

# Patrón de Diseño: Observer






# Patrón de Diseño de Comportamiento



# Sujeto y Observadores





# Relación uno-a-muchos entre objetos



# Objetivo



# Utilidad

# Utilidad

- Mantiene diferentes partes de un programa sincronizadas y actualizadas en tiempo real
- Implementación de GUI
- Comunicación bidireccional entre objetos
- Utilizado en el JDK y al implementar el patrón de arquitectura MVC
- Consistencia entre clases relacionadas, pero con independencia




# Bajo Acoplamiento







# Bajo Acoplamiento

- Lo único que el sujeto sabe sobre un observador es que implementa una determinada interfaz
  - Implementación se puede añadir nuevos observadores cuando queramos
  - Nunca tendremos que modificar el sujeto para añadir nuevos tipos de observadores
  - Los cambios en un sujeto o en un observador no afectarán al otro
- 




# Ventajas y Desventajas






# Ventajas

- Separación de responsabilidades
  - Flexibilidad
  - Desacoplamiento
  - Actualización automática
- 

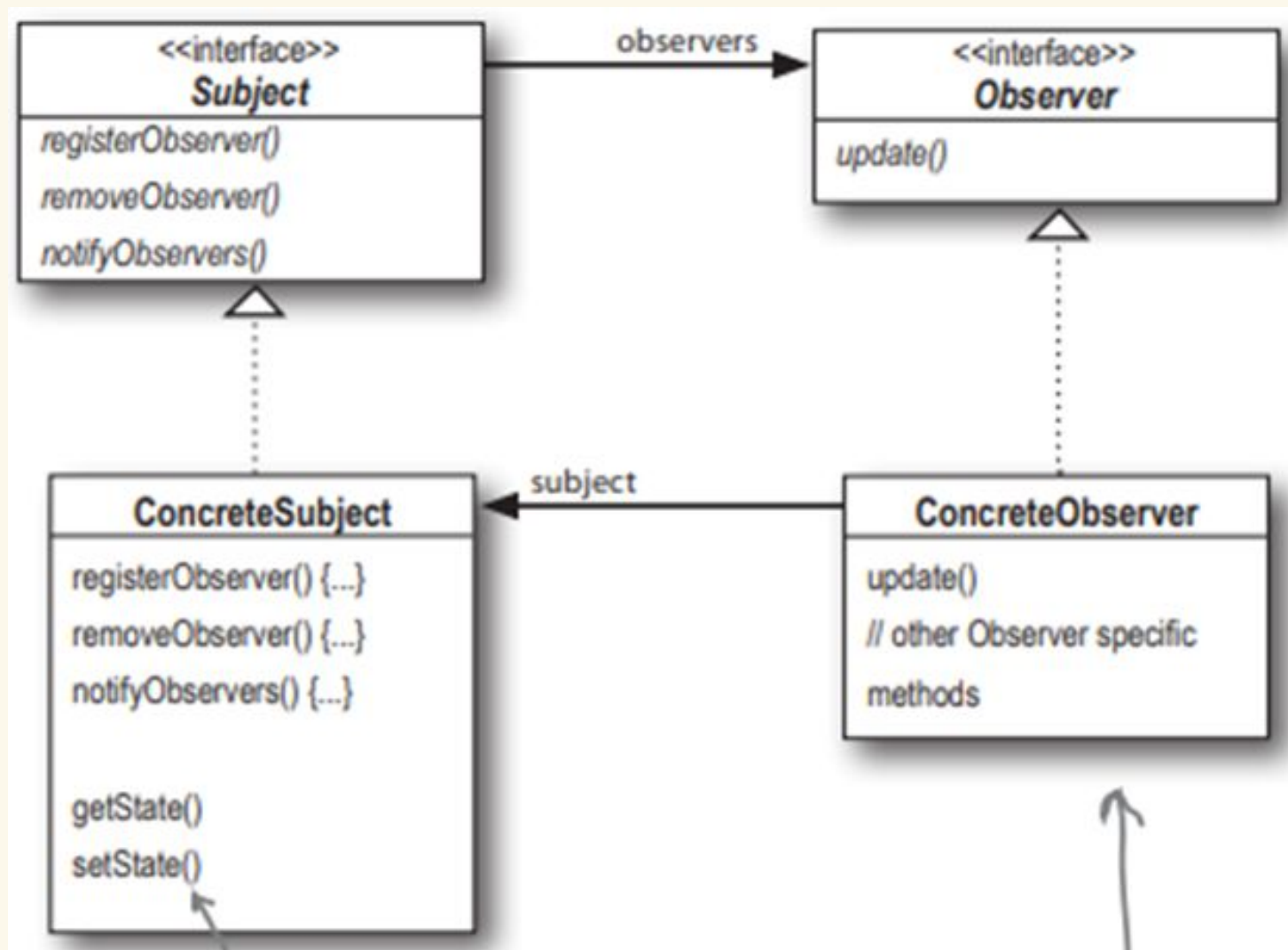


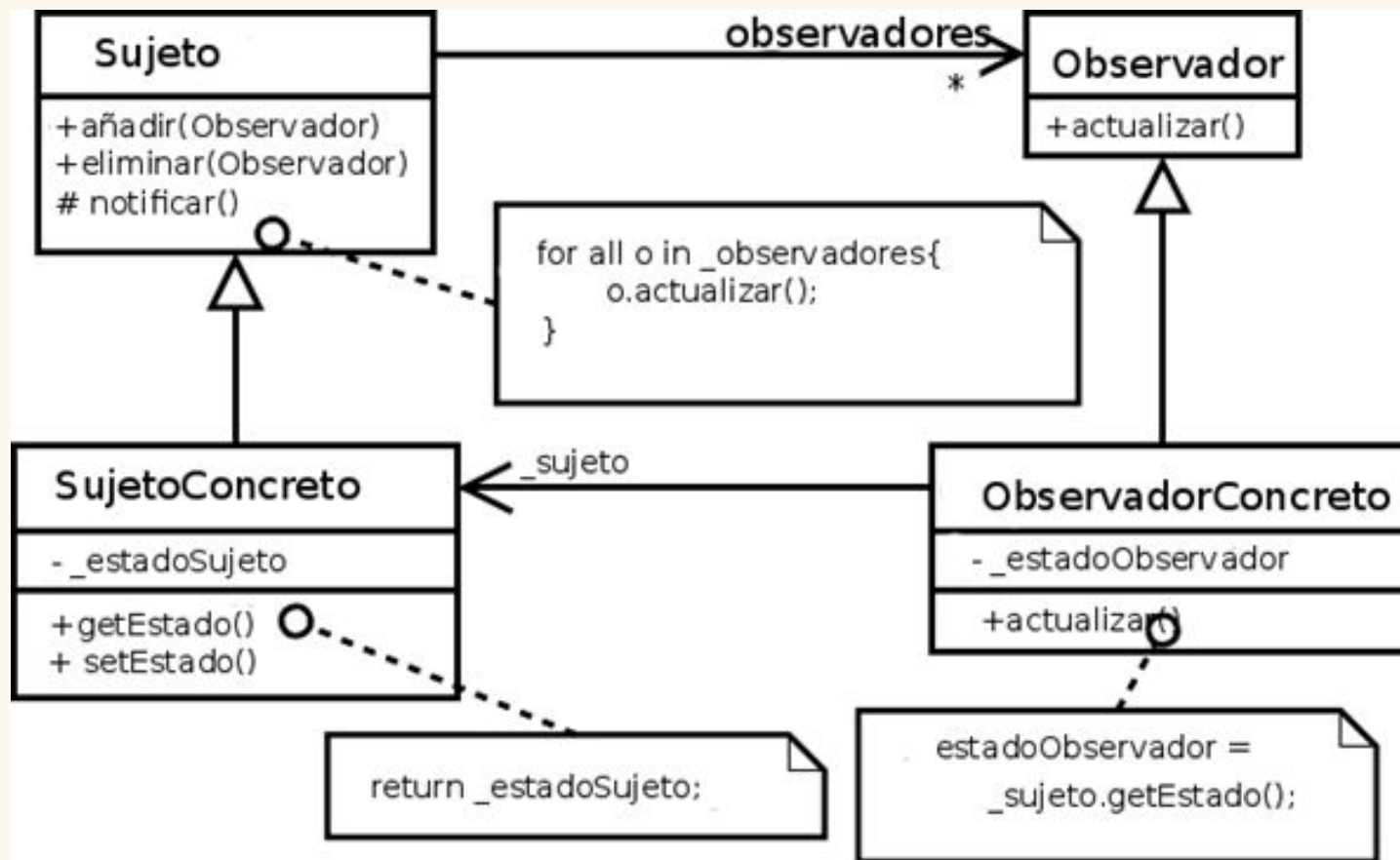
# Desventajas

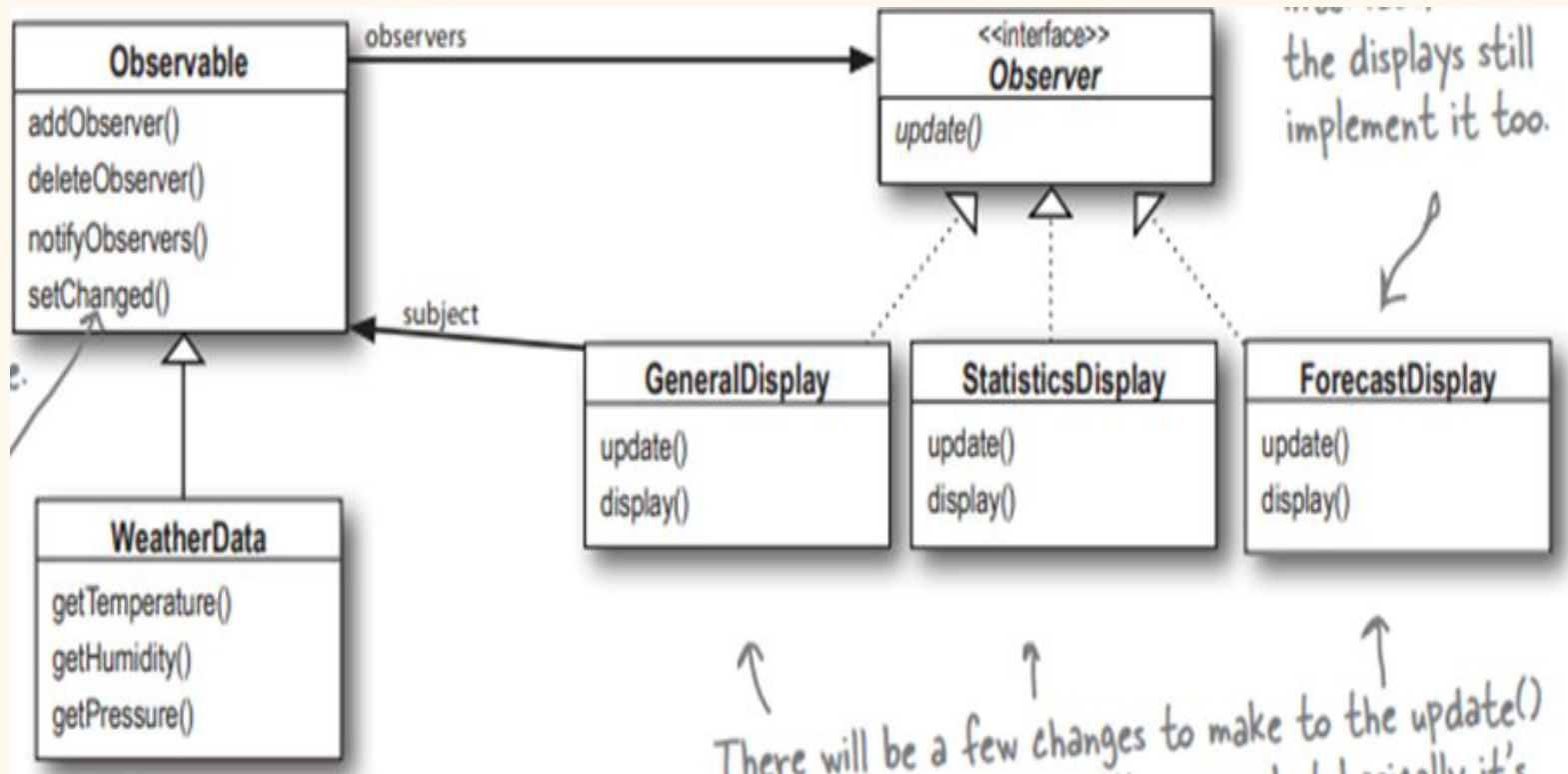
- **Sobrecarga de rendimiento**
  - **Dificultad para mantener el orden de notificación**
  - **Complejidad de implementación**
  - **Posible pérdida de datos**
- 



