7. Instrucciones Condicionales

7.1. Instrucciones Condicionales Simples

Una instrucción condicional nos permite plantear la solución a un problema considerando los distintos casos que se pueden presentar. De esta manera, podemos utilizar un algoritmo distinto para enfrentar cada caso que pueda existir en el mundo. Considere el método de la clase Producto que se encarga de vender una cierta cantidad de unidades presentes en la bodega. Allí, se pueden presentar dos casos posibles, cada uno con una solución distinta: el primer caso es cuando la cantidad que se quiere vender es mayor que la cantidad disponible en la bodega (el pedido es mayor que la disponibilidad) y el segundo es cuando hay suficientes unidades del producto en la bodega para hacer la venta. En cada una de esas situaciones la solución es distinta y el método debe tener un algoritmo diferente.

Para construir una instrucción condicional, se deben identificar los casos y las soluciones, usando algo parecido a la tabla que se muestra a continuación:

Caso 1:

Expresión que describe el caso:

```
cantidad > cantidadBodega
```

Algoritmo para resolver el problema en ese caso:

```
// Vende todas las unidades disponibles
totalProductosVendidos += cantidadBodega;
cantidadBodega = 0;
```

Caso 2:

Expresión que describe el caso:

```
cantidad <= cantidadBodega
```

Algoritmo para resolver el problema en ese caso:

```
// Vende lo pedido por el usuario
totalProductosVendidos += cantidad;
cantidadBodega  -=  cantidad;
```

En el primer caso la solución es vender todo lo que hay en la bodega. En el segundo, vender lo pedido como parámetro. En Java existe la instrucción condicional `if-else`, que permite expresar los casos dentro de un método. La sintaxis en Java de dicha instrucción se ilustra en el siguiente fragmento de programa:

```
public class Producto
{
  ...
  public void vender( int cantidad )
  {
    if( cantidad > cantidadBodega )
    {
      totalProductosVendidos += cantidadBodega;
      cantidadBodega  =  0;
    }
    else
    {
      totalProductosVendidos += cantidad;
      cantidadBodega -= cantidad;
    }
  }
}
```

- En lugar de una sola secuencia de instrucciones, se puede dar una secuencia para cada caso posible. El computador sólo va a ejecutar una de las dos secuencias.
- Es como si el método escogiera el algoritmo que debe utilizar para resolver el problema puntual que tiene, identificando la situación en la que se encuentra el objeto.
- La condición caracteriza los dos casos que se pueden presentar. Note que con una sola condición debemos separar los dos casos.
- Los paréntesis alrededor de la condición son obligatorios.
- La condición es una expresión lógica, construida con operadores relacionales y lógicos.
- La parte del "`else`", incluida la segunda secuencia de instrucciones, es opcional. Si no se incluye, eso querría decir que para resolver el segundo caso no hay que hacer nada.

La instrucción `if-else` tiene tres elementos:

1. Una condición que corresponde a una expresión lógica capaz de distinguir los dos casos (su evaluación debe dar verdadero si se trata del primer caso y falso si se trata del segundo).
2. La solución para el primer caso.
3. La solución para el segundo caso. Al encontrar una instrucción condicional, el computador evalúa primero la condición y decide a partir de su resultado cuál de las dos soluciones ejecutar. Nunca ejecuta las dos.

Si el algoritmo que resuelve uno de los casos sólo tiene una instrucción, es posible eliminar los corchetes, como se ilustra en el ejemplo 11. Allí también se puede apreciar que una instrucción condicional es sólo una instrucción más dentro de la secuencia de instrucciones que implementan un método.

Ejemplo 11

En este ejemplo se presentan algunos métodos de la clase Producto, para mostrar la sintaxis de la instrucción `if-else` de Java. Los métodos aquí presentados no son necesariamente los que escribiríamos para implementar los requerimientos funcionales del caso de estudio, pero sirven para ilustrar distintos aspectos de las instrucciones condicionales.

```
public boolean haySuficiente( int cantidad )
{
  boolean suficiente;
  if( cantidad <= cantidadBodega )
    suficiente  =  true;
  else
    suficiente = false;
    return suficiente;
}
```

Como la secuencia de instrucciones de cada caso tiene una sola instrucción, se pueden eliminar los corchetes.

