Taller de preparación para la OCI - PUC

For

1. Introducción

En este capítulo veremos el manejo de ciclos con `for`. Junto con el `while`, son las formas más comunes para manejar ciclos en la mayoría de los lenguajes de programación.

Hay ocasiones en que nos gustaría que una acción se repitiera un número determinado de veces, pero a la vez nos gustaría llevar un contador de las iteraciones que llevamos hasta el momento. Como han visto la clase anterior, esto ya se puede hacer con `while`. Sin embargo, el `for` permite esto mismo con una sintaxis más clara y concisa.

Los casos de uso más comunes del `for` son:

- Repetir una acción un número definido de veces
- Generar una sucesión de números, de 1 en 1, de 2 en 2, etc.

Estos no son los únicos casos. Con la experiencia verás en qué otras situaciones puedes usar el `for`.

2. Estructura del `for`

La estructura general del `for` consiste en:

```
1 for(<Seteo inicial de variable>; <Condición para continuar>; <Incremento>) {
2   // Código que se ejecuta en cada iteración.
3 }
```

El **seteo inicial** es una sentencia de Java, como lo es una asignación de un valor a una variable. Se ejecuta sólo una vez antes de la primera iteración. Acá podemos configurar el valor inicial de alguna variable que queramos usar dentro del `for`[1]. Es importante notar que esta variable sólo existe dentro del bucle, y cuando éste termine la variable será eliminada.

La **condición para continuar** es una expresión booleana. El siguiente loop se ejecuta sólo si esta condición sea verdadera. Esta condición puede involucrar la variable creada en la **condición inicial**.

Finalmente, tenemos el **incremento**. Esta parte está destinada a alterar el valor de la variable que declaramos en el **seteo inicial**.

Partiremos con un ejemplo donde vamos a imprimir números en cierto intervalo.

Ejemplo 2.1. Queremos imprimir en pantalla todos los números del 0 al 9:

```
1 for(int i = 0; i < 10; i++) {
2   System.out.println("Vamos en el número " + i);
3 }
```

Lo primero que sucede en el Ejemplo 2.1 es que se crea la variable `i` con valor 0. Luego, para ejecutar el primer loop se verifica que la condición `i <10` sea verdadera. Como `i` vale 0 en ese momento la condición se cumple, por lo que se ejecuta el contenido del `for`. Así, lo primero en imprimirse es `"Vamos`

[1] Después veremos que esta estructura es aún más general, pero por ahora nos quedaremos con esto.

`en el número 0"`. Cuando el contenido del `for` ya se ejecutó, se ejecuta el incremento, por lo que `i` aumenta en 1.

El mismo procedimiento se repite hasta que `i` valga 10, pues no es cierto que 10 sea menor que 10. Por lo tanto, el loop deja de ejecutarse. Finalmente, el output completo del Ejemplo 2.1 es:

```
Vamos en el número 0
Vamos en el número 1
Vamos en el número 2
Vamos en el número 3
Vamos en el número 4
Vamos en el número 5
Vamos en el número 6
Vamos en el número 7
Vamos en el número 8
Vamos en el número 9
```

Ejemplo 2.2. El mismo resultado se puede alcanzar con el siguiente código que usa `while`:

```
1  int i = 0;
2  while (i < 10) {
3    System.out.println("Vamos en el número " + i);
4    i++;
5  }
```

Es importante que, al igual que con el `while`, es posible usar código más complejo dentro del `for`, como uso de condiciones `if-else` y otras más. Esto se muestra en el Ejemplo 2.3.

Ejemplo 2.3. Un profesor quiere hacer una prueba de 20 preguntas en las que se pregunta si un número es o no es par. El siguiente código pregunta al usuario 20 veces si un número es par o no, y verifica la respuesta dada por el usuario.

```
1  Scanner scanner = new Scanner(System.in);
2  for (int i = 0; i < 20; i++) {
3    System.out.println("Es el " + i + " par? Si: 1, 2: No");
4    int answer = scanner.nextInt();
5    if (answer == 1 && i % 2 == 1) {
6      System.out.println("Incorrecto!");
7    }
8    else if (answer == 2 && i % 2 == 0) {
9      System.out.println("Incorrecto!");
10   }
11   else {
12     System.out.println("Correcto!");
13   }
14 }
```

Ejercicio 2.1. Piensa cómo podrías mejorar el código del ejemplo 2.3. Puedes pensar en cómo hacer el código más conciso o cómo manejar si el usuario da un número distinto a 1 o 2 como respuesta.

El incremento no es necesariamente de 1 en 1. Puede ser en otra cantidad, incluso, ser un decremento. Al igual que en el `while`, hay que tener cuidado de no dejar loops infinitos o de que los loops nunca se inicien. Las situaciones están expuestas en los Ejemplos 2.4, 2.5 y 2.6 respectivamente.

Ejemplo 2.4. El siguiente código hace lo mismo que el Ejemplo 2.1, pero de dos en dos:

```
1  for(int i = 0; i < 10; i += 2) {
2    System.out.println("Vamos en el número " + i);
3  }
```

Ejemplo 2.5. El siguiente código cuenta hacia atrás, pero no termina nunca ¿Por qué?

```
1  for(int i = 0; i < 10; i--) {
2    System.out.println("Vamos en el número " + i);
3  }
```

Ejemplo 2.6. El siguiente código intenta contar hacia atrás, pero no tiene output ¿Por qué?

```
1  for(int i = 10; i < 10; i--) {
2    System.out.println("Vamos en el número " + i);
3  }
```

Ejercicio 2.2. Crea un código que cuente desde el 10 al 0.

