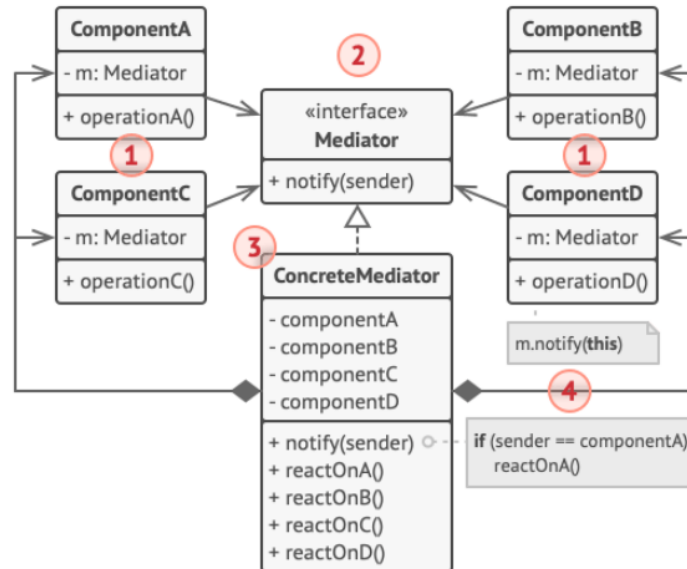
	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		FORMATO DE INFORME DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA ESTUDIANTES	
CARRERA: COMPUTACIÓN		ASIGNATURA: Programación Aplicada	
NRO. PRÁCTICA:		TÍTULO PRÁCTICA: Patrón de diseño Mediator	
OBJETIVO ALCANZADO: <ul style="list-style-type: none"> - Conocer sobre el patrón de diseño mediador - Como aplicarlo - Ventajas y desventajas 			
ACTIVIDADES DESARROLLADAS			
1. Revisar la teoría y conceptos de Patrones de Diseño de Java			
2. Diseñar e implementar cada estudiante un patrón de diseño y verificar su funcionamiento. A continuación se detalla el patrón a implementar.			
3. Probar y modificar el patrón de diseño a fin de generar cuales son las ventajas y desventajas.			
4. Realizar práctica codificando los códigos de los patrones y su estructura.			
RESULTADO(S) OBTENIDO(S): <p style="text-align: center;">Patrón de diseño Mediator</p> <p>Este patrón de diseño nos permite disminuir las dependencias incoherentes entre objetos, restringiendo la comunicación directa entre objetos, haciendo que estos trabajen usando un objeto mediador. La dependencia entre objetos tiende a ser caóticas según como crezca la aplicación.</p> <p>Para entender mejor este patrón plantearemos el problema inicial. Para eso tenemos un apartado de crear y editar un perfil de usuario, donde disponemos de controladores y los componentes gráficos como casilla de texto, botones, etc.</p> <p>Algunos elementos del formulario pueden interactuar con otros. Por ejemplo, podemos tener una casilla “tengo smartphone” y que cuando se marque aparezca un campo oculto para ingresar la marca. Otro ejemplo puede ser con el botón de registrar o enviar información, que tiene que validar toda la información antes de guardar.</p> <p>Al utilizar esta lógica, estamos haciendo que los elementos sean difíciles de reutilizar en otra parte del código, esto porque el formulario este acoplado a smartphone.</p> <p>Solución</p> <p>EL patrón Mediator no propone cada componente para ser independiente deben utilizar un objeto especial mediador, este se encarga de redireccionar las llamadas a los componentes que se desea utilizar. En otras palabras, el objeto mediador se encargará de recolectar la información y validarlo según el criterio de donde se use el formulario.</p> <p>Analogía al mundo real</p>			


Un ejemplo que nos proponen es una pista de aterrizaje, donde tenemos una torre central que todos los pilotos de los aviones se comunican y la torre se encarga de organizarlos y responde a cada uno. En caso de no existir una torreta los aviones deberían conectarse directamente y discutir la prioridad de aterrizaje entre ellos.

Estructura



1. Los Componentes son varias clases que contienen cierta lógica de negocio. Cada componente tiene una referencia a una interfaz mediadora, declarada con su tipo. El componente no conoce la clase de la interfaz mediadora, por lo que puedes reutilizarlo en otros programas vinculándolo a una mediadora diferente.
2. La interfaz Mediadora declara métodos de comunicación con los componentes, que normalmente incluyen un único método de notificación. Los componentes pueden pasar cualquier contexto como argumentos de este método, incluyendo sus propios objetos, pero sólo de tal forma que no haya acoplamiento entre un componente receptor y la clase del emisor.
3. Los Mediadores Concretos encapsulan las relaciones entre varios componentes. Los mediadores concretos a menudo mantienen referencias a todos los componentes que gestionan y en ocasiones gestionan incluso su ciclo de vida.
4. Los componentes no deben conocer otros componentes. Si le sucede algo importante a un componente, o dentro de él, sólo debe notificar a la interfaz mediadora. Cuando la mediadora recibe la notificación, puede identificar fácilmente al emisor, lo cual puede ser suficiente para decidir qué componente debe activarse en respuesta.

Desde la perspectiva de un componente, todo parece una caja negra. El emisor no sabe quién acabará gestionando su solicitud, y el receptor no sabe quién envió la solicitud.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Aplicabilidad

- Utiliza el patrón Mediator cuando resulte difícil cambiar algunas de las clases porque están fuertemente acopladas a un puñado de otras clases.
- Utiliza el patrón cuando no puedas reutilizar un componente en un programa diferente porque sea demasiado dependiente de otros componentes.
- Utiliza el patrón Mediator cuando te encuentres creando cientos de subclases de componente sólo para reutilizar un comportamiento básico en varios contextos.

Cómo implementarlo

1. Identifica un grupo de clases fuertemente acopladas que se beneficiarían de ser más independientes.
2. Declara la interfaz mediadora y describir el protocolo de comunicación deseado entre mediadores y componentes.
3. Esta interfaz es fundamental cuando quieras reutilizar las clases del componente en distintos contextos. Siempre y cuando el componente trabaje con su mediador a través de la interfaz genérica, se puede vincular el componente con una implementación diferente del mediador.
4. Implementa la clase concreta mediadora. Esta clase se beneficiará de almacenar referencias a todos los componentes que gestiona.
5. Los componentes deben almacenar una referencia al objeto mediador. La conexión se establece normalmente en el constructor del componente, donde un objeto mediador se pasa como argumento.
6. Cambia el código de los componentes de forma que invoquen el método de notificación del mediador en lugar de los métodos de otros componentes

Pros y contras

- Principio de responsabilidad única. Puedes extraer las comunicaciones entre varios componentes dentro de un único sitio, haciéndolo más fácil de comprender y mantener.
- Puedes introducir nuevos mediadores sin tener que cambiar los propios componentes.
- Puedes reducir el acoplamiento entre varios componentes de un programa.
- Puedes reutilizar componentes individuales con mayor facilidad.
- Con el tiempo, un mediador puede evolucionar a un objeto todopoderoso. x

CONCLUSIONES:

Conocer varios patrones de diseño nos puede ayudar a mejorar nuestra forma de programación, como este patrón de mediador que nos ayuda a reutilizar ciertos componentes u objetos. También nos da la posibilidad de administrar mejor los objetos y que no interactúen directamente.

Nombre de estudiante: Paul Sebastian Idrovo Berrezueta

Firma de estudiante:

