



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 13

Patrones de diseño

IIC2143 - Ingeniería de Software
Sección 1

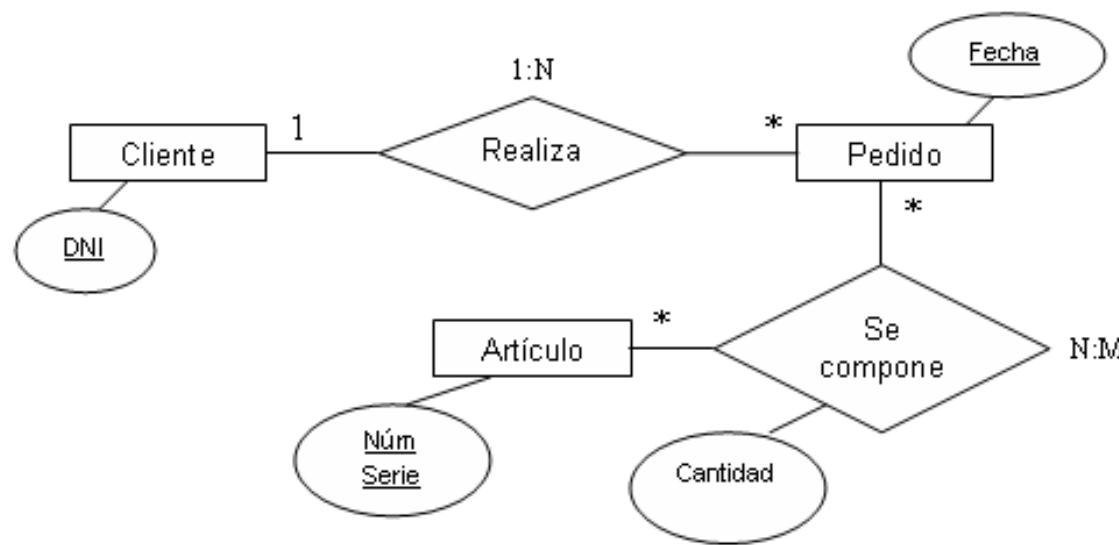
Rodrigo Saffie

rasaffie@uc.cl

23 de abril de 2018

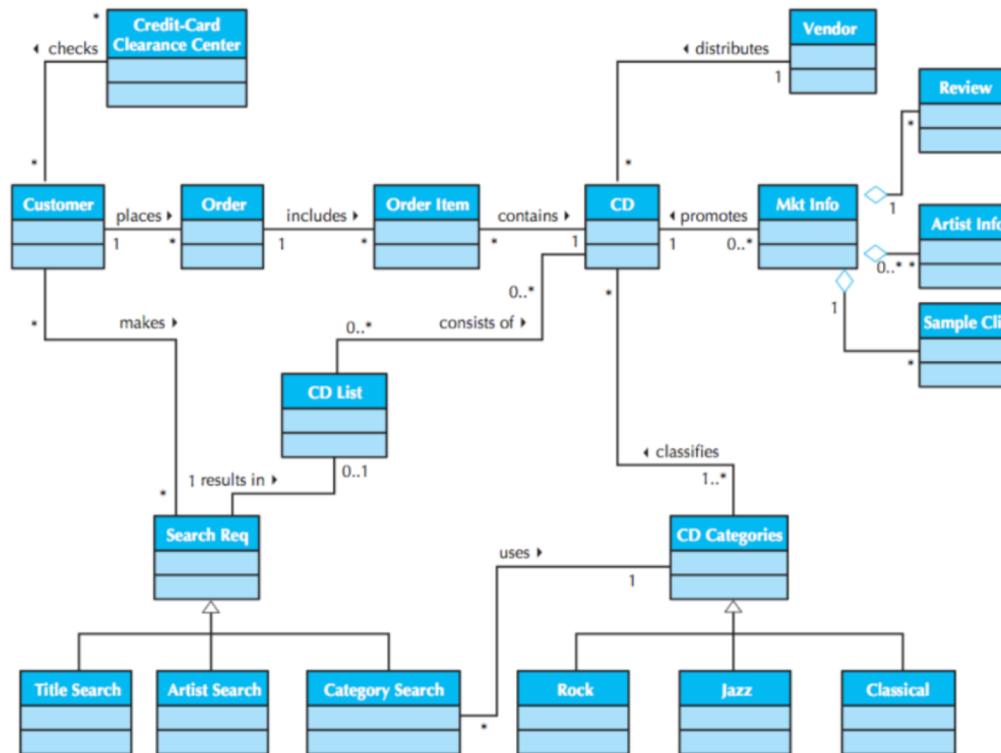
Modelo Entidad-Relación

- Herramienta para modelar datos persistentes de un sistema, junto con sus relaciones



