funciones, argumentos, clases y paquetes. Asignamos nombres a archivos y a directorios, a archivos jar, war y ear. Usamos nombres constantemente. Por ello, debemos hacerlo bien. A continuación, veremos algunas reglas básicas para crear nombres correctos.

Usar nombres que revelen las intenciones

Es fácil afirmar que los nombres deben revelar nuestras intenciones. Lo que queremos recalcar es la *importancia* de hacerlo. Elegir nombres correctos lleva tiempo, pero también ahorra trabajo. Por ello, preste atención a los nombres y cámbielos cuando encuentre otros mejores. Todo el que lea su código se lo agradecerá.

El nombre de una variable, función o clase debe responder una serie de cuestiones básicas. Debe indicar por qué existe, qué hace y cómo se usa. Si un nombre requiere un comentario, significa que no revela su cometido.

```
int d; // tiempo transcurrido en días
```

El nombre `d` no revela nada. No evoca una sensación de tiempo transcurrido, ni de días. Debe elegir un nombre que especifique lo que se mide y la unidad de dicha medida:

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

La elección de nombres que revelen intenciones facilita considerablemente la comprensión y la modificación del código. ¿Para qué sirve el siguiente código?

```
public List<int[]> getThem() {
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
     if (x[0] == 4)
       list1.add(x);
   return list1;
}
```

¿Por qué es complicado saber la función de este código? No hay expresiones complejas. Los espacios y el sangrado son razonables. Sólo hay tres variables y dos constantes. Ni siquiera contiene clases complejas o

métodos polimórficos, sólo una lista de matrices (o eso parece).

El problema no es la simplicidad del código sino su carácter *implícito*: el grado en el que el contexto no es explícito en el propio código. Implícitamente, el código requiere que sepamos las respuestas a las siguientes preguntas:

- ¿Qué contiene `theList`?
- ¿Qué significado tiene el subíndice cero de un elemento de `theList`?
- ¿Qué importancia tiene el valor `4`?
- ¿Cómo se usa la lista devuelta?

Las respuestas a estas preguntas no se encuentran en el código, pero se podrían haber incluido. Imagine que trabaja en un juego de buscar minas. El tablero es una lista de celdas llamada `theList`. Cambiemos el nombre por `gameBoard`.

Cada celda del teclado se representa por medio de una matriz. El subíndice cero es la ubicación de un valor de estado que, cuando es `4`, significa que se ha detectado. Al asignar nombres a estos conceptos mejoramos considerablemente el código:

```
public List<int[]> getFlaggedCells() {
   List<int[]> flaggedCells = new ArrayList<int[]>();
   for (int[] cell : gameBoard)
     if (cell[STATUS_VALUE] == FLAGGED)
       flaggedCells.add(cell);
   return flaggedCells;
}
```

La simplicidad del código no ha cambiado. Sigue teniendo los mismos operadores y constantes y el mismo número de niveles anidados, pero ahora es mucho más explícito. Podemos crear una sencilla clase para celdas en lugar de usar una matriz de elementos `int`. Puede incluir una función que revele el objetivo (con el nombre `isFlagged`) para ocultar los números. El resultado es una nueva versión de la función:

```
public List<Cell> getFlaggedCells() {
   List<Cell> flaggedCells = new ArrayList<Cell>();
   for (Cell cell : gameBoard)
     if (cell.isFlagged())
       flaggedCells.add(cell);
```