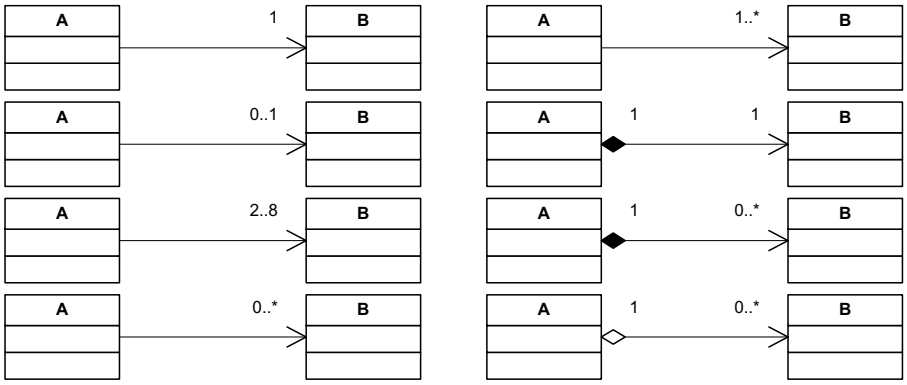
Programación Avanzada

PRÁCTICO 6

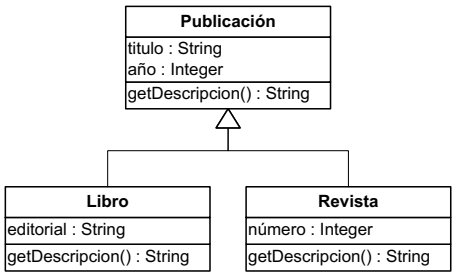
Ejercicio 1 (básico, imprescindible)

Implementar en C++ los siguientes Diagramas de Clases de Diseño. Considerar en la implementación constructores y destructores. Incluir en la clase A operaciones para el manejo de los links con instancias de B.



Ejercicio 2 (básico, imprescindible)

Implementar en C++ el siguiente DCD. Considerar para ello que la operación `getDescripcion()` retorna la concatenación de la información de cada publicación. Incluir operaciones `get` y `set` para cada atributo.



Ejercicio 3 (básico, imprescindible)

Implementar completamente el diseño del sistema de usinas y turbinas realizado en el ejercicio 5 del Práctico 5. Asumir una clase ColTurbina que ofrece los servicios de colección para la clase Turbina.

Ejercicio 4 (medio, imprescindible)

Diseñar la realización de la interfaz ICollection e IIterator mediante cada una de las siguientes estructuras de datos. Implementar en C++ el diseño construido.

a) Lista enlazada

b) Lista doblemente enlazada

c) Arreglo dinámico

Ejercicio 5 (medio, imprescindible)

Diseñar la realización de la interfaz IDictionary mediante cada una de las siguientes estructuras de datos. Implementar en C++ el diseño construido.

a) Lista enlazada

b) Lista doblemente enlazada

c) Arreglo dinámico

d) Árbol binario de búsqueda

Ejercicio 6 (medio, de práctica)

Realizar el diseño genérico de los siguientes tipos abstractos de datos. Implementar en C++ el diseño construido.

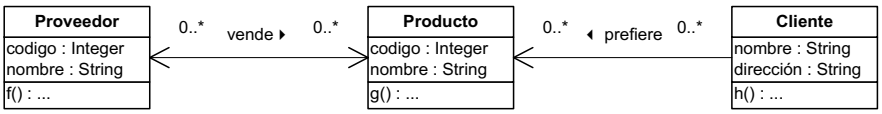
a) Set

b) Stack

c) Queue

Ejercicio 7 (medio, imprescindible)

Considerar el siguiente Diagrama de Clases de Diseño:



a) Implementar en C++ las clases necesarias para los pseudo-atributos de cada una de las clases del diagrama considerando que en el diseño de las interacciones se requirió que:

   i. La operación `Proveedor::f()` hace un `find()` sobre los productos que vende utilizando el nombre.

   ii. La operación `Proveedor::f()` itera sobre todos los productos que vende.

