Ciclos

Programación de Computadoras II

Abdel G. Martínez L.

Agenda

1. Introducción
2. Generando Tablas
3. Encapsulación y Generalización
4. La Sentencia ’for’
5. El Ciclo ’do-while’
6. Las Sentencias ’break’ y ’continue’

Introducción

- Las computadores se utilizan usualmente para hacer tareas repetitivas. Repetir taeeas sin equivocarse es algo que las computadoras hacen bien y las personas pueden equivocarse.
- Ejecutar el mismo código múltiples veces se le conoce como iteración. Existen métodos que utilizan la recursión para iterar.
- A pesar que la recursión pueda ser poderosa y elegante, toma tiempo aprender e implementar.
- Java provee características que permite que las iteraciones sean mucho más sencillas utilizando las sentencias *while* y *for*.

La Sentencia 'while'

- Veamos el siguiente ejemplo:

```
public static void conteo(int n) {
   while (n > 0) {
         System.out.println(n);
         n = n - 1;
   }
   System.out.println("Boom");
}
```

- Las sentencias *while* se pueden leer como cualquier sentencia en idioma natural: "*Mientras n sea mayor a cero, ejecuta imprime su valor y redúcelo en 1. Cuando la condición ya no se cumpla, entonces imprime el mensaje Boom*".

La Sentencia ’while’

- La expresión ubicada entre paréntesis se le conoce como **condición.** Las sentencias entre llaves son el **cuerpo.**
- El flujo de ejecución de una sentencia *while* es la siguiente:
  - Evaluar la condición, arrojando verdadero o falso.
  - Si la condición es falsa, omitir el cuerpo e ir a la siguiente sentencia.
  - Si la condición es cierta, ejecutar el cuerpo y volver al paso 1.
- Este tipo de flujo es un **ciclo**, porque el último paso itera de vuelta al primer paso.
- Un **ciclo infinito** es aquel ciclo donde el cuerpo del ciclo no cambia el valor de la variable evaluada en la condición.

Generando Tablas

- Los ciclos son buenos para generar y mostrar data tabular.
- El siguiente ejemplo muestra la generación de una secuencia para una tabla de logaritmos:

```
int i = 1;
while (i < 10) {
   double x = (double) i;
   System.out.println(x + ”
” + Math.log(x));
   i = i + 1;
}
```

```
1.0   0.0
2.0   0.6931471805599453
3.0   1.0986122886681098
4.0   1.3862943611198906
5.0   1.6094379124341003
6.0   1.791759469228055
7.0   1.9459101490553132
8.0   2.0794415416798357
9.0   2.1972245773362196
```

Encapsulación y Generalización

- La encapsulación y generalización es otro proceso de desarrollo de programas que consta de los siguientes pasos:
  1. Escribir una pocas líneas de código en el método main, luego probarlas.
  2. Cuando funcionen, encerrarlas en un nuevo método, y re-probarlos.
  3. Si es apropiado, reemplazar valores literales con variables y parámetros.
- El paso 2 se le conoce como encapsulación y el paso 3 se le conoce como generalización.
- El problema principal siempre es identificar cómo dividir el programa en métodos. El proceso de encapsulación y generalización permiten diseñarlo.

La Sentencia ’for’

- Los ciclos tienen elementos en común. Ellos inicializan una variable, tienen una condición que dependen de esa variable, y dentro del ciclo se actualiza la variable.
- La sentencia for tiene 3 componentes en paréntesis, separados por punto y coma: el inicializador, la condición y la actualización.

Inicializador

- Se ejecuta una vez al inicio de cada ciclo.

Condición

- Es validado por cada iteración en el ciclo. Si es falso, el ciclo termina.

Actualización

- Al final de cada iteración, se ejecuta y vuelve a la condición.

La Sentencia ’for’

- Existe una diferencia entre los ciclos for y while: si uno declara una variable en el inicializador, solamente existe dentro del ciclo for.
- Ejemplo:

```
public static void imprimirFila(int n, int col) {
   for (int i = 1; i <= cols; i = i + 1) {
        System.out.println(”%4d”, n * i);
   }
   System.out.println(i); // Error de compilacion
}
```

- La última línea no muestra el valor de i.

El Ciclo ’do-while’

- Las sentencias while y for son ciclos pre-evaluados, donde se valida la condición al principio y al inicio de cada iteración.
- La sentencia do-while es un ciclo post-evaluados.
- Este tipo de ciclo es útil cuando se necesita ejecutar el cuerpo del ciclo al menos una vez.

El Ciclo ’do-while’

```
Scanner in = new Scanner(System.in);
boolean okay;
do {
    System.out.print("Enter a number: ");
    if (in.hasNextDouble()) {
        okay = true;
    } else {
        okay = false;
        String word = in.next();
        System.err.println(word + " is not a number");
    }
} while (!okay);
double x = in.nextDouble();
```

Las Sentencias ’break’ y ’continue’

- Algunas veces los ciclos pre-evaluados y post-evaluados no proveen lo que uno necesita. Se evalúa la necesidad en medio de la iteración.
- Normalmente se utiliza una variable bandera junto a una sentencia if-else anidada.
- Sin embargo, una manera fácil de resolver este problema es utilizar la sentencia *break*. Cuando se ejecuta esta sentencia, entonces se sale del ciclo actual.
- Adicionalmente, existe una sentencia *continue* que se mueve a la siguiente iteración, sin salir del ciclo mismo.

Las Sentencias ’break’ y ’continue’

```
Scanner in = new
Scanner(System.in);
while (true) {
    System.out.print("Enter a
number: ");
    if (in.hasNextDouble()) {
        break;
    }
    String word = in.next();
    System.err.println(word + "
is not a number");
}
double x = in.nextDouble();
```

```
Scanner in = new
Scanner(System.in);
int x = -1;
int sum = 0;
while (x != 0) {
    x = in.nextInt();
    if (x <= 0) {
        continue;
    }
    System.out.println("Adding "
+ x);
    sum += x;
}
```

¡HASTA LA PRÓXIMA CLASE!

Tema: Arreglos