Exámenes teóricos y prácticos de patrones de diseño

7 de noviembre de 2019

Índice

1. Ejercicios teóricos **3**

2. Ejercicios prácticos **3**

2.1. Simulador de auto 3
2.1.1. Requerimientos 3
2.1.2. Módulos 2MIL 4
2.1.3. Documentación de patrones 5
2.2. Editor de Statecharts 7
2.2.1. Requerimientos 7
2.2.2. Módulos 2MIL 8
2.2.3. Documentación de patrones 8
2.3. Estación Climatológica 8
2.3.1. Requerimientos 8
2.3.2. Módulos 2MIL 9
2.3.3. Documentación de patrones 10
2.4. Plan de cuentas 11
2.4.1. Requerimientos 11
2.4.2. Módulos 2MIL 12
2.4.3. Documentación de patrones 12
2.5. Teléfonos celulares 12
2.5.1. Requerimientos 12
2.5.2. Módulos 2MIL 13
2.5.3. Documentación de patrones 13

2.6. Cuadros de diálogo 13
2.6.1. Requerimientos 13
2.6.2. Módulos 2MIL 13
2.6.3. Documentación de patrones 13
2.7. Modelos de automóviles 13
2.7.1. Requerimientos 13
2.7.2. Módulos 2MIL 14
2.7.3. Documentación de patrones 14
2.8. Problemas matemáticos 14
2.8.1. Requerimientos 14
2.8.2. Módulos 2MIL 15
2.8.3. Documentación de patrones 15
2.9. Protocolo de comunicación 15
2.9.1. Requerimientos 15
2.9.2. Módulos 2MIL 15
2.9.3. Documentación de patrones 15
2.10. Videojuego 15
2.10.1. Requerimientos 15
2.10.2. Módulos 2MIL 16
2.10.3. Documentación de patrones 16

1. Ejercicios teóricos

1. Explique en pocas líneas la razón por la cual el patrón de diseño Visitor utiliza el mecanismo de *doble despacho*.

2. Describa las características fundamentales del patrón de diseño Visitor.

3. Describa las características fundamentales del patrón de diseño Strategy.

2. Ejercicios prácticos

2.1. Simulador de auto

2.1.1. Requerimientos

Se requiere un Simulador de Auto que permita la simulación física de diferentes tipos de autos con diferentes niveles de dificultad. En los niveles más bajos de dificultad algunas cosas son automáticas (por ejemplo accionar la palanca de cambios), y en otras el manejo es totalmente manual. Además, se pueden agregar accesorios que afectan el andar del auto tales como nitro, spoilers, etc. Cuando el usuario agrega estos elementos, el simulador debe mostrar una nueva imagen del auto con lo que se haya agregado. Los accesorios que el usuario puede agregar dependen de la marca y modelo del auto. El simulador debe permitir que el usuario elimine el ultimo accesorio agregado. A su vez, en la pantalla de selección de automóvil, dado que hay muchos modelos, se desea contar con varias formas de recorrer la lista de acuerdo a diferentes parámetros o aplicando filtros (por ejemplo, marca, tamaño, etc).

Documente apropiadamente un diseño para el simulador descrito más arriba aplicando los siguientes patrones de diseño.

1. Bridge, para organizar correctamente los diferentes niveles de dificultad y autos físicos.

2. Strategy, para implementar los efectos que producen los accesorios sobre al andar del auto.

3. Wrapper (Decorator), para dibujar los autos con accesorios.

4. Iterator, para recorrer los modelos de auto.

2.1.2. Módulos 2MIL

Module	***Componente***
exportsproc	*Dibujar()*

Module	**Autos**
is	Agregado(Auto)

Module	**Filtro**
is	Iterador(Auto)

Module	**Orden**
is	Iterador(Auto)

Module	***Dificultad***
exportsproc	*PresionarAcelerador()*
	SoltarAcelerador()
	PresionarEmbrague()
	SoltarEmbrague()
	Izquierda()
	Derecha()
	SubirCambio()
	BajarCambio()

Generic Module	***Agregado(X)***
imports	*X*
exportsproc	*CrearIterador():Iterador(X)*
	Agregar(i X)
	Borrar(i X)

Module	**Spoiler**
inherits from	Accesorio
exportsproc	Agregar(Componente)

Module	**Auto**
inherits from	Componente
exportsproc	ObtenerDatos():Datos
	ListarAccesorios()