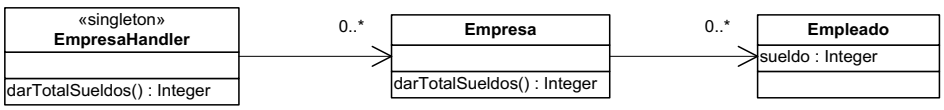
   iii. La operación `Producto::g()` hace un `member()` sobre los proveedores que lo venden por código.

   iv. La operación `Cliente::h()` itera sobre los productos que prefiere.

i) Modificar la implementación realizada considerando ahora que la operación `Proveedor::f()` también hace un `find()` sobre los productos que vende utilizando el código.

Ejercicio 8 (medio, imprescindible)

Implementar en C++ el diseño presentado en el siguiente Diagrama de Clases de Diseño. Incluir en la implementación las clases de colecciones e iteradores que considere necesarias. Implementar además un procedimiento `main()` en el que se utilice la operación de sistema `darTotalSueldos()`.



Ejercicio 9 (avanzado, imprescindible)

Implementar en C++ el diseño construido en el ejercicio 5 del Práctico 6 referente al sistema para el reloj digital. Considere diferentes estrategias para el manejo de memoria de los estados asociados al reloj.

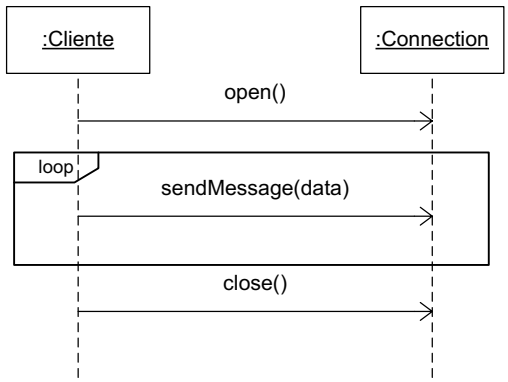
Ejercicio 10 (avanzado, de práctica)

Se desea desarrollar una aplicación estilo File Manager (Administrador de Archivos o Explorador). Esta aplicación permitirá visualizar el contenido de directorios, que pueden ser archivos u otros directorios. Estos ítems tienen un nombre y en el caso de los archivos, tienen además un tamaño; en el caso de directorios tienen además una colección de ítems (los archivos y directorios contenidos en ese directorio). El tamaño de un archivo es directamente el valor almacenado en su atributo correspondiente, mientras que el tamaño de un directorio se calcula como la suma de los tamaños de los archivos y directorios contenidos en él. Se desea tener disponible una operación `getTamaño()` que permita obtener el tamaño de un archivo o un directorio.

Implementar completamente en C++ una solución a este problema, resolviendo previamente el diseño utilizando Design Patterns.

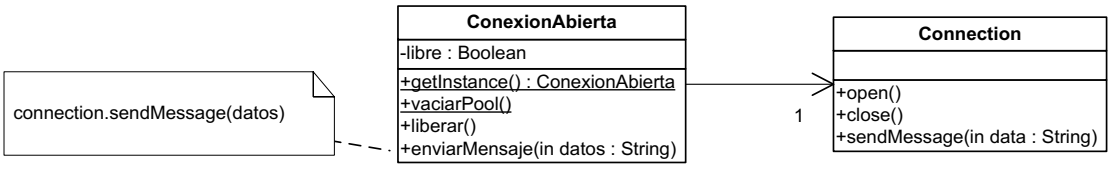
Ejercicio 11 (avanzado, de práctica)

Se necesita construir un mecanismo que permita a la aplicación que se está desarrollando comunicarse con un servidor remoto. Para esto se utilizará una clase provista por terceros llamada `Connection` cuyo uso sigue el siguiente protocolo:



Se busca minimizar la cantidad de veces que se abren conexiones (`open`) porque éste es un proceso pesado. Por ello se decidió construir un *pool de conexiones*, esto es, un grupo de conexiones que se mantienen abiertas y prontas para enviar mensajes. Cabe destacar que de todas formas es importante cerrar las conexiones (`close`) en algún momento para no dejar recursos ocupados en el servidor remoto.

