: ");
      a=teclado.nextInt();
      System.out.println("b: ");
      b=teclado.nextInt();

      if(a==0)
         if(b!=0)
            s=s+b;
         else
            s=s+a;
      System.out.println("s: "+s);
  }
}
```

Ejercicio U3_B1_E8. Vuelve a escribir el siguiente programa de forma que en el if sólo utilice condiciones simples (suprime las condiciones compuestas con &&)

```
import java.util.Scanner;
class Unidad3{
   public static void main(String[] args) {
      Scanner teclado= new Scanner(System.in);
      int a,b,s=5;
      System.out.println("a: ");
```

```
            a=teclado.nextInt();
            System.out.println("b: ");
            b=teclado.nextInt();

            if(a==0 && b!=0)
                  s=s/b;
            else if (a==0 && b==0)
                  s++;
            System.out.println("s: "+s);
        }
      }
```

Ejercicio U3_B1_E9.

Vuelve a escribir el siguiente programa utilizando condiciones simples en la instrucción if.

```
import java.util.Scanner;
class Unidad3{
   public static void main(String[] args) {
      Scanner teclado= new Scanner(System.in);
      int a,b,s=5; //mejor, es cada variable en su línea pero existe esta posibilidad
      System.out.println("a: ");
      a=teclado.nextInt();
      System.out.println("b: ");
      b=teclado.nextInt();

      if(a==0 && b!=0)
         s=s/b;
      else if(a==0 && b==0)
         ++s;
      else if(a!=0 && b==0)
         s=999+a;
      else if(a!=0&&b!=0)
         s=888+a+b;
      System.out.println("s: "+s);
  }
}
```

Ejercicio U3_B1_E10.

Vuelve a escribir el siguiente código utilizando el operador condicional y sin utilizar condiciones complejas. Pista: utiliza un operador condicional anidado en otro para obtener una solución compacta y fácil.

```
class Unidad3 {
   public static void main(String args[]) {
      int x;//suponemos que x puede tomar valores 1,2 o 3
      x = 3;
      if (x == 1) {
         System.out.println("x es uno");
      } else if (x == 2) {
         System.out.println("x es dos");
      } else {
         System.out.println("x es tres");
```

```
    }
  }
}
```

VARIOS IF SECUENCIALES SIN ANIDAR

Colocar varios if secuencialmente, uno tras otro, es una estructura que también se suele usar mucho. Muchas veces lo que se escribe con else if se puede escribir de esta otra manera. Otras veces no hay equivalencia.

```
class Unidad3{
   public static void main(String args[]){
      int x;
      x=3;
      if (x==1)
         System.out.println("x es uno");
      else if (x==2)
         System.out.println("x es dos");
      else if (x==3)
         System.out.println("x es tres");
      else if (x==4)
         System.out.println("x es cuatro");
      else if (x==5)
         System.out.println("x es cinco");
      else
         System.out.println("x no se encuentra entre uno y cinco");
   }
}
```

EL SIGUIENTE CÓDIGO CONSIGUE EL MISMO EFECTO DE EJECUCIÓN PERO TIENE UNA LÓGICA DIFERENTE. Observa que NO HAY ANIDAMIENTO DE IFs

```
class Unidad3{
   public static void main(String args[]){
      int x;
      x=9;
      if (x==1)
         System.out.println("x es uno");
      if (x==2)
         System.out.println("x es dos");
      if (x==3)
         System.out.println("x es tres");
      if (x==4)
         System.out.println("x es cuatro");
      if (x==5)
         System.out.println("x es cinco");
      if(x<1||x>5)
         System.out.println("x no se encuentra entre uno y cinco");
   }
}
```

Si el problema que queremos resolver se puede escribir de las dos formas es mejor usar else ya que la intención del código es más clara y es más eficiente. Prueba con el tracer de bluej los dos códigos anteriores y comprueba la diferencia de condiciones evaluadas con x=1

En el siguiente ejemplo quiero expresamente demostrar que dado el valor de un número en algunos casos se pueden ejecutar muchos if no sólo uno. Es el típico ejemplo para resolver con bucles pero aquí lo hacemos con if para observar en un ejemplo simple esta estructura.

```
import java.util.Scanner;
class Unidad3 {
   public static void main(String[] args) {
      Scanner sc= new Scanner(System.in);
      System.out.println("introduce un número del 1 al 9: ");
      int numero=sc.nextInt();
      System.out.print("sus antecesores son: ");
      if(0<numero){
```

```
            System.out.print(0+" ");
        }
        if(1<numero){
            System.out.print(1+" ");
        }
        if(2<numero){
            System.out.print(2+" ");
        }
        if(3<numero){
            System.out.print(3+" ");
        }
        if(4<numero){
            System.out.print(4+" ");
        }
        if(5<numero){
            System.out.print(5+" ");
        }
        if(6<numero){
            System.out.print(6+" ");
        }
        if(7<numero){
            System.out.print(7+" ");
        }
        if(8<numero){
            System.out.print(8+" ");
        }

