lee rápido la parte que corresponde al while del siguiente enlace
http://manuais.iessanclemente.net/index.php/Elementos_esenciais_da_linguaxe_Java#Control_de_fluxo

Al hacer los ejercicios de este boletín se puedes consultar rápidamente las soluciones aunque no te salgan puesto que más adelante comenzaremos a hacer ejercicios de bucles con CodeRunner y en la plataforma acepta el reto.

Ejercicio U3_B3_1:
Imprime, usando while, los números del 0 al 10

Ejercicio U3_B3_2:
Imprime, usando while, los números pares que se encuentran entre 0 y 10.

Bucles infinitos

Son bucles con infinitas iteraciones. En consola para parar un programa que entra en un bucle infinito: ctrl-c
Ejemplo: ejecuta y para el siguiente programa

```
class Unidad3{
  public static void main(String args[]){
    while(true)
          System.out.println("Hasta el fin de los tiempos....");
  }
}
```

Aunque ahora pueda resultar sorprendente, los bucles infinitos se utilizan en diversas situaciones de forma muy útil, también hay que tener en cuenta que podemos generar un bucle infinito de forma no intencionada si escribimos mal el código del bucle haciendo por tanto que el programa se "cuelgue" debido a que jamás se llega a cumplir la condición de fin de bucle, normalmente porque el contador que suele ir en la condición del bucle se incrementa incorrectamente o no se incrementa.

Ejercicio U3_B3_3 Una solución incorrecta al ejercicio anterior es la siguiente, ejecuta el código y explica por qué es erróneo. **Utiliza el tracer de BlueJ para**

investigar el problema

```
class Unidad3{
   public static void main(String[] args) {
      int i=0;
      while(i<=10)
         if(i%2==0)
            System.out.println((i++)+" es par");
    }
}
```

Ejercicio U3_B3_4: Imprime las letras minúsculas del abecedario utilizando un bucle while.

Ten en cuenta lo que ya sabes: un char es un tipo especial de entero de 16 bits sin signo (valores de 0 a 65536). Según UNICODE ese entero estará asociado a un carácter. Tratar a un carácter como un entero sin signo, permite hacer algunas operaciones aritméticas como sumas y restas. En la tabla unicode las letras minúsculas del abecedario son números enteros contiguos. El caracter 'a' se corresponde con el entero 97.

Al estudiar downcasting vimos excepciones que se aparten de la norma genérica, por ejemplo podíamos asignar una constante int a una variable byte si estaba en rango

byte b=127;//OK 127 está en rango

También había algunas particularidades en la relación char/int. Ahora fíjate en la siguiente:

char c='a'; // c contiene el entero 97 asociado al carácter 'a'

c++; //ahora c contiene el entero 98, o sea el carácter 'b'

System.out.println(c); //imprime el caracter 'b'

Por lo tanto podemos hacer c++ pero siendo equivalente

c=c+1;

nos da un error de tipos ya que c+1 pasa a ser una expresión entera de 32 bits

Podemos pensar que o *c++* es en este caso no da error porque es equivalente a *c=(char)(c+1)*

bucles anidados

Se puede escribir un bucle en el interior de otro, es decir, podemos anidar un

bucle en otro. Hablamos en este caso del bucle exterior y del bucle interior o anidado.

Ejemplo:

```
class Unidad3 {
   public static void main(String[] args) {
      int i=1;
      while(i<4){
         System.out.println("iteración bucle exterior número " +i);
         int j=1;
         while(j<3){
            System.out.println("\titeración bucle interior número " +j);
            j++;
         }
         i++;
      }
   }
}
```

Ya sabes la manía de los informáticos de empezar a contar en 0. EL código anterior es más común encontrarlo comenzando i en 0 y aumentar en 1 su valor al imprimirlo. Lo mismo para j.

```
class Unidad3 {
   public static void main(String[] args) {
      int i=0;
      while(i<3){
         System.out.println("iteración bucle exterior número " +(i+1));
         int j=0;
         while(j<2){
            System.out.println("\titeración bucle interior número " +(j+1));
            j++;
         }
         i++;
      }
   }
}
```