Module	**Nitro**
inherits from	Accesorio
exportsproc	Agregar(Componente)

Module	**Cabina**
imports	Dificultad
exportsproc	Acelerar()
	Frenar()
	Izquierda()
	Derecha()
	PonerCambio(i Int)
	ElegirDificultad(Dificultad)

Module	**Fácil**
inherits from	Dificultad

Module	***Accesorio***
inherits from	*Componente*

Module	**Difícil**
inherits from	Dificultad

Generic Module	***Iterador(X)***
imports	*X*
exportsproc	*Primero()*
	Siguiente()
	HaTerminado()
	ElementoActual():X

Module	**Simulador**
imports	Autos, Cabina, Componente
exportsproc	ElegirAuto()
	ElegirAccesorios()
	ElegirDificultad()
	Empezar()

2.1.3. Documentación de patrones

Pattern	**Modelos**		
based on	Iterator		
because	▪ Permite recorrer los autos sin saber como están almacenados. ▪ Se pueden recorrer los autos de varias formas diferentes.		
where	Cliente	**is**	Simulador
	Agregado	***is***	*Agregado*
	crearIterador()	***is***	*crearIterador()*
	AgregadoConcreto	**is**	Autos
	crearIterador()	**is**	crearIterador()
	Iterador	***is***	*Iterador*
	primero()	***is***	*Primero()*
	siguente()	***is***	*siguente()*
	haTerminado()	***is***	*haTerminado()*
	elementoActual()	***is***	*elementoActual()*
	IteradorConcreto	**is**	Filtro / Orden
	primero()	**is**	primero()
	siguente()	**is**	siguente()
	haTerminado()	**is**	haTerminado()
	elementoActual()	**is**	elementoActual()
comments	El iterador concreto filtro permite recorrer solo una parte de los modelos, mientras que el orden permite hacerlo en diferentes ordenes.		

Pattern	**Niveles**		
based on	Bridge		
because	▪ Facilita agregar nuevas dificultades. ▪ Permite cambiar la dificultad en tiempo de ejecución.		
where	Cliente	**is**	Simulador
	Abstraccion	**is**	Cabina
	Operacion1()	**is**	Acelerar()
	Operacion2()	**is**	Frenar()
	...	***is***	...
	Implementador	***is***	*Dificultad*
	OperacionImp1()	***is***	*PresionarAcelerador()*
	OperacionImp2()	***is***	*SoltarAcelerador()*
	...	***is***	...
	ImplementadorConcretoA	**is**	Fácil
	OperacionImp1()		PresionarAcelerador()
	...	**is**	...
	ImplementadorConcretoB	**is**	Difícil
	OperacionImp1()		PresionarAcelerador()
	...	**is**	...
comments	En el nivel de dificultad fácil la implementación de PonerCambio es nula, pero Acelerar y Frenar deben llamar a SubirCambio y BajarCambio.		

Pattern	**Dibujante**		
based on	Wrapper		
because	▪ Pueden agregarse y quitarse accesorios con facilidad.		
where	*Componente*	***is***	*Componente*
	Operacion()	***is***	*Dibujar()*
	ComponenteConcreto	**is**	Auto
	Operacion()	**is**	Dibujar()
	Decorador	***is***	*Accesorio*
	Operacion()	***is***	*Dibujar()*
	DecoradorConcretoA	**is**	Spoiler
	Operacion()	**is**	Dibujar()
	DecoradorConcretoB	**is**	Nitro
	Operacion()	**is**	Dibujar()
comments	El simulador obtendrá los posibles accesorios del auto y creara el objeto indicado por el usuario.		

2.2. Editor de Statecharts

2.2.1. Requerimientos

Aplique los patrones de diseño Composite, Strategy, Visitor, Iterator y Command para diseñar la aplicación que se describe a continuación. Concentre el esfuerzo de documentación en documentar el uso de los patrones de diseño.

La aplicación es un editor y simulador de Statecharts. Posee una interfaz gráfica donde el usuario dibuja el modelo de su sistema (recuerde que los Statecharts son máquinas jerárquicas). Ademas hay un motor de simulación que muestra gráficamente cómo la maquina descrita por el usuario reacciona ante eventos externos. Hay funciones para ocultar los componentes de un super-estado y otra para hacerlos visibles. Hay funciones para iniciar, detener y continuar una simulación. La simulación puede ser interactiva (el usuario da uno a uno los eventos), automática (el usuario da todos los eventos) o aleatoria (el motor genera secuencias de eventos).

