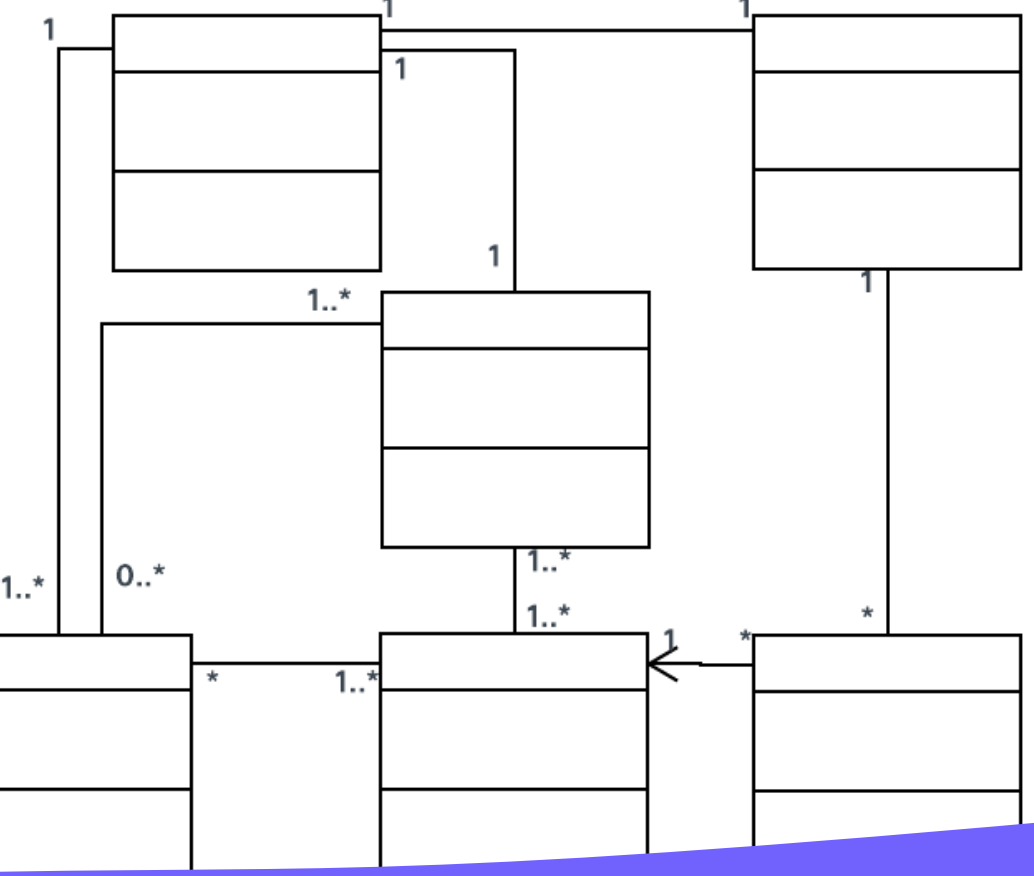
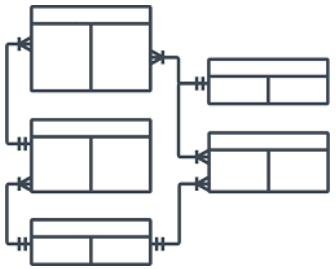




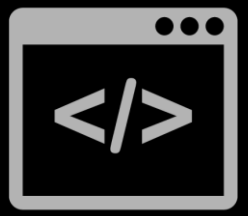
Diseño de Sistemas

UTN.BA
DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CÁTEDRA DISEÑO DE SISTEMAS

Agenda

- Testing & Tests Unitarios
- Componentes Stateless y Stateful
- Patrón Strategy
- Patrón Adapter

Testing & Test Unitarios



¿Qué es Testing?

Testing es la verificación dinámica de la adecuación del Sistema a los requerimientos/requisitos.

