Programación Imperativa

Manuel Collado

Programación II

Septiembre 2013

Índice general

Bases de la Programación Imperativa

Aquí se resumen algunos conceptos básicos de la **Programación Imperativa**, como preparación para introducir los conceptos de **Programación Orientada a Objetos**.

Fundamentos de la Programación Imperativa

- Un programa se expresa como una colección de órdenes o **instrucciones**, con indicación de en qué orden hay que ejecutarlas.
- Se pueden almacenar valores en **variables**. Se puede modificar el valor de una variable durante la ejecución del programa.

Fundamentos de la Programación Imperativa

Instrucciones

- Son las **acciones** que debe ir ejecutando el computador.
- Se codifican como **sentencias** (*statements*) en el lenguaje de programación.
- Pueden ser instrucciones **simples** o **compuestas**.

Fundamentos de la Programación Imperativa

Instrucciones simples

- Son acciones que se expresan como una única sentencia del programa.
- Instrucciones simples típicas:
  - **Asignación**: Realizar un cálculo y almacenar el resultado en una variable.
  - **Entrada-Salida**: Leer o escribir un dato simple.

Fundamentos de la Programación Imperativa

Estructuras de control

- Corresponden a acciones compuestas, que se describen como una colección de acciones más sencillas.
- Las estructuras de control básicas son:
  - **Secuencia**: Realizar una tras otra una serie de acciones, en el orden indicado.
  - **Selección**: Realizar una acción entre varias posibles, dependiendo de ciertas condiciones.
  - **Iteración**: Repetir una acción cierto número de veces, dependiendo de una condición.

Fundamentos de la Programación Imperativa

Datos simples

- Son **valores elementales**, que no se descomponen en partes.
- Ejemplos típicos:
  - Valores numéricos **enteros**.
  - Valores **reales**, con parte fraccionaria.
  - Valores **booleanos**.
  - Valores de tipo **carácter**.
- Cada lenguaje de programación tiene un repertorio limitado de tipos de datos simples.

Fundamentos de la Programación Imperativa

Estructuras de datos

- Son una agrupación lógica de datos más sencillos, bien datos simples u otras estructuras.
- Estructuras básicas:
  - **Agregados**: colección formada por un número fijo de datos, cada uno con un nombre, que pueden ser de tipos diferentes.
  - **Formaciones** (*arrays*): colección numerada de datos, todos del mismo tipo. Dependiendo del lenguaje de programación el número de componentes se mantiene fijo, o bien puede variar durante la ejecución del programa.
- **Cadenas de caracteres** (*strings*): Son un caso particular de *array* de caracteres. Hay lenguajes de programación que los tratan de forma diferente al resto de los *arrays*.

Fundamentos de la Programación Imperativa

Ejemplo

- Fecha: día, mes año
- Nombre completo: apellido1, apellido2, nombre

Desarrollo por refinamiento progresivo

La técnica de **refinamiento** consiste en expresar inicialmente el programa a desarrollar como una acción global, que si es necesario se irá descomponiendo en acciones más sencillas hasta llegar a acciones simples que puedan ser expresadas directamente como sentencias del lenguaje de programación.
En otros contextos este modo general de desarrollo de designa como "*divide y vencerás*", y también como desarrollo descendente (*top-down*).

Desarrollo por refinamiento progresivo

Refinamiento progresivo

1. Plantear el programa como una operación global.
2. Si la operación puede codificarse directamente, hacerlo. Fin del desarrollo.
3. En caso contrario, si la operación es compleja:
   1. Identificar operaciones más sencillas.
   2. Identificar la manera de combinar las operaciones componentes para conseguir el efecto global.

- El paso 3 se repite mientras haya operaciones componentes que todavía no puedan codificarse directamente, para cada una de ellas.
- El paso 3.2 se hará atendiendo a las estructuras de control disponibles en el lenguaje de programación.

