Métodos con Valores

Programación de Computadoras II

Abdel G. Martínez L.

Agenda

1. Valores para ’return’
2. Escribiendo Métodos
3. Composición de Métodos
4. Sobrecargar Métodos (Overloading)
5. Métodos Booleanos
6. Etiquetas Javadoc
7. Otro Ejemplo de Recursividad

Valores para ’return’

- Cuando invocamos un método *void*, la invocación es usualmente todo en una línea, sin esperar un valor de vuelta:

```
public static void conteo(int n) {
   if (n == 0) {
        System.out.println(”Boom!”);
   } else {
        conteo(n-1);
        System.out.println(n);
   }
}
```

- A continuación la invocación:

```
conteo(3);
```

Valores para ’return’

- Cuando invocamos un método que retorna un valor, se tiene que hacer algo con el valor retornado. Usualmente, eso se asigna a una variable o forma parte de una expresión.
- Comparado con los métodos void, los métodos por valor difieren en dos maneras:
  - Declaran el tipo de datos a retornar por la sentencia *return*.
  - Utilizan al menos una sentencia *return* dentro del bloque de código.

Valores para ’return’

- Ejemplos:

```
public static double calcularArea(double radio) {
   double resultado = Math.PI * radio * radio;
   return resultado;
}

public static double calcularArea(double radio) {
   return Math.PI * radio * radio;
}

double resultado = calcularArea(25.0);
```

Escribiendo Métodos

- Un problema típico es escribir mucho código antes de compilar y ejecutarlo. Este alcance novato requiere mucha depuración.
- Recomiendo utilizar un alcance de **desarrollo incremental**:
  - Iniciar con un programa funcional y pequeño, realizando cambios incrementales. Ante cualquier error, se sabe qué línea de código revisar.
  - Utilizar variables para almacenar, de forma intermedia, los valores para poder verificarlos, imprimirlos o depurarlos.
  - Una vez el programa esté funcionando se puede consolidar múltiples sentencias en expresiones compuestas.

Escribiendo Métodos

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Primer paso:**

```
public static double distancia
     (double x1, double y1, double x2, double y2)
{
     return 0.0
}
```

Escribiendo Métodos

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Segundo paso:**

```
public static double distancia
      (double x1, double y1, double x2, double y2)
{
      double dx = x2 - x1;
      double dy = y2 - y1;
      System.out.println("dx = " + dx);
      System.out.println("dy = " + dy);
      return 0.0;
}
```

Escribiendo Métodos

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Tercer paso:**

```
public static double distancia
      (double x1, double y1, double x2, double y2)
{
      double dx = x2 - x1;
      double dy = y2 - y1;
      double dsquared = dx * dx + dy * dy;
      System.out.println("dsquared = " + dsquared);
      return 0.0;
}
```

Escribiendo Métodos

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Tercer paso:**

```
public static double distancia
      (double x1, double y1, double x2, double y2)
{
      double dx = x2 - x1;
      double dy = y2 - y1;
      double dsquared = dx * dx + dy * dy;
      double resultado = Math.sqrt(dsquared);
      return resultado;
}
```

Composición de Métodos

- Una vez se define un nuevo método, se puede utilizar como parte de una expresión o para construir nuevos métodos utilizando métodos existentes.
- El uso de variables temporales es útil para desarrollar y depurar, pero cuando el programa ya es funcional debemos ser concisos en la composición de los llamados de los métodos.
- La **descomposición funcional** consiste en romper computación compleja en métodos simples, probar los métodos de forma aislada y luego componer los métodos para realizar la computación. Este proceso facilita el mantenimiento.

Composición de Métodos

```
public static double areaCirculo
   (double xc, double yc, double xp, double yp) {
     double radio = distancia(xc,yc,xp,yp);
     double area = calcularArea(radio);
     return area;
}

public static double areaCirculo
   (double xc, double yc, double xp, double yp) {
     return calcularArea(distancia(xc,yc,xp,yp));
}
```

Sobrecargar Métodos (Overloading)

- Cuando dos métodos hacen lo mismo, es natural darle el mismo nombre. Tener más de un método con el mismo nombre se le conce como **overloading**.
- En Java, esta práctica es válida siempre y cuando se mantengan cada versión con diferentes lista de parámetros.
- Muchos métodos propios de Java son sobrecargados, significando que existen diferentes versiones que aceptan diferentes números y tipos de parámetros.
- Por ejemplo, en la clase Math existe una versión del método abs que funciona con *double*, mientras que otra funciona con *int*.

