Programación Orientada a Objetos

Examen final

Junio 2010

1. **Crear clase para generar un sistema de información geográfica para localizar servicios. El sistema de información contendrá varias capas según el tipo de servicio. Tendrá una capa por cada localización de servicios de cada tipo de servicio, es decir, una capa para hospitales (que representaremos por H), una capa de restaurante (R) y una capa de parada de autobús (B). El programa permitirá mostrar el mapa con varias capas y se debe poder desactivar y activar el mapa por capas.**

   Ej salida:

```
 -------------------------
|   |   |   | R |   | H |
-------------------------
|   | B |   |   |   |   |
-------------------------
|   | R |   |   | R|   |
-------------------------
```

a.1) Crear clase capa (5 atributos):

- **Dos enteros sin signo (x_, y_). Dimensiones de la capa**
- **Un carácter: icono. Letra de servicios**
- **Booleano: activada_**
- **Pares de coordenadas: cuadricula_**

```
using namespace std;
class capa {
    private:
        unsigned int x_,y_;        //dimensiones de la capa
        char icono_;               //letra de servicio
        bool activada_;            //si la capa está o no activada
        char**cuadricula_;         //cuadricula de coordenadas (espacio si casilla vacía o icono si no lo está)
                                   //Nota: la matriz se crea dinamicamente al construir la capa
        bool existente_; //si existe o no
    public:
        capa() {     //constructor por defecto
        x_=0;
        y_=0;
        activada_=false;
        existente_=false;
        cuadricula_=NULL;
    }
```

a.2) Constructor que recibe una capa y un icono y crea una capa vacía y activada.

```
//constructor que recibe una capa y un icono y crea una capa vacia y activada.
capa(unsigned int x, unsigned int y, char icono) {
    x_=x;
    y_=y;
```

```
//creación dinamica de una matriz de dimensiones x_, y_
    cuadricula_ = new char*[x_];
    for (int i=0;i<x_;i++) cuadricula_[i]=new char[y_];
    for(int i=0;i<x_;i++) {
        for (int j=0;j<y_;j++) {
            cuadricula_[i][j]=' ';
        }
    }
    icono_=icono;
    activada_=true;
    existente_=false;
}

//constructor de copia
capa(const capa& c) {
    x_=c.x_;
    y_=c.y_;
    cuadricula_ = new char*[x_];
    for (int i=0;i<x_;i++) cuadricula_[i]=new char[y_];
    for(int i=0;i<x_;i++) {
        for (int j=0;j<y_;j++) {
            cuadricula_[i][j]=c.cuadricula_[i][j];
        }
    }
    icono_=c.icono_;
    activada_=c.activada_;
    existente_=c.existente_;
}

//destructor
~capa() {
    if (cuadricula_!=NULL) {
        for (int i=0;i<x_;i++)  delete cuadricula_[i];
        delete cuadricula_;
    }
}
```

a.3) Función observadora: icono( ) que devuelve el icono del mapa

```
inline char icono() const { return icono_; }
inline unsigned int x() const {return x_;}
inline unsigned int y() const {return y_;}
```

a.4) activa ( ) y desactiva ( ) : activar y desactivar la capa

```
inline void activa() { activada_=true; }
inline void desactiva() { activada_=false; }
```

a.5) activada( )

```
inline bool activada() const { return activada_; }  //Devuelve el estado de la capa.
```

a.6) ocupada( ): recibirá coordenada de la capa (dos enteros sin signo) para saber si la casilla se encuentra ocupada.

```
inline char casilla(unsigned int fil, unsigned int col) const {
    return cuadricula_[fil][col]; //Devuelve V si es ≠ de vacio.
}
inline bool ocupada(unsigned int fil, unsigned int col) const {
    return (cuadricula_[fil][col]!=' ');  //Me da el icono carácter de la casilla fil, col
}
```

a.7) agrega_elemento ( ) : recibe una coordenada y añade un elemento. Deberá comprobar si las coordenadas están fuera de las dimensiones o sino mostrará una excepción: out_of_range.

```
inline void agrega_elemento(unsigned int fil, unsigned int col) {
    if (fil<0 || fil>=x_ || col<0 || col>=y_) { //se salio de la cuadricula
        throw out_of_range("Error, elemento fuera de la cuadricula\n");
    }
    cuadricula_[fil][col]=icono_;
}
```

b.1) Crear clase mapa (3 atributos)