Fíjese que dejamos el resultado de cada caso en la misma variable, de manera que al hacer el retorno del método siempre se encuentre allí el resultado. ¿Qué hace este método? Una instrucción condicional se puede ver como otra instrucción más del método. Puede haber instrucciones antes y después de ella.

El método anterior también se podría escribir de esta manera, un poco más sencilla. ¿Qué ganamos escribiéndolo así?

```
public boolean haySuficiente( int cantidad )
{
  return cantidad <= cantidadBodega;
}
```

Si en el segundo de los casos de una instrucción condicional no es necesario hacer nada, no se debe escribir ninguna instrucción, tal como se muestra en el ejemplo. ¿Está claro el problema que resuelve el método?

```
public double darPrecioPapeleria( boolean conIVA )
{
  double  precioFinal  =  valorUnitario;
  if( conIVA )
    precioFinal = precioFinal *( 1+IVA_PAPEL );
  return precioFinal;
}
```

En este método, si se han vendido menos de 100 unidades, se hace un descuento del 20% en el precio del producto.

Si se han vendido 100 o más unidades, se aumenta en un 10% el precio. En las instrucciones condicionales, incluso si sólo hay una instrucción para resolver cada para facilitar la lectura del código. En algunos casos, incluso, son indispensables para evitar ambigüedades.

```
public void ajustarPrecio( )
{
  if( totalProductosVendidos < 100 )
  {
    valorUnitario = valorUnitario * 80 / 100;
  }
  else
  {
    valorUnitario = valorUnitario * 1.1;
  }
}
```

> Tenga cuidado de no escribir un ";" después de la condición, porque el computador lo va a interpretar como si la solución al caso fuera no hacer nada (una instrucción vacía).

7.2 Condicionales en Cascada

Cuando el problema tiene más de dos casos, es necesario utilizar una cascada (secuencia) de instrucciones `if-else`, en donde cada condición debe indicar sin ambigüedad la situación que se quiere considerar. Suponga por ejemplo que queremos calcular el IVA de un producto. Puesto que el valor que se paga de impuestos por un producto depende de su tipo, es necesario considerar los tres casos siguientes:

Caso 1

Expresión que describe el caso:

```
( tipo == SUPERMERCADO )
```

Algoritmo para resolver el problema en ese caso:

```
return IVA_MERCADO;
```

Caso 2

Expresión que describe el caso:

```
( tipo == DROGUERIA )
```

Algoritmo para resolver el problema en ese caso:

```
return IVA_FARMACIA;
```

Caso 3

Expresión que describe el caso:

```
( tipo == PAPELERIA )
```

Algoritmo para resolver el problema en ese caso:

```
return IVA_PAPEL;
```

El método de la clase Producto para determinar el IVA que hay que pagar sería de la siguiente forma (no es la única solución, como veremos más adelante):

```
public double darIVA( )
{
  if( tipo == PAPELERIA )
  {
    return IVA_PAPEL;
  }
  else if( tipo == SUPERMERCADO )
  {
    return IVA_MERCADO;
  }
  else
  {
    return IVA_FARMACIA;
  }
}
```

- Para representar los tres casos posibles, utilizamos una instrucción condicional en el

"else" del primer caso. Esa manera de encadenar las instrucciones condicionales para poder considerar cualquier número de casos se denomina "en cascada".

- Una instrucción condicional puede ir en cualquier parte donde pueda ir una instrucción del lenguaje Java. Esto lo retomaremos en capítulos posteriores.

```
public double darIVA( )
{
  double resp = 0.0;
  if( tipo == PAPELERIA )
  {
    resp = IVA_PAPEL;
  }
  else if( tipo == SUPERMERCADO )
  {
    resp = IVA_ MERCADO;
  }
  else
  {
    resp = IVA_ FARMACIA;
  }
  return resp;
}
```

En esta segunda solución del método, en lugar de hacer un retorno en cada caso, guardamos la respuesta en una variable y luego la retornamos al final.

> Al usar varias instrucciones if en cascada hay que tener cuidado con la ambigüedad que puede surgir con la parte else. Es mejor usar siempre corchetes para asegurarse de que el computador lo va a interpretar de la manera adecuada.