Sobrecargar Métodos (Overloading)

```
public static double calcularArea
                    (double x, double y) {
    return x * y;
}

public static double areaCirculo(double x) {
    return Math.PI  * x * x;
}
```

Métodos Booleanos

- Los métodos pueden retornar valores boolean.
- Son convenientes para esconder pruebas dentro de métodos.
- Ejemplo:

```
public static boolean esUnicoDigito(int x) {
     if (x > -10 && x < 10) {
          return true;
     } else {
          return false;
     }
}
```

Métodos Booleanos

- Se puede optimizar la cantidad de líneas del ejemplo anterior:

```
public static boolean esUnicoDigito(int x) {
    return x > -10 && x < 10;
}
```

- En el método principal, se puede invocar el método normalmente:

```
System.out.println(esUnicoDigito(2));
boolean digitoGrande = !esUnicoDigito(17);
```

Etiquetas Javadoc

- Los comentarios para escribir documentación utilizan /**.
- La idea es documentar cada clase y método, para que los otros programadores puedan entender lo que hace el código sin leerlo.
- Para organizar cada sección dentro de la documentación, Javadoc soporta etiquetas opcionales que inician con un símbolo de arroba (@). Por ejemplo, se puede usar @param y @return para proveer información adicional sobre los parámetros y valores a retornar.
- Los métodos con múltiples parámetros deben tener varias etiquetas @param separadas para describir cada parámetro.
- Los métodos void no deben tener etiqueta @return.

Etiquetas Javadoc

```
/**
 * Tests whether x is a single digit integer.
 *
 * @param x the integer to test
 * @return true if x has one digit, false otherwise
 */
public static boolean isSingleDigit(int x) {
```

isSingleDigit

`public static boolean isSingleDigit(int x)`

Tests whether x is a single digit integer.

Parameters:

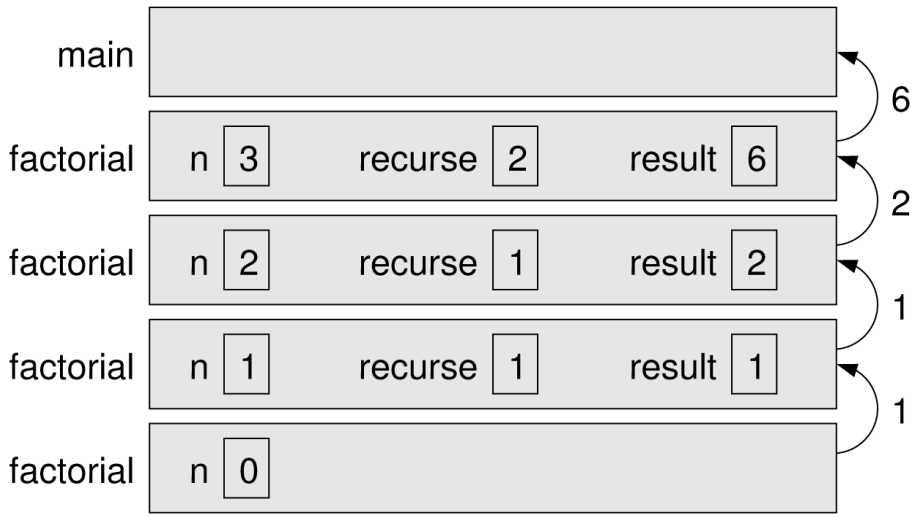
x - the integer to test

Returns:

true if x has one digit, false otherwise

Otro Ejemplo de Recursividad

```
public static int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    int recurse = factorial(n - 1);
    int result = n * recurse;
    return result;
}
```



¡HASTA LA PRÓXIMA CLASE!

Tema: Ciclos