Modelo de Dominio

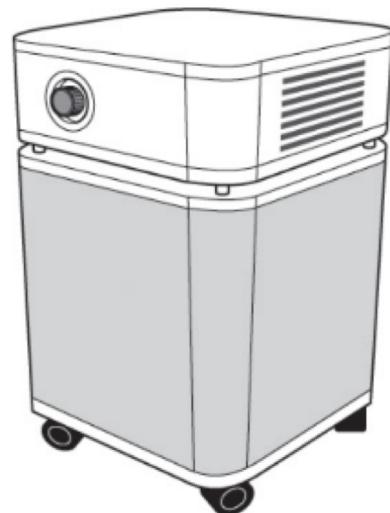
- Similar a un modelo de datos Entidad-Relación (E-R), pero no necesariamente los objetos son persistentes



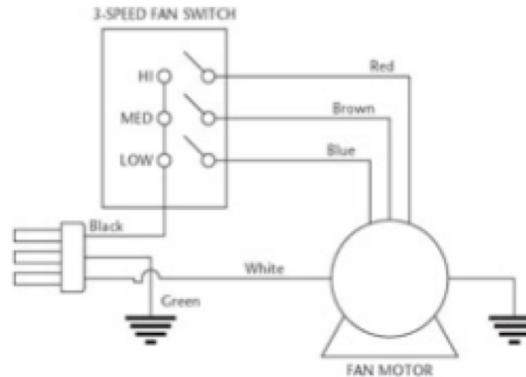
Unified Modeling Language 2.0

Lenguaje visual para describir artefactos de software

artefacto



lenguaje visual



Unified Modeling Language 2.0

```
public class Payment {  
    private float amount;  
    public Payment (float cashTendered){this.amount = cashTendered;}  
    public float getAmount() { return amount; }  
}  
  
public class ProductCatalog{  
    private HashTable productSpecifications = new Hashtable();  
    public ProductCatalog(){  
        ProductSpecification ps = new ProductSpecification(100, 1, "product 1");  
        productSpecifications.put (new Integer(100), ps);  
        ps = new ProductSpecification(200, 1, "product 2");  
        productSpecifications.put (new Integer(200), ps);  
    }  
  
    public ProductSpecification getSpecification (int upc){  
        return (ProductSpecification)productSpecifications.get(new Integer(upc));  
    }  
}  
  
class POST {  
    private ProductCatalog productCatalog;  
    private Sale sale;  
    public POST(ProductCatalog catalog){productCatalog = catalog;}  
    public void endSale() {sale.becomeComplete();}  
    public void enterItem(int upc, int quantity){  
        if (isNewSale() ) sale = new Sale();  
        ProductSpecification spec = productCatalog.specification(upc);  
        sale.makeLineItem(spec, quantity);  
    }  
    public void makePayment(float cashTendered){  
        sale.makePayment(cashTendered);  
    }  
    private boolean isNewSale(){return (sale == null) ||(sale.isComplete() );}  
}  
  
public class ProductSpecification{  
    private int upc = 0;  
    private float price = 0;  
    private String description = "";  
    public ProductSpecification(int upc, float price, String description){  
        this.upc = upc;  
        this.price = price;  
        this.description = description;  
    }  
    public int getUPC {return upc; }  
    public float getPrice() {return price; }  
    public String getDescription() {return description; }  
}  
  
class Sale {  
    private Vector lineItems = new Vector();  
    private Date date = new Date();  
    private boolean isComplete = false;  
    private Payment payment;  
    public float getBalance () {return payment.getAmount() - total(); }  
    public void becomeComplete() {isComplete = true; }  
    public boolean isComplete() {return isComplete; }  
    public void makeLineItem(ProductSpecification spec, int quantity) {  
        lineItems.addElement(new SaleLineItem(spec, quantity));  
    }  
    public float total(){  
        float total = 0;  
        Enumeration e = lineItems.elements();  
        while(e.hasMoreElements() ){  
            total += ( (SaleLineItem) e.nextElement() ).subtotal();  
        }  
        return total;  
    }  
    public void makePayment (float cashTendered) {  
        payment = new Payment (cashTendered);  
    }  
}  
  
class SaleLineItem{  
    private int quantity;  
    private ProductSpecification productSpec;  
    public SaleLineItem (ProductSpecification spec, int quantity) {  
        this.productSpec = spec;  
        this.quantity = quantity;  
    }  
    public float subtotal() { return quantity* productSpec.getPrice(); }  
}  
  
class Store {  
    private ProductCatalog productCatalog = new ProductCatalog();  
    private POST post = new POST (productCatalog);  
    public POST getPOST() {return post; }  
}
```

