Tema 3: Práctica guiadas

Objetivos

-Implementar asociaciones entre clases.

-Utilización de colecciones para implementar relaciones. Se usarán los tipos array (C`++) ,List (C++)y LinkedList (Java).

Nota: Este documento constituye material de apoyo y de estudio para las prácticas obligatorias que posteriormente se deberán entregar.

Práctica guiada 1 C++

(Profesora Elisa García)

Un juego de luces está formado por un número variable de focos (entre 4 y 6), que se determina exactamente cuando se fabrica. El técnico de luces puede realizar varias operaciones sobre el juego de luces: *encenderlo completamente, encender una luz en particular, apagarlo completamente o apagar un foco concreto*. Cada foco puede ser encendido y apagado por el operario de manera independiente.

Se pide:

Realizar un programa principal que realice las siguientes acciones:

- Crear un juego de luces con 5 luces.
- Encender todas las luces.
- Apagar la primera.
- Imprimir el estado de todas las luces del juego de luces.

Estudio de la solución:

- Las **dos abstracciones** necesarias son **Juego de Luces y Foco.** Y tienen una asociación

- **Juego de luces** juega el **rol de agregado** respecto a Foco. Pues el juego de luces se puede concebir como un conjunto de Focos. Identificamos que la relación es de agregación. **La agregación es fuerte** (Composición) ya que el ciclo de vida de las instancias del tipo Foco no son **controladas completamente dentro del agregado(en este caso Juego de Luces)**, luego tenemos una **composición entre Juego de Luces y Foco.**

- La multiplicidad de la relación es 4..6 en la parte del agregado. Es una multiplicidad variable con un valor máximo. Necesitamos un parámetro en el constructor que “informe “del número de focos exactos que tiene el juego de luces.

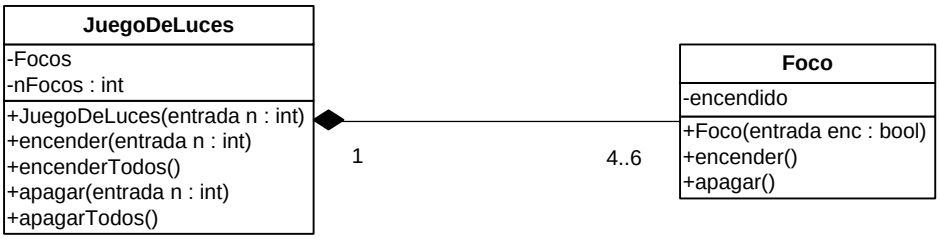
- Las **responsabilidades individuales de cada clase** son:

    - **Clase JuegoDeLuces: tiene que permitir al operador encender** cada una de los Focos, encender un Foco en particular, apagar todos y apagar uno en concreto.

- **Clase Foco:** debe poder ser apagado y encendido.

Tomando en cuenta todas estas consideraciones tenemos el siguiente diseño:

Diagrama de clases UML propuesto para la solución:



La implementación en C++ propuesta es la siguiente:

****** foco.h ***

```
#ifndef FOCO_H
#define FOCO_H

class Foco
{
private:
   bool encendido;
public:
  Foco(bool enc);
 ~Foco();
 void encender();
 void apagar();
 bool estaEncendido();
};

#endif
```

****** foco.cpp ***

```
#include "foco.h"

Foco::Foco(bool enc){
 encendido = enc;
}
void Foco::apagar(){
  encendido = false;
}
void Foco::encender(){
 encendido = true;
}

bool Foco::estaEncendido()
{
     return encendido;
}

Foco::~Foco()
{}
```

Juego de Luces tiene los siguientes atributos:

- El **conjunto de Focos o colección de Focos**: Esto se va a modelar **como un array de objetos tipo Foco**.
- Y el número exacto de estos focos: Esto permite controlar cuantos Focos hay que encender por que el número es variable (unas veces 6 otras 4) y con un número máximo de 6.

********** juegodeluces.h ***

```
#ifndef _JUEGOLUCES_H
#define _JUEGOLUCES_H
#include "Foco.h"

#define MIN_LUCES 4
#define MAX_LUCES 6

class JuegoDeLuces{
private:
unsigned int numLuces;
Foco* Focos;
public:
JuegoDeLuces(int n=MIN_LUCES);
~JuegoDeLuces();
void encender(int);
void encenderTodos();
void apagar(int n);
void apagarTodos();
void verEstado();
};
#endif
```