Tarea 5

Objetivo: Practicar el uso de las instrucciones condicionales simples para expresar el cambio de estado que debe hacerse en un objeto, en cada uno de los casos identificados.

Escriba el código de cada uno de los métodos descritos a continuación. Tenga en cuenta la clase en la cual está el método y la información que se entrega como parámetro.

Para la clase: Producto

Aumentar el valor unitario del producto, utilizando la siguiente política: si el producto cuesta menos de $1000, aumentar el 1%. Si cuesta entre $1000 y $5000, aumentar el 2%. Si cuesta más de $5000 aumentar el 3%.

```
public double subirValorUnitario( )
{









}
```

Recibir un pedido, sólo si en bodega se tienen menos unidades de las indicadas en el tope mínimo. En caso contrario el método no debe hacer nada.

```
public void hacerPedido( int cantidad )
{









}
```

Modificar el precio del producto, utilizando la siguiente política: si el producto es de droguería o papelería debe disminuir un 10%. Si es de supermercado debe aumentar un 5%.

```
public void cambiarValorUnitario( )
{









}
```

Para la clase: Tienda

Vender una cierta cantidad del producto cuyo nombre es igual al recibido como parámetro. El método retorna el número de unidades efectivamente vendidas. Suponga que el nombre que se recibe como parámetro corresponde a uno de los productos de la tienda. Utilice el método vender de la clase Producto como parte de su solución.

```
public int venderProducto( String nombreProducto, int cantidad )
{









}
```

Calcular el número de productos de papelería que se venden en la tienda.

```
public int cuantosPapeleria( )
{









}
```

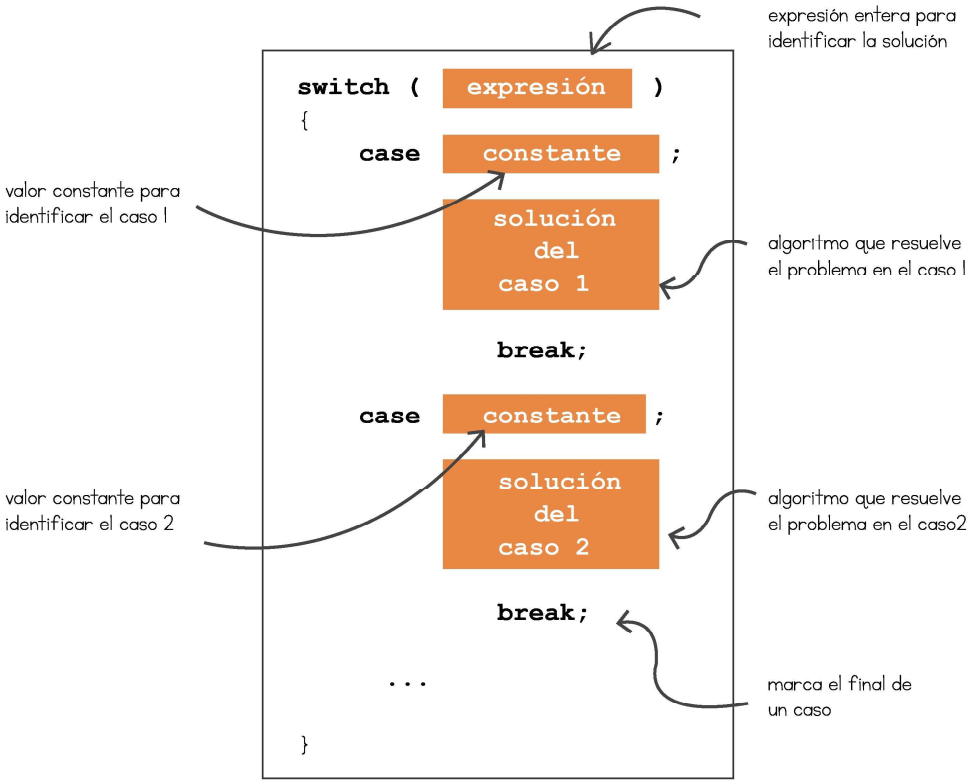
7.3. Instrucciones Condicionales Compuestas

Una instrucción condicional compuesta (`switch`) es una manera alternativa de expresar la solución de un problema para el cual existe un conjunto de casos, cada uno con un algoritmo distinto para resolverlo. Esta instrucción tiene la restricción de que cada caso debe estar identificado con un valor entero.

