Normas para la presentación de programas y proyectos

Este documento tiene como objetivo guiar al estudiante con respecto a la presentación de actividades programadas que se realizan en los cursos de programación de la Escuela de Informática de la UNA. Es necesario que el estudiante tome conciencia de la importancia que tiene no sólo el hecho de elaborar un buen programa, sino también, el de elaborar una muy buena documentación tanto para sí mismo, como para las personas que lleguen a tener contacto con el programa.

La etapa de mantenimiento de una aplicación informática es la de mayor duración, por eso es conveniente que llevarla a cabo sea lo más fácil posible, para ello se debe disponer de toda la documentación posible, es decir de todos los documentos que se han ido generando en todas las etapas anteriores: diagramas, algoritmos, códigos fuentes, manuales de usuario, manual técnico, etc. A esta documentación se le llama **documentación externa**.

Además, existe otro tipo de documentación llamada interna. La **documentación interna** de un programa son los comentarios que el programador puede escribir en el código fuente de un programa. Los comentarios de un programa son explicaciones o aclaraciones que ayudarán al programador en un futuro, cuando quiera revisar o modificar el código fuente de dicho programa, y todavía serán de más ayuda si la modificación la tiene que realizar un programador distinto al que escribió el código fuente en un primer momento

Además la documentación debe tener las siguientes características:

> ***Validez***: debe estar acorde con los algoritmos, procesos, clases, etc, que contempla el programa.
>
> ***Completitud*****:** debe contener toda la información sobre la definición, el planteamiento y la solución del problema, así como cualquier requerimiento que sea necesario especificar.
>
> ***Legibilidad*****:** debe ser clara tanto gramatical como sintácticamente y debe ser redactada de tal forma que sea legible para lectores con diferentes niveles de conocimiento.

1. Documentación externa

Esta documentación se presenta como un documento aparte, en el cual se debe presentar información importante para entender el programa, al menos los siguientes puntos:

1. **Descripción del problema**: el programador define el objetivo de su programa en palabras propias. Es necesario desarrollar un enunciado definitivo del problema por resolver, una descripción de las restricciones del problema y de las metas que se lograrán.

2. **Análisis de la solución del problema**: se debe efectuar un análisis de la solución que planteó al problema y dar una descripción detallada de la misma, esta solución puede incorporar diagramas (dfd, de clases,etc), .

3. **Descripción de los aspectos más importante a utilizar**: descripción de las variables, clases, algoritmos, métodos, funciones más importantes (nombre, tipo y función que cumplen dentro del programa).

4. **Descripción de datos de prueba**: los datos de prueba constituyen una referencia de gran importancia, tanto para el programador como para los lectores, sobre el ambiente bajo el cual el programa fue probado. Los datos de prueba deben considerar todos los casos extremos, así como casos que se consideren problemáticos.

5. **Limitaciones del programa**: se incluye en el caso de que el programa no funcione correctamente o no contemple alguna situación. Se deben incluir los objetivos o requerimientos de la tarea que no se pudieron cumplir y el motivo de los mismos.

6. **Observaciones generales**: incluye cualquier tipo de situación que se considere de importancia y no califique en ninguna de las categorías anteriores. Es importante en este apartado mostrar aquellas funcionalidades que sin ser solicitadas fueron sumadas a la solución y que vienen a mejorarla.

7. **Manual del usuario**: consiste en una descripción detallada para mostrarle a un usuario poco experto como debe utilizar el programa, es bueno incorporar imágenes de pantallas que sean explicativas sin abusar de las mismas, debe estar escrito en un lenguaje muy simple.

2. Documentación interna

El objetivo de la documentación interna es facilitar el posterior entendimiento del código fuente. Debe incluir:

1. Nombre completo del programador/ Institución
2. Fecha
3. Objetivo del programa
4. Comentarios explicando el propósito de cada subprograma, el uso de las variables más importantes y cualquier otro aspecto que se desee aclarar o documentar (por ejemplo secciones de código complicado o difícil de entender). En este punto es importante tener en cuenta que el exceso de comentarios podría entorpecer el entendimiento del programa, que es lo que se debe evitar, ver ejemplo en el Anexo #1. Exceso de comentarios.

Además es recomendable el uso de ciertas convenciones para la escritura de los programas, esto por cuanto las convenciones de código son importantes para los programadores por un gran número de razones: un alto porcentaje del costo del código de un programa se va en su mantenimiento, casi ninguna aplicación se mantiene igual toda su vida útil, las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.