Según la IEEE, el testing es una actividad en la cual un Sistema o componente de software es ejecutado, bajo condiciones específicas, para que los resultados de dicha ejecución sean observados y/o registrados; y a partir de los mismos realizar una evaluación de algún aspecto del Sistema o componente.

¿Qué es Testing?

- El testing busca encontrar fallas en el producto.
- Busca hacerlo lo más rápido y barato posible (busca la eficiencia).
- Busca encontrar la mayor cantidad de fallas
- Intenta no detector fallas que en realidad no lo son.
- Pretende encontrar las fallas más importantes.

¿Qué es Testing?

Una prueba es exitosa si encuentra fallas.

Testing – Conceptos generales

- ***Equivocación***: es una acción humana que produce un resultado incorrecto.
- ***Defecto***: paso, proceso o definición de dato incorrecto. También puede considerarse como la ausencia de cierta característica.
- ***Falla***: resultado de ejecución incorrecto (o distinto al valor esperado).

Testing – Conceptos generales

- Una *equivocación* lleva a uno o más *defectos* que están presentes en el Código.
- Un *defecto* lleva a cero, una o más *fallas*.
- La *falla* es la manifestación del *defecto*.
- Una *falla* tiene que ver con uno o más *defectos*.

Testing – Conceptos generales

- ***Condiciones de prueba***: son descripciones de situaciones que quieren probarse ante las cuales el Sistema debe responder.

- ***Casos de prueba***: son lotes de datos necesarios para que se dé una determinada condición de prueba.

Testing – Tipos de Tests

- *Pruebas Unitarias (de caja negra y de caja blanca)*
- *Pruebas de Integración*
- *Pruebas de Aceptación de Usuario (UAT)*
- *Pruebas de Stress*
- *Pruebas de Volumen y Performance*
- *Pruebas de Regresión*
- *Pruebas Alfa y Beta*

Tests Unitarios

Los tests unitarios se realizan sobre unidades de código claramente definidas.

Con la definición anterior, los tests unitarios podrían validar el correcto funcionamiento de una función en particular, o de un método de un objeto.

Tests Unitarios

Se suele realizar un test por cada método o por cada función para probar que cada parte aislada funciona de forma correcta.

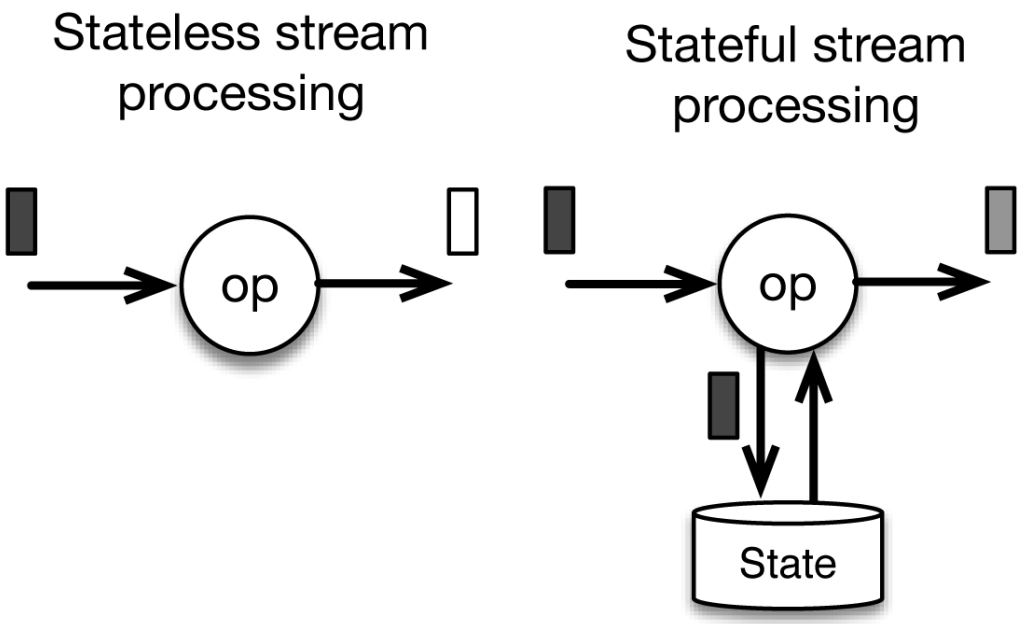
Tests Unitarios

- Generalmente las realizan las mismas personas que construyeron el módulo que se quiere probar
- Los módulos altamente cohesivos son los más sencillos de probar

