

---

---

# Observer

— Padrões de Projeto —  
Dra. Alana Moraes

---

---

# Sumário

- O que já foi visto?
  - Padrões Comportamentais: Template Method
- Hoje:
  - Observer

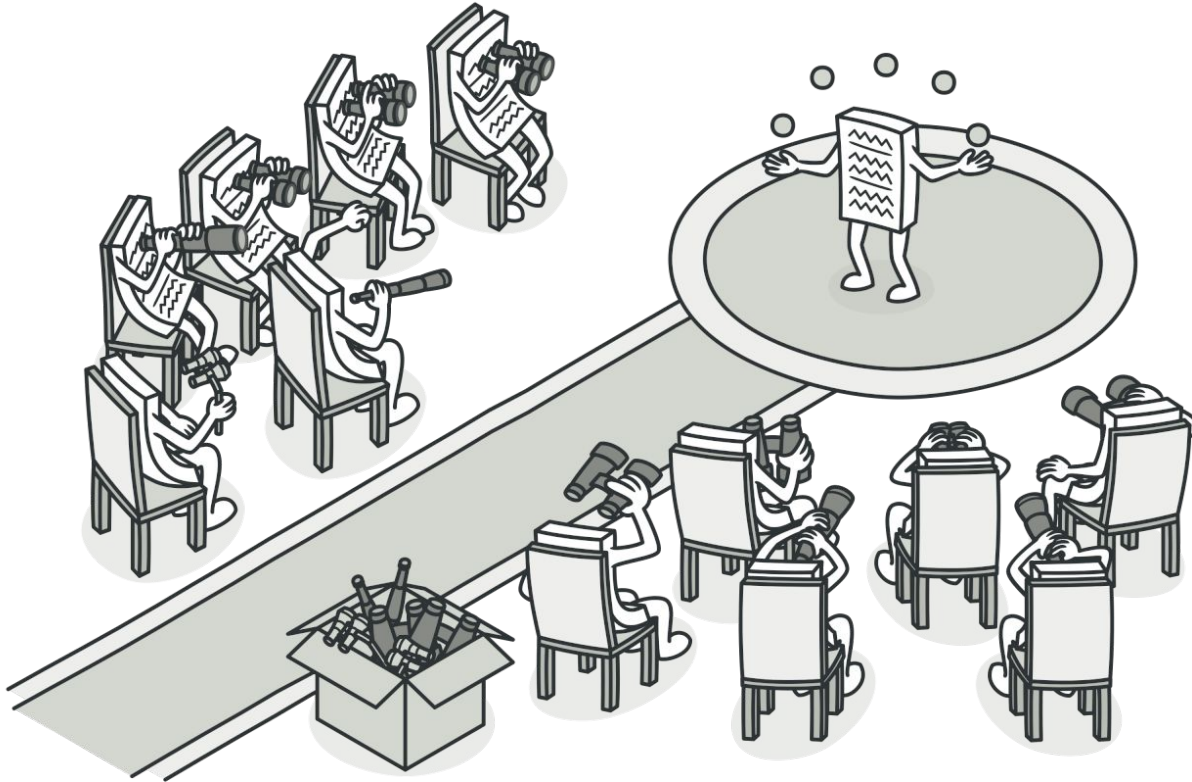
# Observer

“DEFINIR UMA DEPENDÊNCIA UM-PARA-MUITOS ENTRE OBJETOS PARA QUE QUANDO UM OBJETO MUDAR DE ESTADO, TODOS OS SEUS DEPENDENTES SEJAM NOTIFICADOS E ATUALIZADOS AUTOMATICAMENTE”.

