
	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:	3	TÍTULO PRÁCTICA: Patrones en Java	
OBJETIVO ALCANZADO: Identificar los cambios importantes de Java Diseñar e Implementar las nuevas tecnicas de programación Entender los patrones de Java			
ACTIVIDADES DESARROLLADAS			
Patrón de diseño Command			
1. ¿Para que sirve el patrón “Command”? <p>El patrón Command tiene como funcionalidad la separación entre quien hace la petición y quien lleva acabo esa petición. Es decir, un objeto lleva acabo unas acciones y otro objeto necesita que se lleven acabo esas acciones. Se debe tomar en cuenta que los objetos no van a estar conectados directamente.</p> <p>La idea principal del patrón “Command” es convertir una solicitud en un objeto independiente de la cual el objeto creado va a contener todos los datos de la solicitud. La solicitud al transformarse en un objeto se puede parametrizar los métodos con diferentes solicitudes.</p>			
2. ¿Cuándo se utiliza? <p>En una programación normal se crea un programa que pueda ejecutar varios subprocesos o subprogramas, pero cuando se presenta una situación en donde es mas conveniente desacoplar la invocación de determinados procesos del área en donde se encuentran, se utiliza el patrón de diseño “Command”.</p> <p>También se utilizan cuando:</p> <ul style="list-style-type: none"> • El programa utiliza colas, pilas u otras estructuras para gestionar invocaciones. • Se crea la probabilidad de cancelar operaciones. • Se necesite parametrizar de manera uniforme las invocaciones. • Las ordenes que debamos ejecutar son de alto nivel y por debajo son implementadas por ordenes simples (primitivas) 			
3. Consecuencias positivas y negativas del patrón Command Positivo <ul style="list-style-type: none"> • Desacoplamiento de la aplicación que invoca las ordenes y la implementación de estos. • Los comandos se convierten en objetos lo cual permite la aplicación de herencias a las mismas o realizar composición de ordenes • El conjunto de ordenes es escalable. • Permite la modificación de ordenes a ejecutar en el tiempo de ejecución. 			

- Permite la implementación de métodos tales como undo/redo

Negativo

- Aumento de volumen de código.

4. Estructura

Cliente: Establece la configuración del Command y del Reciever.

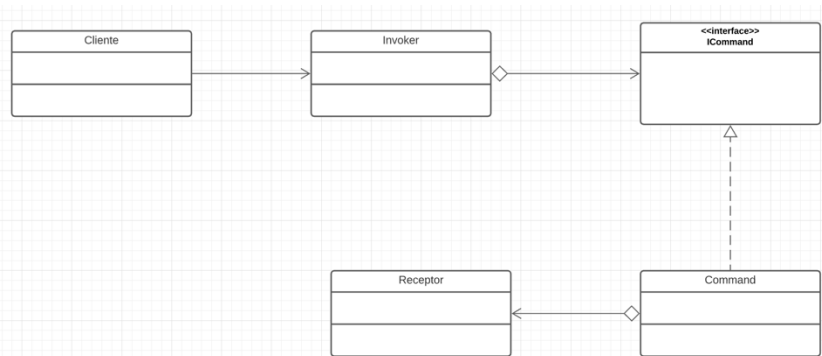
Receiver: este objeto implementa la funcionalidad que deseamos implementar en el ICommand.

Uno o varios de sus métodos serán llamados a la hora de invocar a la función `execute` del `ICommand`.

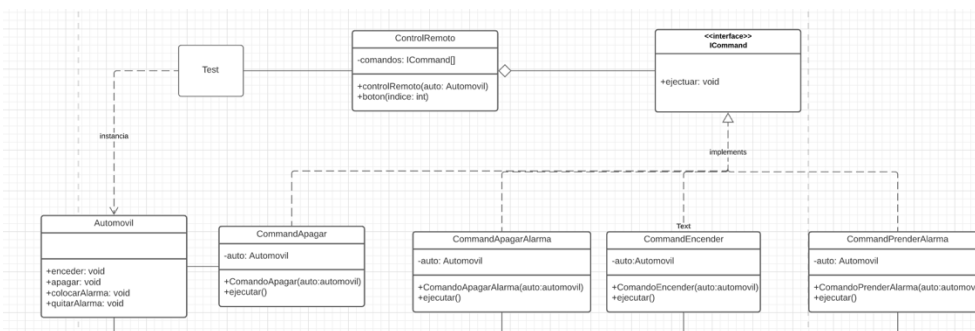
Command: implementa la interfaz "ICommand" y contiene una referencia al objeto

ICommand: interfaz que representa el orden que se pretende ejecutar de manera desacoplada, permitiendo su ejecución mediante su método.

Invoker: clase encargada de invocar el método `ICommand.execute()`. En otras palabras, le pide a las clases `Command` que lleven acabo las acciones.



5. A continuación, se ha creado un ejemplo de este patrón para poner en practica su funcionalidad.




6. Primero se crearon los siguientes paquetes de los cuales cada uno va a representar los componentes del patrón de diseño Command.

- Ec.edu.ups.ICommand;
- Ec.edu.ups.command;
- Ec.edu.ups.cliente;
- Ec.edu.ups.invoker;
- Ec.edu.ups.reciever;

7. Ec.edu.ups.reciever

Este paquete va a comportarse como el receptor del patrón Command;

Tiene solamente una clase creada

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

- Automóvil

8. Automóvil

Esta clase va a tener todos los métodos ya realizados.

