Programación 2

Patrones de diseño

Patrones de diseño

¿Qué es un patrón?

Solo con observar a nuestro alrededor encontraremos conjuntos de elementos que estén dispuestos de una determinada manera, siguiendo una regla.

Patrones de diseño

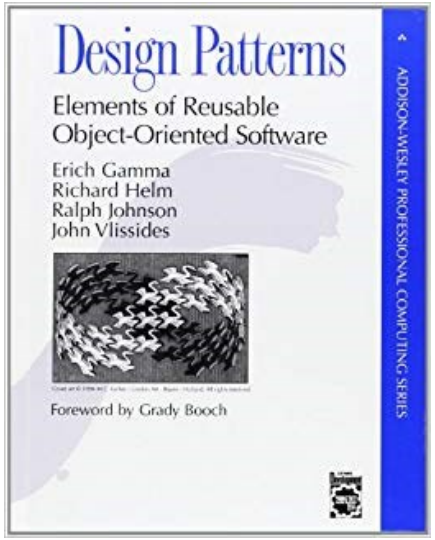
- El diseño orientado a objetos no es una tarea sencilla, y el diseño reusable es aún más difícil.
- Los diseñadores expertos no resuelven todos los problemas desde cero, sino que reutilizan soluciones que han funcionado anteriormente para problemas similares.
    - Buscan conjuntos de clases y objetos y formas de relacionarlos que son recurrentes en muchos sistemas orientados a objetos

Patrones de diseño

- En el diseño de software, un patrón de diseño no es más que una forma “estandarizada” de resolver problemas comunes
- Definen un “molde” de una solución para un determinado problema que es recurrente.
- Resuelven problemas específicos de diseño y hacen que el diseño sea flexible y reusable.

Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre si adaptada para resolver un problema de diseño general en un contexto particular [Gamma, 1994]



Patrones de diseño

- Ventajas
  - Conforman un amplio catálogo de problemas y soluciones
  - Estandarizan la resolución de determinados problemas
  - Condensan y simplifican el aprendizaje de las buenas prácticas
  - Proporcionan un vocabulario común entre desarrolladores
  - Evitan “reinventar la rueda”

Patrones de diseño

- Elementos de un patrón
  - **Nombre**: describe el problema de diseño
  - **Problema**: describe cuándo aplicar el patrón
  - **Solución**: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.

Patrones de diseño

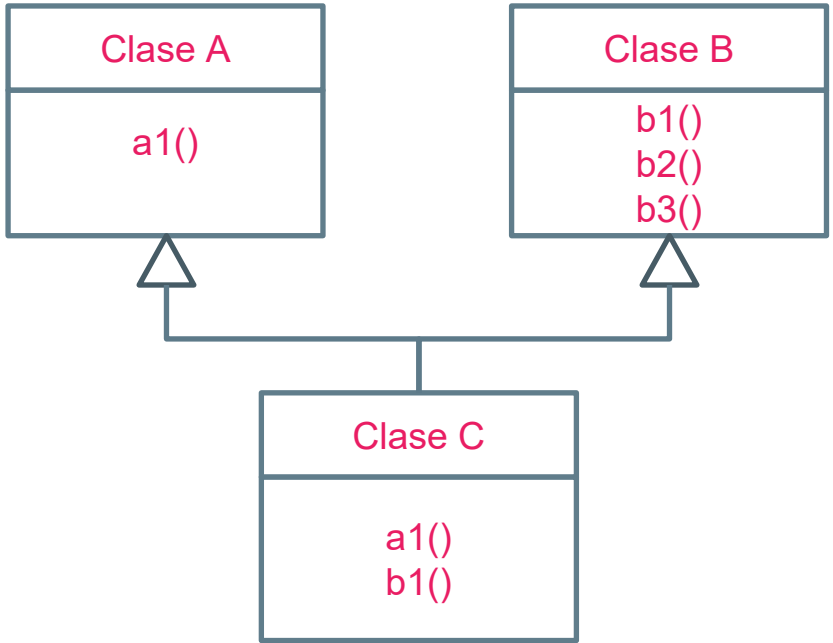
- Además de los patrones de diseño “de catálogo” existen otros que son tomados como principios y ya estuvimos trabajando:
  - Delegación
  - Interface

Patrones de diseño - Delegación

- Utilidad: cuando se quiere extender y reutilizar la funcionalidad de una clase sin utilizar herencia.