    }
}
```

Ejercicio U3_B1_E11. Escribe el siguiente código con if secuenciales

```
import java.util.Scanner;
class Unidad3 {
    public static void main(String[] args) {
        //suponemos 3 enteros distintos
        Scanner sc= new Scanner(System.in);
        System.out.println("Mete 3 números separados: ");
        int a=sc.nextInt();
        int b=sc.nextInt();
        int c=sc.nextInt();
        int menor;
        if(a<b && a<c){
            menor=a;
        }else if (b<c && b<a){
            menor=b;
        }else{
            menor=c;
        }
        System.out.println("El menor: "+ menor);

    }
}
```

COMBINACIÓN IF/RETURN

Para simplificar la lógica del flujo de control de programa, se suele a menudo utilizar en los métodos el hecho de que un return obliga a concluir la ejecución del método lo que ayuda a evitar el uso de *else* o condiciones muy largas y complicadas. Por ejemplo el método sonBuenaPareja() devuelve true cuando los números que recibe son ambos pares, o bien son las parejas 3,5 o 5,6

```
class Unidad3 {
```

```
    static boolean sonBuenaPareja(int a, int b) {
        if (a % 2 == 0 && b % 2 == 0) {
            return true;
        }
        if (a == 3 && b == 5) {
            return true;
        }
        if (a == 5 && b == 6) {
            return true;
        }
        return false;

    }

    public static void main(String args[]) {

        System.out.println(sonBuenaPareja(4, 6));
        System.out.println(sonBuenaPareja(3, 5));
        System.out.println(sonBuenaPareja(5, 6));
        System.out.println(sonBuenaPareja(11, 12));
        System.out.println(sonBuenaPareja(13, 27));

    }
}
```

Ejercicio U3_B1_E12. Escribe un método que calcula el máximo de dos formas, una valiéndose del return múltiple y otra con un único return. El main

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

        System.out.println(max1(4, 6, 5));
        System.out.println(max1(9, 6, 5));
        System.out.println(max1(1, 6, 88));

        System.out.println(max2(4, 6, 5));
        System.out.println(max2(9, 6, 5));
        System.out.println(max2(1, 6, 88));

    }
```

EJERCICIOS PARA HACER CON CODE RUNNER

(ver cuestionario en moodle)

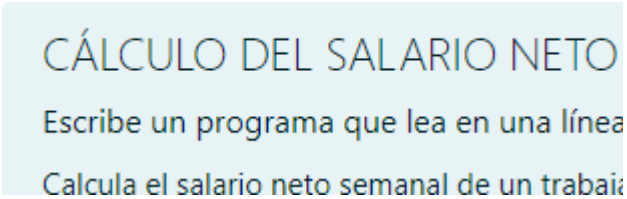
INTRO A USAR PSEUDOCÓDIGO

El pseudocódigo es una especie de lenguaje "light" de especificación de algoritmos. Se puede utilizar como un primer borrador del programa en la fase de diseño, para "perfilar" el código fuente, centrándose en la lógica y los puntos de control de éste sin tener que ceñirse a las restricciones sintácticas de un lenguaje de programación. El pseudocódigo es una herramienta de "boli y papel" PARA AYUDAR en la concepción de la solución. Si se convierte en una molestia, mejor no usarlo.

Hay muchos enfoques de uso de pseudocódigo pero nosotros por el momento lo vamos a usar sólo para hacer un "borrador" del programa.

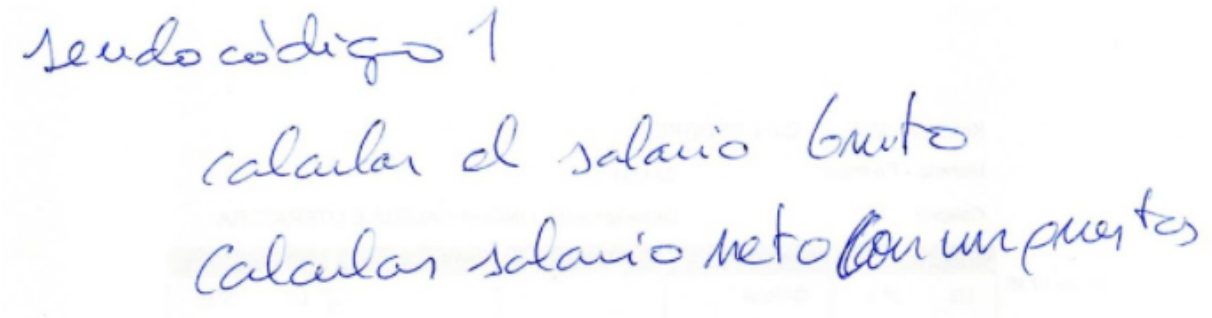
Nos va a ser útil, cuando la lógica o solución nos resulta difícil y al trabajar con boli y papel podemos mezclar con las variables e instrucciones dibujos, anotaciones etc. También puede ser útil cuando el programa es un poco largo y nos ayuda a estructurar la solución.

