IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

10

Padrão de Desenho: Observer

Bruno Silva, Patrícia Macedo

Sumário 📝

- Padrão **Observer**
  - Enquadramento
  - Problema
  - Solução Proposta (pelo padrão)
  - Exemplo de Aplicação
    - Exercícios
  - o padrão Observer e o Java

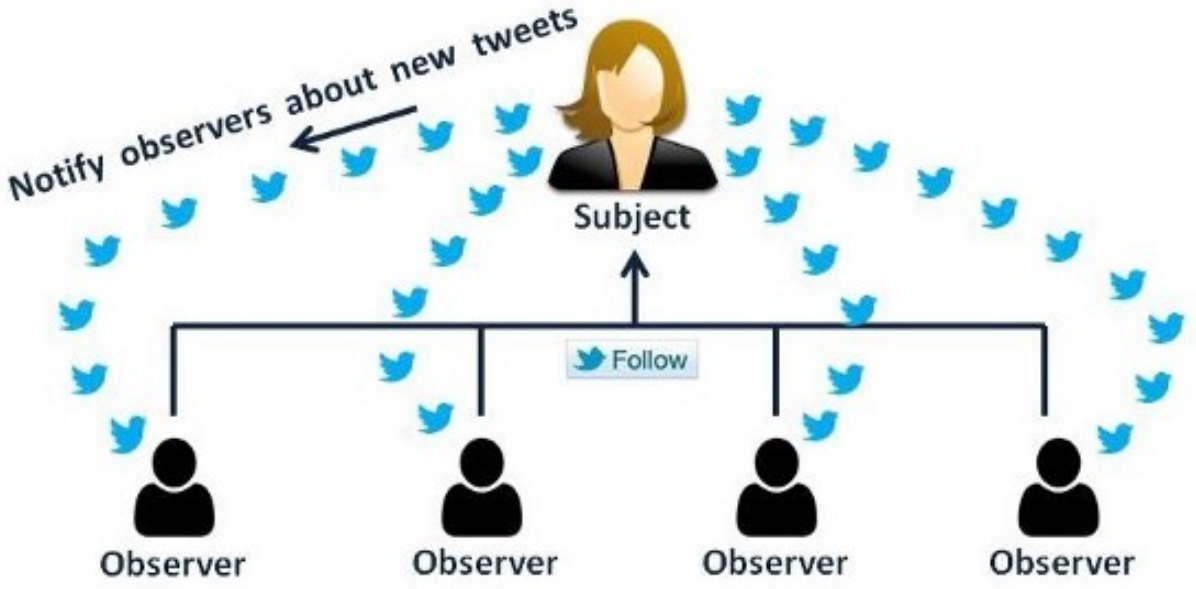
Enquadramento 

Existem várias situações em que:

- Pretende-se assegurar que, quando um objecto muda de estado, um número de objectos dependentes é actualizado automaticamente.
- Um objecto pretende notificar um conjunto de outros objetos.

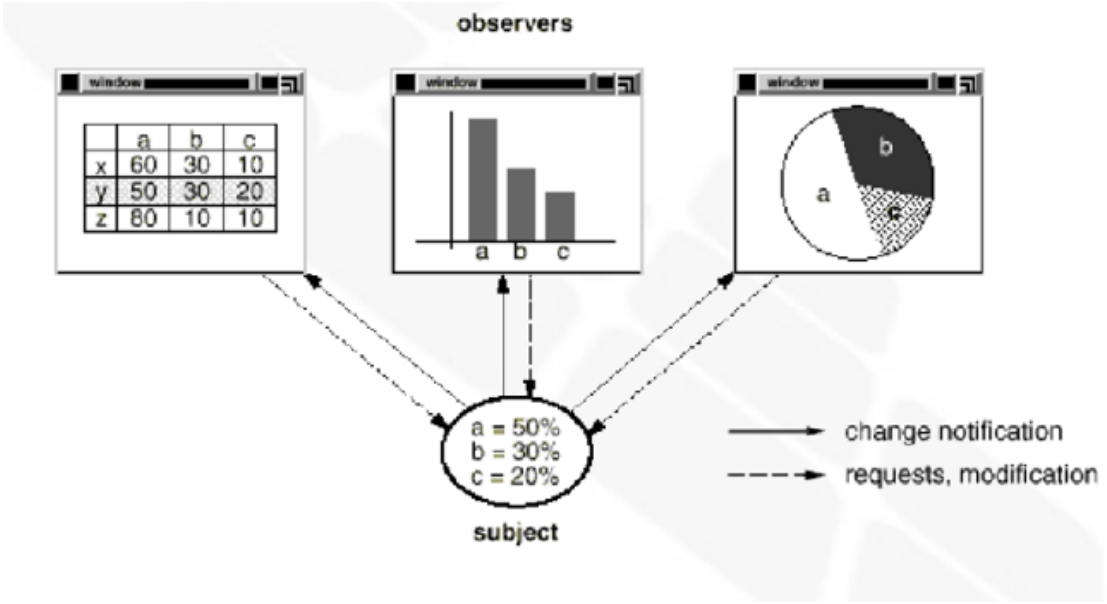
Motivação 🤔

Notificar um conjunto de susbcritores de uma plataforma que uma nova versão foi lançada.