Artefacto

Unified Modeling Language 2.0

Representación Visual del Artefacto

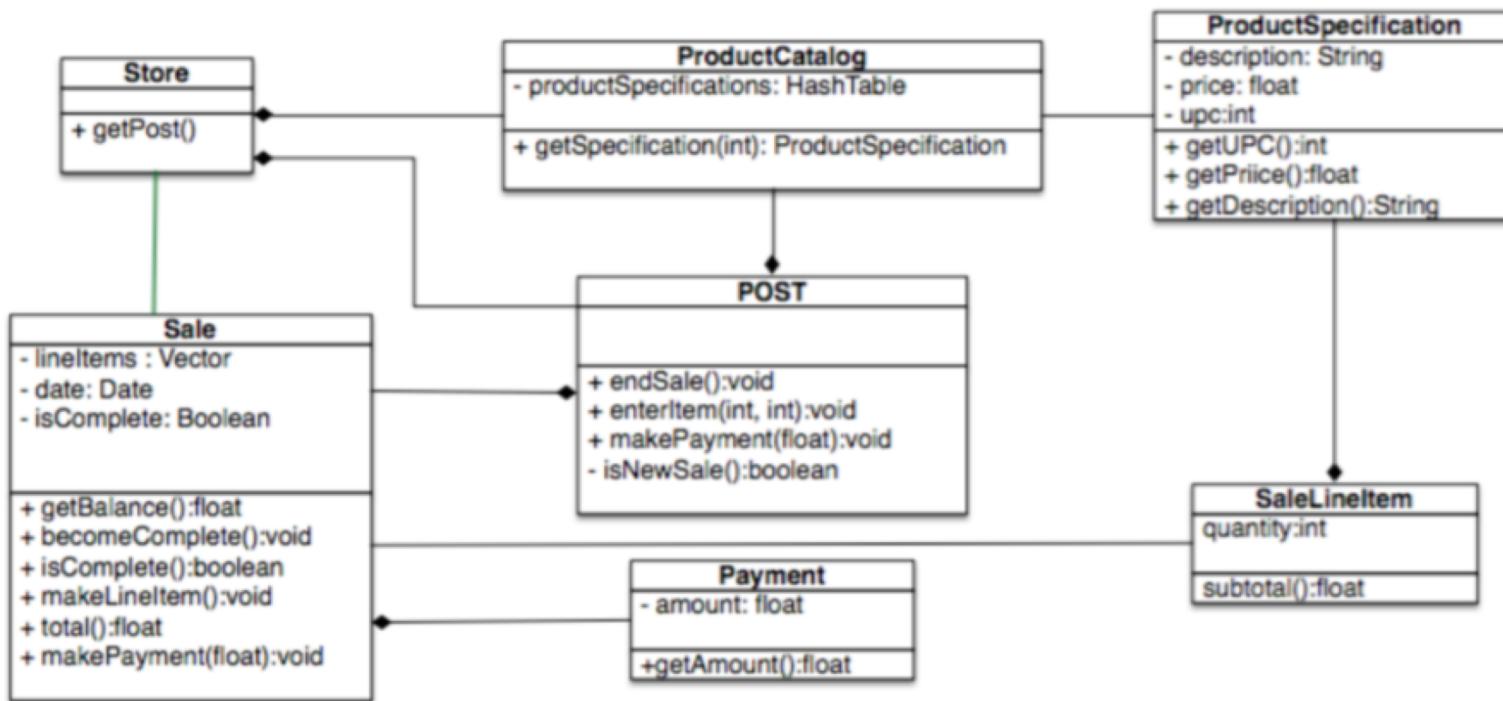
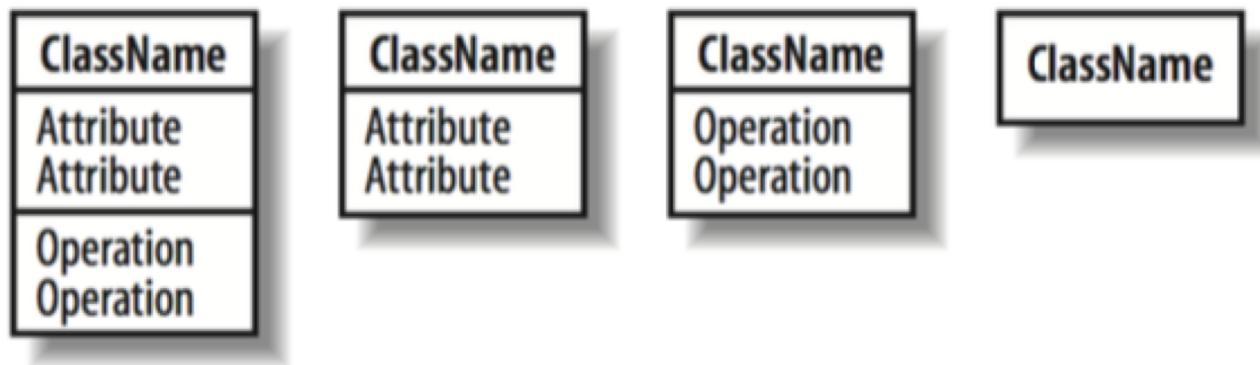