- Public void encender();
- Public void apagar();
- Public void colocarAlarma();
- Public void quitarAlarma();

9. Ec.edu.ups.reciever

Este paquete se va a comportar como el Command del patrón Command. Este paquete contiene todas las clases que implementa la interfaz ICommand y operan todos los métodos del objeto Automóvil.

- CommandApagar
- CommandApagarAlarma
- CommandEncender
- CommandEncenderAlarma

10. CommandApagar

Esta clase implemente la interfaz ICommand. Como la interfaz ICommand solamente tiene un método solo es método se ejecutará en esta clase con la única excepción de que este método se sobre escribe para que se ejecute la acción de apagar el automóvil.

Atributos:

- private Automovil auto;

El constructor recibe un objeto de tipo Automóvil en sus parámetros, después inicializa el objeto auto.

11. CommandApagarAlarma

Esta clase implemente la interfaz ICommand. Como la interfaz ICommand solamente tiene un método solo es método se ejecutará en esta clase con la única excepción de que este método se sobre escribe para que se ejecute la acción de apagar la alarma del automóvil.

Atributos:

- private Automovil auto;

El constructor recibe un objeto de tipo Automóvil en sus parámetros, después inicializa el objeto auto.


12. CommandEncender

Esta clase implemente la interfaz ICommand. Como la interfaz ICommand solamente tiene un método solo es método se ejecutará en esta clase con la única excepción de que este método se sobre escribe para que se ejecute la acción de encender el automóvil.

Atributos:

- private Automovil auto;

El constructor recibe un objeto de tipo Automóvil en sus parámetros, después inicializa el objeto auto.

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

13. CommandEncenderAlarma

Esta clase implemente la interfaz ICommand. Como la interfaz ICommand solamente tiene un método solo es método se ejecutará en esta clase con la única excepción de que este método se sobre escribe para que se ejecute la acción de encender la alarma del automóvil.

Atributos:

- private Automovil auto;

El constructor recibe un objeto de tipo Automóvil en sus parámetros, después inicializa el objeto auto.

14. Ec.edu.ups.Icommand

Este paquete se comportará como la interfaz del patrón Command

- ICommando

15. ICommand

Esta interfaz solamente tiene un método

- ejecutar

16. ec.edu.ups.invoker

este paquete se comportará como el invoker del patrón Command. Se creo una clase ControlRemoto

17. ControlRemoto

Paquetes importados:

- import ec.edu.ups.Icommand.ICommando;
- import ec.edu.ups.command.*;
- import ec.edu.ups.receiver.Automovil;

Atributos:

- private ICommando[] comandos

El constructor de esta clase recibe un objeto de tipo Automovil, también se inicializan todos los comandos existentes que implementan la interfaz. A la misma vez se les pasa el objeto de tipo Automovil a cada comando.


Métodos:

- botón: este método recibe un objeto de tipo entero en su parámetro. Este método ejecuta el comando que el usuario esta queriendo que se realice.

18. Ec.ups.edu.cliente

Esta clase se comportará como las solicitudes que desea que se operen del patrón Command.

Este paquete solamente tiene una clase en la cual el usuario ejecutara todas las acciones que desea que se realicen.

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

RESULTADO(S) OBTENIDO(S):

QUE DESEA HACER?

1. ENCENDER AUTO
2. APAGAR AUTO
3. ENCENDER ALARMA
4. APAGAR ALARMA
5. SALIR

1
EL AUTO SE HA ENCENDIDO

QUE DESEA HACER?

1. ENCENDER AUTO
2. APAGAR AUTO
3. ENCENDER ALARMA
4. APAGAR ALARMA
5. SALIR

4
ALARMA APAGADA

QUE DESEA HACER?

1. ENCENDER AUTO
2. APAGAR AUTO
3. ENCENDER ALARMA
4. APAGAR ALARMA
5. SALIR

2
EL AUTO SE HA APAGADO

QUE DESEA HACER?

1. ENCENDER AUTO
2. APAGAR AUTO
3. ENCENDER ALARMA
4. APAGAR ALARMA
5. SALIR

3
ALARMA ENCENDIDA

CONCLUSIONES: En conclusión, el patrón Command es muy útil al momento que se desea parametrizar objetos con operaciones, también es de mucha utilidad cuando se desea poner operaciones en colas o cuando se desea implementar operaciones reversibles. Este patrón presenta una forma sencilla y versátil de implementar un sistema basado en comandos, facilitando su uso y ampliación.

RECOMENDACIONES:

- REFACTORING GURU. (2020, noviembre 17). <https://refactoring.guru/es/design-patterns/command>.
- Patrones de Diseño (XV): Patrones de Comportamiento – Command. (2020, noviembre 17). https://programacion.net/articulo/patrones_de_diseno_xv_patrones_de_comportamiento_command_1018.
- Command Design Pattern. (2020, noviembre 17). https://sourcemaking.com/design_patterns/command.
- MitoCode. (2018, Agosto 06). Curso de Patrones de diseño - 9 Command [Video]. Youtube. <https://www.youtube.com/watch?v=hDBOfyzFKEU>.
- Gernot.Klinger. (2014, Febrero 10). Implementing undo/redo with Command Pattern. Gernot.Klinger|blog. <https://gernotklinger.com/blog/implementing-undoredo-with-the-command-pattern/>

Nombre de estudiante: _____ Denys Dutan _____

Firma de estudiante: _____