Desarrollo por refinamiento progresivo

Esquemas de refinamiento

- **Secuencia**: Una operación compleja se plantea como un número fijo de operaciones parciales a realizar una tras otra, en un orden fijo.
- **Selección**: Una operación compleja se plantea como la ejecución de una única operación parcial elegida entre varias posibles, dependiendo de ciertas condiciones.
- **Iteración**: Una operación compleja se plantea como la repetición de una misma operación más sencilla, tantas veces como sea necesario para completar el proceso.

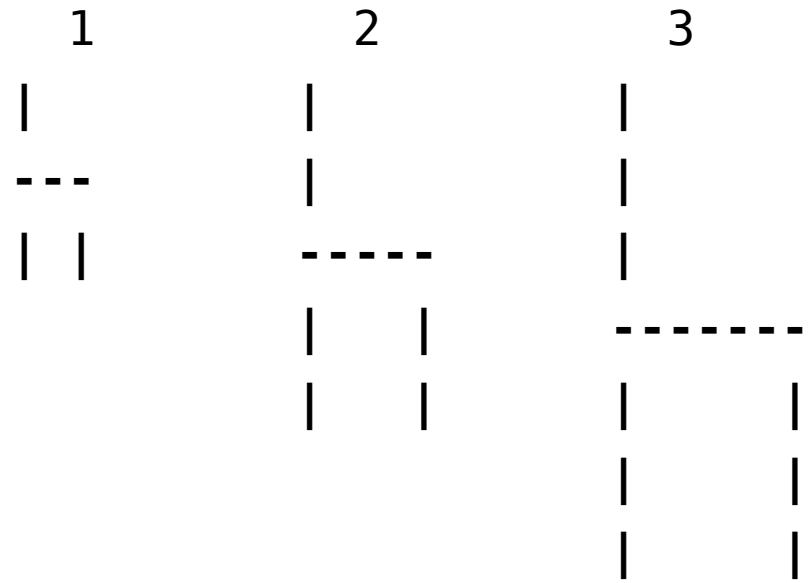
Desarrollo por refinamiento progresivo
Ejemplo (I)

- Imprimir la silueta de una silla con caracteres de texto:

```
 |
---
| |
```

Desarrollo por refinamiento progresivo
Ejemplo (II)

- Imprimir la silueta de una silla del tamaño que se indique:



Desarrollo por refinamiento progresivo
Ejemplo (III)

- Imprimir una pirámide de asteriscos de la altura que se indique:

```
3
   *
  ***
 *****
```

Desarrollo por refinamiento progresivo
Ejemplo (IV)

- Imprimir un triángulo de asteriscos de la altura que se indique:

```
4
   *
  * *
 *   *
* * * *
```

El mecanismo de abstracción

En esencia, la idea de abstracción consiste en plantear algo de manera global o general, atendiendo sólo a lo esencial y prescindiendo de los detalles.
En el contexto de programación, la abstracción consiste en codificar por separado ciertos elementos de un programa que tengan sentido en sí mismos, dándoles un nombre. De esta manera cuando haya que usar esos elementos bastará hacer referencia a ellos por su nombre, con lo que es más fácil tener una visión global del programa, evitando que *"los árboles no dejen ver el bosque"*.
El mecanismo de abstracción puede aplicarse a:

- **operaciones**, mediante **abstracciones funcionales**.
- **datos**, planteándolos como **tipos abstractos de datos** (TADs).

El mecanismo de abstracción

Subprogramas

- Corresponden a operaciones que se codifican por separado, dándoles un nombre.
- Para mandar ejecutar la operación simplemente se hace referencia a ella por su nombre, en lugar de escribir el código completo cada vez.
- Un subprograma tiene una **interfaz** y una **implementación**:
  - La interfaz especifica simplemente **qué hace** esa operación.
  - La implementación contiene el código completo que **realiza** la operación.
- Para invocar la operación basta conocer la interfaz. No hace falta saber los detalles de implementación.