Ayuda: Use Composite para la composición recursiva de las máquinas jerárquicas; use Strategy para las diferentes formas de simulación; use Iterator para recorrer el Statechart; use Visitor para ocultar o mostrar elementos de un super-estado; use Command para operar la simulación.

2.2.2. Módulos 2MIL

2.2.3. Documentación de patrones

2.3. Estación Climatológica

2.3.1. Requerimientos

Una estación climatologica portátil es un aparato que se ubica en el campo para medir varios parámetros relacionados con el clima tales como temperatura, presion, velocidad del viento, etc. El funcionamiento de la estación es controlado y coordinado por un software. El aparato cuenta con una pantalla LCD y un teclado reducido que el operador utiliza para configurar y ver algunos resultados. También tiene una unidad de almacenamiento secundario extraible donde se almacenan los datos que se fueron registrando. Los sensores deben ser activados o consultados cada un cierto intervalo de tiempo; para ello el hardware incluye un *timer*.

Documente apropiadamente un diseño para el software de control de la estación climatológica teniendo en cuenta los puntos que siguen:

1. Aplique el patro de diseño Bridge para que el software de la estación climatológica pueda funcionar sobre diferentes tipos de sensores. Por ejemplo puede darse que diferentes termómetros tengan interfaces de software diferentes pero el software de control siempre necesita el mismo dato.

2. Aplique el patrón de diseño Abstract Factory para que los mensajes, textos, magnitudes, etc. que son mostrados en la pantalla puedan aparecer en diferentes idiomas y sistemas de medición (por ejemplo grados centígrados o Farenheit).

3. Aplique el patrón de diseño Command para interactuar con el timer de manera que este avise cada *timeout*, como alternativa a un *callback*.

2.3.2. Módulos 2MIL

Module	**Sensor**
imports	InterfazSensor
exportsproc	Sensar()
	ObtenerValor():Int
	ElegirInterfaz(InterfazSensor)

Module	***InterfazSensor***
exportsproc	*ComenzarMedicion()*
	TerminarMedicion()
	SetearPuerto(i Int)
	ObtenerValor():Int

Module	**InterfazACME**
inherits from	InterfazSensor

Module	**InterfazEMCA**
inherits from	InterfazSensor

Module	***Region***
imports	*Unidades, Mensaje*
exportsproc	*CrearUnidades():Unidades*
	CrearMensaje(Codigo):Mensaje

Module	**Español**
inherits from	Region

Module	**Ingles**
inherits from	Region

Module	***Unidades***
exportsproc	*Longitud(i Int)*
	Temperatura(i Int)

Module	**Internacional**
inherits from	Unidades

Module	**Imperial**
inherits from	Unidades

Module	***Mensaje***
exportsproc	*ObtenerTexto():String*

Module	**MensajesEspañol**
inherits from	Mensaje

Module	**MensajesIngles**
inherits from	Mensaje

Module	**Timer**
imports	Orden
exportsproc	ElegirTiempo(i Int)
	ElegirOrden(i Orden)

Module	***Orden***
exportsproc	*Ejecutar()*

Module	**Medicion**
inherits from	Orden

2.3.3. Documentación de patrones

Pattern	**UI**		
based on	Abstract Factory		
because			
where	Cliente	**is**	
	FabricaAbstracta	***is***	*Region*
	CrearProductoA()	***is***	*CrearUnidade()*
	CrearProductoB()	***is***	*CrearMensaje()*
	FabricaConcreta1	**is**	Español
	CrearProductoA()	**is**	CrearUnidades()
	CrearProductoB()	**is**	CrearMensaje()
	FabricaConcreta2	**is**	Ingles
	CrearProductoA()	**is**	CrearUnidades()
	CrearProductoB()	**is**	CrearMensaje()
	ProductoAbstractoA	***is***	Unidades
	ProductoA1	**is**	Imperial
	ProductoA2	**is**	Internacional
	ProductoAbstractoB	***is***	Mensaje
	ProductoB1	**is**	MensajesIngles
	ProductoB2	**is**	MensajesEspañol
comments			