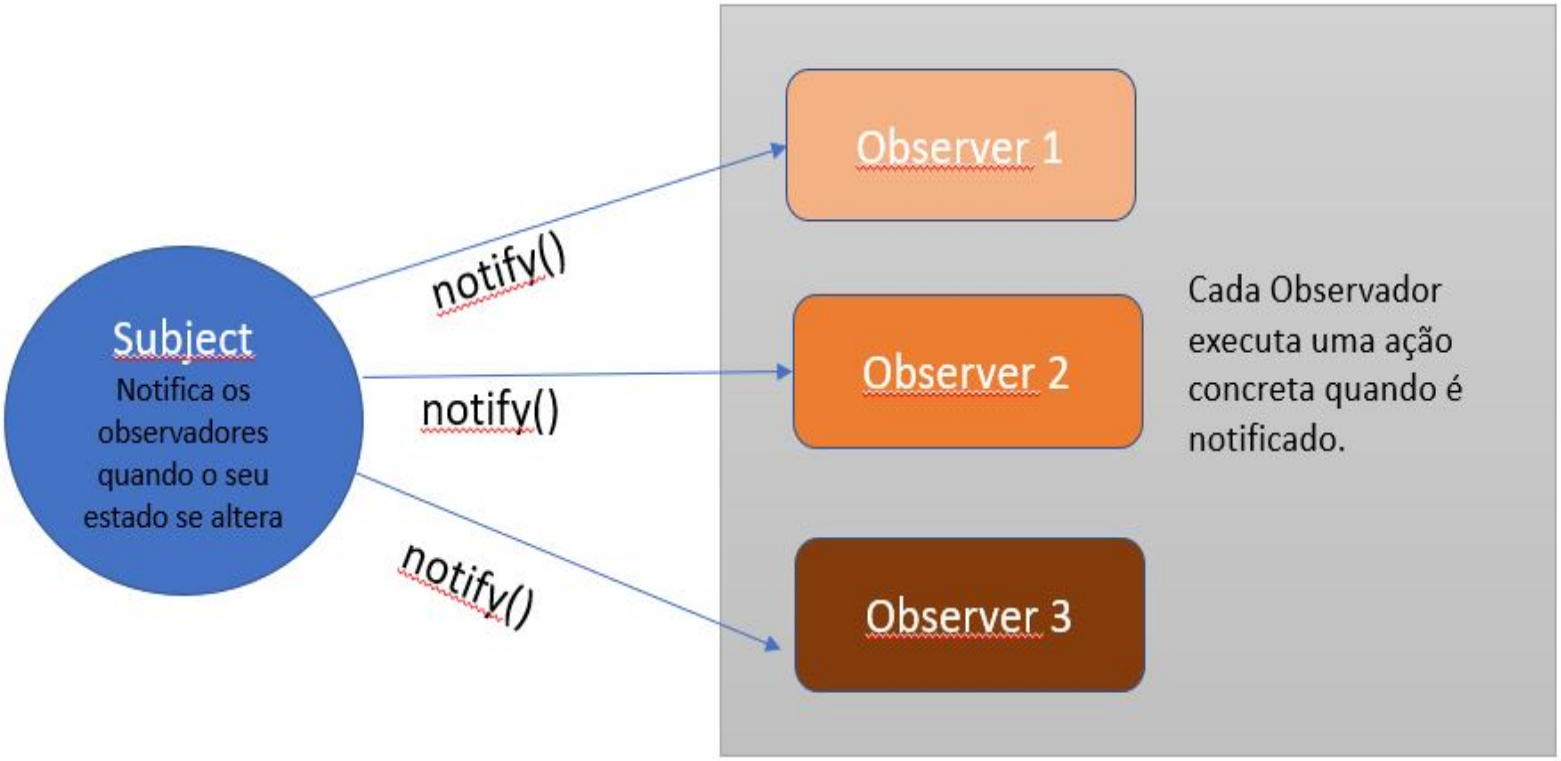


Motivação 🤔

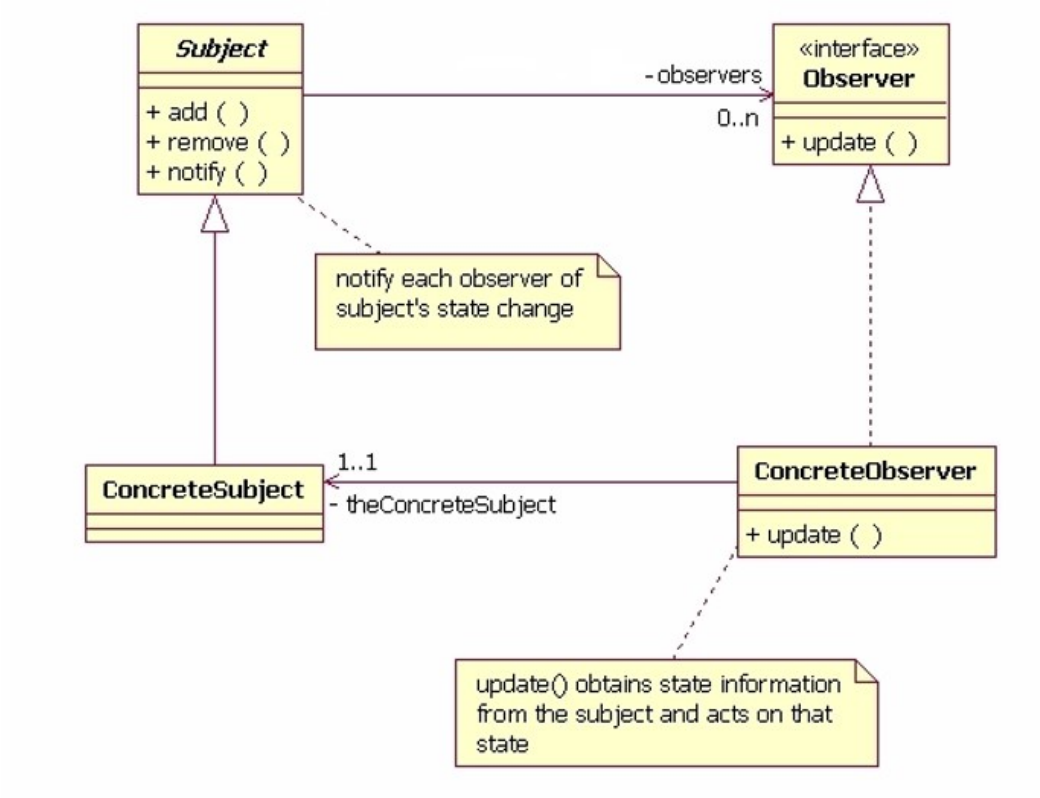
Atualizar a visualização dos dados automáticamente cada vez que os dados são alterados



Solução Proposta: Padrão observer



Solução Proposta: Padrão observer



Participantes do Padrão:

- Subject
  - Conhece os seus observadores.
  - Qualquer número de objectos Observer pode observar um Subject.
  - Fornece uma interface para adicionar e remover objectos Observer.
- ConcreteSubject
  - Armazena os estados que interessam ao ConcreteObserver.
  - Envia uma notificação aos Observers, quando o estado se altera.

Participantes do Padrão:

- Observer
  - Define uma interface de alteração para os objectos que devam ser notificados de alterações verificados no Subject.
- ConcreteObserver
  - Mantém uma referência para o objecto ConcreteSubject.
  - Armazena o estado consistente com o de Subject.
  - Implementa a interface de actualização definida em Observer, para que o seu estado seja consistente com o de Subject.

Variante à implementação do Padrão Observer

- Um dos problemas que levanta a utilização do padrão Observer, é o facto de ele definir uma classe abstrata Subject em vez de uma interface.
- Como não existe em JAVA herança multipla isso dificulta quando queremos tornar uma subclasse "observable".
- Por esta razão vamos implementar uma variante do padrão Observer onde é acrescentado uma interface Observable.