- **Dos enteros sin signo (x_, y_). Dimensiones del mapa.**
- **Cadena (string)**

```
class mapa {
    private:
        unsigned int x_, y_;//dimensiones del mapa (y x tanto de todas sus capas)
        capa cap_hospitales_; //capa para hospitales
        capa cap_restaurantes_; //capa para restaurantes
        capa cap_bus_; //capa para el bus
    public:
        class Capa_Inexistente { };  //excepción de capa inexistente.
};
```

b.2) Define un constructor que recibe la dimensión del mapa y crea un mapa vacío.

```
mapa(unsigned int x, unsigned int y) {
    x_=x;
    y_=y;
    cap_hospitales_ = capa(x,y,'H');   //creo la capa de hospitales
    cap_restaurantes_ = capa(x,y,'R'); //creo la capa de restaurantes
    cap_bus_ = capa(x,y,'B'); //creo la capa de bus
}

mapa(const mapa& m) { // Constructor de copia.
    x_=m.x_;
    y_=m.y_;
    cap_hospitales_= m.cap_hospitales_;
    cap_restaurantes_= m.cap_restaurantes_;
    cap_bus_=m.cap_bus_;
}
```

b.3) activa_capa ( ) y desactiva_capa( ): Buscan si existe la capa cuyo nombre es recibido como parámetro y si existe, activa o desactiva la capa. Si no existe la capa devuelve una excepción.

```
inline void activa_capa(char icono) {
    switch(icono) {
        case 'H':
                if (!cap_hospitales_.existente()) throw Capa_Inexistente();
                cap_hospitales_.activa();
                 break;
        case 'R':
                if (!cap_restaurantes_.existente()) throw Capa_Inexistente();
                 cap_restaurantes_.activa();
                break;
        case 'B':
                if (!cap_bus_.existente()) throw Capa_Inexistente();
                cap_bus_.activa();
                break;
    }
}
```

```
inline void desactiva_capa(char icono) {
    switch(icono) {
        case 'H':
            if (!cap_hospitales_.existente()) throw Capa_Inexistente();
            cap_hospitales_.desactiva();
            break;
        case 'R':
            if (!cap_restaurantes_.existente()) throw Capa_Inexistente();
            cap_restaurantes_.desactiva();
            break;
        case 'B':
            if (!cap_bus_.existente()) throw Capa_Inexistente();
            cap_bus_.desactiva();
            break;
    }
}
```

b.4) agrega_capa ( ): Recibe el nombre de la capa y su icono, crea la capa e inserta en el mapa.

```
inline void agrega_capa(const capa& c) {    //Copio 1 capa externa a 1 interna.
    if (c.icono()=='H') {
        cap_hospitales_=c;
        cap_hospitales_.existe();
    }
    else {
        if (c.icono()=='R') {
            cap_restaurantes_=c;
            cap_restaurantes_.existe();
        }
        else {
            if (c.icono()=='B') {
                cap_bus_=c;
                cap_bus_.existe();
            }
        }
    }
}
```

2. Dado el siguiente programa:

```
#include<iostream>
struct B {
    void f ( ) { std::cont << "f ( ) de B" << std::endl; }
};
struct D:B {
    void f ( ) { std::count<<"f ( ) de D" << std::ende;}
};
void f ( B b) {
    std::count << "f ( ) externa"<< std::endl; b.f( );
}
int main( ) {
    B b;
    D d;
    f(b);
    f(d);
}
```

a) Diga si tiene error de compilación o ejecución. Modifique el código para solucionarlo y después escribe lo que imprime. Si no, sólo lo que imprime.

No hay errores ni de compilación ni de ejecución.

B b;	b es un objeto de la clase B
D d;	d es un objeto de la clase D que deriva de B
f(b);	f es una función que toma un objeto del tipo B al recibir b (que es del tipo B) se ejecuta la función f de la clase B. Imprime: f ( ) externa f ( ) de B
f(d);	f recibe un objeto del tipo B, puede, por tanto recibir objetos de clases derivadas por lo que esto es válido, ya que d es un objeto derivado de la clase B. Sin embargo, al ejecutarse, se toma por defecto la función miembro de B. Imprime: f ( ) externa f ( ) de B

b) Repite el anterior pero suponiendo que char B :: f ( ) como virtual.

El poner virtual no soluciona nada, imprime lo mismo.

c) Repite el anterior pero teniendo en cuenta que el parámetro función externa f ( ) se recibe por referencia.