Tests Unitarios

- Particularmente, en Java solemos utilizar JUnit como framework de testeo unitario.
- Además, para mockear objetos se suele utilizar mockito o realizar una implementación propia.

Recomendamos la lectura del siguiente artículo para comprender la utilización de los mockeos.



Componentes Stateless y Stateful

Componentes Stateless y Stateful

"El estado de una aplicación (o de cualquier otra cosa) hace referencia a su condición o cualidad de existir en un momento determinado. Es su capacidad de estar. Que un sistema tenga estado depende del tiempo durante el cual se registra interacción con él y de la forma en que se debe almacenar esa información. "

Red Hat – Sistemas con estado y sin estado

Componentes Stateless

- Un proceso, una aplicación o, genéricamente, un componente sin estado se refiere a los casos en que éstos están aislados.
- No se almacena información sobre las operaciones anteriores ni se hace referencia a ellas.
- Cada operación se lleva a cabo desde cero, como si fuera la primera vez.

Componentes Stateless – Objetos

- Los objetos Stateless no deberían tener estado interno; o si lo tienen, éste no debería condicionar el funcionamiento frente a las distintas peticiones que podría recibir el objeto en cuestión.

- Si pensamos a nuestros objetos Stateless, podríamos llegar a la conclusión de que ellos son reutilizables; es decir, que no dependen de otros objetos para sobrevivir, sino que tienen sentido de existencia propio.

Componentes Stateful

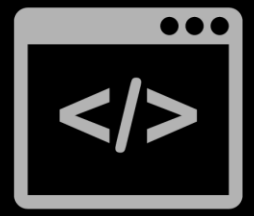
- Las aplicaciones, los procesos o, genéricamente, los componentes con estado son aquellos a los que se puede volver una y otra vez y éstos recuerdan "*quiénes somos*" y "*qué hicimos anteriormente*".

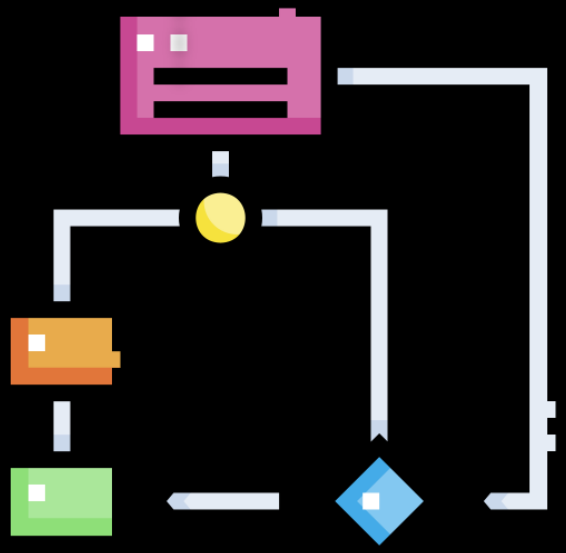
- Se realizan con el contexto de las operaciones anteriores, y la operación actual puede verse afectada por lo que ocurrió previamente.

Componentes Stateful – Objetos

- Si pensamos a nuestros objetos Stateful, podríamos llegar a la conclusión de que *generalmente* ellos no serían reutilizables; es decir, que por estar – al menos - levemente acoplados a algún otro objeto, no podrían servir para que otro objeto realice operaciones con ellos.