Interface Observable

```
    public interface Observable {
    /**
     *  Attach  observers to the subject.
     * @param observers  to be attached
     */
    public void addObservers(Observer... observers);
    /**
     *  Attach  observers to the subject.
     * @param observer to be removed
     */
    public void removeObservers(Observer observer);

    /**
     * notify all observer
     * @param object, argument of update method
     */
    public void notifyObservers(Object object);
}
```

Classe Abstrata Subject

```
public abstract class Subject implements Observable{
 private List<Observer> observerList;

    public Subject() {
        this.observerList = new ArrayList<>();
    }

    @Override
    public void addObservers(Observer... observers) {
        for (Observer obs : observers) {
            if (!observerList.contains(obs))
                this.observerList.add(obs);
        }
    }

    @Override
    public void removeObservers(Observer observer) {
        this.observerList.remove(observer);
    }

    @Override
    public void notifyObservers(Object obj) {
        for (Observer observer : this.observerList)
            observer.update(obj);

    }
}
```

Interface Observer

```
    public interface Observer {
    /**
     * When a observer is notified execute this method
     * @param obj - argument of the method
     */
    void update(Object obj);
}
```

Exemplo de aplicação 

1 - Temos uma classe tipica que controla um "carrinho de compras" : `ShoppingCart` , com as operações de adicionar, e remover produtos.

2 - Com vista a desacoplar o modelo (ShoppingCart) das vistas sobre o mesmo, aplicou-se o padrão Observer.

```
- ConcreteSubject: ShoppingCart
- Observer: ShoppingCartCostView
```

Cada vez que se faz uma alteração ao modelo (adiciona, ou removem produtos do Carrinho de Compras), o Valor Total do Carrinho de Compras é atualizado e mostrado.

ShoopingCart

```
public class ShoppingCart extends Subject {

    private String name;
    private List<Product> products;

    public ShoppingCart(String name) {
        this.name = name;
        products = new ArrayList<>();
    }

    public void addProduct(Product p) {
        products.add(p);
        notifyObservers(this);
    }

    public void removerProduct(Product p) {
        products.remove(p);
        this.notifyObservers(this);
    }
```

ShoopingCartCostView

```
    public class ShoppingCartTotalCostView implements Observer {
    @Override
    public void update(Object arg) {
        if(arg instanceof ShoppingCart) {
            ShoppingCart cart = (ShoppingCart)arg;
            String name = cart.getName();
            System.out.printf("(%s) total cost: %.2f \n", name,  cart.getTotal());
        }
    }

}
```

Client

- É necessário adicionar explicitamente o "observador" ao "subject" :

```
List<Product> productList= generateProductList();
ShoppingCart cart1 = new ShoppingCart("Bruno's Cart");
ShoppingCartTotalCostView costView = new ShoppingCartTotalCostView();

// add Observer to the Subject
cart1.addObservers(costView);

cart1.addProduct(productList.get(0));
cart1.addProduct(productList.get(1));
cart1.addProduct(productList.get(5));
cart1.removerProduct(productList.get(0));
```

- Output:

```
(Bruno's Cart) total cost: 30.00
(Bruno's Cart) total cost: 380.00
(Bruno's Cart) total cost: 680.00
(Bruno's Cart) total cost: 650.00
```

Exercícios

Repositório de apoio à aula no GitHub:
https://github.com/patriciamacedo/ObserverPatternJava

1. Teste o programa fornecido e analise cuidadosamente as classes fornecidas
2. Adicione uma nova classe que assume o papel de `ConcreteObserver` denominada `ShoppingCartListView` que tem como objectivo imprimir com o seguinte formato a lista de compras.
   Lista Ordenada por IDs dos produtos

```
<shopping cart name>
<ordem>: <nome> - <cost> euros
```

2. Faça as modificações necessárias no `main`, para adicionar ao `cart1`, este novo observador.

Exercícios (cont):

3. Adicione uma nova classe que assume o papel de `ConcreteObserver` denominada `ShoppingCartAllert`. Esta classe cada vez que tem o atributo maxValue. Cada vez que este observador é notificado, verifica se o ultimo produto adicionado tem um valor superior ao máximo e imprime uma mensagem com a seguinte configuração:

```
"ALLERT!!! - The product <productName> has exceeded the maximum value configured <maxValue>
```

4. Faça as modificações necessárias no main, para adicionar ao `cart2`, este novo observador.

O padrão Observable em JAVA

Nota: O JAVA descontinuou a classe `Observable` e a Interface `Observer`.

- Em alternativa para resolver o problema que o Padrão Observer responde, propõe-se o uso do mecanismo de Listeners e mais especificamente `PropertyChangeListener`.
  - Leia mais em: https://www.baeldung.com/java-observer-pattern
- Em JavaFX o padrão Observer continua a ser implementado através das `ObservableList` e `ObservableMap`
  - Leia mais em: https://docs.oracle.com/javafx/2/collections/jfxpub-collections.htm

Referências web 📖

- https://www.journaldev.com/1739/observer-design-pattern-in-java
- http://www.javaworld.com/jw-09-1998/jw-09-techniques.html