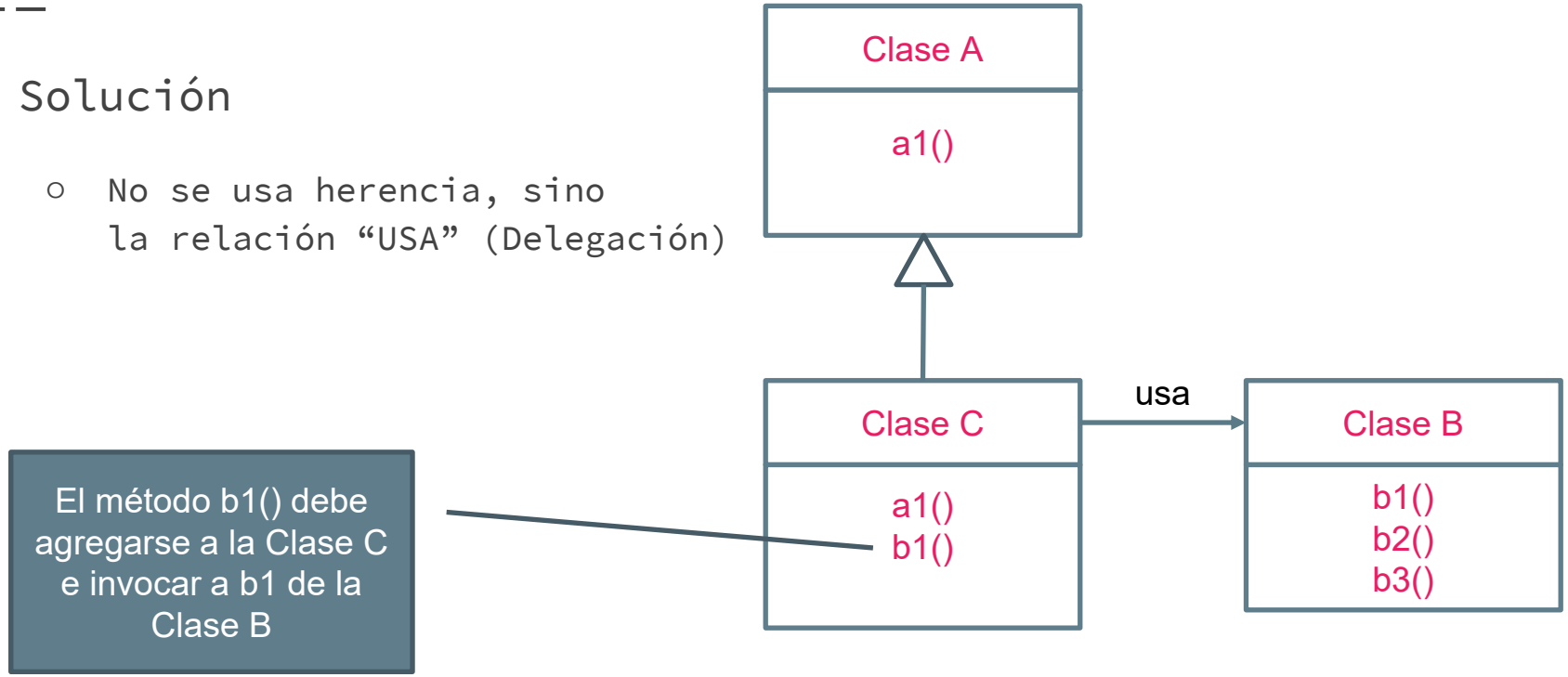
Patrones de diseño - Delegación

- Utilidad: cuando se quiere extender y reutilizar la funcionalidad de una clase sin utilizar herencia.
- Problema
  - El lenguaje utilizado no posee herencia múltiple
  - La Clase C no desea todos los métodos de B



Patrones de diseño - Delegación

- Solución
  - No se usa herencia, sino la relación “USA” (Delegación)