B b;	b es un objeto de la clase B
D d;	d es un objeto de la clase D que deriva de B
f(b);	f es una función que toma un objeto del tipo B al recibir b (que es del tipo B) y ser la función f virtual, el C++ deduce que debe usar la función f de la clase B. Imprime: f ( ) externa f ( ) de B
f(d);	f recibe un objeto del tipo B, puede, por tanto recibir objetos de clases derivadas por lo que esto es válido, ya que d es un objeto derivado de la clase B. Además, al paso por referencia, y ser la función miembro f virtual el C++ sabe que debe llamar a la función miembro f de la clase D, ya que el objeto pasado es de la clase D. Imprime: f ( ) externa f ( ) de D

d) Repita el 2º apartado suponiendo que la definición de f( ) (externa) se ha cambiado a :

```
void f ( B *b) { std::count<< " f ( ) externa" << std :: endl; b-> f ( ); }
```

Cambios en el main:

f(b) → f(&b);

f(d) → f(&d);	Como usamos punteros y f es virtual C++ sabe que debe llamar a la función f de la clase D, ya que se pasa el objeto de la clase D. Imprime lo mismo que en c.

3. Responde a las siguientes cuestiones:

a) A la hora de resolver las ambigüedades en la sobrecarga y buscar qué función sobrecargada llamar, el compilador de C++ realiza una serie de pasos. Indique cuáles son esos pasos y en qué orden.

Sobrecarga: Consiste en declarar varias funciones con el mismo nombre dentro de una clase. Se distingue cuando se quiere usar una u otra según los parámetros que tenga.

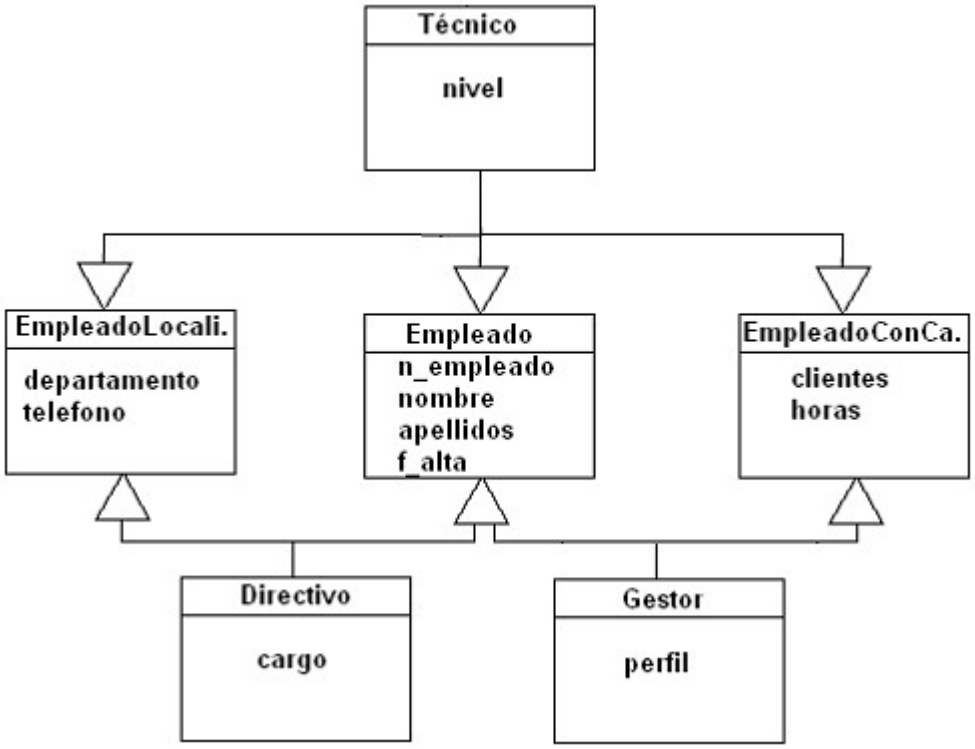
- Facilitan el uso de una misma función de forma distinta para objetos de una clase concreta.
- La sobrecarga se da en una misma clase.
- La sobrecarga es ventajosa cuando se quiere dotar de versatilidad a los objetos de la clase.

b) Además indique diferencias y semejanzas entre sobrecargar una función y definirla paramétricamente. ¿Cuándo es adecuado sobrecargar una función y cuándo hacer una función paramétrica?

- Funciones paramétricas (polimorfismo): Consiste en funciones redefinidas en clases derivadas. El concepto general es el mismo, aunque la forma de hacerse es distinto (un coche y una moto pueden "arrancarse", pero la forma en que se arrancan es distinta)
- Permiten que el programa sepa como ejecutar la función según el tipo de objeto al que se le esté pidiendo que ejecute dicha función.
- Son funciones redefinidas en distintas clases.
- Las funciones paramétricas son útiles cuando se quiere usar una función miembro sin tener en cuenta el objeto sobre el que se ejecuta la función.
- Se programan a través de funciones virtuales y herencia.
- Se usan a través de punteros a objetos.