Patrón Strategy





Patrón Strategy

Es un patrón de Comportamiento

¿Qué hace?

- Encapsula distintas formas (o algoritmos) de resolver el mismo problema en diferentes clases.
- Permite intercambiar en momento de ejecución la forma en que un tercero resuelve un problema.

Patrón Strategy

Se sugiere su utilización cuando:

- Se requiere que un objeto realice una misma acción pero con un algoritmo distinto o de una forma distinta.
- Existen muchas formas de realizar la misma acción (pero con distintos pasos) en el mismo objeto.
- Se requiere permitir configurar en momento de ejecución la forma en que un objeto realizará una acción.

Patrón Strategy

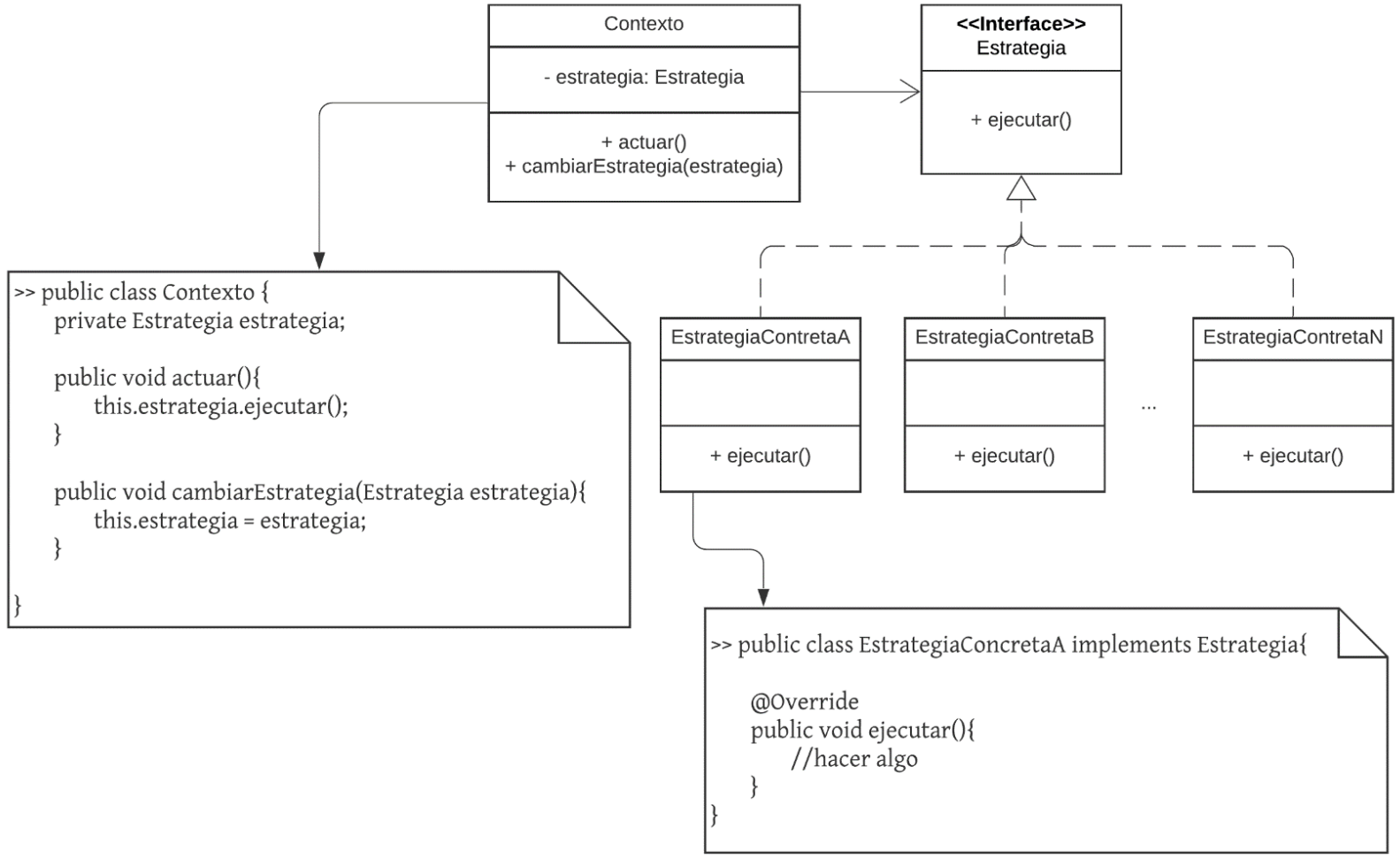
Componentes:

- ***Interface (o clase abstracta) Strategy***: Define la firma del método que se utilizará como estrategia de resolución de algún problema.

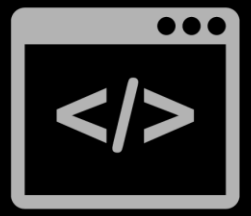
- ***Clases de estrategias concretas***: Clases que implementan la interface Strategy (o que heredan de ella, si ésta fuera clase abstracta), es decir, que tienen la implementación real del algoritmo.

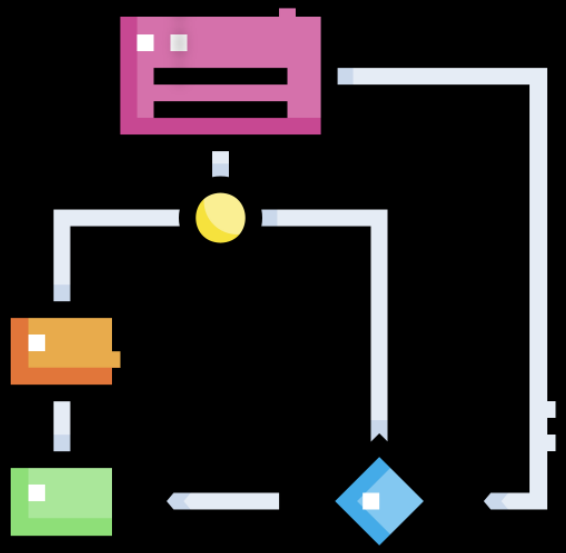
- ***Contexto***: Clase que tiene referencia a la interface/clase abstracta Strategy, cuyos objetos van a delegar la responsabilidad de resolución de algún problema en la estrategia. Estos objetos utilizarán de forma polimórfica a las estrategias concretas.

Patrón Strategy



Patrón Adapter





Patrón Adapter

Es un patrón Estructural

¿Qué hace?

- Encapsula el uso (llamadas/envío de mensajes) de la clase que se quiere adaptar en otra clase que concuerda con la interfaz requerida.

Patrón Adapter

Se sugiere su utilización cuando:

- Se requiere seguir adelante con el diseño y/o implementación (código) sin conocer exactamente cómo, quién y cuándo resolverá una parte necesaria; y solamente se conoce qué responsabilidad tendrá dicha parte.

- Se requiere usar una clase que ya existe, pero su interfaz no concuerda con la que se necesita.

Patrón Adapter

Componentes:

