UNIVERSIDAD RICARDO PALMA **ESCUELA DE INGENIERÍA INFORMÁTICA**
FACULTAD DE INGENIERIA **Ciclo 2013-II**

Examen Parcial

CURSO: Arquitectura de Software
GRUPO(s): Todos

PROFESOR(s): Miguel Arrunátegui
FECHA: 9 de Mayo de 2013
HORA: 17:45
DURACION DEL EXAMEN: 2 horas
DEVOLUCION DEL EXAMEN CALIFICADO/ Fecha, hora y aula: 13-5-2013 En clase de Teoría

IMPORTANTE:

- El examen es sin copias ni apuntes.
- Está prohibido el préstamo de calculadoras, correctores, uso de celulares, consumo de bebidas, comidas y cigarrillos.
- Guarde sus entregables en D:\Arquitectura20132\<ap_paterno>-<ap_materno>

Nombre: Código:		Número de máquina:		Nota:	

Pregunta 1.-

a. (1 Punto, Word) Explique el patrón GRASP **Controlador** y cite un ejemplo

b. (2 Puntos, Hoja de examen) Una clase que implementa una Interface: (marque los verdaderos)

- ☐ No puede tener métodos propios
- ☐ Puede implementar también otras interfaces
- ☐ Tiene que implementar todos los métodos de la interface
- ☐ No tiene que implementar todos los métodos de la interface
- ☐ Puede tener sus propios métodos

c. (2 Puntos, Word) Del conjunto de requerimientos de un sistema, ¿Cuáles son los más relevantes para la Arquitectura? Cite un ejemplo.

Pregunta 2 (8 Puntos).- Patrón Strategy

El patrón Estrategia (Strategy) es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina como se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.

Motivación

Suponiendo un editor de textos con diferentes algoritmos para particionar un texto en líneas (justificado, alineado a la izquierda, etc.), se desea separar las clases clientes de los diferentes algoritmos de partición, por diversos motivos:

- Incluir el código de los algoritmos en los clientes hace que éstos sean demasiado grandes y complicados de mantener y/o extender.
- El cliente no va a necesitar todos los algoritmos en todos los casos, de modo que no queremos que dicho cliente los almacene si no los va a usar.
- Si existiesen clientes distintos que usasen los mismos algoritmos, habría que duplicar el código, por tanto, esta situación no favorece la reutilización.
- La solución que el patrón estrategia supone para este escenario pasa por encapsular los distintos algoritmos en una jerarquía y que el cliente trabaje contra un objeto intermediario contexto. El cliente puede elegir el algoritmo que prefiera de entre los disponibles, o el mismo contexto puede ser el que elija el más apropiado para cada situación.

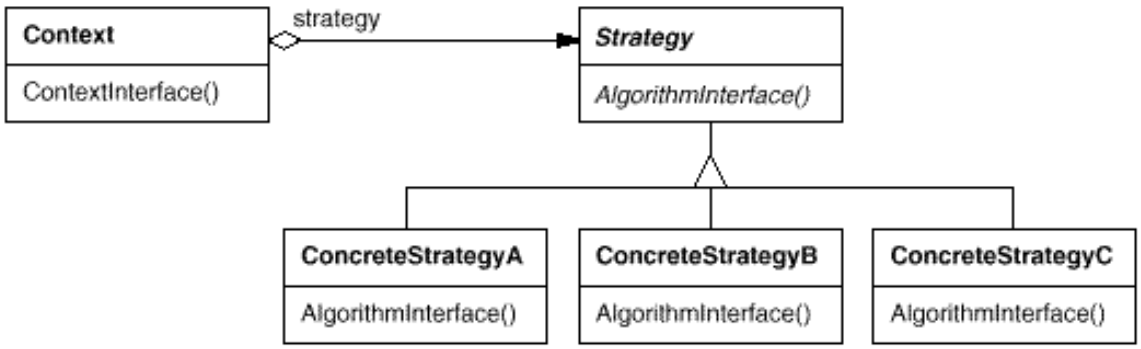
Aplicabilidad

Cualquier programa que ofrezca un servicio o función determinada, que pueda ser realizada de varias maneras, es candidato a utilizar el patrón estrategia. Puede haber cualquier número de estrategias y cualquiera de ellas podrá ser intercambiada por otra en cualquier momento, incluso en tiempo de ejecución. Si muchas clases relacionadas se diferencian únicamente por su comportamiento, se crea una superclase que almacene el comportamiento común y que hará de interfaz hacia las clases concretas.

