Como se mencionó anteriormente, los objetos en la Programación Orientada a Objetos interactúan entre sí. Existen distintos tipos de interacciones y cada una de ellas tiene sus características específicas y, por supuesto, su nombre.

Composición

Una de las relaciones más habituales es la que permite que los atributos de una clase sean objetos de otra clase. Es decir, una clase está compuesta por objetos de otras clases.

Si seguimos con el ejemplo que hemos visto en el apunte anterior, un examen está compuesto de elementos complejos como una fecha y hora, una materia y el alumno que lo rinde. También podría registrar otros elementos más simples como la nota.

Un atributo como la nota podemos representarlo fácilmente con un float o int dependiendo del contexto. Eventualmente será necesario validarlo para que no pueda almacenar un valor inconsistente como -1 o 1000000. Pero no tendremos grandes dificultades de representar este dato. Sin embargo, la fecha, hora, materia y alumno sí son elementos más complejos.

Por ejemplo, la fecha del examen va a ser un objeto de la clase Fecha. Que podrá representar el día, mes y año en que se realiza la evaluación. Lo mismo con la Hora que representará la hora, minutos y segundos en que comienza el examen (eventualmente podríamos guardar también la hora en que el examen termina o la duración en minutos y calcularlo). La Materia podría almacenar el código de materia, el nombre de materia y carrera a la que pertenece. Al igual que el Alumno podría almacenar un sinfín de atributos que sean de utilidad para el sistema que estamos creando.

Un posible código que represente la declaración de nuestra clase Examen podría ser el siguiente:

```
class Examen{
  Fecha _fecha;
  Horario _horario;
  Materia _materia;
  Alumno _alumno;
  float _nota;
};
```

Si cada clase tuviese métodos get y set para todos sus atributos. Y a su vez, nuestra clase tuviese métodos get para cada uno de los atributos. Entonces podríamos realizar algo como lo siguiente:

```
void miFuncion(){
  Examen examen;
  // Se asignan valores a los atributos del objeto datos al examen
  // mediante alguna función o método

  // cargarExamen(examen);
  if (examen.getFecha().getMes() == 12){
    // El examen se rindió en Diciembre
  }
}
```

Analicemos un poco esta sucesión de getters que estamos haciendo. El objetivo final del if es preguntarse si el examen fue rendido en diciembre. Para ello, debemos obtener este dato de nuestro objeto examen pero claro está que el objeto que almacena el mes no es directamente el examen sino que es la fecha que lo compone.
Luego, la fecha es un atributo privado del objeto examen. Por lo que no podríamos acceder directamente desde la función. Un código como el que se describe a continuación nos generaría un error de compilación:

```
if (examen._fecha.getMes() == 12)
```

Código • Código incorrecto ya que _fecha es privado y no puede ser accedido desde fuera de la clase Examen.

Entonces, hacemos uso de nuestro getter para obtener el objeto fecha que está dentro de nuestro objeto examen.

`examen.getFecha()` nos devolverá en tiempo de ejecución una instancia de la clase Fecha con todos los atributos que registramos, entre ellos, el mes. Bien podríamos asignar dicha instancia en un objeto auxiliar de la clase Fecha.

```
Fecha aux = examen.getFecha();
if (aux.getMes() == 12){
// Es el mes de Diciembre
}
```

Código · Otra manera de resolverlo pero haciendo uso de una variable auxiliar.

Si bien parece complejo, es simplemente llevar la idea del encapsulamiento a atributos que no sean tipos de datos simples sino instancias de clases.

A modo de repaso veamos a la clase Examen de la manera que la habíamos trabajado en el apunte anterior. La representación UML de la misma podría ser la siguiente:

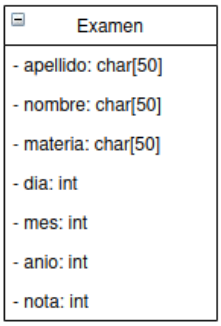

Diagrama · Representación de la clase Examen sin hacer uso de composición.