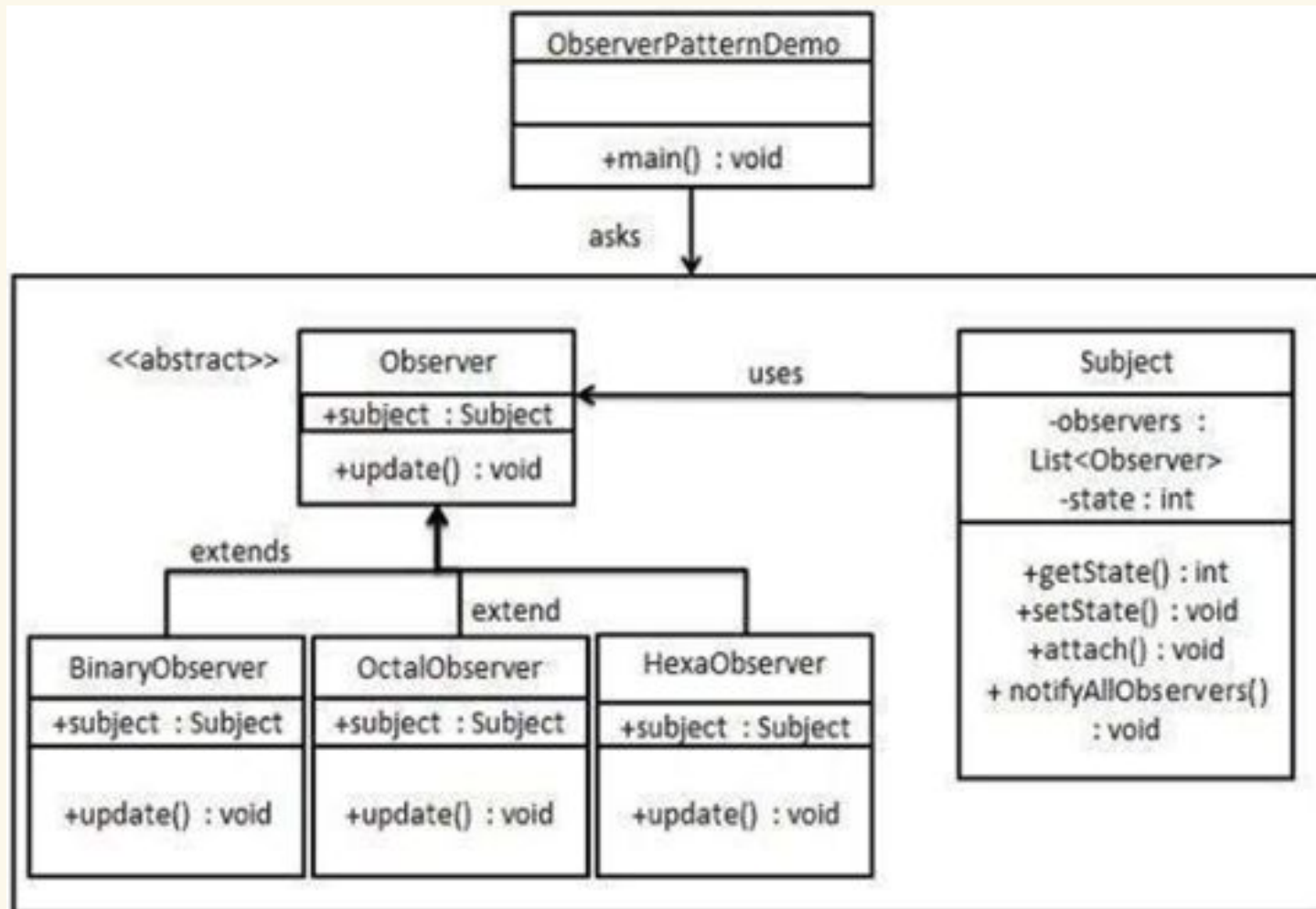
# Estructura e Implementación

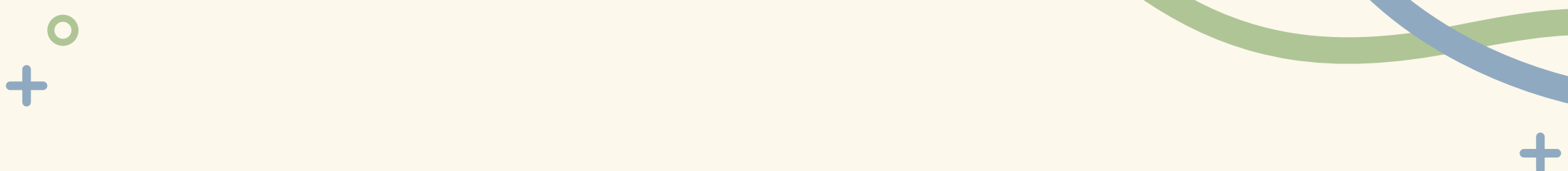












```
public class ObserverPattern {
```

Run | Debug

```
public static void main(String[] args) {
```

```
    Product product = new Product(name:"Smartphone", price:800);
```

```
    User user1 = new User(name:"Juan");
```

```
    User user2 = new User(name:"Ana");
```

```
    product.addObserver(user1);
```

```
    product.addObserver(user2);
```

```
    product.setPrice(price:700);
```

```
    // Notifica a usuarios interesados en el producto sobre el cambio de precio.
```

```
}
```

```
}
```



```
import java.util.ArrayList;
import java.util.List;

public class Product {
    private String name;
    private double price;
    private List<ProductObserver> observers = new ArrayList<>();

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public void addObserver(ProductObserver observer) {
        observers.add(observer);
    }

    public void removeObserver(ProductObserver observer) {