El mecanismo de abstracción

Funciones

- Son operaciones de **cálculo**, que obtienen un valor como **resultado** de operar con ciertos **argumentos**.
- La interfaz abstracta especifica el nombre de la función, el tipo del resultado, y el tipo de cada argumento. Debe completarse con una descripción del significado de esa operación:

```
/** Máximo de dos valores */
int maximo2( int valor1, int valor2 )
```

- La función se invoca usándola como un término de una expresión aritmética:

```
alturaTotal = maximo2( altura1, altura2 );
```

El mecanismo de abstracción

Procedimientos

- Son **acciones**, cuya ejecución puede invocarse como una orden en un programa.
- Las acciones pueden estar parametrizadas, es decir, pueden tener **argumentos** o parámetros.
- La interfaz abstracta especifica el nombre de la operación y el tipo de cada argumento, si los hay. Debe completarse con una descripción del significado de esa operación:

```
/** Escribir un texto n veces seguidas */
void escribirN( String texto, int n )
```

- El procedimiento se invoca como una orden del programa:

```
escribirN( "*", 2*k-1 );
```

El mecanismo de abstracción

Refinamiento progresivo mediante abstracciones

1. Plantear el programa como una operación global.
2. Si la operación puede codificarse directamente, hacerlo. Fin del desarrollo.
3. **Si la operación tiene sentido en sí misma, plantearla como subprograma.**
4. En otro caso:
   1. Identificar operaciones más sencillas.
   2. Identificar la manera de combinar las operaciones componentes para conseguir el efecto global.
5. El proceso se repite hasta que no queden operaciones sin codificar o desarrollar como subprogramas.

El mecanismo de abstracción

Ejemplo (I)

- Imprimir una pirámide de asteriscos de la altura que se indique:

```
3
  *
 ***
*****
```

-
```
{ /*--- Imprimir una pirámide de la altura indicada */
    for (int k=1; k<=altura; k++) {
        /*--- Imprimir altura-k espacios */
            ...
        /*--- Imprimir 2k-1 asteriscos */
            ...
        /*--- Saltar de línea */
            ...
    }
}
```

El mecanismo de abstracción (...)
Ejemplo (I)

El mecanismo de abstracción
Ejemplo (II)

```
/*--- Escribir un texto n veces seguidas */
void escribirN( String texto, int n ) {
    for (int k=1; k<=n; k++) {
        System.out.print( texto );
    }
}

{ /*--- Imprimir una pirámide de la altura indicada */
    for (int k=1; k<=altura; k++) {
        escribirN( " ", altura - k );  //-- espacios
        escribirN( "*", 2*k - 1 );     //-- asteriscos
        System.out.println();         //-- fin de línea
    }
}
```

El mecanismo de abstracción
Ejemplo (III)

- Desarrollar un programa que lea la hora de comienzo y final de una serie de actividades, y escriba una línea por cada una de ellas indicando la hora de comienzo, hora de final y la duración.
  - Definir un tipo de dato que permita almacenar valores de tiempo dado por hora y minuto. Servirá tanto para designar una hora del día como una duración en tiempo.
  - Desarrollar el programa mediante refinamiento progresivo, identificando operaciones que convenga plantear como subprogramas.

El mecanismo de abstracción

Ejemplo (IV)

Un ejemplo de ejecución del programa por consola sería el siguiente, donde los datos de entrada aparecen en negrita:

```
10 30 12 15
10:30 12:15 1h45m
17 0 22 14
17:00 22:14 5h14m
23 15 0 31
23:15 00:31 1h16m
15 0 19 0
15:00 19:00 4h0m
```

Como puede verse, los valores de tiempo deben poder escribirse de dos maneras diferentes, según se considere que son una hora del día (`12:34`) o una duración (`12h34m`).