El problema es que para que funcionen las convenciones, cada persona que escribe software debe seguirlas, algunas de las propuestas son:

1. Escriba únicamente una instrucción por línea.
2. Los archivos deben iniciar de la siguiente forma:

```
/ *  nombre del archivo
  *  información de la versión
  *  fecha
  * Autor/derechos:
  */
```

3. Luego, según el lenguaje utilizado deben seguir las instrucciones que especifican nombres de bibliotecas o archivos a utilizar.
4. Los nombres de clases deben ser sustantivos y significativos, cuando son compuestos tendrán primera letra de cada palabra que lo forma en mayúsculas. Las constantes, instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de las variables deben significativos, mnemotécnicos, designado para indicar cual su función. Los nombres de variables de un solo carácter se deben evitar, excepto para variables de indicadores. Sin embargo en este punto puede buscar algún estándar, por ejemplo CamelCase.
5. Use la sangría (indentación) adecuada para darle claridad al programa, se sugieren 4 espacios.
   5.1. No hay una restricción sobre el orden de definición de los métodos, algunos establecen que es conveniente definir los métodos privados primero, luego públicos, etc. Sin embargo es importante que los métodos de cálculo, estén agrupados funcionalidad, por ejemplo, un método de clase privado puede estar entre dos métodos públicos, con el único objetivo de hacer el código más legible y comprensible.
6. Haga una declaración por línea, por ejemplo:

```
int tamaño;
int cantidad;
```

7. Haga las declaraciones siempre al inicio de los bloques de programación, no espere a declarar una variable en el momento en que la va a usar, pues puede causar confusión y en la medida de lo posible inicialícelas de una vez, la única razón para no hacerlo es si su valor es el resultado de un cálculo, la excepción a la regla son los índices del ciclo FOR.
8. Las llaves de apertura "{" aparecen al final de la misma línea de la sentencia declaración. La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

```
class Ejemplo {
        int ivar1;
        int ivar2;

        Ejemplo (int i, int j) {
            ivar1 = i;
            ivar2 = j;
        }
        int metodoVacio() {}
```

9. Como recomendación las llaves se usan en todas las instrucciones, incluso las simples, cuando forman parte de una estructura de control, como en la instrucción if-else o for., por cuanto esto hace más sencillo añadir más instrucciones eliminando la probabilidad de cometer errores accidentalmente por olvidar las llaves.

3. Criterios de evaluación.

Para obtener la calificación completa, es necesario implementar el programa y entregar toda la documentación correspondiente. La implementación será evaluada según el uso de conceptos y técnicas, de manera proporcional, de acuerdo con el criterio del profesor, el que el programa corra adecuadamente es sólo uno de los objetivos. Para la documentación, se entregará un documento en formato PDF.

4. Consideraciones de implementación.

A la hora de implementar un programa es importante considerar efectuar las pruebas necesarias para verificar el funcionamiento de los métodos de cada clase y del programa en general, y es recomendable definir un procedimiento de prueba para poder verificar el funcionamiento general de la aplicación, según los requerimientos descritos.

En caso de que la aplicación no funcione adecuadamente, efectúe un análisis de los resultados obtenidos, indicando las razones por las cuales la aplicación no trabaja correctamente, y cuáles son las posibles correcciones que se le puedan hacer al programa. Durante la revisión del proyecto, es muy importante poder defender adecuadamente la solución propuesta.

5. Entrega del proyecto

La entrega del proyecto se realizará según las indicaciones de cada profesor.