        observers.remove(observer);
    }

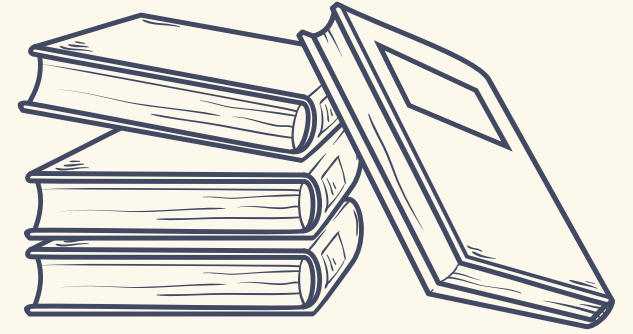
    public void setPrice(double price) {
        this.price = price;
        notifyObservers();
    }

    private void notifyObservers() {
        for (ProductObserver observer : observers) {
            observer.update(name, price);
        }
    }
}
```

```
public abstract class ProductObserver {  
    public abstract void update(String name, double price);  
}
```

```
public class User extends ProductObserver {  
    private String name;  
  
    public User(String name) {  
        this.name = name;  
    }  
  
    public void update(String name, double price) {  
        System.out.println(name + " ha cambiado su precio a " + price + ". El usuario " + this.name + " ha sido notificado.");  
    }  
}
```

```
Smartphone ha cambiado su precio a 700.0. El usuario Juan ha sido notificado.  
Smartphone ha cambiado su precio a 700.0. El usuario Ana ha sido notificado.
```



# ¡Gracias!

¿Preguntas?