4. Tres tipos de empleados: gestores, técnicos y directivos. Todos tienen un número de empleados, nombre, apellidos y fecha de alta. Los gestores y los técnicos almacenan el número de clientes atendidos y el número de horas trabajadas. Los directivos y técnicos almacenan un teléfono de la empresa para localizarlos y el departamento. Los gestores almacenan su perfil, los técnicos su nivel y los directivos su cargo.

a) Dibujar el diagrama de clases.



b) Escriba el operador de inserción (<<) para empleado y complete las clases para que el operador de inserción sea correcto.
Ejemplo de Salida:
Técnico: 111, Antonio Pérez Muñoz, 12/12/2000, 50 clientes, 100 horas, Redes, 956111111, 3.
Gestor: 112, Felipe Fernández Márquez, 15/08/2002, 30 clientes, 50 horas, Finanzas.
Directivo: 113, Manuel González Jiménez, 03/07/2005, Atención al cliente, 956222222, Gerente.
(NOTA: Técnico, Gestor y Directivo no tiene que mostrarlo el operador de inserción).

```
ostream& operator<<(ostream& salida, const Tecnico& t) {
    salida << t.n_empleado() <<","<<t.nombre()<<","<<t.apellidos()<<","<<t.f_alta()<<",";
    salida << t.clientes() << "clientes, "<<t.horas()<<" horas, ";
    salida << t.departamento() << "," << t.telefono() <<","<<t.nivel();
    salida << endl;
    return salida;
}


class Empleado {
    public:
        //…
        Empleado(string n, string nom, string ape, string fe):
                    n_empleado_(n),nombre_(nom),apellidos_(ape),f_alta_(fe) {}
        Empleado() {}
        string n_empleado() const { return n_empleado; }
        string nombre() const { return nombre_;
         string apellidos() const { return apellidos_; }
         string f_alta() const { return f_alta_; }
    private:
        string n_empleado_;
        string nombre_;
        string apellidos_;
        string f_alta_;
};


Class EmpleadoConCartera{
    public:
        //…
        EmpleadoConCartera(){}
        EmpleadoConCartera(unsigned cli, unsigned ho):clientes_(cli),horas_(ho){}
        unsigned clientes()const {return clientes_;}
        unsigned horas() const {return horas_;}
    protected:
        unsigned  clientes_;
        unsigned horas_;
};
```

```
Class Gestor: public Empleado, public EmpleadoConCartera{
    public:
        //…
        Gestor() {}
        Gestor(string n, string nom, string ape, string fe,
        unsigned cli, unsigned ho,
        string pe):perfil_(pe){
                n_empleado_=n;
                nombre_=nom;
                apellidos_=ape;
                f_alta_=fe;
                clientes_=cli;
                horas_=ho;
        }
    private:
        string perfil_;
};


Class EmpleadoLocalizable{
    public:
        //…
        EmpleadoLocalizable(){};
        EmpleadoLocalizable(string de, string te):departamento_(de),telefono_(te){}
        string departamento() const {return departamento_;}
        string telefono() const {return telefono_;}
    protected:
        string departamento_;
        string telefono_;
};


Class Tecnico:public Empleado, public EmpleadoConCartera, public EmpleadoLocalizable{
    public:
        //…
        Tecnico() {}
        Tecnico(string n, string nom, string ape, string fe,
        unsigned cli, unsigned ho,
        string de, string te,
        unsigned ni):nivel_(ni){
                n_empleado_=n;
                nombre_=nom;
                apellidos_=ape;
                f_alta_=fe;
                clientes_=cli;
                horas_=ho;
                departamento_=de;
                telefono_=te;
        }
        unsigned nivel()const {return nivel_;}
        friend ostream& operator<<(ostream& salida, const Tecnico& t);
    private:
        unsigned nivel_;
};
```

```
ostream& operator<<(ostream& salida, const Tecnico& t) {
    salida << t.n_empleado() <<","<<t.nombre()<<" "<<t.apellidos()<<","<<t.f_alta()<<",";
    salida << t.clientes() << "clientes, "<<t.horas()<<" horas, ";
    salida << t.departamento() << "," << t.telefono() <<","<<t.nivel();
    salida << endl;
    return salida;
}


Class Directivo:public Empleado, public EmpleadoLocalizable{
    public:
        //…
        Directivo(){}
        Directivo(string n, string nom, string ape, string fe,
        string de, string te,
        string ca):cargo_(ca){
                n_empleado_=n;
                nombre_=nom;
                apellidos_=ape;
                f_alta_=fe;
                 departamento_=de;
                telefono_=te;
        }
    private:
        string cargo_;
};
```