Diagrama de clases

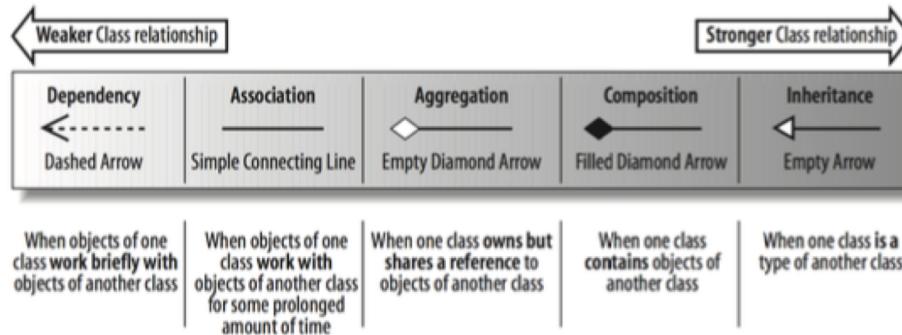
- Representa las clases de un sistema, junto con sus atributos, operaciones y relaciones.
- Sirve para:
 - Traducir el modelo de dominio en una implementación de *software*
 - Representar el diseño de un *software* orientado a objetos

Diagrama de clases - Elementos

- Rectángulos para representar clases



- Líneas para representar asociaciones entre clases



Actividad: Reserva aviones

Considera un sistema de reserva de vuelos con las siguientes características:

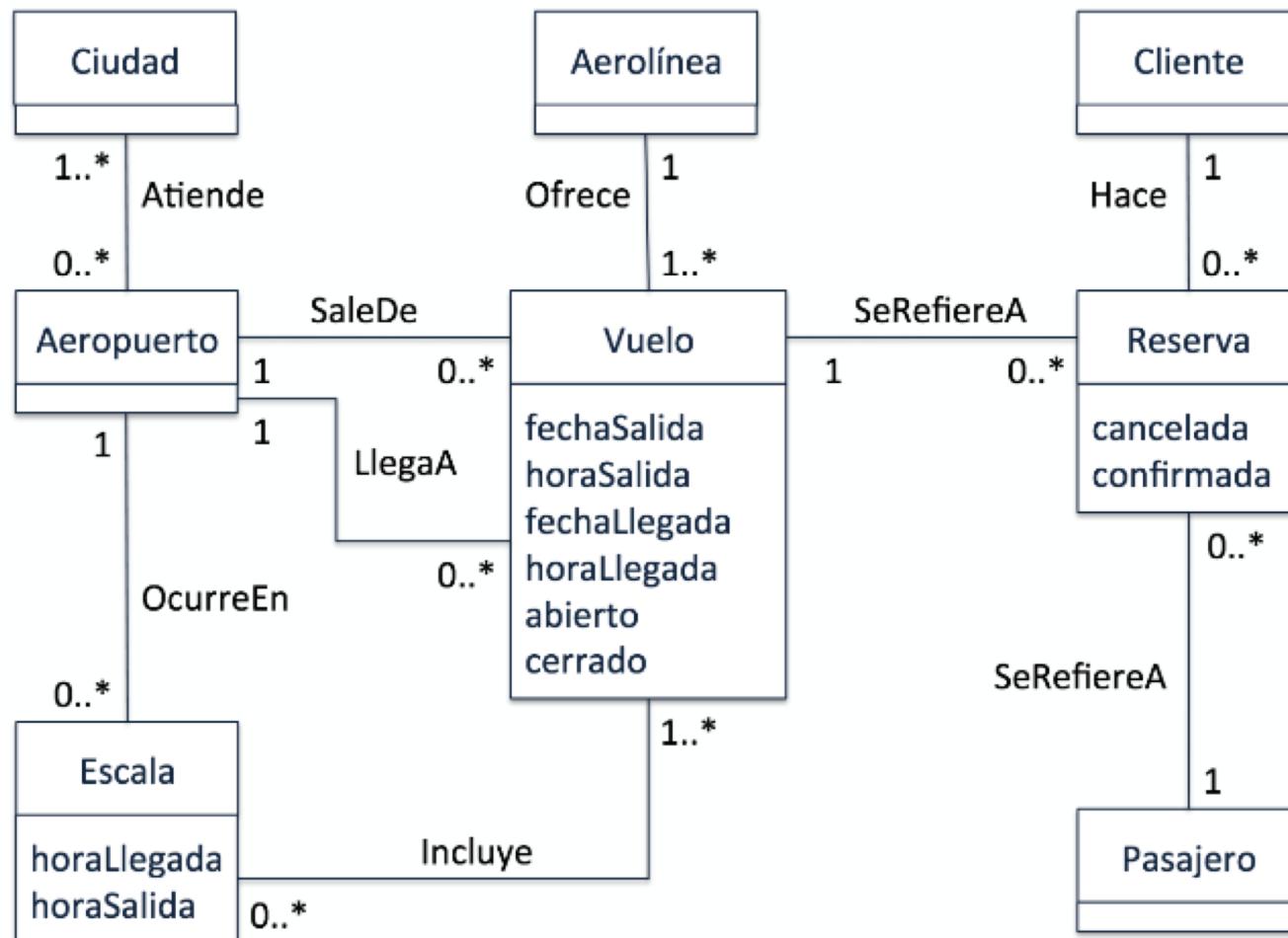
Las líneas aéreas ofrecen varios vuelos. Un vuelo es abierto para hacer reservas y nuevamente cerrado por orden de la compañía. Un cliente puede reservar uno o más vuelos y para diferentes pasajeros. Una reserva se refiere a un único vuelo y a un único pasajero, y puede ser cancelada o confirmada. Los vuelos tienen aeropuerto, fecha y hora de salida y de llegada, y pueden incluir escalas en aeropuertos. Cada escala tiene una hora de llegada y una hora de salida. Cada aeropuerto atiende a una o más ciudades.