Pattern	**Abstraccion de sensores**		
based on	Bridge		
because			
where	Cliente	**is**	
	Abstraccion	**is**	Sensor
	Operacion()	**is**	Sensar()
	Implementador	***is***	InterfazSensor
	OperacionImp1()	***is***	*ComenzarMedicion()*
	OperacionImp2()	***is***	*TerminarMedicion()*
	OperacionImp3()	***is***	*SetearPuerto(i Int)*
	OperacionImp4()	***is***	*ObtenerValor():Int*
	ImplementadorConcretoA	**is**	InterfazEMCA
	OperacionImp()	**is**	...
	ImplementadorConcretoB	**is**	InterfazACME
	OperacionImp()	**is**	...
comments			

Pattern	**Ordenes**		
based on	Command		
because			
where	Invocador	**is**	Timer
	Orden	***is***	*Orden*
	Ejecutar()	***is***	*Ejecutar()*
	OrdenConcreta	**is**	Medicion
	Ejecutar()	**is**	Ejecutar()
	Receptor	**is**	Sensor
	Accion()	**is**	Sensar()
comments			

2.4. Plan de cuentas

2.4.1. Requerimientos

Un plan de cuentas de una contabilidad está formado por grupos de cuentas dentro de los cuales puede haber otros grupos de cuentas o cuenta reales. Las cuentas reales contienen asientos. Tanto a los grupos de cuentas como a las cuentas se los denomina genéricamente cuentas. Las cuentas y asientos tienen un saldo, pueden borrarse, tienen un nombre, descripción y fecha. Los

asientos tienen una cuenta de origen (que es de la cual forman parte) y una cuenta de destino, el saldo puede estar en el debe o en el haber. El saldo de una cuenta se calcula en base al saldo de cada uno de los asientos que la conforman; el saldo de un grupo de cuentas se calcula en base al saldo de cada una de las cuentas que lo forman. Borrar una cuenta significa borrar todo lo que contiene.

1. Aplique y documente el patrón Composite para estructurar el plan de cuentas.

2. Explique como re-implementaría las funciones principales de cada participante del punto anterior.

3. Aplique y documente el patrón Visitor para visualizar el plan de cuentas y guardarlo y recuperarlo de una base de datos. Los grupos de cuentas se visualizan en forma de árbol, en tanto que cuando el usuario hace doble click sobre una cuenta real se listan todos sus asientos en una ventana separada.

4. Explique como re-implementaría las funciones principales de cada participante del punto anterior.

5. La empresa que desarrolla el software de contabilidad planea venderlo en Argentina, Perú e Italia. En cada país los formularios como Factura, Cliente y Remito son ligeramente diferentes. Aplique y documente el patrón Abstract Factory para internacionalizar el sistema.

2.4.2. Módulos 2MIL

2.4.3. Documentación de patrones

2.5. Teléfonos celulares

2.5.1. Requerimientos

Una compañía desarrolla software para teléfonos celulares. Uno de los problemas que más costo de mantenimiento tiene es el de la interacción con los dispositivos de cada teléfono. Todos los teléfonos tienen más o menos los mismos dispositivos que hacen más o menos las mismas cosas pero cada fabricante (y posiblemente cada modelo) lo hace de forma ligeramente diferentes.

Utilice el patrón de diseño Bridge para resolver este problema en lo referente a la pantalla y la placa de sonido. Debe quedar muy claro en su diseño la diferencia entre interfaz e implementación; en particular debe aclararse las razones por las cuales cierta funcionalidad se ubica en la «interfaz» y otra en la «implementación».

2.5.2. Módulos 2MIL

2.5.3. Documentación de patrones

2.6. Cuadros de diálogo

2.6.1. Requerimientos

Un cuadro de diálogo toma dos datos que ingresa el usuario. Cuando el usuario pulsa el botón «Aceptar» se debe consultar un archivo y con los datos que allí se encuentren se debe abrir otro cuadro de dialogo que muestre algunos de esos datos.

Suponga que el código que muestra el primer cuadro de diálogo y lee la entrada del usuario está agrupado en el módulo CuadroD1; la función que consulta el archivo está en el módulo Consult; y el código del segundo cuadro de diálogo en el módulo CuadroD2.

Utilice el patrón de diseño Command para implementar la interacción entre CuadroD1 y CuadroD2 y Consult. Muestre el pseudo-código que implementaría su diseño. Muestre con detalle cómo sería el flujo de datos entre los tres módulos y los que usted defina.

