Estándar de programación

Contenido

1. Notación

a) Nombres de los Ficheros:

Ficheros de código fuente:

- *Nombre:* Nombre de la clase que implementa.
- *Extensión:* .java
- *Ejemplo:* Dia.java

Ficheros ejecutables:

- *Nombre:* Nombre de la clase que implementa.
- *Extensión:* .class
- *Ejemplo:* Dia.class

b) Nombres de los elementos del código:

Clases y package:

- *Nombre:* El nombre debe tener relación con la semántica de la funcionalidad de la clase.
- *Formato:* El nombre debe comenzar en mayúscula y el resto en minúscula. En el caso que el nombre se componga de diversas palabras, la primera letra de las diferentes palabras en mayúscula.
- *Ejemplo:* `class Dia {...}`

Variables locales y atributos (variables de instancia y de clase):

- *Nombre:* El nombre debe tener relación con la semántica de la funcionalidad del elemento.
- *Formato:* El nombre debe estar en minúscula. En el caso que el nombre se componga de diversas palabras, la primera letra de las siguientes palabras en mayúscula.
- *Ejemplo:* `private int segundos; int segundosDeUnaHora;`

Variables globales y constantes:

- *Nombre:* El nombre debe tener relación con la semántica de la funcionalidad del elemento.
- *Formato:* El nombre debe estar en mayúscula. En el caso que el nombre se componga de diversas palabras, éstas se separarán con un guión bajo.
- *Ejemplo:* `private final static int HORAS_MAXIMAS = 24;`

Operaciones:

- *Nombre:* El nombre debe tener relación con la semántica de la funcionalidad que implementan.
- *Formato:* El nombre debe estar en minúscula. En el caso que el nombre se componga de diversas palabras, la primera letra de las siguientes palabras en mayúscula.
- *Ejemplo:* `public int totalDeSegundos(int h) {...}`

2. Organización de los ficheros

a) Comentario de cabecera:

Al inicio de todo fichero .java debe haber un comentario de cabecera donde aparezcan:

- El nombre de la clase que implementa
- La descripción de la clase
- El nombre del autor de la clase (usuario racó)
- La fecha y hora de la última revisión

El formato debe ser el siguiente:

```
/* Class "Nombre de la clase":
    Descripcion: "Descripción de la clase"
    Autor: "Nombre del autor"
    Revisado: "Fecha y hora de la última revisión" */
```

Ejemplo:

```
/* Class Dia:
    Descripcion: Esta clase representa un dia y su
        funcionamiento.
    Autor: mi.user.del.raco
    Revisado: 27/10/2009 21:48 */
```

b) Declaración del Package y los Imports:

Después del comentario de cabecera debe estar la declaración del **package** y posteriormente y separado por una línea en blanco deben estar las declaraciones de los **imports**.

Ejemplo:

```
package Tiempo;

import Minuto;
import Mes;
```

c) Declaraciones de Clase:

Seguidamente de la declaración de package y los imports tendremos la declaración de la clase.

La organización de una clase será la siguiente:

- Cabecera de la clase.
- Constantes, variables de clase y variables de instancia (por ese orden y uno por línea).
- Operaciones Constructoras (cada una con su precondición y poscondición antes y después de la operación respectivamente).
- Resto de operaciones (cada una con su precondición y poscondición antes y después de la operación respectivamente).

Cada uno de los apartados estarán separados mediante una línea en blanco y de igual manera las operaciones entre sí.

Ejemplo:

```
class Dia {

    private final static int HORAS_MAXIMAS;
    private static int totalInstancias;
    private int segundos;
    private int horas;
    private int minutos;

    /* PRE: - */
    public Dia() {...}
    /* POST: Todos los atributos de Dia quedan
        inicializados. */

    /* PRE: - */
    /* EXC: HoraNoValida: h es un valor negativo. */
    public int totalDeSegundos(int h) {...}
    /* POST: 'segundosDeUnaHora' son los segundos de
        'h'. */

    /* PRE: - */
    public void generaHora(int horas, int segundos) {...}
    /* POST:  Crea horas de 50 segundos a partir de
        'horas' y 'segundos'. */
}
```

c. 1) Operaciones:

Cada operación debe tener la siguiente organización:

- Cabecera de la operación.
- Variables locales e inicialización de estas (una por línea).
- Instrucciones (una por línea).
- Instrucción de retorno.

Cada uno de los apartados estarán separados mediante una línea en blanco.

Ejemplo:

```
/* PRE: - */
/* EXC: HoraNoValida: h es un valor negativo. */
public int totalDeSegundos(int h) {

    /* h representa horas */
    int segundosDeUnaHora = h*60;

    return segundosDeUnaHora;
}
/* POST: 'segundosDeUnaHora' son los segundos de 'h'. */
```

*Todos los bloques descritos estarán separados mediante una línea en blanco.

3. Indentación

a) Tabulaciones:

Las tabulaciones deben ser de cuatro espacios (insertados).

b) Longitud máxima de línea:

Las líneas deben tener entre 80 y 90 carácteres como máximo.

b. 1) Cortes de línea:

En el caso de que una línea tenga una longitud mayor al límite, esta se cortará en el lugar más apropiado para su entendimiento a la hora de ver el código, donde los cortes deben estar indentados a la mitad de un tabulador (dos espacios).

Ejemplo:

```
if (totalDeSegundos(horas) > 50 and
  segundos > 50) {
    while (this.horas < HORAS_MAXIMAS) {
        this.minutos = totalDeMinutos(horas, segundos);
        ++this.horas;
    }
}
```

c) Estructuras de control (if-else, switch, while, for, try-catch):

Las estructuras de control deben tener la siguiente indentación:

- Las condiciones de las estructuras deben tener un espacio justo antes y después de los respectivos paréntesis.
- La llave ({) de inicio del bloque debe estar en la misma línea que la cabecera de la estructura de control.
- La llave (}) de final del bloque debe estar a la misma indentación que la cabecera de la estructura de control.