Actividad: Reserva aviones

Entidades:

- Aerolíneas
- Vuelos
- Reservas
- Aeropuertos
- Pasajeros
- Clientes
- Ciudades

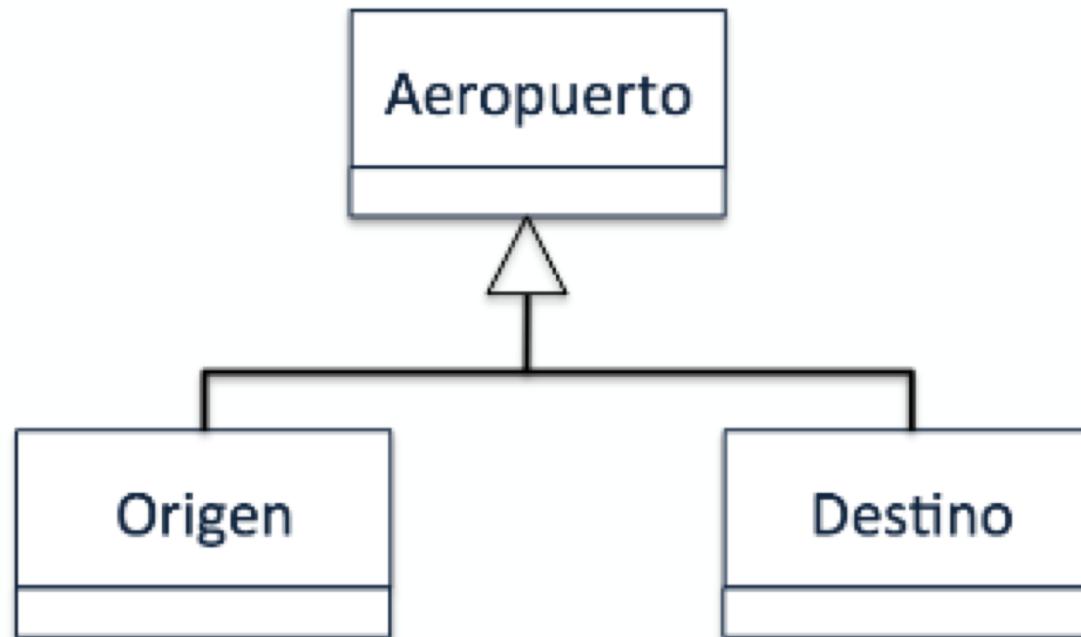
Actividad: Reserva aviones



Actividad: Reserva aviones

Alternativas diagrama de clases:

- Aeropuertos de origen y destino como subclases



Actividad: Reserva aviones

Alternativas diagrama de clases:

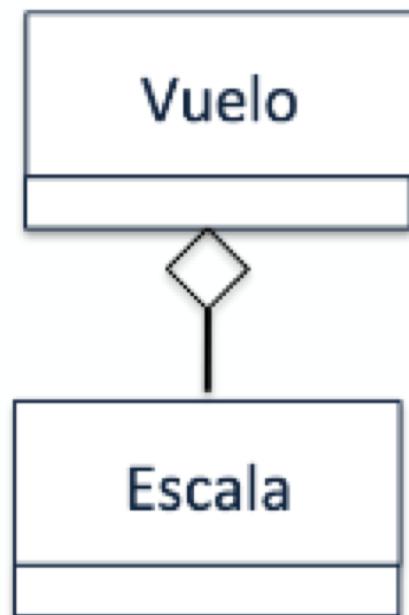
- Aeropuertos de origen y destino como subclases
- Escala como subclase de aeropuerto



Actividad: Reserva aviones

Alternativas diagrama de clases:

- Aeropuertos de origen y destino como subclases
- Escala como subclase de aeropuerto
- Vuelo contiene número de escalas



Patrones de diseño

¿Qué es un patrón de diseño?

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” [Christopher Alexander](#)

Patrones de diseño

¿Qué es un patrón de diseño?

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” [Christopher Alexander](#)

Este concepto fue acuñado por GoF en [Design Patterns: Elements of Reusable Object-Oriented Software](#) en 1994

Patrones de diseño

¿Qué es un patrón de diseño?

“Design patterns make it easier to reuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new systems. Design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability.” [GoF, 1994]

Patrones de diseño

¿Qué es un patrón de diseño?

“[...] allows the software engineering community to capture design knowledge in a way that enables it to be reused.” [Pressman, 2009]

Patrones de diseño

¿Qué caracteriza un patrón de diseño?