En la instrucción `switch` va inicialmente una expresión entera entre paréntesis, cuya evaluación va a servir para identificar el caso que se está presentando. Después de dicha expresión, se introduce la solución de cada uno de los casos identificados, usando la

instrucción case y el valor entero constante que lo identifica. Al final del bloque de instrucciones que implementa la solución de un caso se debe utilizar la instrucción break, antes de arrancar el siguiente. En la figura 2.7 se ilustra la estructura de una instrucción condicional compuesta. En el ejemplo 12 se presenta la solución del método que calcula el IVA de un producto, usando una instucción condicional compuesta.

Fig. 2.7 Estructura de la instrucción switch de Java



Ejemplo 12

Objetivo: Ilustrar el uso de instrucciones condicionales compuestas.

En este ejemplo se presenta el método que calcula el IVA que debe pagar un producto, dependiendo de su tipo.

```
public double darIVA( )
{
  double iva = 0.0;
  switch( tipo )
  {
    case PAPELERIA:
      iva = IVA_PAPEL;
    break;
    case SUPERMERCADO:
      iva = IVA_MERCADO;
    break;
    case DROGUERIA:
      iva = IVA_FARMACIA;
    break;
  }
  return iva;

}
```

- Este método de la clase Producto tiene tres casos posibles, cada uno identificado con un valor constante entero (candidato ideal para la instrucción switch). Recuerde que esa es una característica indispensable para poder utilizar esta instrucción.
- La expresión que va a permitir distinguir los casos se construye simplemente con el atributo "tipo".
- Cada caso se introduce con la palabra reservada de Java "case" y se cierra con un "break". Después de la instrucción "case" va el valor que identifica el caso. En nuestro ejemplo, el valor se identifica con las constantes que representan los tres tipos posibles de productos: PAPELERIA, SUPERMERCADO o DROGUERIA.

Dado que siempre es posible escribir una instrucción condicional compuesta como una cascada de condicionales simples, la pregunta que nos debemos hacer es ¿cuándo usar una instrucción condicional compuesta? La respuesta es que siempre que se pueda utilizar la instrucción `switch` en lugar de una cascada de `if` es conveniente hacerlo, por dos razones:

1. Eficiencia, ya que de este modo sólo se evalúa una vez la expresión aritmética, mientras que en la cascada se evalúan una a una las condiciones, descartándolas.
2. Claridad en el programa, porque es más fácil de leer y mantener un programa escrito de esta manera.

Y la segunda pregunta es, ¿cuándo no intentar usar una instrucción condicional compuesta? La respuesta es: cuando los casos no están identificados por valores enteros. Si se tienen, por ejemplo, los nombres de los productos como identificadores de los casos, la única opción es usar una cascada de instrucciones `if-else`, como veremos en la tarea que sigue.

Tarea 6

Objetivo: Utilizar instrucciones condicionales para expresar un conjunto de casos y soluciones asociadas con los mismos.

Escriba el código de cada uno de los métodos descritos a continuación. Tenga en cuenta la clase en la cual está el método y la información que se entrega como parámetro.

Para la clase: Producto

Dar el nombre del tipo del producto. Por ejemplo, si el producto es de tipo SUPERMERCADO, el método debe retornar la cadena: "El producto es de supermercado".

```
public String nombreTipoProducto( )
{









}
```

Aumentar el precio del producto, siguiendo esta regla: si es un producto de droguería debe aumentar el 1%, si es de supermercado el 3% y si es de papelería el 2%.

```
public void aumentarValorUnitario( )
{









}
```

Para la clase: Tienda

Retornar el precio final del producto identificado con el número que se entrega como parámetro. Por ejemplo, si numProd es 3, debe retornar el precio del tercer producto (`p3`). Suponga que el valor que se entrega como parámetro es mayor o igual a 1 y menor o igual a 4.

```
public double darPrecioProducto( int numProd )
{









}
```

Este método debe hacer lo mismo que el anterior, pero en lugar de recibir como parámetro el número del producto, recibe su nombre. Puede suponer que el nombre que se entrega como parámetro corresponde a un producto perteneciente a la tienda.

```
public double darPrecioProducto( String nomProd )
{









}
```