# Observer

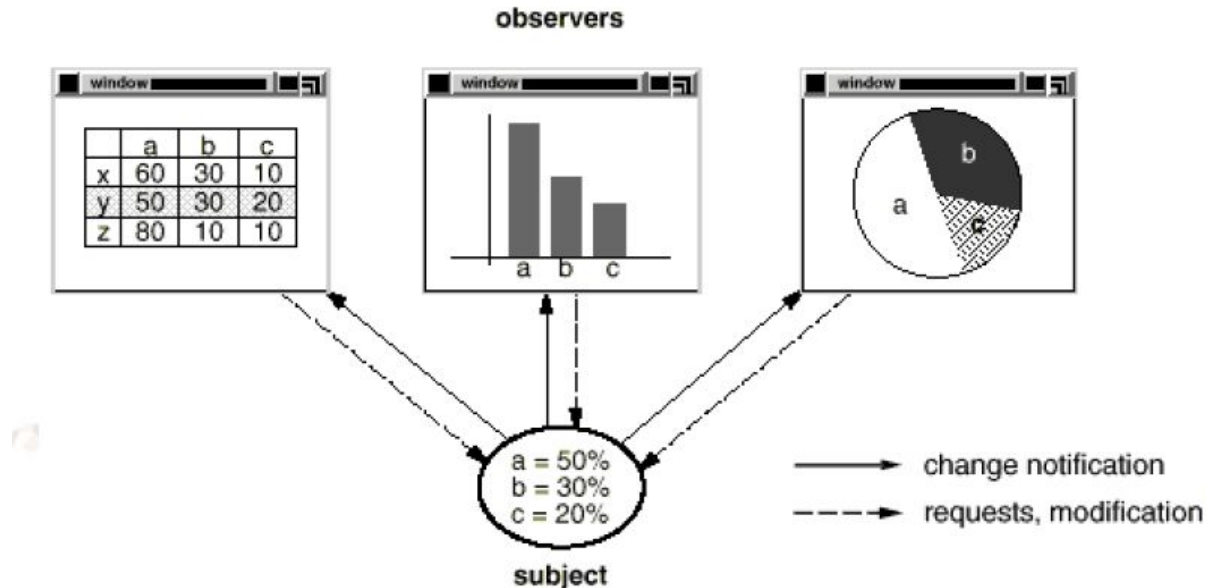
- Desacopla um objeto do conhecimento de que outros objetos que dependem dele.
- Também conhecido como:
  - Dependents, Publish-Subscribe.
- Como garantir que objetos que dependem de outro objeto fiquem em dia com mudanças naquele objeto?
  - Como fazer com que os observadores tomem conhecimento do objeto de interesse?
  - Como fazer com que o objeto de interesse atualize os observadores quando seu estado mudar?

# Observer - Problema



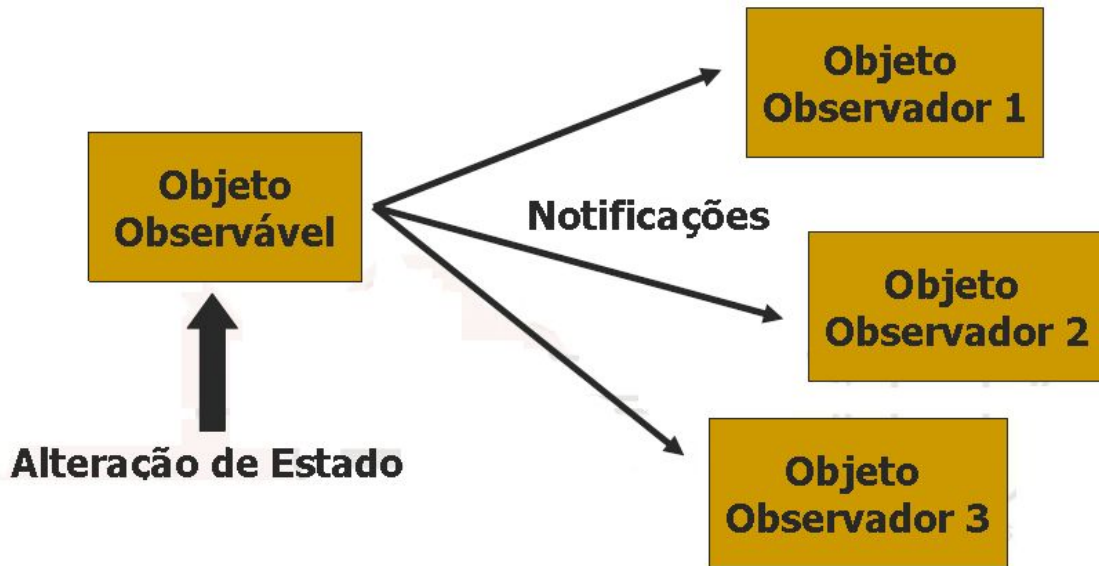
# Observer - Problema

O efeito colateral de distribuir responsabilidade entre objetos é manter a consistência entre eles.