Según [Pressman, 2009]:

- Un contexto
- Un problema
- Una solución

Patrones de diseño

¿Qué caracteriza un patrón de diseño?

Según [GoF, 1994]:

- Un nombre
- Un problema
- Una solución
- Sus consecuencias

Patrones de diseño

En este curso se estudiarán algunos patrones de los 23 definidos en el libro de [GoF, 1994]

Estos patrones son aplicados a OOP, pero existen **muchos** más

Patrones de diseño

¿Por qué estudiarlos?

- Permiten reutilizar soluciones
- Permiten establecer terminología común: mejora la comunicación del equipo
- Nos dan una perspectiva de más alto nivel para el diseño de *software*
- Código es más fácil de modificar
- Permiten aprender estrategias generales de diseño

Patrones de diseño

Existen 3 clasificaciones:

- Creacionales
- Estructurales
- De comportamiento

Patrones de diseño

Creacionales:

- Se centran en la creación, composición y representación de los objetos

Patrones de diseño

Estructurales:

- Se centran en cómo los objetos se organizan e integran en un sistema

Patrones de diseño

De comportamiento:

- Se centran en las interacciones y responsabilidades entre objetos

Patrones de diseño

¿Cómo seleccionar un patrón de diseño?

- Detectar el problema: causa del rediseño
- Analizar el propósito de cada patrón
- Identificar si hay relación entre el problema y un propósito
- Estudiar los patrones de la misma categoría
- Considerar qué podría variar en el diseño

Patrones de diseño



Patrones estructurales

Façade (Fachada)

Provee una interfaz unificada a un conjunto de interfaces en un sub-sistema. *Façade* define un interfaz de alto nivel que hace a un sistema más fácil de utilizar.

Patrones estructurales

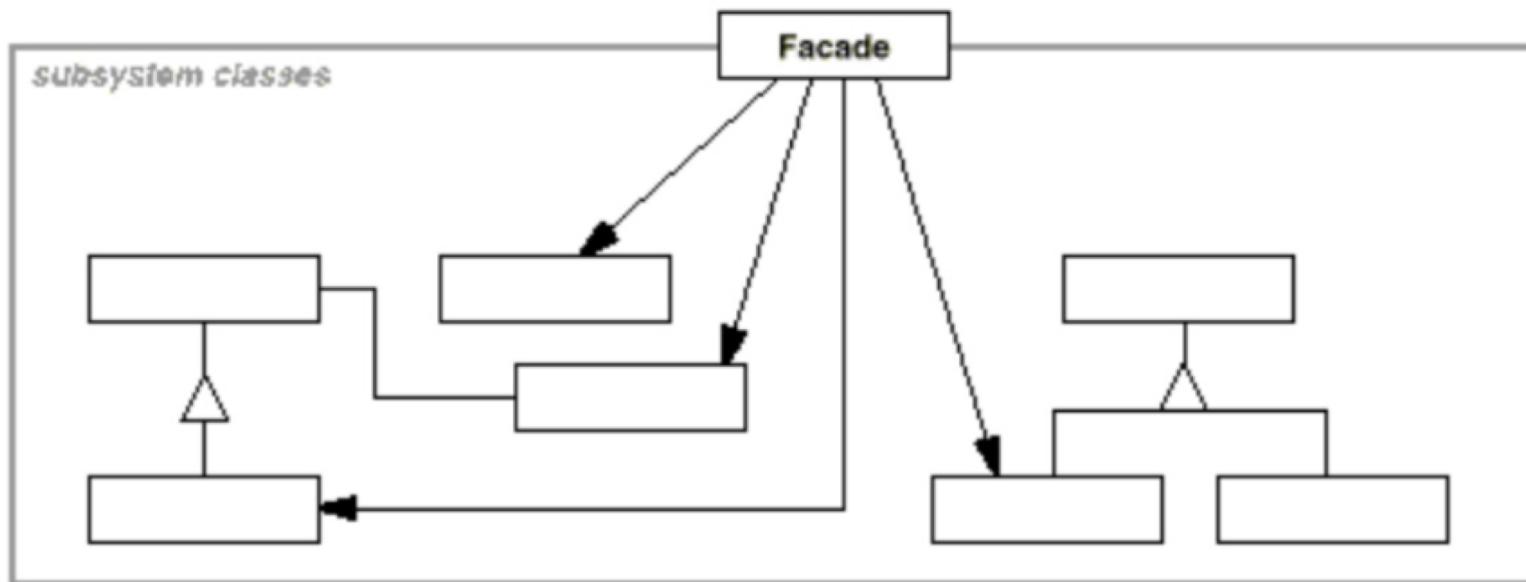
Façade (Fachada)

¿Cuándo se utiliza?