Si un algoritmo utiliza información que no deberían conocer los clientes, la utilización del patrón estrategia evita la exposición de dichas estructuras. Aplicando el patrón a una clase que defina múltiples comportamientos mediante instrucciones condicionales, se evita emplear estas instrucciones, moviendo el código a clases independientes donde se almacenará cada estrategia.

Efectivamente, como se comenta anteriormente, este patrón de diseño nos sirve para intercambiar un sin número de estrategias posibles.

Estructura



Participantes

Cliente (Context): Es el elemento que usa los algoritmos, por tanto, delega en la jerarquía de estrategias. Configura una estrategia concreta mediante una referencia a la estrategia necesaria. Puede definir una interfaz que permita a la estrategia el acceso a sus datos en caso de que fuese necesario el intercambio de información entre el contexto y la estrategia. En caso de no definir dicha interfaz, el contexto podría pasarse a sí mismo a la estrategia como parámetro.

Estrategia (Strategy): Declara una interfaz común para todos los algoritmos soportados. Esta interfaz será usada por el contexto para invocar a la estrategia concreta.

EstrategiaConcreta (ConcreteStrategy): Implementa el algoritmo utilizando la interfaz definida por la estrategia.

Colaboraciones

Es necesario el intercambio de información entre estrategia y contexto. Este intercambio puede realizarse de dos maneras:

- Mediante parámetros en los métodos de la estrategia.
- Mandándose, el contexto, a sí mismo a la estrategia.
- Los clientes del contexto lo configuran con una estrategia concreta. A partir de ahí, solo se interactúa con el contexto.

Consecuencias

La herencia puede ayudar a factorizar las partes comunes de las familias de algoritmos (sustituyendo el uso de bloques de instrucciones condicionales). En este contexto es común la aparición conjunta de otros patrones como el patrón plantilla.

El uso del patrón proporciona una alternativa a la extensión de contextos, ya que puede realizarse un cambio dinámico de estrategia.

Los clientes deben conocer las diferentes estrategias y debe comprender las posibilidades que ofrecen.

Como contrapartida, aumenta el número de objetos creados, por lo que se produce una penalización en la comunicación entre estrategia y contexto (hay una indirección adicional).

Implementación

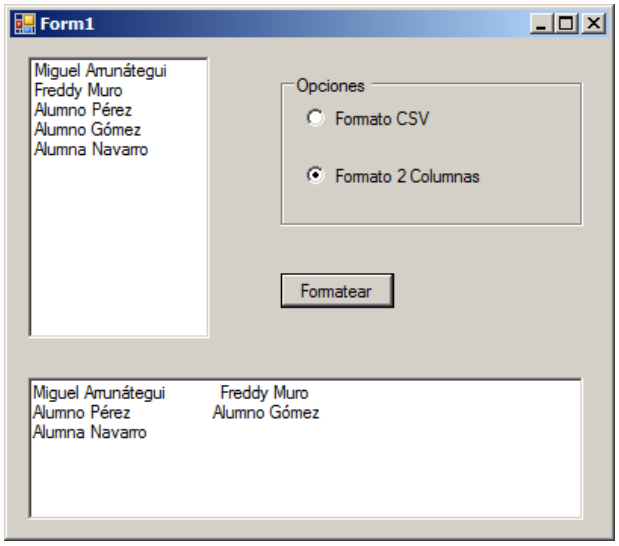
Entre las posibilidades disponibles a la hora de definir la interfaz entre estrategia y contexto, están:

