Nombre: ..... Fecha: / 11 / 2008

HOJA DE EJERCICIOS 1
NOCIONES DE CLASE Y OBJETO EN PROGRAMACIÓN ORIENTADA A OBJETOS

Esta hoja de ejercicios nos servirá para repasar las nociones más importantes del Tema 1.

1. Señala las principales diferencias entre las propiedades de una clase en un lenguaje orientado a objetos y la representación de TAD's por medio de registros más funciones que actúan sobre los mismos.

2. Resume en una breve descripción el concepto de "encapsulación de la información" o de modularización.

3. Distingue algunas de las consecuencias del uso de constructores para la inicialización de objetos.

4. Define los ámbitos de visibilidad de atributos y métodos "public" y "private".

5. Explica las ventajas de tener ciertas propiedades "private" en una clase con respecto a la ocultación de la información.

6. ¿Cuáles son las mayores diferencias entre un atributo de instancia (o de objeto) y un atributo estático? ¿Y entre un método de instancia (o de objeto) y un método estático?

7. Observa el siguiente fragmento de código en Java y los fragmentos en C++ (supón definida una clase **Pelicula**):

JAVA:

```
public class Principal {
        public static void main(String [] args){

                Pelicula et = new Pelicula ("ET", "Steven Spielberg", 115);
                auxiliar (mi_pelicula);
                System.out.println ("El titulo de la película es "+ mi_pelicula.getTitulo());
        }

        public static void auxiliar (Pelicula z){
                z.setTitulo ("Parque Jurasico");
        }
}
```

1. C++:

```
int main (){
        Pelicula mi_pelicula ("ET", "Steven Spielberg", 115);
        auxiliar (mi_pelicula);
        cout << "El nuevo titulo de la película es " << mi_pelicula.getTitulo() << endl;
        system ("PAUSE");
        return 0;
}

void auxiliar (Pelicula z){
        z.setTitulo ("Parque Jurasico");
}
```

2. C++:

```
int main (){
        Pelicula * mi_pelicula = new Pelicula ("ET", "Steven Spielberg", 115);
        auxiliar (mi_pelicula);
        cout << "El nuevo titulo de la película es " << mi_pelicula->getTitulo() << endl;
```

```
        system ("PAUSE");
        return 0;
}

void auxiliar (Pelicula * z){
        z->setTitulo ("Parque Jurasico");
}
```

a) Responde cuál será el resultado de ejecutar los tres fragmentos de código anteriores. Basa y justifica tus respuestas en las nociones de paso por valor y paso por referencia.
b) ¿Por qué el método "auxiliar()" ha sido definido como "static" en Java? ¿Qué hubiera pasado de no haberlo hecho? Recuerda las nociones de método de clase y de método de instancia.

8. Observa la siguiente operación en C++ y Java:

C++: strcpy (this->titulo, "La delgada linea roja");
Java: this.titulo = "La delgada linea roja";

a) ¿Cómo explicas la ausencia del método "strcpy ()" en Java?
b) ¿Qué hace el método "strcpy()" en C++?
c) Relaciona tus respuestas anteriores con los modelos de gestión de memoria de Java y C++.

9. Dado el siguiente diagrama UML, declara y define en C++ y Java la clase que lo implementa

CuentaBancaria
-numeroCuenta: string -nombreTitular : string -saldo : double -numeroOperaciones : int -MAXIMOREINTEGRO : double = 5000 -cuentasCreadas : int = 0
+CuentaBancaria(entrada : string, entrada : string) +getNumeroCuenta() : string +getNombreTitular() : string +getSaldo() : double +setSaldo(entrada : double) : void +getNumeroOperaciones() : int +ingresarCantidad(entrada : double) : void +retirarCantidad(entrada : double) : void +getCuentasCreadas() : int +setCuentasCreadasNull() : void

Notas:

- El atributo de instancia "numeroOperaciones" guarda el número de operaciones (se consideran como operaciones únicamente "ingresarCantidad(double): void" y "retirarCantidad(double): void") que se han realizado sobre una cuenta.
- MAXIMOREINTEGRO representa una constante de clase (cuyo valor es 5000).
- cuentasCreadas es un atributo de clase (o estático) cuyo valor inicial es 0 (su valor debe actualizarse cada vez que creemos una nueva cuenta bancaria).
- El método "retirarCantidad(double): void" debe comprobar que la cantidad retirada no sea mayor que el "MAXIMOREINTEGRO" ni que el "saldo" existente en la cuenta.