NO ESTÁ ESCRITO EN NINGÚN LADO PERO…

En los bucles "que cuentan" por diversos motivos y por tradición:

- La variable que cuenta comienza en 0 en lugar de en 1
- La condición del bucle se razona con < en lugar de <=

Así es como hacen la mayoría de los programadores, lo mejor es hacer como ellos para que cuando consultes sus ejemplos te sientas cómodo y no tengas que andar "traduciendo" la lógica de la comunidad a tu lógica. Y dicho esto, hay excepciones por un tubo a esta pseudo norma.

Ejercicio U3_B3_5

Vuelve a escribir el ejemplo anterior de forma que los contadores i y j se incrementen en la condición del while pero se obtenga exactamente el mismo resultado que con el código anterior.

Ejercicio U3_B3_6. Usando while, escribe un programa que va leyendo palabras hasta que introducimos la palabra "fin". Al acabar de introducir palabras las imprime todas.

```
E:\Programacion>java Unidad3
Palabra:
hola
Palabra:
adios
Palabra:
chao
Palabra:
fin
lista de palabras introducidas:   hola  adios  chao

E:\Programacion>
```

El método hasNext() en la condición de bucle.

Este método lo vamos a tener que usar de vez en cuando en cierto tipo de ejercicios, concretamente en ejercicios de la plataforma "acepta el reto". Nos interesa pues saber cómo funciona y su relación con la entrada por teclado.

Las condiciones de bucles pueden ser todo lo complejas que queramos, por ejemplo pueden contener invocaciones a métodos, lo único que deben cumplirse es que la condición debe ser una expresión de tipo boolean, es decir, una expresión que al evaluarse es true/false, por lo tanto emplear un método que devuelve true/false será un recurso muy utilizado.

En el siguiente ejemplo se utiliza en la condición del while el método hasNext() que devuelve false cuando no quedan más tokens que analizar.

```
import java.util.Scanner;
public class Unidad3{
   public static void main(String[] args) {
      Scanner sc = new Scanner("CHAO adios \n DEICALOGO goodbye");
      while (sc.hasNext()) {
         System.out.println(sc.next());
      }
   }
}

run:
CHAO
adios
DEICALOGO
goodbye
```

¿cuando da false hasNext()?

Los métodos como hasNext() suelen utilizarse cuando se lee la información de un fichero. Usaremos ficheros más adelante. Con ficheros hasNext() devuelve false cuando se encuentra la marca de fin de fichero. Con teclado se simula la marca fin de fichero si tecleamos:

- Enter + ctrl-d en linux
- Enter + ctrl-z en windows
- En las consolas virtuales de los IDE pueden no funcionar las combinaciones anteriores . Estamos hablando de consolas físicas.

Las combinaciones anteriores pueden tener variantes según versiones etc..

```
import java.util.Scanner;
public class Unidad3{
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      while (sc.hasNext()) {
         System.out.println("Repito: "+sc.next());
      }
   }
```

```
}
```

```
lozano@infol06:~/programacion$ java Unidad3
hola a todos
Repito: hola
Repito: a
Repito: todos
voy a escribir enter y luego ctrl-d
Repito: voy
Repito: a
Repito: escribir
Repito: enter
Repito: y
Repito: luego
Repito: ctrl-d
lozano@infol06:~/programacion$
```

Ejercicio U3_B3_7. Se introducen por teclado una serie de palabras y al terminar de teclear se indica al usuario la cantidad de palabras introducidas. Se indica el fin del tecleo con enter+ ctrl-z (en windows, en linux es enter + ctrl-d). Por lo tanto el usuario puede teclear un número indeterminado de palabras.

```
PS C:\Users\donlo\programacion> java Unidad3
teclea palabras separadas por delimitador
hola
que tal
estais todos
^Z
palabras introducidas: 5
PS C:\Users\donlo\programacion>
```