Convenciones de código en Java

Programación I

Grado en Ingeniería Informática

JCRdP y JDGD

Importancia de las convenciones de código

- El 80% del coste del código de un programa va a su mantenimiento
- Casi ningún software lo mantiene toda su vida el autor original
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo
- Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto
  - Para que funcionen las convenciones, cada persona que escribe software debe seguir la convención. Todos

Convenciones de nombres

- Variables
  - Los atributos y las variables locales, al igual que los nombres de los métodos empiezan con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúscula
  - Los nombres de las variables deben ser cortos pero con significado. La elección del nombre de una variable debe ser un mnemónico, designado para indicar su función
  - Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales.
    - Nombres comunes para variables temporales son i, j, k, m, y n para **enteros**; c, d, y e para **caracteres**
- Constantes
  - Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("_")

Convenciones de nombres

- Clases
  - Los nombres de las clases deben ser **sustantivos**, cuando son compuestos tendrán la **primera letra de cada palabra** que lo forma en **mayúscula**
  - Intentar mantener los nombres de las clases simples y descriptivos
  - Usar palabras completas, evitar acrónimos y abreviaturas
- Métodos
  - Los métodos deben ser **verbos**, cuando son compuestos tendrán la **primera letra en minúscula**, y la **primera letra de las siguientes palabras** que lo forma en **mayúscula**

Organización del código

- Llaves
  - Las **llaves de comienzo de bloque** del cuerpo de un método, clase u otra entidad con cabecera se situarán en la misma línea que la cabecera, siempre que ésta no haya alcanzado la longitud máxima
  - Las **llaves de cierre de un bloque** se alinearán verticalmente con el comienzo del mismo

Organización del código

- Indentación
  - Las sentencias o bloques que estén anidados dentro de un bloque se desplazarán con respecto al margen izquierdo del mismo un total de **4 espacios por cada nivel de anidamiento**. En ningún caso se usarán caracteres de tabulación para este propósito ni para ningún otro
- Longitud de las líneas
  - Las líneas del código fuente, incluidas las líneas de comentarios, tendrán una **longitud máxima de 80 caracteres**, en este máximo están incluidos los espacios

Organización del código

- Romper líneas después de una coma

```
unMetodo(expresionLarga1, expresionLarga2, expresionLarga3,
         expresionLarga4, expresionLarga5);

var = unMetodo1(expresionLarga1,
                unMetodo2(expresionLarga2,
                          expresionLarga3));
```

- Romper antes de un operador

```
nombreLargo1 = nombreLargo2 * (nombreLargo3 + nombreLargo4
               - nombreLargo5) + 4 * nombreLargo6;
```

Organización del código

- Saltar de líneas por sentencias if

```
if ((condicion1 && condicion2)
        || (condicion3 && condicion4)
        ||!(condicion5 && condicion6)) {
    hacerAlgo();
}

if ((condicion1 && condicion2) || (condicion3 && condicion4)
        ||!(condicion5 && condicion6)) {
    hacerAlgo();
}
```

Organización del código

- Líneas en blanco
  - Mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas
    - Entre las secciones de un fichero fuente
    - Entre las definiciones de clases e interfaces
    - Entre métodos
    - Entre las variables locales de un método y su primera sentencia

Organización del código

- Espacios en blanco
  - Debe aparecer un espacio en blanco **después de cada coma** en las listas de argumentos de un método
  - No se debe usar un espacio en blanco **entre el nombre de un método y su paréntesis** de apertura
  - Todos los operadores binarios excepto . se deben separar de sus operandos con espacios en blanco

```
a += c + d;
a = (a + b) / (c * d);

while (d++ == s++) {
    n++;
}
```

  - Las expresiones en una sentencia `for` se deben separar con espacios en blanco

```
for (expr1; expr2; expr3)
```

Referencias

- *Java Language Specification*, de Sun Microsystems, Inc.