11. Crea un cliente de la clase anterior (en C++ y Java) que simule el siguiente comportamiento:

a) Creamos una cuenta con número "55551123231234567890" a nombre de "Manuel Ramírez".
b) Hacemos un ingreso en la cuenta de 15.000 euros.
c) Solicitamos el saldo de la cuenta.

d) Retiramos 2.500 euros.
e) Comprobamos de nuevo el saldo de la cuenta.
f) Retiramos 800 euros.
g) Comprobamos el número de operaciones que se han realizado sobre la cuenta.
h) Creamos una nueva cuenta con número “55551123230987654321” a nombre de “Manuel Ramírez”.
i) Comprobamos el número de cuentas que han sido creadas.

12. Partiendo de la siguiente implementación del TAD “Collection” en C++, prográmalo en Java y en C++ haciendo uso de los conocimientos adquiridos a lo largo del Tema 1 sobre POO. Modifica también el cliente de la clase de manera acorde.

```
//Vamos a trabajar con colecciones de enteros
typedef int telemento;
//número máximo de datos que habrá en la colección
const int MAX= 1000;
struct Collection
{
        int num;//número de datos que hay en la colección
        int elementoApuntado;
        telemento  datos [MAX];
};

void createCollection(Collection & C);
//Inicia C como una colección vacía
int size(Collection C);
//Devuelve el número de elementos en la colección
bool isEmpty(Collection C);
//Si la coleccción C está vacía devuelve el valor verdad
//En caso contrario devuelve el valor falso
bool contains(Collection C, telemento c);
//Devuelve verdadero si el elemento c está en la colección C
bool add(Collection & C, telemento c);
//Asegura que el elemento c está en la colección C
//Si no estaba lo añade, si ya estaba no hace nada
bool remove (Collection & C, telemento c);
//Elimina el elemento C de la colección c, si está
bool hasNext (Collection c);
//Devuelve verdadero si hay más elementos en la colección
telemento next (Collection & C);
//Devuelve el siguiente elemento en la colección

//Collection.cpp
#include "Collection.h"
void createCollection(Collection & C){
    C.num = 0;
    C.elementoApuntado = 0;
};
int size(Collection C){
   return C.num;
};
bool isEmpty(Collection C){
    return C.num == 0;
};
bool contains(Collection C, telemento c){
    bool aux = false;
    for (int i = 0; i < C.num; i++){
       if (C.datos [i] == c)
         {aux = true;}
    }
    return aux;
};
bool add(Collection & C, telemento c){
    if (contains (C, c)){
```

```
        return true;
        }
    else {
         C.num++;
         C.datos [C.num] = c;
         }
};
bool remove (Collection & C, telemento c){
    if (contains (C, c)){
        int aux;
        for (int i = 0; i< C.num; i++){
           if (C.datos[i] == c){
             aux = i;
           }
        }
        for (int i = aux; i < C.num - 1; i++){
             C.datos[i] = C.datos[i + 1];
           }
        C.num--;
        return true;
        }
    else {return false;}
};
bool hasNext (Collection C){
    return C.elementoApuntado < C.num;
};
telemento next (Collection & C){
    C.elementoApuntado++;
    return (C.datos[C.elementoApuntado-1]);
};

#include <iostream>
#include "Collection.h"

using namespace std;

int main (){
   //Declaramos una colección
   Collection c1;
   //La iniciamos vacía
   createCollection (c1);
   //Le añadimos 3 elementos
   add (c1, 5);
   add (c1, 45);
   add (c1, 17);
   //Comprobamos su tamaño
   cout << "El numero de elementos en la coleccion es " << size(c1) << endl;
   //Le añadimos una serie de elementos:
   for (int i = 0; i <= 15; i++){
      add (c1, 3 * i);
   }
   //Eliminamos alguna posible aparición de "30"
   cout << "El resultado de remover el 30 es " << remove (c1, 30) << endl;
  //Recorremos la colección con los métodos propios de la misma
  while (hasNext (c1)){
    cout << "El siguiente elemento en la colección es " << next (c1) << endl;
  }
  system ("PAUSE");
  return 0;
};
```