Si analizamos con más detalle un examen y nos abstraemos de los datos, o mejor dicho, los representamos de una forma más orientada a objetos. Podríamos decir que nuestra clase Examen en realidad está compuesto por un Alumno, una Materia, una Fecha y una nota. Luego, podríamos decir que un Alumno está compuesto por un apellido y un nombre. Y le agregaremos un número de legajo.
Una Materia contiene el nombre de la materia. Le agregaremos un código y el nombre de la carrera (que podría ser un objeto de la clase Carrera si estamos con ganas de codificar). Y por último, una Fecha está formada por un día, mes y año.
El diagrama UML quedaría algo así:

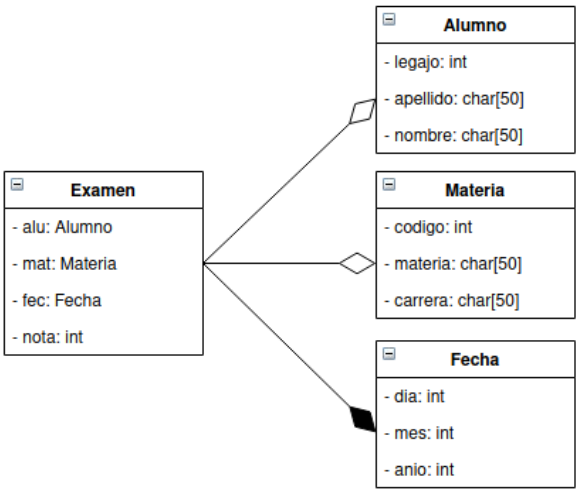

Diagrama · Representación de la clase Examen compuesta por objetos de las clases Alumno, Materia y Fecha.

Veamos cómo sería la declaración de la clase Examen y definición de sus métodos.

```
#ifndef EXAMEN_H
#define EXAMEN_H
#include "Alumno.h"
#include "Materia.h"
#include "Fecha.h"

class Examen
{
	public:
	Examen();
	Alumno getAlumno();
	void setAlumno(Alumno);
	Materia getMateria();
	void setMateria(Materia);
	Fecha getFecha();
	void setFecha(Fecha);
	int getNota();
	void setNota();

	private:
	Alumno _alumno;
	Materia _materia;
	Fecha _fecha;
	int _nota;
};

#endif // EXAMEN_H
```

Código · Nueva clase Examen codificada en C++.

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

void Examen::setFecha(Fecha fecha){
	_fecha = fecha;
}
void Examen::setAlumno(Alumno alumno){
	_alumno = alumno;
}
void Examen::setMateria(Materia materia){
	_materia = materia;
}
void Examen::setNota(int nota){
	_nota = nota;
}
Fecha Examen::getFecha(){
	return _fecha;
}
Alumno Examen::getAlumno(){
	return _alumno;
}
Materia Examen::getMateria(){
	return _materia;
}
int Examen::getNota(){
	return _nota;
}
void Examen::mostrar(){
	_alumno.mostrar();
	_materia.mostrar();
	_fecha.mostrar();
	cout << "Nota: " << _nota << endl;
}
```

Código · Definición de los métodos de la clase Examen

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

int main()
{
```

```
    Examen e;
    Materia mat;
    Alumno alu;
    Fecha f(15, 4, 1999);
    mat.setNombre("Programación II");
    mat.setCodigo(21);
    mat.setCarrera("TUP");
    alu.setApellido("Perez");
    alu.setNombre("Juan");
    alu.setLegajo(1111);
    e.setAlumno(alu);
    e.setMateria(mat);
    e.setFecha(f);
    e.setNota(10);
    e.mostrar();
    return 0;
}
```

Código · Programa main que demuestra el uso de la clase Examen.

```
Perez, Juan (1111)
Programación II (21) - TUP
15/4/1999
Nota: 10
```

Salida · Salida por pantalla del programa en ejecución.