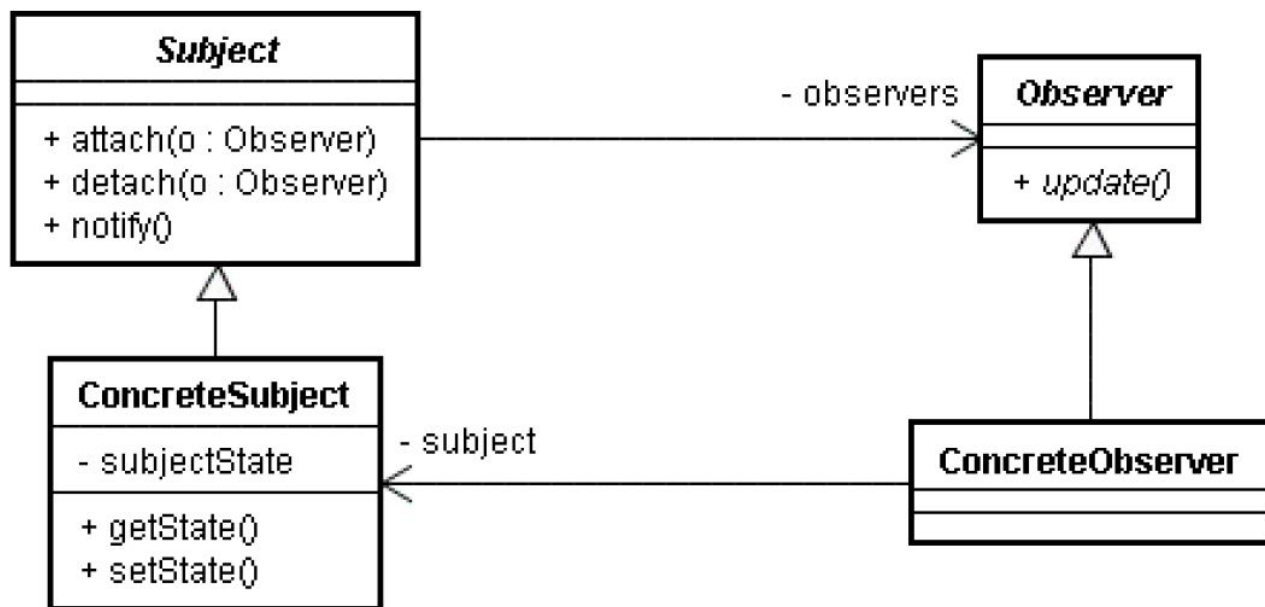


# Observer - Problema

Objeto observável registra os observadores e os notifica sobre qualquer alteração



# Observer - Solução





# Observer - Quando usar?

- Uma abstração possui dois aspectos e é necessário separá-los em dois objetos para variá-los;
- Alterações num objeto requerem atualizações em vários outros objetos não-determinados;
- Um objeto precisa notificar sobre alterações em outros objetos que, a princípio, ele não conhece.

# Observer - Vantagens e Desvantagens

## Flexibilidade:

- Observável e observadores podem ser quaisquer objetos;
- Acoplamento fraco entre os objetos: não sabem a classe concreta uns dos outros;
- É feito broadcast da notificação para todos, independente de quantos;
- Observadores podem ser observáveis de outros, propagando em cascata.