********** juegodeluces.cpp ***

```
#include <iostream>
#include "juegodeluces.h"
using namespace std;
JuegoDeLuces::JuegoDeLuces(int n){
if (n>0 && n<MAX_LUCES)
   numLuces = n;
else
   numLuces = MIN_LUCES;
Focos= new Foco[numLuces];
}
JuegoDeLuces::~JuegoDeLuces(){
delete [] Focos;
}
void JuegoDeLuces::encender(int n){
Focos[n].encender();
}
void JuegoDeLuces::encenderTodos(){
unsigned int i;
   for (i=0; i<numLuces; i++)
    Focos[i].encender();
}
void JuegoDeLuces::apagar(int n){
        Focos[n].apagar();
}
void JuegoDeLuces::apagarTodos(){
        unsigned int i;
for (i=0; i<numLuces; i++)
    Focos[i].apagar();
```

```
}

void JuegoDeLuces::verEstado(){
unsigned int i;
   for (i=0; i<numLuces; i++)
     if (Focos[i].estaEncendido())
      cout << "Foco ["<< i<<"]"<<" encendido"<<endl;
     else
      cout  << "Foco ["<< i<<"]"<<" apagado"<<endl;
}
```

Vamos a probar estas clases creando un programa que realice lo siguiente:

- Crear un juego de luces.
- Encender todos los focos del juego de luces.
- Apagar el primero foco del juego de luces.

*** MAIN.CPP ***

```
#include <iostream>
#include <stdlib.h>
#include "juegodeluces.h"

using namespace std;

int main(int argc, char *argv[]){

 JuegoDeLuces ju1(5);

 ju1.encenderTodos();
 ju1.apagar(0);
 ju1.verEstado();

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

Nota: en esta solución se utilizan **objetos estáticos en C++**

En este programa a diferencia de los anteriores no hemos creado los objetos de forma dinámica.

- La siguiente instrucción declara una variable `(ju1)` de tipo JuegoDeLuces **e implícitamente se llama al constructor** de JuegoDeLuces pasándole 5 como argumento

```
JuegoDeLuces ju1(5);
```

- En la siguiente instrucción se accede a una función miembro mediante el operador de acceso (.)

```
ju1.encenderTodos();
```

El programa principal equivalente con memoria dinámica sería el siguiente.

```
#include <iostream>
#include <stdlib.h>
#include "juegodeluces.h"
using namespace std;
```

```
int main(int argc, char *argv[]){

 JuegoDeLuces *ju1=new JuegoDeLuces(5);
 ju1->encenderTodos();
 ju1->apagar(0);
 ju1->verEstado();
 delete jul;//Borramos
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Actividad propuesta

- Realice un programa principal que realice lo siguiente.
- Cree un juego de luces con 4 focos.
- Solicite una secuencia de longitud fija de 4 caracteres “0” o “1” (Ejemplo “0011”). Donde “1” significa foco encendido y “0” foco apagado.
- Y en función de esta secuencia active o desactive los focos del juego de luces.

Práctica guiada 2.C++

Modelar la información acerca de departamentos y empleados de una empresa. Para ello considere las siguientes características:

La información que se desea representar sobre un departamento es la siguiente: nombre del departamento (Marketing, I+D, etc) y los empleados que pertenecen al departamento. Por otro lado sobre los empleados sólo es necesario guardar su nombre.

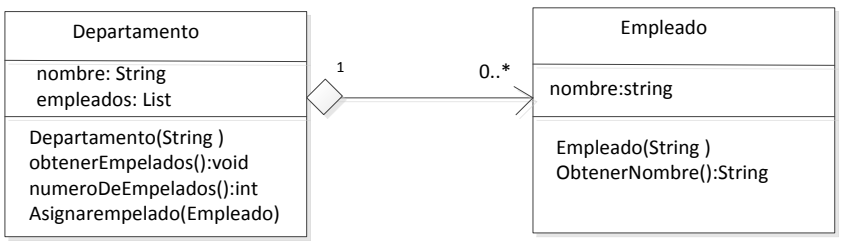
Se pide:

Escribir un programa principal con las siguientes acciones

- Crear un departamento de I+D
- Crear dos empleados
- Asignar los empleados al departamento creado.

Estudio de la solución C++

- Las dos conceptos que manejamos son: Departamento y Empleado
- La asociación es de tipo 1:N .Se puede plantear como una agregación débil
- El modelo de clase en UML es el siguiente:



- Modelamos la clase Departamento como un **agregado** compuesto por Empleados.
- El tipo de **agregación** es **débil** ya que el ciclo de vida de los objetos de tipo Empleado no están controlados por el agregado y podrían formar parte de otras asociaciones

Implementamos ahora cada una de las clases

Archivo **empleado.h**

```
#ifndef EMPLEADO_H
#define EMPLEADO_H
class Empleado
{
   protected:
             char *nombre;

       public:
              //  constructores
              Empleado();
              Empleado(char * nomb);
              // destructor
              ~Empleado();
              //funciones
              char * getNombre();
};
#endif
```