Para la construcción del pool se decidió utilizar una variante del patrón Singleton que maneje N instancias en lugar de una única, siendo N una constante definida en tiempo de compilación. Se realizó el siguiente diseño parcial:



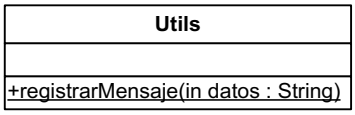
Cuando una conexión es devuelta por la operación `getInstance` se marca como "ocupada", y es responsabilidad del programador que la utiliza marcarla como "liberada" (mediante la operación `liberar`) al terminar de usarla. Por lo tanto, la operación `getInstance` debe:

- Devolver sólo conexiones libres
- Crear conexiones a demanda si no tiene ninguna conexión libre y no llegó a N conexiones abiertas
- Devolver NULL si las N conexiones están ocupadas

La operación `vaciarPool` libera totalmente la memoria reservada por el pool de conexiones.

i. Implementar en C++ completamente la clase `ConexionAbierta`, de forma que resuelva todos los puntos mencionados por sí misma. Incluir constructor(es) y destructor.

Luego de tener el mecanismo de comunicación funcionando surgió la necesidad de registrar todos los mensajes enviados. Para esto se cuenta con la siguiente clase utilitaria:



ii. Implementar en C++ el cambio teniendo en cuenta las siguientes restricciones
   - NO se puede modificar el cuerpo de la operación `enviarMensaje` de la clase `ConexionAbierta`
   - La modificación debe ser totalmente transparente para el resto de la aplicación
   - Evite que la solución contenga código duplicado

Ejercicio 12 (medio, imprescindible)

Considerar la siguiente definición de clase en C++, suponiendo que todas las operaciones están correctamente implementadas. Considerar también el siguiente procedimiento `main()`.

```
class Clase {
    private:
        int entero;
        float real;
    public:
        Clase();
        Clase(int, float);
        Clase(Clase &);
        ~Clase();
        Clase operator=(Clase &);
        Clase & operator+(Clase);
        void desplegar();
};
```

```
void main() {
    Clase c1;
    Clase c2(3,2.54);
    Clase c3=c2;
    c1=c2+c3;
    c1.desplegar();
}
```

Suponer que se ejecuta el programa. Hacer una lista enumerando ordenadamente cada operación invocada a raíz de dicha ejecución, justificando. Recordar que una línea de código puede causar que se invoque más de una operación.

Ejercicio 13 (avanzado, imprescindible)

```
class claseA {
   public:
      void func1(int a);
      virtual void func2(short a);
      virtual void func3(int a);
};

class claseB : public claseA {
   public:
      virtual void func1(int a);
      virtual void func2(int a);
      void func3(int a);
      void operator=(claseB &)
};

class claseC : public claseB {
   public:
      virtual void func1(int a);
      virtual void func2(short a);
      void func3(int a);
};

class claseD : public claseA {
   public:
      void func1(int a);
      virtual void func2(short a);
};
```

a) Considerando la definición de clases anterior, indicar a qué clase corresponde el método invocado en cada invocación:

```
claseA *a;
claseB *b;
claseC *c;
claseD *d;
claseA *ap;
claseB *bp;
int in = 0;
short sh = 4;

a = new claseA();
b = new claseB();
c = new claseC();
d = new claseD();

claseB bb;
bb = (*c);
bb.f1(in);

ap = b;
bp = c;

ap->func1(in);
ap->func3(in);
bp->func3(in);
bp->func1(in);
bp->func2(in);

ap = c;
ap->func2(in);
ap->func2(sh);

ap = b;
ap->func2(sh);

ap = d;
ap->func1(in);
a->func1(in);
```

b) En este ejercicio se han manejado conceptos como polimorfismo, binding dinámico, overriding y sobrecarga de funciones. Indicar en qué parte de (a) se utilizan los mismos.