EJEMPLO DE PSEUDOCÓDIGO PARA EL EJERCICIO DE SALARIO NETO DE CODERUNNER

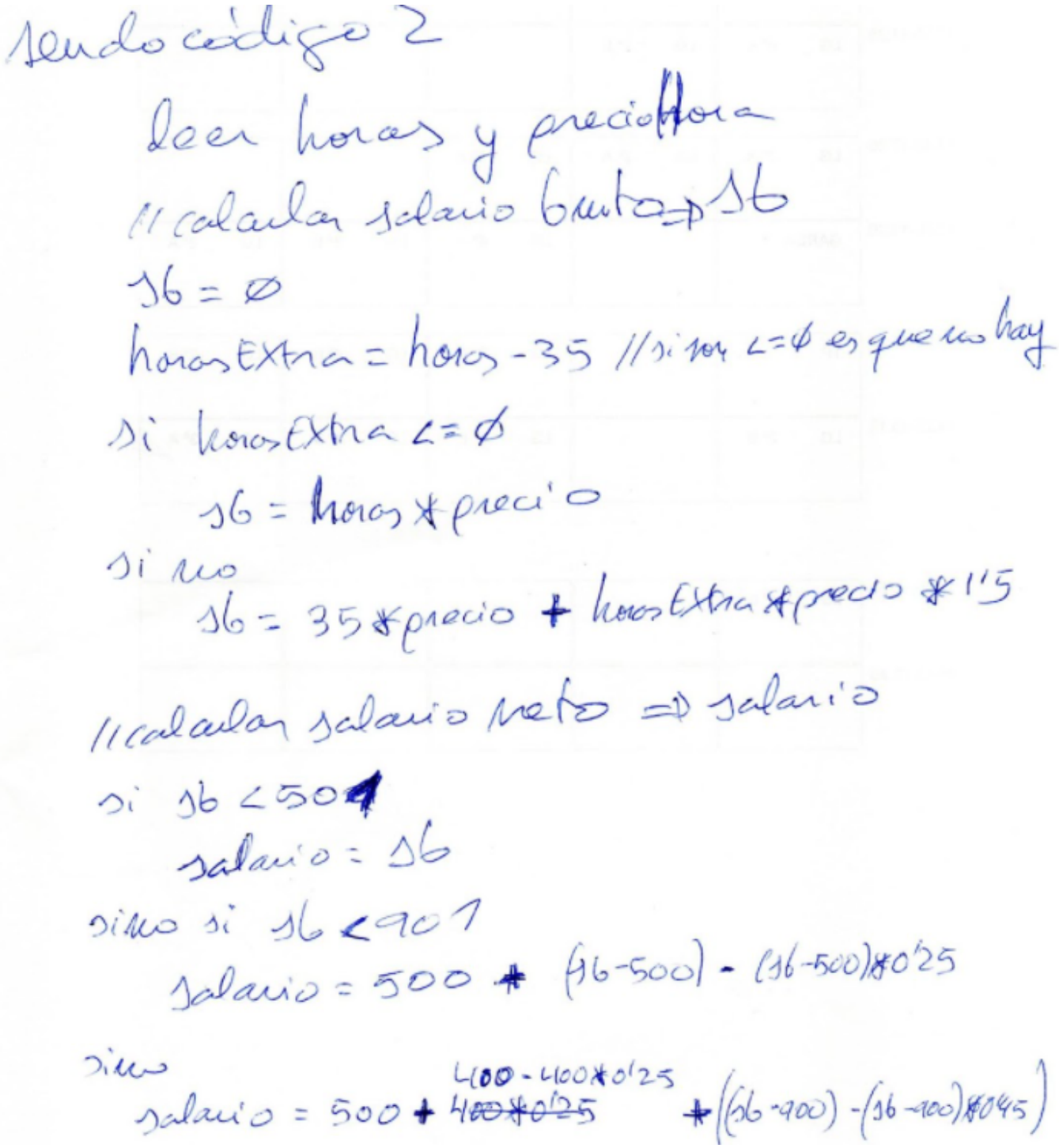


En estos momentos, el ejercicio del salario del cuestionario nos resulta largo y usar pseudocódigo nos puede ayudar a arrancar.

Cogo un boli y un papel y esta es mi primera versión del programa con pseudocódigo



Otra persona a lo mejor ya hubiera pensado algo más detallado, pero a mi esto es lo primero que se me vino a la cabeza y me sirvió para "arrancar". Parece que ya tengo al menos la idea más básica de lo que tengo que hacer. El pseudocódigo anterior son frases del lenguaje normal "castellano". Ahora que ya me situé un poco en el problema voy a pensar a continuación de forma un poco más cercana a la máquina usando variables, asignaciones, etc. pero de forma informal, es decir, de nuevo usando pseudocódigo, sólo que escribo un pseudocódigo más cercano a la solución final



Ahora ya tengo en mi mente como voy a resolver el problema y pasaría a escribirlo en Java.