2.6.2. Módulos 2MIL

2.6.3. Documentación de patrones

2.7. Modelos de automóviles

2.7.1. Requerimientos

El programa a diseñar debe armar diversos modelos de automoviles y mostrarlos en pantalla. Para armar los automoviles el programa cuenta con objetos gráficos que representan las diversas partes de gran cantidad de modelos de automóviles; se acompañan a los objetos con *metadatos* que describen distintas características técnicas de los objetos (tamaño, dimensiones, color,

potencia, etc; depende del objeto las características que aparecen). Tan importante como obtener un automóvil terminado es obtener los subsistemas más importantes tales como chasis, carrocería, motor, sistema eléctrico, etc. e incluso componentes de esos subsistemas tales como tablero, carburador, alternador, diferencial, etc.

Aplique el patrón de diseño Abstract Factory para crear instancias de los componentes de cada modelo de automóvil según la fábrica (marca de automóviles) que utilice el software.

2.7.2. Módulos 2MIL

2.7.3. Documentación de patrones

2.8. Problemas matemáticos

2.8.1. Requerimientos

Existen diversas bibliotecas de funciones que resuelven los siguientes problemas matemáticos:

- Multiplicación de matrices.
- Cálculo de determinantes.
- Factorial.
- x^y para x e y números reales.

Cada biblioteca fue implementada por una universidad o centro de investigación diferente por lo que no todas tienen la misma interfaz. Por otro lado, existen diversos métodos de resolución de ecuaciones diferenciales que usan como operaciones primitivas las operaciones provistas por las bibliotecas mencionadas más arriba.

Una empresa desea desarrollar un programa que provea los métodos de resolución de ecuaciones diferenciales utilizando las distintas bibliotecas según diferentes parámetros de configuración.

2.8.2. Módulos 2MIL

2.8.3. Documentación de patrones

2.9. Protocolo de comunicación

2.9.1. Requerimientos

La capa de transporte de un protocolo de comunicación debe interactuar con la placa de red de una computadora para poder proveer datos a las capas superiores del protocolo. Hay muchos fabricantes de placas de red que proveen interfaces diferentes para los programadores de protocolos. Por otro lado, se desea que las capas superiores del protocolo no estén al tanto de las diferencias entre los diversos modelos de placas de red.

A nivel de la placa de red existen secuencias de bytes con bytes de control. Precisamente las placas de red difieren en la interfaz que proveen para acceder a estas secuencias de bytes y en la forma en que codifican los bytes de control. Estas secuencias deben ser interpretadas por la capa de transporte de forma tal de proveer a las capas superiores tres tipos de paquetes de datos: inicio de conexión, paquete de datos (payload) y fin de conexión.

1. Aplique el patrón de diseño Bridge para diseñar la capa de transporte del protocolo mencionado.

2. Explique como deberían utilizarlo los programadores de las capas superiores.

2.9.2. Módulos 2MIL

2.9.3. Documentación de patrones

2.10. Videojuego

2.10.1. Requerimientos

Un juego de computadora medianamente complejo consiste en que un personaje tome ciertos objetos evitando ser capturado por diversos tipos de enemigos o caer en trampas. El escenario donde el personaje actúa consiste de habitaciones con puertas, escaleras, ascensores, tubos para descender, etc. En cada escenario hay un cierto número de objetos que el personaje debe tomar. Cuando el personaje los toma a todos, el juego pasa de nivel a otro

escenario semejante al anterior pero diferente (más complejo). Esto se repite al infinito.

El jugador puede mover el personaje en todas las direcciones, puede ordenarle que suba, baje, tome un ascensor, tome un objeto, etc. A medida que el jugador pasa de nivel, el personaje puede contar con ciertas armas para defenderse de los enemigos.

Los enemigos aparecen y desaparecen de forma dinámica dependiendo de lo que haga el personaje. Las trampas son fijas pero pueden activarse o desactivarse según lo que haga el personaje; el jugador tiene que ver en la pantalla si una trampa está activada o no.

1. Aplique el patrón de diseño Composite para representar la estructura física de los niveles y escenarios del videojuego.

2. Aplique el patrón de diseño Decorator para dotar de armas al personaje del videojuego.

2.10.2. Módulos 2MIL

2.10.3. Documentación de patrones