Ejercicio 14 (medio, de práctica)

Explicar qué método se ejecuta al realizarse cada una de las sentencias del siguiente código. Considerar que todas las operaciones declaradas en cada clase están correctamente definidas.

```
void main(){
	ClaseA a;
	ClaseB b;
	ClaseC c;
	ClaseD d;
	ClaseA * A[4];

	A[0]= &a;
	A[1]= &b;
	A[2]= &c;
	A[3]= &d;

	a=b;
	a=c;
	b=c;
	a.F1();
	a.F2();
	a.F3();
	a.F4();
	b.F1();
	b.F2();
	b.F3();
	b.F4();
	c.F1();
	c.F2();
	c.F3();
	c.F4();
	d.F1();
	d.F2();
	d.F3();
	d.F4();

	A[0]->F1();
	A[1]->F1();
	A[2]->F1();
	A[3]->F1();
	A[0]->F2();
	A[1]->F2();
	A[2]->F2();
	A[3]->F2();
	A[0]->F3();
	A[1]->F3();
	A[2]->F3();
	A[3]->F3();
	A[0]->F4();
	A[1]->F4();
	A[2]->F4();
	A[3]->F4();
}
```

```
class ClaseA {

public:
	ClaseA();
	void F1();
	virtual void F2();
	virtual void F3();
	virtual void F4();
};

class ClaseB:public ClaseA {

public:
	ClaseB();
	void F2();
	void F3();
};

class ClaseC:public ClaseB {

public:
	ClaseC();
	void F1();
	void F2();

};

class ClaseD:public ClaseA{

public:
	ClaseD();
	void F4();
};
```

Ejercicio 15 (básico, imprescindible)

La AUF (Asociación Uruguaya de Fútbol) desea desarrollar un sistema que automatice la actividad contractual en el fútbol profesional uruguayo. El equipo de desarrollo decidió implementar un pequeño prototipo que brinde las funcionalidades básicas del sistema. La primera tarea realizada fue la construcción de la Visión del problema, presentado a continuación:

> La AUF cuenta con jugadores afiliados de los cuales se tiene conocimiento de su nombre y edad. Al igual que los jugadores, existen clubes afiliados, de los cuales se conoce el nombre. Un jugador puede haber jugado en el mismo club durante varios períodos.
>
> Los contratos firmados por un jugador son de carácter anual por un monto en moneda nacional. Como estrategia de control se decidió fijar el período de contratos al comienzo del año y no permitir firmar contratos por más de un año.
>
> Los contratos firmados entre un jugador y un club determinado se archivan en un mismo expediente el cual es identificado por un número de expediente.
>
> Los contratos se firman por medio de un contratista. Cada contratista puede participar en varios contratos durante el mismo año.
>
> Cada entidad afiliada a la AUF (jugadores, clubes y contratistas) poseen número (que los identifica entre sus pares) y año de afiliación. No interesa conocer datos de estos antes de su afiliación.

Para la realización del prototipo, el equipo de desarrollo determinó ocho etapas a seguir.

Se pide:

Llevar a cabo cada una de las etapas para culminar con el desarrollo del prototipo.

a) Realizar el Modelo de Dominio que represente la realidad planteada. Escribir en lenguaje natural las restricciones del modelo. No incluir las restricciones que indiquen que un atributo es positivo ni de unicidad de identificadores.

b) Realizar el Diagrama de Secuencia del Sistema para el siguiente caso de uso:

> "Dar los números de expediente de los contratos realizados por un contratista determinado, con un club en particular, a partir del año indicado"

Los datos necesarios (contratista, club, y año) serán indicados por el usuario.

c) Realizar los contratos de las operaciones del proceso anterior. Las pre- y poscondiciones deben ser expresadas en lenguaje natural.

d) Diseñar las interacciones realizando los diagramas de comunicación de las operaciones involucradas en el caso de uso anterior.

e) Diseñar la estructura realizando el Diagrama de Clases de Diseño.

f) Escribir los .h, excepto los relativos a las colecciones. Suponer la existencia de clases ColXX representando las colecciones de XX.

g) Implementar completamente la solución junto con un caso de prueba.