Referencia bibliográfica

1. Hommel, Scott. (1999). *Convenciones de código para el lenguaje de programación JAVA™***.** Sun MicroSystem Inc.

Anexo #1. Programa ejemplo con exceso de comentarios,

```
#include <iostream.h> //Libreria para la entrada/salida de datos

main() //Comienza la funcion principal
{
 class vehiculo  //Declaramos la superclase vehiculo
 {
  protected:  //Le ponemos tipo protegido a los datos para que se pasen a la variante
        float peso; //Variable tipo decimal que guardara el peso
        int ruedas; //Variable tipo intero que guardara el nº de ruedas
  public:  //Le ponemos tipo publico a los constructores para poder utilizarlos en todo el
programa
       void iniciar(float p,int r) //Este constructor se dedica a enlazar las variables del
       { //main principal con las de las clases por eso se pasan por parametros
         peso = p;  //La variable peso sera igual a la variable p
         ruedas = r;  //La variable ruedas sera igual a la variable r
       }; //Fin del constructor iniciar
       float ob_peso(void) //Constructor de tipo decimal que muestra el peso
       { return peso; }; //Devuelve el peso con return
       int ob_ruedas(void) //Constructor de tipo entero que muestra el nº de ruedas
       { return ruedas; }; //Devuelve las ruedas con return
 };  //Fin de la declaracion de la superclase vehiculo
 class motocicleta:public vehiculo //Declaramos la clase variante llamada motocicleta
 { //Hacemos que se pase todo lo protegido y publico a esta variante con :public vehiculo
  private: //Aunque por defecto si no se pone el tipo es private, pero lo ponemos
        int pasajeros;  //Variable de tipo entero que guardara el nº de pasajeros
        int combustible; //Variable de tipo entero que guardara el combustible
        char tipo; //Variable de tipo caracter que guardara una tecla
  public:  //Le ponemos tipo publico a los constructores para poder utilizarlos en todo el
programa
       void iniciar(int pj,int com,int r,float p) //Este constructor se dedica a enlazar las
       { //variables del main principal con las de las clases por eso se pasan por parametros
         pasajeros = pj;  //La variable pasajeros sera igual a pj
         combustible = com;  //La variable combustible sera igual a com
         ruedas = r; //La variable ruedas sera igual a r
         peso = p; //La variable peso sera igual a p
```

```
        };  //Fin del constructor iniciar
        int ob_pasajeros(void) //Constructor tipo entero para obtener el nº de pasajeros
        { cout << endl << "Que tipo de moto es ?";  //Datos de salida
          cout << endl << "Grande: Pulse G - Chica: Pulse C\\n";  //Datos de salida
              cin >> tipo;  //Dato de entrada hacia la variable tipo char llamada tipo
          if((tipo == 'g') || (tipo == 'G')) //Si hemos pulsado g o G
          { pasajeros=pasajeros+1; }  //pasajeros se incrementara 1
          if((tipo == 'c') || (tipo == 'C')) //Si hemos pulsado c o C
          { pasajeros=pasajeros-1; }  //pasajeros se decrementara 1
          return pasajeros;  //Devuelve con return el valor de la variable pasajeros
        };  //Fin del constructor ob_pasajeros
        int ob_combustible(void) //Constructor tipo entero para obtener el nº de
combustible
        { //Se utiliza el mismo valor anterior con la variable char tipo
          if((tipo == 'g') || (tipo == 'G')) //Si habiamos pulsado g o G
          { combustible=combustible+20; }  //El valor de combustible sera 20 mas
          if((tipo == 'c') || (tipo == 'C')) //Si habiamos pulsado c o C
          { pasajeros=20; }             //El valor de combustible sera 20
          return combustible;  //Devuelve con return el valor de combustible
        };  //Fin del constructor ob_combustible
 }; //Fin de la declaracion de la clase variante motocicleta

 int r=2,pj=2,com=20; //Declaramos variables enteras:
                 //r sera el numero de ruedas
                 //pj sera el numero de pasajeros
                 //com sera el numero de combustible
 float p=96.5;  //Declaramos variables decimales:
            //p sera el peso

 motocicleta moto;  //Declaramos una variable tipo sub-clase moticicleta llamada moto

 moto.iniciar(pj,com,r,p); //Llamamos al constructor iniciar desde moto, le pasamos los
parametros
 pj=moto.ob_pasajeros(); //Asignamos el valor del constructor ob_pasajeros a pj

 cout << endl << "Caracteristicas de un ciclomotor";  //Datos de salida
```

```
 cout << endl << "Peso: " << moto.ob_peso() << "kg";   //Datos de salida, llamamos a
ob_peso
 cout << endl << "Ruedas: " << moto.ob_ruedas();        //Datos de salida, llamamos a
ob_ruedas
 cout << endl << "Pasajeros: " << pj;                  //Datos de salida, ponemos a mostrar pj
 cout << endl << "Combustible: " << moto.ob_combustible() << "litros"; //Datos de salida,
llamamos
 cout << endl;                                        //a ob_combustible

 cin.get();  //Limpia el buffer
 cin.get();  //Espera a que pulses una tecla
} //Fin de la funcion principal
```