Programación 4

Conceptos Básicos de Orientación a Objetos (2da parte)

Operación y Método

- **Operación**: especificación de una transformación o consulta que un objeto puede ser llamado a ejecutar
- **Método**: implementación de una operación para una determinada clase

Operación y Método

```
class Usuario {
private:
      int idUsuario;
      DateTime* nacimiento;

public:
      int getEdad()
      {
            ... // un cierto algoritmo
      }
};
```

Polimorfismo y Redefinición

- Es la capacidad de asociar diferentes métodos a la misma operación
- Cuando en una jerarquía de generalización se encuentra más de un método asociado a la misma operación, se dice que dicha operación está redefinida

Redefinición de Operaciones

```
Class Usuario {
  private:
     int permisos; // set de bits
  public:
    virtual int getPermisos();
};

int Usuario::getPermisos(){
        return permisos;
}

class Admin: public Usuario {
public:
    static int ADMIN_SISTEMA;
public:
    int getPermisos();
};

int Admin::getPermisos(){
        int p = Usuario::getPermisos();
        return (p | ADMIN_SISTEMA);
}
```

Interfaz

- Una interfaz "**es un conjunto de operaciones al que se le aplica un nombre**"
- No define un estado para las instancias de estos elementos, ni tampoco asocia un método a sus operaciones
- Este conjunto de operaciones caracteriza el (o parte del) comportamiento de instancias de clases

Interfaz (2)

- Una clase realiza una interfaz en forma análoga a cómo un tipo implementa un TAD
- Cuando una clase C realiza una interfaz `I`, puede decirse que una instancia de C:
  - "Es de C" o "es un C" pero tambien que,
  - "Es de `I`" o "es un `I`"
- Esto permite quebrar las dependencias hacia "las implementaciones" cambiándolas por una sola dependencia hacia "la especificación" (la interfaz)

Interfaz (3)

```
// Una interfaz para implementar lecturas secuenciales sobre algo
class Archivo {
public:
        // lee hasta n caracteres y avanza
        // la última posición leída y devuelve
        // su contenido en el string
        virtual string read(int n) = 0;
        // pone la última posición leída al
        // principio del archivo
        virtual void rewind() = 0;
        // dice en qué posición se va a leer
        virtual int position() = 0
        // dice cuántos caracteres hay disponibles
        virtual int available() = 0

        virtual ~Archivo() {};
};
```

Cualquier clase derivada de Archivo será un Archivo.

Realización

- Es una relación entre una especificación y su implementación
- Una forma posible de realización se produce entre una interfaz y una clase
  - Se dice que una clase *C* realiza una interfaz *I* si *C* implementa todas las operaciones declaradas en *I*, es decir provee un método para cada una

Realización (2)

```
class CPPArchivo: public Archivo {
private:
        fstream f;
public:
        CPPArchivo(const string& ruta);

        // implementación de interfaz Archivo
        string read(int n);
        void rewind();
        int position();
        int available();

        // operaciones extra
        void close();
        void write(const string& s);
        void setPosition();

        ~CPPArchivo();
};
```

Realización(3)

```
CPPArchivo::CPPArchivo(const string& ruta):
        f(ruta)
{}
// ...
string CPPArchivo::read(int n){
        char *buffer = new char[n];
        f.read(buffer, n);
        string res = string(buffer, n);
        delete buffer;
        return res;
}
// ...
int CPPArchivo::position(){
        return f.tellg();
}
// ...
CPPArchivo::~CPPArchivo(){
        f.close();
}
```

Realización (4)

- Una interfaz puede ser entendida como la especificación de un *rol* que algún *objeto* debe desempeñar en un sistema
- Un objeto puede desempeñar más de un rol:
  - Una clase puede realizar cualquier cantidad de interfaces
- Un rol puede ser desempeñado por objetos de características diferentes:
  - Una interfaz puede ser realizada por cualquier cantidad de clases

Realización (5)

- Es posible tipar a un objeto (además de como es usual mediante la clase de la cual es instancia) también mediante *una* de las interfaces que su clase realiza
- Por lo que si un objeto es declarado como de tipo *I* (en una lista de parámetros, como atributo, etc.), siendo *I* una interfaz, significa que ese objeto no es una instancia de *I* (lo cual no tiene sentido) sino que es instancia de una clase que realiza la interfaz *I*

Realización (5)

```
class Comparable {
public:
        // devuelve algo mayor a 0 si this > b
        // 0 si this = b, o algo negativo si this < b
        virtual int comparar(Comparable *b) = 0;
        virtual ~Comparable() {};
};

class Representable {
public:
        // devuelve una representación del objeto
        virtual string toString();
        virtual ~Representable() {};
}
```

Realización (6)

```
class Integer: public Comparable, public Representable {
private:
        int val;
public:
        Integer(int val);
        int getVal();
        string toString();
        int comparar(Comparable *c);
};

ostream& operator <<(ostream& o, Representable& r)
{
        return o << r.toString();
}

void Utilidades::ordenar(Comparable **arr, int largo)
{
        // ;)
}
```

Realización (7)

- Este mecanismo permite abstraerse de la implementación concreta del objeto declarado
- En lugar de exigir que dicho objeto presente una implementación determinada (es decir, que sea instancia de una determinada clase), se exige que presente un determinado comportamiento parcial (las operaciones declaradas en *I*)
- Este comportamiento es implementado por una clase que realice la interfaz, y de la cual el objeto en cuestión es efectivamente instancia

Realización (8)

- Notar que en la definición previa se asume que la clase que realiza la interfaz es concreta
- Es posible sin embargo que una interfaz sea realizada por una clase abstracta
- En cuyo caso debe declarar todas las operaciones de la interfaz aunque no esta obligada a implementarlas a todas
- Si *C* es abstracta y realiza la interfaz *I*, entonces un objeto declarado como de tipo *I* debe ser instancia de alguna subclase concreta de *C* (o de otra clase que realice la interfaz *I*)

Dependencia

- Es una relación asimétrica entre un par de elementos donde el elemento independiente se denomina *destino* y el dependiente se denomina *origen*
- En una dependencia, un cambio en el elemento destino puede afectar al elemento origen
- Las asociaciones, generalizaciones y realizaciones caen dentro de esta definición general
  - Pero son una forma más fuerte de dependencia
  - En esos casos la dependencia se considera asumida y no se expresa explícitamente (por ejemplo en Diagramas de Clases de Diseño)