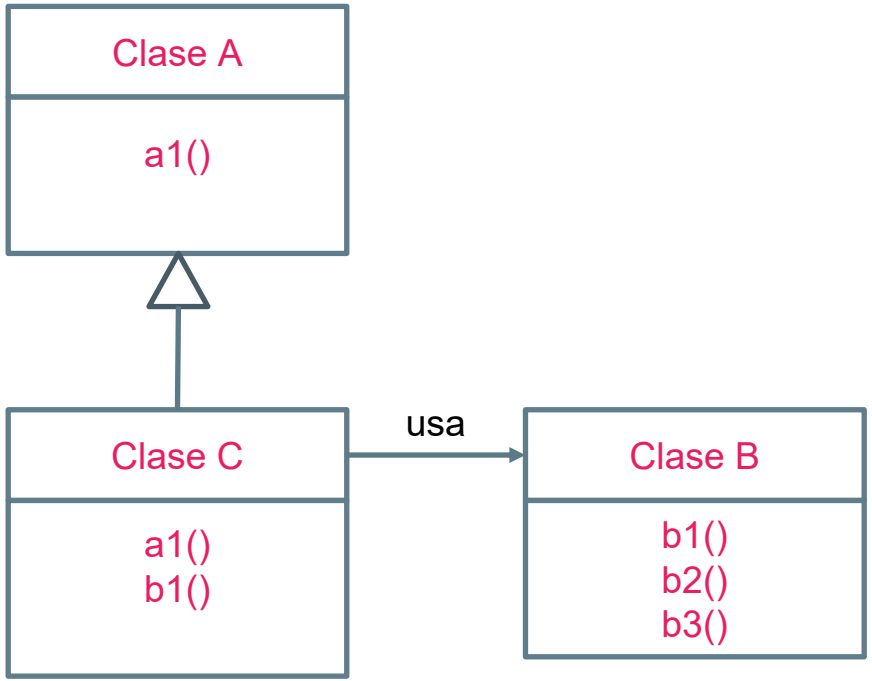


Patrones de diseño - Delegación

- Solución
  - No se usa herencia, sino la relación “USA” (Delegación)

```
public class ClaseC extends ClaseA {
    ClaseB objetoB;
    public ClaseC(){
        objetoB = new ClaseB();
    }

    public void b1(){
        objetoB.b1();
    }
```



Patrones de diseño - Delegación

- Ventajas:
  - Cuando una clase que hereda de otra quiere ocultar algunos de los métodos heredados
  - Compartir código que no se puede heredar
  - Permite cambiar el comportamiento de un objeto en tiempo de ejecución

Patrones de diseño - Interface

- Utilidad: definir un comportamiento, independientemente de dónde vaya a ser utilizado
- Ventajas:
  - Desacople entre comportamiento y clase

Patrones de diseño - Interface

- Problema, se desea ordenar una lista de objetos independientemente del tipo de objeto que contenga

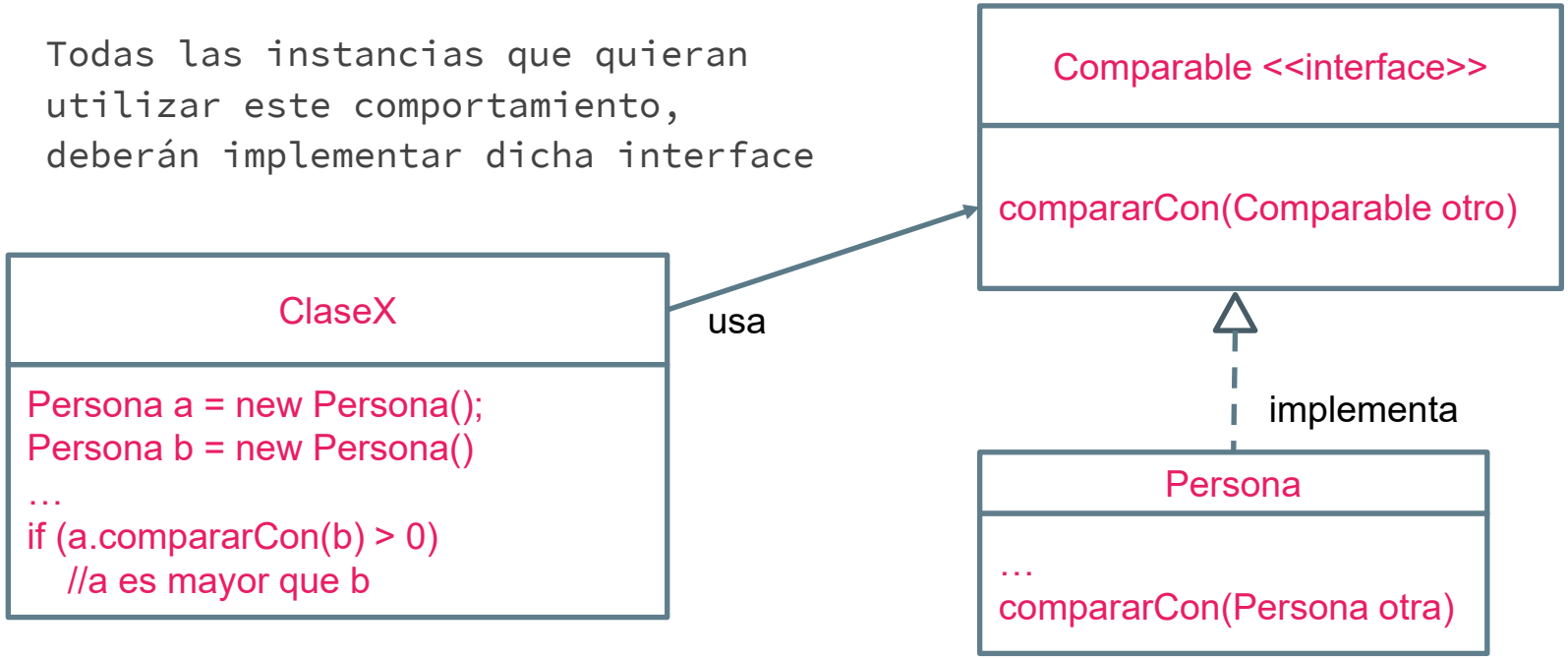
```
public static void ordenar(List lista){
   int n = lista.size();
   for (int i = 0; i < n-1; i++)
      for (int j = 0; j < n-i-1; j++)
         if (lista.get(j) > (lista.get(j+1))) {
            // intercambiamos lista[j+1] con lista[j]
            ?? temp = lista.get(j);
            lista.set(j,lista.get(j+1));
            lista.set(j+1, temp);
         }
}
```