# Observer em Java - (Prefira criar na mão)

<b>java.util.Observable</b>
<ul style="list-style-type: none"><li>+ addObserver(o : Observer) : void</li><li>+ clearChanged() : void</li><li>+ countObservers() : int</li><li>+ deleteObserver(o : Observer) : void</li><li>+ deleteObservers() : void</li><li>+ hasChanged() : boolean</li><li>+ notifyObservers(arg : Object) : void</li><li>+ setChanged() : void</li></ul>

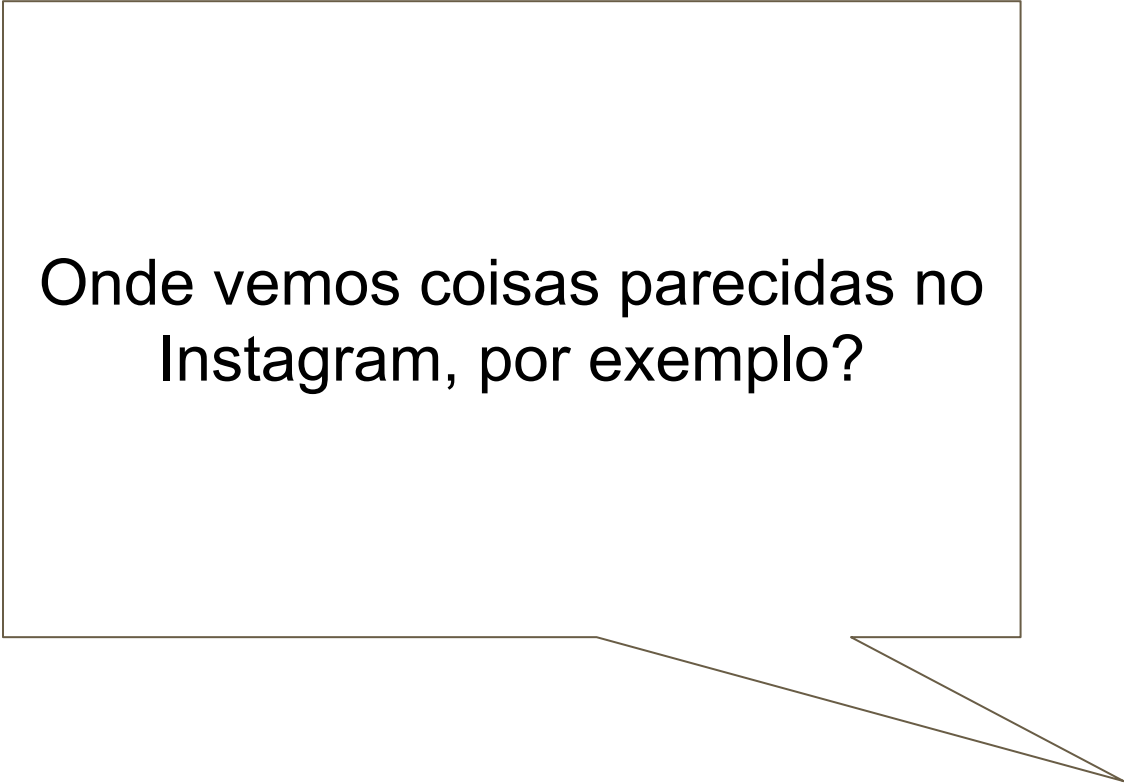
<b>&lt;&lt;interface&gt;&gt; java.util.Observer</b>
<ul style="list-style-type: none"><li>+ <i>update(o : Observable, arg : Object)</i></li></ul>

# Exemplo

Precisamos converter um número para diferentes bases numéricas (hexadecimal, binária e octal).

Além disso, uma demanda de projeto é que essa atualização fosse feita de modo automático.

Como seria o diagrama de classe?



Onde vemos coisas parecidas no  
Instagram, por exemplo?

# Exercício (Observer)



No nosso programa nós teremos uma classe Editora e uma classe Pessoa. Obviamente, Editora é o nosso subject e Pessoa o nosso observer.

Crie um programa que permita uma Editora publicar revistas para seus assinantes.

Como testar?

- Crie quatro objetos do tipo Pessoa, mas inicialmente apenas três deles assinarão a Editora.
- Faça a editora publicar um exemplar (revista). Faça a pessoa que não era assinante se tornar assinante. Publique outra revista.
- Cancele a assinatura de qualquer pessoa e publique a última revista.

# Dúvidas?

[alanamm.prof@gmail.com](mailto:alanamm.prof@gmail.com)