Ingeniería de Software I
Ocultamiento de Información y Encapsulamiento

Objetivo

Aplicar los conceptos de *ocultamiento* y *encapsulamiento* para ocultar los detalles de implementación de clases.

Descripción

El *ocultamiento de información* permite evitar que el código de una *aplicación cliente* tenga acceso a los datos de una *clase* que bien pueden requerir un trato cuidadoso, así cualquier solicitud para modificar o consultar el estado de un *objeto* deba ser mediante el uso de *métodos de interfaz* (métodos públicos) y bajo las restricciones que en ellos se especifiquen. El *encapsulamiento* permite que ocultemos los detalles de implementación de una clase, y que si necesitamos mejorar su funcionamiento, siempre y cuando se mantengan sin cambios sus *métodos de interfaz*, las aplicaciones cliente continuarán funcionando correctamente.

La práctica trata acerca de la implementación de una clase que modele una Fecha, la cual siempre mantenga un estado consistente en la representación de cualquier objeto fecha posible de la realidad.

<u>Años</u>

Cualquier año sea válido, esto es, todo el rango de los enteros positivos, cero o negativos.

<u>Meses</u>

El número correspondiente a cada mes a validar es como sigue:

Mes	Número de mes
Enero	1
Febrero	2
Marzo	3
Abril	4
Mayo	5
Junio	6
Julio	7
Agosto	8
Septiembre	9
Octubre	10
Noviembre	11
Diciembre	12

Días

Los días válidos para cada mes son como sigue:

Mes	Días
Enero	Del 1 al 31
Febrero	Del 1 al 28 para todos los años 29 para los años bisiestos
Marzo	Del 1 al 31
Abril	Del 1 al 30
Mayo	Del 1 al 31
Junio	Del 1 al 30
Julio	Del 1 al 31
Agosto	Del 1 al 31
Septiembre	Del 1 al 30
Octubre	Del 1 al 31
Noviembre	Del 1 al 30
Diciembre	Del 1 al 31

Años bisiestos

Un año bisiesto se presenta cada 4 años, contados a partir del año cero hacia adelante y hacia atrás, esto es:

-4, 0, 4, 8, 12…etc.

…son años bisiestos.

Hay una excepción para los años bisiestos, esto es cada 100 años:

100, 200, 300, 500, 600, 700…etc.

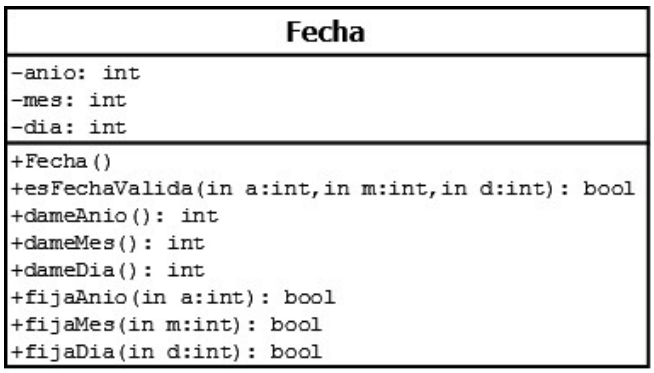
…no son años bisiestos.

Cómo se habrá notado en la serie anterior, hay una excepción a la excepción, esto es que cada 400 años:

400, 800, 1200…etc.

…sí son años bisiestos.

Requerimientos Generales



1. Entregar archivo(s) *fuente* para aplicación de consola que cumpla(n) con los siguientes requerimientos.
2. Codificación de la clase Fecha tal cual se ilustra en el diagrama de clase UML.
3. Inicializar todos los atributos con datos válidos por medio de un constructor sin parámetros, usando la fecha de nacimiento del/la alumn@.
4. Declarar y definir todos los métodos *dame()*.
5. Declarar y definir todos los métodos *fija()*.
6. Contemple que un año bisiesto es múltiplo de 4, no múltiplo de 100 pero sí puede ser múltiplo de 400.
7. Programarla lo más completa y sintética posible de forma que siempre represente en sus atributos una fecha válida; es decir, si se invocara en main:
   cout << f.dameAnio() << "/" << f.dameMes()<< "/" << f.dameDia()
   …en cualquier momento debiera aparecer una fecha válida.
8. Validación en cada uno de los métodos fija para que los atributos en conjunto representen siempre una fecha correcta (al momento del return en el método *fija()*) sea cual fuere el orden posible de invocación a los métodos *fija()* desde cualquier punto del programa.
9. Si al tratar de cambiar algún atributo mediante un método *fija()*, la nueva fecha no sería válida, los atributos permanezcan sin modificación.
10. Evitar leer de o imprimir a la consola desde métodos al interior de la clase (en Java el método main no forma parte de ninguna clase).
11. Invocar al método esFechaValida() dentro de cada uno de los métodos *fija()*.
12. Codificar en un primer archivo fuente, en su programa principal (*main)* lo siguiente:
    a) Declaración de *variable de instancia* del tipo Fecha e instanciación de objeto;
    b) Permitir modificar todos los atributos del *objeto* instanciado, mediante los 3 métodos de interfaz (los *fija()*), pudiendo estos ser invocados en cualquier orden;
    c) Cada que se modifique el objeto, por invocar un método *fija()*, imprimir en consola todos y cada uno de los atributos del objeto;
    d) Un menú que permita modificar cualquier atributo del objeto, una opción de menú para cada atributo y una opción para salir del programa;
    e) Ejecutar de forma cíclica el menú.

Requerimientos para Puntos Extras

13. Entregar en el mismo comprimido carpetas diferentes para cada lenguaje:
   a) Una nombrada “C” con los códigos fuente en C (ANSI);
   b) Una nombrada “C++” con los códigos fuente en C++;
   c) Una nombrada “Java” con los códigos fuente en Java;
   d) Una nombrada “C#” con los códigos fuente en C#;
   e) Una nombrada “Python” con los códigos fuente en Python;
   f) Una nombrada “English” con los códigos fuente escritos en su totalidad en inglés (solo para el lenguaje de programación de su preferencia).

   …las primeras opciones se redacten absolutamente en español y la última absolutamente en inglés. Si incluye 2 carpetas con lenguajes OO, uno de ellos puede redactarse en inglés y sin necesidad de entregar la versión en español para ese lenguaje.

Criterios de Evaluación

- Los establecidos en las “Reglas de Operación y Evaluación” del curso.
- Cumplir con la fecha límite de entrega citada en el Excel de Actividades.
- Calificación en base a cobertura de requerimientos.
- Cumplir con Requerimientos de Valor Agregado en Código Fuente (hasta el req. “MM”).
- Entrega en tres lenguajes, ambos en el mismo archivo: C (ANSI), C++ (ANSI) y Java.
- Es indispensable la entrega de un programa cuyo código fuente sea completamente en idioma Español (a excepción de lo correspondiente a la API del lenguaje).