Patron de diseño ***CallBack.***

<https://java-design-patterns.com/patterns/callback/>

patron:

public interface ***Retrollamada*** {

void llamar();

}

El patrón de Retrollamada (Callback) es muy útil en situaciones donde necesitamos realizar alguna operación que puede tomar tiempo y no queremos bloquear el hilo principal de ejecución.

Aquí tienes un ejemplo de cómo se podría usar este patrón en Java:

public interface Retrollamada {

void llamar();

}

public class MiClase {

private Retrollamada retrollamada;

public MiClase(Retrollamada retrollamada) {

this.retrollamada = retrollamada;

}

public void ejecutar() {

// Realiza alguna operación que puede tomar tiempo...

// Una vez que la operación ha terminado, llama al método de retrollamada

retrollamada.llamar();

}

}

public class Principal {

public static void main(String[] args) {

MiClase miClase = new MiClase(new Retrollamada() {

@Override

public void llamar() {

System.out.println("¡La operación ha terminado!");

}

});

miClase.ejecutar();

}

}

En este ejemplo, MiClase realiza una operación que puede tomar tiempo.

En lugar de bloquear el hilo principal de ejecución hasta que la operación termine, MiClase llama al método llamar() de la interfaz Retrollamada una vez que la operación ha terminado. Esto permite que el hilo principal de ejecución continúe con otras tareas mientras la operación se está realizando.

Este patrón es especialmente útil en programación de interfaces de usuario, donde no queremos bloquear la interfaz de usuario mientras se realizan operaciones que pueden tomar tiempo, como cargar datos de la red o realizar cálculos intensivos. También es muy común en programación de red y en sistemas basados en eventos.

Ejemplo 2.

Es una "alternativa" al return. En lugar de que un método retorne algo, lo que hará ese método es llamar a un método que le habremos pasado, poniendo como parámetros lo que haya que retornar.

En el siguiente ejemplo, observa que al llamar al método **hacerAlgo** le hemos pasado un objeto de clase

Algo.Callback

. El método

hacerAlgo

, en lugar de retornar el texto

"algo a retornar"

, lo pone como parámetro en la llamada de vuelta al método

call

del objeto que le habíamos pasado.

class Algo {

interface Callback {

void call(String valorDeRetorno);

}

void hacerAlgo(Callback callback){

callback.call("algo a retornar");

}

}

public class Main {

public static void main(String[] args) {

Algo algo = new Algo();

algo.hacerAlgo(new Algo.Callback() {

@Override

public void call(String valorDeRetorno) {

System.out.println(valorDeRetorno);

}

});

}

}

Cuando un interficiece solo tiene **un único** método, podem usar una expresión lambda cuando haya que pasar como parámetro una instancia de dicho interface.

En el ejemplo anterior, el

interface Callback

solo tiene el método

call

. Podemos pasar la instancia de

Algo.Callback

con una expresión lambda:

algo.hacerAlgo(valorDeRetorno -> System.out.println(valorDeRetorno));

Observa que una expresión lambda no es más que una simplificación de la instanciación del objeto; se indica solamente el nombre del parámetro y el cuerpo del método:

algo.hacerAlgo(new Algo.Callback() {

@Override

public void call(String valorDeRetorno) {

System.out.println(valorDeRetorno);

}

});

algo.hacerAlgo(valorDeRetorno -> System.out.println(valorDeRetorno));

Por otra parte, un callback puede tener varios métodos, y el que lo recibe puede llamar a uno u otro según sea conveniente. De esta forma, un método puede "retornar" distintos tipos de datos, o quizá "retornar" errores,...

class Algo {

interface Callback {

void retornoUnString(String valorDeRetorno);

void retornoUnInt(int valorDeRetorno);

void retornoUnBoolean(boolean valorDeRetorno);

void retornoUnError(String valorDeRetorno);

}

void hacerAlgo(Callback callback){

if (...) {

callback.retornoUnString("algo a retornar");

} else if (...) {

callback.retornoUnInt(324342);

} else if (...) {

callback.retornoUnBoolean(true);

} else if (...) {

callback.retornoUnError("Error xyz");

}

}

}

public class Main {

public static void main(String[] args) {

Algo algo = new Algo();

algo.hacerAlgo(new Algo.Callback() {

@Override

public void retornoUnString(String valorDeRetorno) {

System.out.println("El resultado es: " + valorDeRetorno);

}

@Override

public void retornoUnInt(int valorDeRetorno) {

System.out.println("El resultado es: " + valorDeRetorno);

}

@Override

public void retornoUnBoolean(boolean valorDeRetorno) {

System.out.println("El resultado es: " + valorDeRetorno);

}

@Override

public void retornoUnError(String valorDeRetorno) {

System.out.println("Ha habido un error: " + valorDeRetorno);

}

});

}

}

En este caso, como el

interface Algo.Callback

tiene más de un método, no se puede usar una expresión lambda.

En la vida real:

- La librería [retrofit](https://gerardfp.github.io/dam/entorns/patterns/) sirve para hacer llamadas HTTP. La respuesta se retorna mediante un callback:

[Callback](https://github.com/square/retrofit/blob/master/retrofit/src/main/java/retrofit2/Callback.java)

api.obtenerNombre().enqueue(new Callback<String>() {

@Override

public void onResponse(Call<String> call, Response<String> response) {

System.out.println("Nombre: " + response.body());

}

@Override

public void onFailure(Call<String> call, Throwable t) {

System.out.println("Fallo al obtener el nombre");

}

});

- La implementación [tyrus](https://eclipse-ee4j.github.io/tyrus-project.github.io/documentation/latest/index/getting-started.html#d0e79) del protocolo websockets utiliza callbacks para gestionar los eventos de conexión a un servidor:

[Endpoint](https://github.com/jakartaee/websocket/blob/master/api/client/src/main/java/jakarta/websocket/Endpoint.java)

ClientManager.createClient().connectToServer(

new Endpoint() {

@Override

public void onOpen(Session session, EndpointConfig config) {

System.out.println("Conectado al servidor");

}

@Override

public void onClose(Session session, CloseReason closeReason) {

System.out.println("Desconectado del servidor");

}

@Override

public void onError(Session session, Throwable thr) {

System.out.println("Error conectando al servidor");

}

},

new URI("ws://websocket.server:1234"));