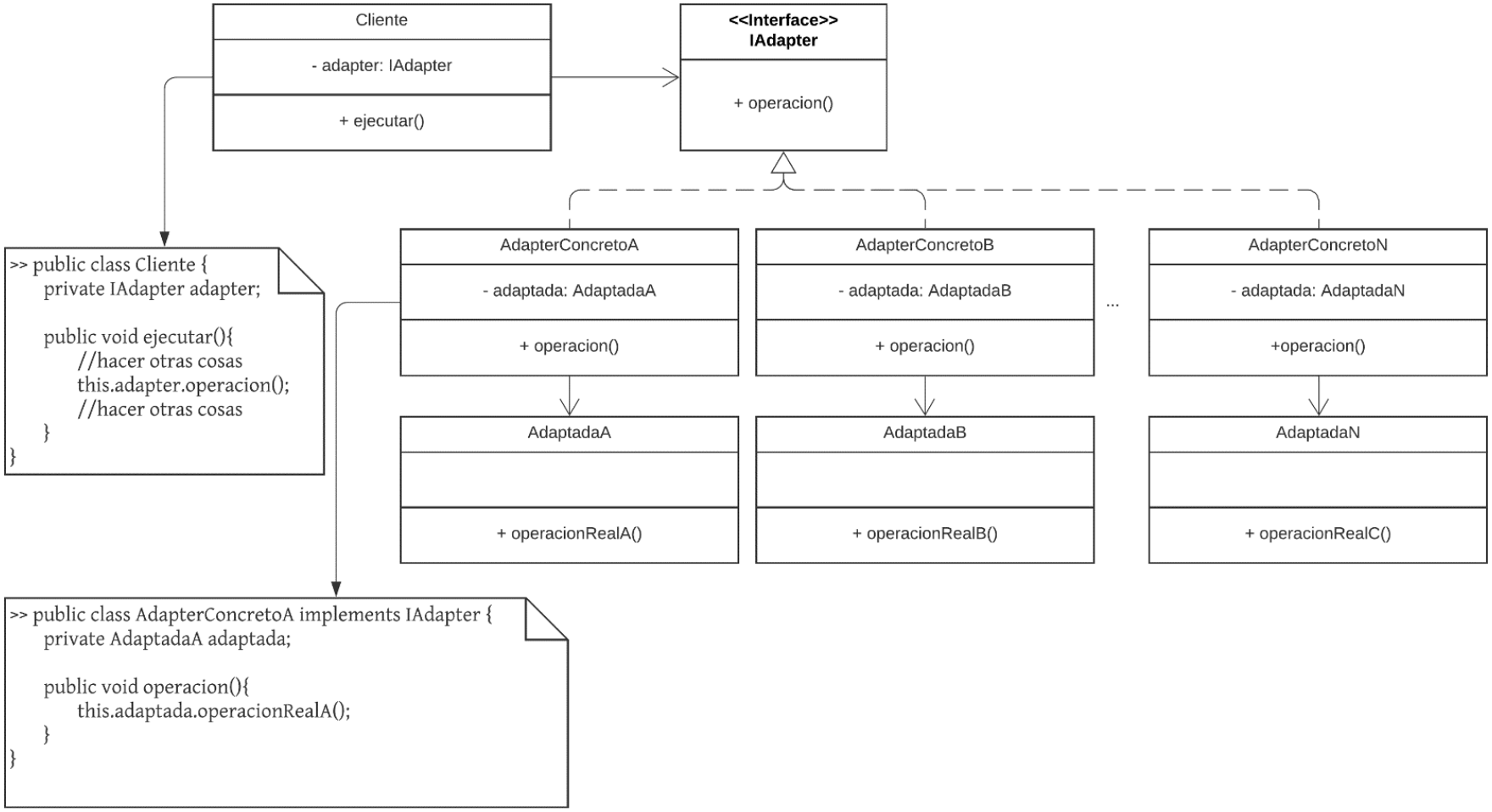
- ***Interface Adapter***: Define la firma del método que se utilizará como "puente" de acoplamiento a nuestro dominio.

- ***Clases de Adapters concretos***: Clases que implementan la interface Adapter, que se encargan de acoplar los componentes externos al dominio. Guardan referencia o hacen uso de las clases Adaptadas. Llaman a los métodos "reales" que resuelven el problema.

- ***Clases Adaptadas***: Clases externas al dominio, o no, que posee firmas incompatibles, o que todavía no conocemos (puede que todavía no estén creadas). Estas clases no se pueden/deben tocar.

- ***Cliente***: Clase que tiene referencia a la interface Adapter, cuyos objetos van a hacer uso de la funcionalidad que ésta les brinda.

Patrón Adapter



Gracias