3. Intrucciones `break` y `continue`

Al igual que en el `while`, se pueden utilizar las instrucciones `break` y `continue` dentro del loop. Recuerda que el `break` sale del loop inmediatamente, no importando si quedaban loops por hacer. En cambio, el `continue` se salta todo el código que falta por ejecutar en cierta iteración y comienza con la siguiente inmediatamente. En los siguientes ejemplos veremos aplicaciones directas de aquellas instrucciones.

Ejemplo 3.1. Acá hacemos lo mismo que en el Ejemplo 2.4, pero a la primera respuesta incorrecta el loop termina. Esto es porque se lee la instrucción `break` en esos casos.

```
1  Scanner scanner = new Scanner(System.in);
2  for (int i = 0; i < 20; i++) {
3    System.out.println("Es el " + i + " par? Si: 1, 2: No");
4    int answer = scanner.nextInt();
5    if (answer == 1 && i % 2 == 1) {
6      System.out.println("Incorrecto! Mala suerte, se acabó");
7      break;
8    }
9    else if (answer == 2 && i % 2 == 0) {
10     System.out.println("Incorrecto! Mala suerte, se acabó");
11     break;
12   }
13   else {
14     System.out.println("Correcto!");
15   }
16 }
```

Ejemplo 3.2. Mallory es un/una alumno/a que te está pidiendo ayuda para terminar una tarea de 20 preguntas. Él/Ella te pregunta si tienes la respuesta a un ítem de la tarea, y si contestas que sí, te pregunta la respuesta y te da las gracias. Este programa simula el diálogo entre Mallory y tú.

```java
Scanner scanner = new Scanner(System.in);
int items = 20;
for (int i = 1; i <= 20; i++) {
  System.out.println("Tienes la respuesta del ítem " + i + "? 1: Si, 2: No");
  int has = scanner.nextInt();
  if(has == 2) {
    continue;
  }
  System.out.println("Por favor, dame la respuesta");
  String answer = scanner.nextLine();
  System.out.println("Gracias!");
}
```

El ejemplo funciona como se espera porque si el usuario responde que **no**, entonces alcanza la instrucción `continue`, que lleva al loop a la siguiente iteración sin tomar en cuenta el código que viene después.

4. Profundizando un poco más

Como ya hemos mostrado, la forma más común de usar un `for` es inicializando un contador, tener como condición de término del loop que el contador alcance cierto valor y actualizar ese valor al final de cada iteración. Sin embargo, el `for` es mucho más flexible que eso. Recordemos la estructura del `for` vista antes:

```java
for(<Seteo inicial de variable>; <Condición para continuar>; <Incremento>) {
  // Código que se ejecuta en cada iteración.
}
```

En realidad el `for` es mucho más versátil. La estructura general es:

```java
for(<Ejecución pre-for>; <Condición para continuar>; <Ejecución final iteración>) {
  // Código que se ejecuta en cada iteración.
}
```

Cada sección puede tener código arbitrario. Además, ninguno de los campos es obligatorio. De hecho, si la condición para continuar no se especifica se considera como `true`. Veremos varios ejemplos de posibles casos más abajo.

Ejemplo 4.1. Este `for` no especifica la condición para continuar, por lo tanto, siempre se considera verdadera.

```java
for (int i = 0; ; i++) {
  System.out.println("Vamos en el número " + i);
}
```

Ejemplo 4.2. Este `for` no especifica nada. Sólo imprime hola por siempre.

```java
for ( ; ; ) {
  System.out.println("Hola");
}
```

Ejemplo 4.3. En este código, se imprimen los números desde el 0 hasta que el usuario decide terminar. Este `for` tiene como condición para continuar un valor booleano que no involucra la variable inicializada.

```
1  Scanner scanner = new Scanner(System.in);
2  boolean stop = false;
3  for (int i = 0; !stop; i++) {
4    System.out.println(i);
5    System.out.println("Quieres continuar contando? 1: Si, 2: No")
6    if (scanner.nextInt() == 2) {
7      stop = true;
8    }
9  }
```

Ejemplo 4.4. En este ejemplo veremos un programa que recibirá un input más complejo (tal como lo hicimos en el capítulo de `while`). Se hará un programa que multiplique varios pares de números. Se debe pedir al usuario que ingrese un número n que represente cuantos pares de números desea sumar. Se debe pedir al usuario n veces dos números a y b. Para cada par de números a y b se debe imprimir la multiplicación.

```
1  import java.util.Scanner;
2
3  public class Main {
4    public static void main(String[] args) {
5
6    Scanner scanner = new Scanner(System.in);
7
8    System.out.println("Cuantos pares de números desea sumar? ");
9    int n = scanner.nextInt();
10
11   int a;
12   int b;
13   for (int i = 0; i < n; i++) {
14     System.out.println("Ingrese a: ");
15     a = scanner.nextInt();
16     System.out.println("Ingrese b: ");
17     b = scanner.nextInt();
18     int suma = a+b;
19     System.out.println("La suma vale: " + suma);
20   }
21   }
22 }
```