Archivo **Empelado.cpp**

```
#include "empleado.h"
//  constructor
Empleado::Empleado()
{
       nombre="nombre no asignado";
}

Empleado::Empleado( char *nomb){
nombre=nomb;
}
Empleado::~Empleado()
{}
char * Empleado::getNombre(){
     return nombre;
}
```

Archivo **departamento.h**

```
#ifndef DEPARTAMENTO_H
#define DEPARTAMENTO_H
#include "Empleado.h"
#include <list.h>
class Departamento
{

    protected:
              char* nombre;
              list<Empleado *> empleados;

       public:
              // contsructor
              Departamento();
              Departamento(char *nomb);
              // destructor
              ~Departamento();
              //función miembro
              void obtenerEmpleados();
              void numeroDeEmpleados();
              void asignarEmpleado(Empleado *e);
```

```
};
#endif
```

Nota: Observe la siguiente declaración

list<Empleado *> empleados;

Esta línea declara una variable con nombre empleados de tipo list. list es tipo colección que se encuentra en la librería estándar de C++

Departamento.cpp

```
#include "departamento.h" // class's header file

Departamento::Departamento(char * nomb)
{
       nombre=nomb;
}

Departamento::Departamento()
{
       nombre="Departamento sin nombre";
}

Departamento::~Departamento()
{
       //se borra cada elemento de la lista empleados;
       list<Empleado *>::iterator iterador_lista=empleados.begin();

       for (;iterador_lista!=empleados.end();iterador_lista++){
         delete *iterador_lista;
        }
       delete nombre;
}
void Departamento::obtenerEmpleados(){
    //itereador_lista se posiciona al principio de la lista
    list<Empleado *>::iterator iterador_lista=empleados.begin();

    Empleado *e;
    //Recorrer cada elemento de la lista

    for (;iterador_lista!=empleados.end();iterador_lista++){
         e=*iterador_lista; //Asignar el contenido del iterador
         cout<<"Empleado :"<<e->getNombre()<<endl;
        }

    }
void Departamento::numeroDeEmpleados(){
    cout<<"Dpto "<<nombre<<" tiene "<< empleados.size()<<" empleados";
    }
void Departamento::asignarEmpleado(Empleado *e){
        //inserta un empleado al principio de la lista
        empleados.push_front(e);
}
```

El programa principal sería el siguiente:

```
#include <cstdlib>
#include <iostream>
#include "Departamento.h"
#include "Empleado.h"
using namespace std;
```

```
int main(int argc, char *argv[])
{
    Departamento *dpto=new Departamento("DESARROLLO");
    Empleado *emp1=new Empleado("Juan");
    Empleado *emp2=new Empleado("Pedro");
    dpto->asignarEmpleado(emp1);
    dpto->asignarEmpleado(emp2);
    dpto->obtenerEmpleados();
    dpto->numeroDeEmpleados();

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

Práctica guiada 2. Java

Estudio de la solución en Java

El mismo estudio realizado anteriormente es válido.

Clase **Empleado.java**

```
public class Empleado {
    String nombre;

    public Empleado(String nombre) {
        this.nombre = nombre;
    }
    public String getNombre() {
        return nombre;
    }
}
```

Clase **Departamento.java**

```
import java.util.LinkedList;

public class Departamento {
    String nombre;
    LinkedList empleados;

    public Departamento(String nombre) {
        this.nombre = nombre;
        empleados=new LinkedList(); //instanciar colección
    }
    public void asignarEmpleado( Empleado e)
    {
     empleados.add(e);
    }
    public void obtenerEmpleados()
    {
        Empleado e;

         int tam= empleados.size();
         int pos=0;
       //Mientras haya elementos
         while ( pos<=tam-1)
         {
```

```
        //obtener elemento
        e=(Empleado)empleados.get(pos);
        System.out.println(e.getNombre());
        pos++;
        }
    }
    public void numeroDeEmpleados()
    {
    System.out.println("Numero de empleados:" +empleados.size());
    }
}
```

Observa que para implementar la relación con multiplicidad * usamos un tipo colección (la clase LinkedList).

```
LinkedList empleados;
```

LinkedList es una clase tipo colección que implementa una lista enlazada. Podríamos haber usados otras disponibles como por ejemplo **Vector**. LinkedList se encuentra el paquete **java.util**, luego antes de usarla debemos importarla:

import java.util.LinkedList;

Clase **ProgramaPrincipal.java**

```
public class ProgramaPrincipal {

    public static void main(String[] args) {
        Departamento dpto = new Departamento("DESARROLLO");
        Empleado emp1 = new Empleado("Juan");
        Empleado emp2 = new Empleado("Pedro");
        dpto.asignarEmpleado(emp1);
        dpto.asignarEmpleado(emp2);
        dpto.obtenerEmpleados();
        dpto.numeroDeEmpleados();
    }
}
```