Todas las estructuras del tipo if-else, if-else-if y try-catch deben seguir la indentación siguiente:

```
if (condición) {
    ...
}
else if (condicion) {
    ...
}
else {
    ...
}
```

En el caso de las estructuras de control que únicamente tengan una instrucción, estas no deberán tener las llaves ({}).

Ejemplo:

```
while (this.horas < HORAS_MAXIMAS) {
    this.minutos = totalDeMinutos(horas, segundos);
    ++this.horas;
}
```

d) Operaciones:

Las operaciones deben tener la siguiente indentación:

- Los parámetros deben tener un espacio justo después del paréntesis de cierre. Así mismo, los parámetros deben estar separados con un espacio entre sí después de la coma.

- La llave ({) de inicio del bloque debe estar en la misma línea que la cabecera de la operación.

- La llave (}) de final del bloque debe estar a la misma indentación que la cabecera de la operación.

Ejemplo:

```
/* PRE: - */
public void generaHora(int horas, int segundos) {

    if (totalDeSegundos(horas) > 50 and
      segundos < 50) {
        while (this.horas < HORAS_MAXIMAS) {
            this.minutos = totalDeMinutos(horas, segundos);
            ++this.horas;
        }
    }
}
/* POST:  Crea horas de 50 segundos a partir de 'horas' y
    'segundos'. */
```

e) Operaciones binarias (=, +, -, *, ...):

Las operaciones binarias deben tener un espacio antes y después.

Ejemplo:

```
this.minutos = totalDeMinutos(horas, segundos);
```

4. Comentarios

Todos los comentarios del código fuente deben declararse con /* */.

Ejemplo:

```
/* h representa horas */
```

a) Comentarios obligatorios:

Como comentarios obligatorios en el código fuente tenemos los siguientes:

- **Comentario de cabecera**, el cual debe estar al inicio del fichero y debe tener el formato siguiente:

```
/* Class "Nombre de la clase":
    Descripcion: "Descripción de la clase"
    Autor: "Nombre del autor"
    Revisado: "Fecha y hora de la última revisión" */
```

Ejemplo:

```
/* Class Dia:
    Descripcion: Esta clase representa un dia y su
        funcionamiento.
    Autor: mi.user.del.raco
    Revisado: 27/10/2009 21:48 */
```

- **Precondiciones y poscondiciones** de todas las operaciones, las cuales deben situarse justo antes de la cabecera y después del final de la operación respectivamente. Deben tener el siguiente formato:

```
/* PRE: "explicación" */
/* POST: "explicación" */
```

Si se hace referencia a alguna de las variables de la operación, estas estarán entre comillas simples.

Ejemplo:

```
/* PRE: - */
/* POST: Crea horas de 50 segundos a partir de
    'horas' y 'segundos'. */
```

b) Comentarios opcionales:

Todas aquellas variables, procedimientos, fragmentos de código, etc., que su funcionalidad o semántica no sea entendible fácilmente, deberá acompañarse con un comentario que clarifique el significado de dicho fragmento.

En el caso que una operación deba hacer alguna comprovación y esta comprovación provoque una excepción, debe añadirse un comentario justo despues de la precondición donde se explique el significado de la excepción. El formato de dicho comentario es el siguiente:

```
/* EXC: "nombreExcepción": "explicación" */
```

Ejemplo:

```
/* EXC: HoraNoValida: h es un valor negativo. */
```

5. Limitaciones del lenguaje

a) No utilización del Do While:

En el código fuente únicamente deben aparecer bucles de tipo while y for, por lo que se deja de utilizar el do-while.

b) Breaks y continues:

Ningún bucle del código fuente debe tener breaks ni continues. Únicamente pueden aparecer breaks en las estructuras de control de tipo switch.

c) Operaciones unárias (++, --, ...):

En el código fuente no deben aparecer operaciones unárias dentro de otras operaciones o condiciones, es decir, las operaciones únarias deben aparecer por si solas en una instrucción. Así mismo todas las operaciones unárias deben estar delante de la variable que afectan.

Ejemplo:

```
++this.horas;
```

6. Ejemplo Completo

Este es un ejemplo ficticio de todo lo detallado en los puntos anteriores:

```
/* Class Dia:
    Descripcion: Esta clase representa un dia y su
        funcionamiento.
    Autor: mi.user.del.raco
    Revisado: 27/10/2009 21:48 */

package Tiempo;

import Minuto;
import Mes;

class Dia {

    private final static int HORAS_MAXIMAS = 24;
    private static int totalInstancias = 0;
    private int segundos;
    private int horas;
    private int minutos;

    /* PRE: - */
    public Dia() {
        ++totalInstancias;
        segundos = 0;
        horas = 0;
        minutos = 0;
    }
    /* POST: Todos los atributos de Dia quedan inicializados. */

    /* PRE: - */
    /* EXC: HoraNoValida: h es un valor negativo. */
    public int totalDeSegundos(int h) {

        /* h representa horas */
        int segundosDeUnaHora = h*60;

        return segundosDeUnaHora;
    }
    /* POST: 'segundosDeUnaHora' son los segundos de 'h'. */
```

```
/* PRE: - */
    public void generaHora(int horas, int segundos) {
        if (totalDeSegundos(horas) > 50 and
          segundos < 50) {
            while (this.horas < HORAS_MAXIMAS) {
                this.minutos = totalDeMinutos(horas, segundos);
                ++this.horas;
            }
        }
    }
    /* POST:  Crea horas de 50 segundos a partir de 'horas' y
        'segundos'. */
}
```