COLEXIO VIVAS S.L.						
	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación			CURSO:	1º
	PROTOCOLO:	Programa	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Normas de estilo en programación Java

A continuación se presentan las normas recomendadas cuando se realizan programas en Java y que vamos a tratar de cumplir en el desarrollo de de este curso.

La principal utilidad de las normas de estilo es asegura un mantenimiento rápido del código que normalmente está gestionado por más de un programador. Hay que pensar que en torno al 80% del tiempo del programador se destina al mantenimiento de software ya escrito, y que si hay unas normas de escritura la facilidad de lectura e interpretación del mismo serán mucho más rápidas.

Dichas normas están principalmente basasdas en las normas de estilo de Google y las de Oracle. Puedes verlas completas en los enlaces (se recomienda su lectura por ser más amplio que este documento):

https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

https://google.github.io/styleguide/javaguide.html

También se usó como referencia la web :

https://github.com/twitter/commons/blob/master/src/java/com/twitter/common/styleguide.md

que toma como base las dos citadas anteriormente.

Reglas principales

Nombres de identificadores

- UpperCamelCase para tipos (clases, interfaces, enumerados,...)
- lowerCamelCase para variables, métdodos y objetos.
- UPPER_SNAKE para constantes.
- todo.en.minuscula para los packages.

Ejemplos:

BIEN:

```
public class Hola
int temperaturaSemanal;
final String NOMBRE_Y_APELLIDOS = "Nathan Drake";
package colegio.vivas;
```

MAL:

```
public class hola
int TemperaturaSemanal;
int temperatura_semanal;
final String nombreYApellido = "Nathan Drake";
package Colegio.Vivas;
```

Siempre se usan llaves en las estructuras y en estilo K&R

Esto es así aún cuando el contenido sea de una única línea en i*f, else, do y while.* Además seguirán la norma establecida para C por Kerningahan& Ritchie.

- No hay retorno de carro antes de {
- Hay retorno de carro después de { y antes de }.
- Retorno de carro después de } si termina una sentencia, el cuerpo de un método o una clase, pero no por ejemplo si va seguido de un else, de una coma o de un while.

BIEN:

```
for (int i = 0; i < 10; i++) {
    System.out.print(i);
}

if (a>0){
     System.out.println("Positivo");
} else {
     System.out.println("Negativo");
}

do{
  // Cuerpo del bucle
} while (condicion);
```

MAL:

```
for (int i = 0; i < 10; i++)
{
     System.out.print(i);
}

if (a>0){
     System.out.println("Positivo");
}
else
     System.out.println("Negativo");

do{
  // Cuerpo del bucle
}
while (condicion);
```

Evitar varias sentencias en la misma línea y asignaciones múltiples.

Aunque estas sean cortas, siempre en lineas distintas, y tampoco debe haber asignaciones múltiples "por comodidad". También se aconseja realizar una declaración por cada línea, sin agrupar variables. Solo se aceptarán declaraciones múltiples en la cabecera de un bucle for.

BIEN:

```
int a = 1;
int b = 1;
a++;
System.out.println(a);
for (int i = 0, k = 1; i < 10; i++, k--) {
    System.out.print("%d, %d", i, k);
}
```

MAL:

```
int a, b;
a = b = 1;
a++; System.out.println(a);
```

Identación

Identación mínima de cuatro espácios.

La identación comienza a partir de una apertura de llave (o inicio de estructura) y se finaliza al cerrar dicha llave que estará a la misma altura de identación que la sentencia que abre.

En el switch no se escriben sentencias junto a los case, y se identa a partir de la siguiente línea al case y hasta el break (que lleva la misma identación que las previas.

BIEN:

```
switch (input) {
     case 1:
     case 2:
       funcion1();
     case 3:
       funcion2();
      break;
     default:
       otraFuncion();
}
if (a>0){
    System.out.println("Positivo");
}
```

MAL:

```
switch (input) {
     case 1:
     case 2: funcion1();
     case 3: funcion2();
     break;
     default:
      otraFuncion();
}
if (a>0){
    System.out.println("Positivo");
   }
```

Tamaño de línea

Tamaño máximo de fila 100 caracteres (80 también es válido). A partir de ahí se debe trocear la sentencia en varias líneas tras un operador, un paréntesis o una coma buscando claridad en las expresiones. Si es una cadena se usará concatenación. La línea siguieente se identará.

BIEN:

```
static String nuevFuncion(int parametro1, int parametro2,
                          double parametro3, String parametro4){
   String auxiliar = "Esta cadena es muy" +
                     " larga por lo que debería estar"+
                     " escrita en más de una línea.";
   if (a>1 && a<10 &&
       b>100 && b<100 ||
       funcionBooleana()) {
   //Realiza algo
  }
}
```

MAL:

```
static String nuevFuncion(int parametro1, int parametro2,
double parametro3, String parametro4){
    String auxiliar = "Esta cadena es larga por lo que debería estar escrita en más de una línea.";
   if (a>
        1 && a<
        10 && b>
        100 && b<100 || funcionBooleana()){
    //Realiza algo
   }
  }
```

Otras reglas de interés

Se usarán los caracteres de escape especiales en lugar de sus equivalentes unicode.

`P.ej. \t en lugar de \u0009`

Poner las unidades como parte del nombre de una variable.

`P.ej. DistanciaKm, capacidadDiscoGb.`

El uso de paréntesis opcional para clarificar código es recomendado.

`P.ej. if ((a<0) || (a>10))`

Aun no siendo obligatorio siempre se usa *@Override* en las sobreescrituras.

Enuna clase con sobrecarga de un método o sobrecarga de constructores, estos irán seguidos.