¿a qué objetos puedo comparar con > o <

¿de qué tipo debo declarar a temp?

Patrones de diseño - Interface

- Solución: Definir los objetos del un tipo Interface
  - Todas las instancias que quieran utilizar este comportamiento, deberán implementar dicha interface



Patrones de diseño

- Gamma et al. clasifican los patrones según su propósito:
  - **De creación**: se utilizan para crear y configurar clases y objetos
  - **De estructura**: se utilizan para desacoplar las interfaces e implementar clases y objetos. Crean grupos de objetos con determinadas características
  - **De comportamiento**: se enfocan en la interacción entre asociaciones de clases y objetos, definiendo cómo se comunican entre sí

Patrones de diseño

Creación	Estructural	Comportamiento
• Factory Method • Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Flyweight • Facade • Proxy	• Interpreter • Template method • Chain of responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

Patrones de diseño – Patrones de Creación

- Como su nombre indica, estos patrones vienen a solucionar o facilitar las tareas de creación o instanciación de objetos.

- Estos patrones hacen hincapié en la encapsulación de la lógica de la instanciación, ocultando los detalles concretos de cada objeto y permitiéndonos trabajar con abstracciones.

Patrones de diseño – Patrones de Creación

- Los más importante son:
  - **Factory Method**: Expone un método de creación, delegando en las subclases la implementación de este método.
  - **Abstract Factory**: Nos provee una interface que delega la creación de una serie de objetos relacionados sin necesidad de especificar cuáles son las implementaciones concretas.
  - **Builder**: Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción nos puede servir para crear representaciones diferentes.

Patrones de diseño – Patrones Estructurales

- Nos ayudan a definir la forma en la que los objetos se componen. Los más habituales son:
  - **Adapter**: Nos ayuda a definir una clase intermedia que sirve para que dos clases con diferentes interfaces puedan comunicarse. También se conoce como Wrapper.
  - **Decorator**: Permite añadir funcionalidad extra a un objeto (decora el objeto) sin modificar el comportamiento del resto de instancias.
  - **Facade**: Una fachada es un objeto que crea una interfaz simplificada para tratar con otra parte del código más compleja.
  - **Composite**: Permite componer objetos en jerarquías todo-parte y permitir a los clientes tratar objetos simples y compuestos de manera uniforme

Patrones de diseño – Patrones de Comportamiento

- Nos ayudan a definir la forma en la que los objetos interactúan entre ellos.
- Algunos de los más conocidos son:
  - **Command**: Son objetos que encapsulan una acción y los parámetros que necesitan para ejecutarse.
  - **Observer**: Los objetos son capaces de suscribirse a una serie de eventos que otro objeto va a emitir, y serán avisados cuando esto ocurra.
  - **Strategy**: Permite la selección del algoritmo que ejecuta cierta acción en tiempo de ejecución.
  - **Template Method**: Especifica el esqueleto de un algoritmo, permitiendo a las subclases definir cómo implementan el comportamiento real.