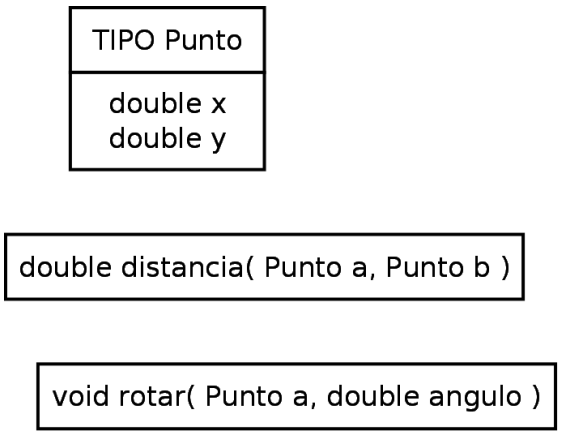
El mecanismo de abstracción

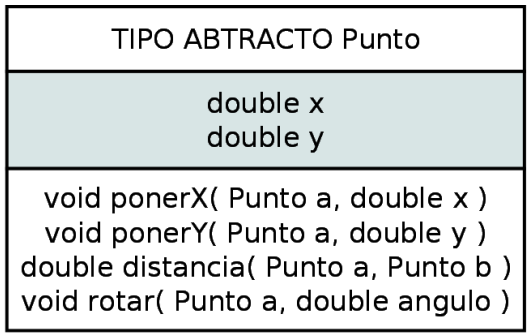
Tipos abstractos de datos

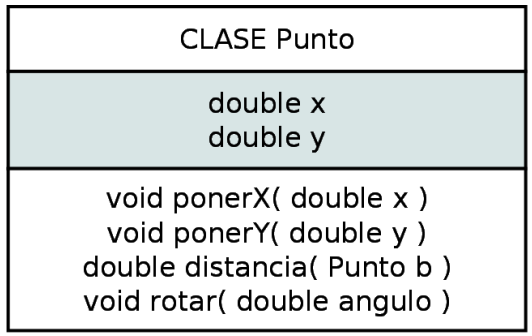
- El concepto de abstracción puede aplicarse no sólo a las operaciones sino también a los datos.
- En un tipo abstracto de datos se prescinde del detalle de cómo se representan los valores de los datos, y se atiende sólo a las operaciones que pueden realizarse con esos datos.
- Las valores internos del dato no pueden ser vistos o modificados directamente. Hay que disponer de operaciones para leer o escribir esos valores, si hace falta.
- El lenguaje Java permite definir tipos abstractos de datos mediante el mecanismo de clases. Hay un cambio de notación: las operaciones tienen implícitamente un primer argumento del tipo abstracto.
  - Tipo abstracto ---> Clase
  - Operaciones ---> Métodos

El mecanismo de abstracción

Tipos vs. tipos abstractos vs. clases








```
puntoA.x = 3.3
puntoA.y = 4.4
rotar( puntoA, 3.1416 )
```

```
ponerX( puntoA, 3.3 )
ponerY( puntoA, 4.4 )
rotar( puntoA, 3.1416 )
```

```
puntoA.ponerX( 3.3 )
puntoA.ponerY( 4.4 )
puntoA.rotar( 3.1416 )
```

El mecanismo de abstracción

Tipos vs. clases (I)

- Tipo de datos (tradicional):
  - El tipo define solamente el contenido de información.
  - Las operaciones se plantean por separado.
  - Las operaciones se invocan por sí solas.
    operacion(argumento1, argumento2, ...)
- Clases (Programación Orientada a Objetos):
  - La clase define el contenido de información (**atributos**).
  - La clase define también las operaciones (**métodos**).
  - Las operaciones se invocan sobre un objeto.
    objeto . operacion(argumento2, ...)

El mecanismo de abstracción

Tipos vs. clases (II)

```
class Punto {      ____________TIPO_____________
  double x;
  double y;
}
                   ___________APLICACION_________
static double distancia( Punto p1, Punto p2) {...}
static void rotar( Punto p1, double angulo ) {...}
    ..... main() .....
Punto a, b;
double d;
a.x = 14.0;
a.y = 20.5;
d = distancia( a, b );
rotar( a, 0.5*pi );
```

El mecanismo de abstracción

Tipos vs. clases (III)

```
class Punto {      ____________CLASE_____________
  private double x;
  private double y;
  public Punto( double x; double y ) {...}

  public double distancia( Punto p2) {...}
  public void rotar( double angulo ) {...}
}
                   __________APLICACION__________
Punto a, b;
double d;
a = new Punto(14.0, 20.5);
d = a.distancia( b );
a.rotar( 0.5*pi );
```