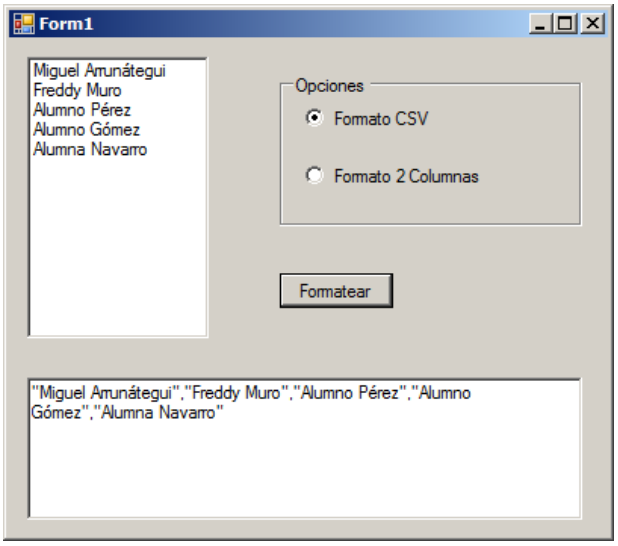
- Pasar como parámetro la información necesaria para la estrategia implica un bajo acoplamiento y la posibilidad de envío de datos innecesarios.
- Pasar como parámetro el contexto y dejar que la estrategia solicite la información que necesita supone un alto acoplamiento entre ellos.
- Mantener en la estrategia una referencia al contexto (similar al anterior).
- También puede ocurrir que se creen y se utilicen los objetos estrategia en el contexto sólo si es necesario, en tal caso las estrategias serán opcionales.

Problema a resolver: Implementar el patrón para un formateador de texto que toma como input una lista de strings, provenientes de un ListBox y los puede formatear de 2 maneras posibles:

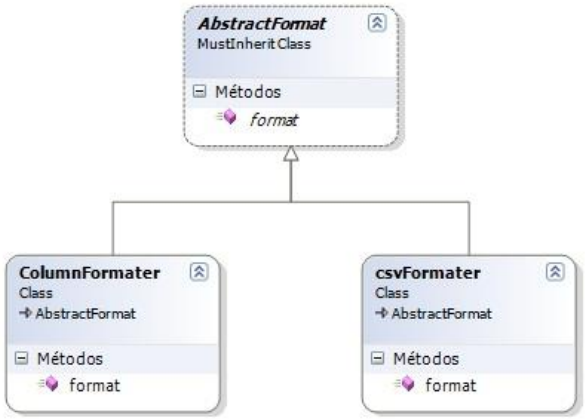
A) Como un string CSV (comma separated values) de valores entre comillas y separados por comas
B) Como 2 columnas

El resultado debe ser un String con el formato elegido, que se mostrará en una caja de texto. Ejemplos:





Diseño del patrón:



Pregunta 3.- (Solamente para los del Grupo 1 del laboratorio)

a. **(3 Puntos, Word)** Elabore una Regular Expression que detecte la línea del número de infracción de tránsito (de 6 dígitos) en el texto de un e-mail en donde se pide, entre otras cosas, especificar lo siguiente:

Número de infracción: 123456

Detectar las posibles formas en que el usuario pudiera haber escrito la línea, por ejemplo:
numero de infraccion : 123456 (sin acentos, mayúsculas y con varios espacios)

b. **(2 Puntos, Word)** Hacer una RE que haga match con el contenido de una línea numerada, separando como Grupo 1 el texto de la línea excluyendo la numeración. (Los grupos se arman poniendo la parte de la RE entre paréntesis)

Ejemplo:
1).- esta es la primera línea
2).- esta es la segunda
3).- esta es la tercera

123).- Esta es la 123

c. **(2 Puntos, Word)** Hacer una RE que haga match con el código de alumno de la URP. Por ejemplo: su propio código de alumno.

Pregunta 3.- (Solamente para los del Grupo 2 del laboratorio)

a. **(1 punto)** Diseñar la interface de despacho de productos de una expendedora automática. La dispensadora puede despachar 3 tipos de producto: Café, Infusión y paquetes (galletas, Halls, etc.). El método:

String **despachar**(int param1, int param2), despachará cualquiera de los 3 tipos de producto.

Los parámetros son como sigue:

Producto	**param1**	**param2**
Café	Tintura: 1=Normal, 2=Cargado	Azúcar: 1=normal, 2= extra
Infusión	Tipo de infusión: 1=te, 2= anís, 3=Hierba Luisa	Azúcar: 1=normal, 2= extra
Paquete	Fila	Columna

b. **(3 Puntos)** Implementar la interface vía 3 clases: Café, Infusion y Paquete. El método **despachar**, devuelve un String diciendo qué es lo que está despachando. Por ejemplo: "Despachando Infusión, anís con azúcar extra"

c. **(3 Puntos)** Implementar una GUI para demostrar el funcionamiento sacando provecho del uso de la interface