- Se quiere proveer una interfaz simple para un sistema complejo (altamente configurable)
- Desacoplar los clientes de los sub-sistemas
- Ordenar los sub-sistemas por capas, con una *façade* como punto de entrada para cada capa

Patrones estructurales

Facade (Fachada)



Ejemplo

Patrones estructurales

Adapter

Convierte la interfaz de una clase en otra interfaz que los clientes esperan. Adapter permite a ciertas clases interactuar, sin lo cual serían incompatibles.

Patrones estructurales

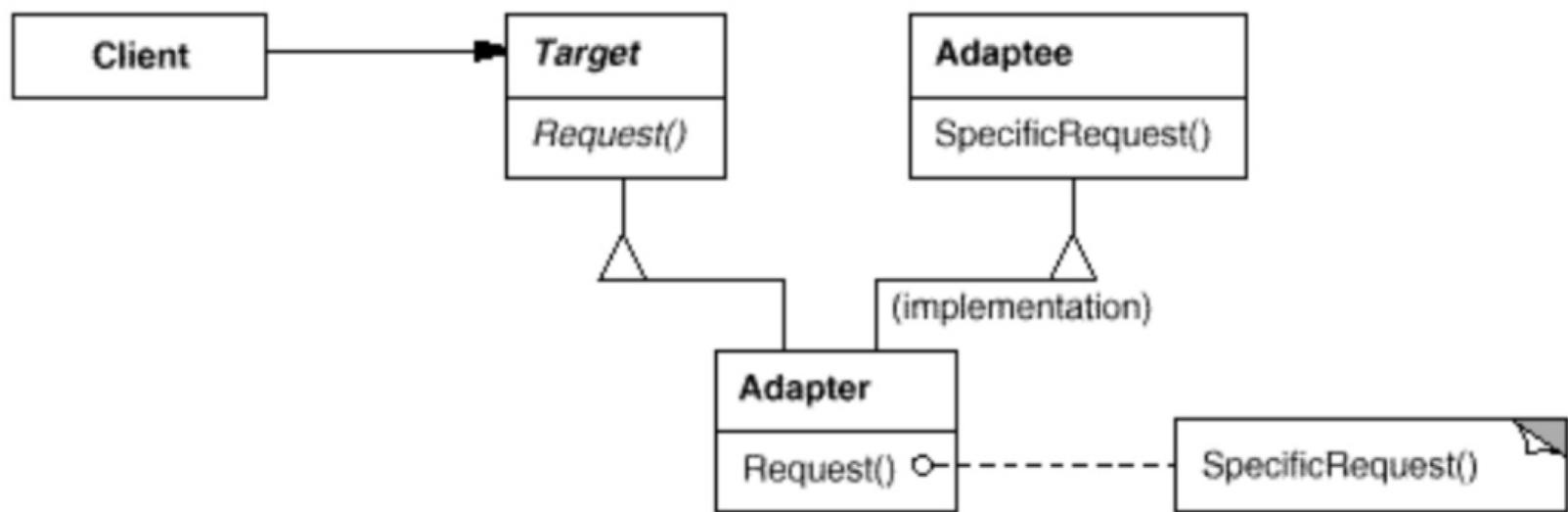
Adapter

¿Cuándo se utiliza?

- Se necesita utilizar una clase existente, pero su interfaz es incompatible con lo que se necesita
- Se quiere crear una clase que coopere con otras clases que utilizan interfaces incompatibles

Patrones estructurales

Adapter



Ejemplo



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Clase 13

Patrones de diseño

IIC2143 - Ingeniería de Software
Sección 1

Rodrigo Saffie

rasaffie@uc.cl

23 de abril de 2018