Padrões de Projeto

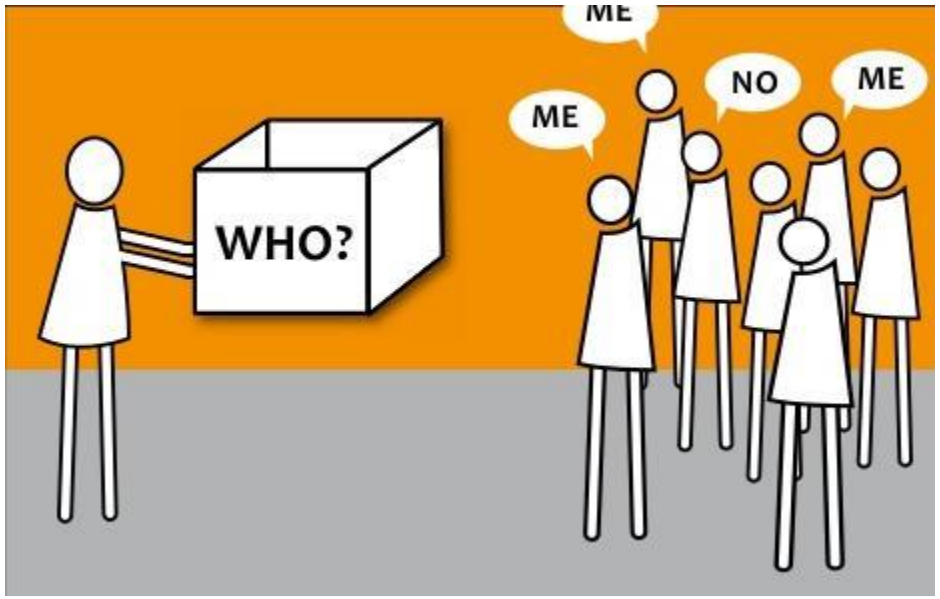
Prof. Adilson Vahldick

Departamento de Engenharia de Software

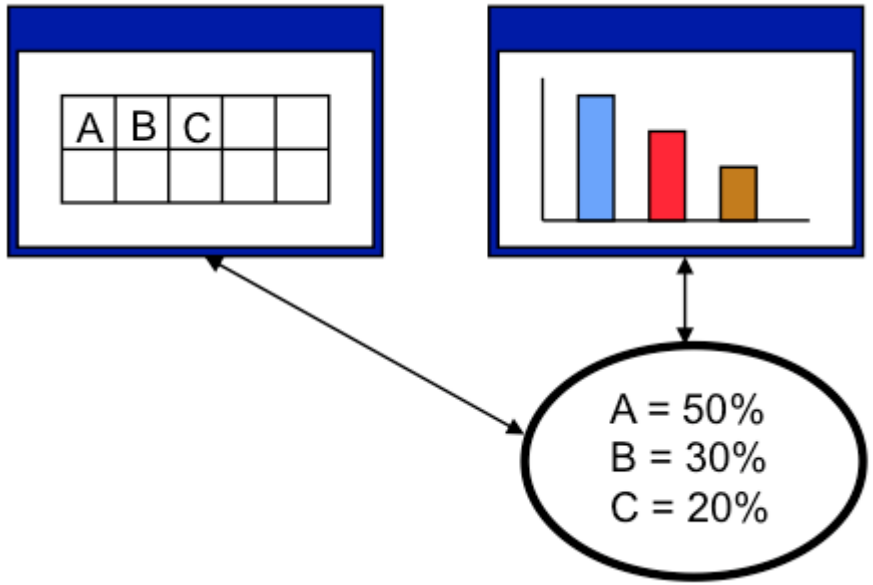
Udesc Ibirama

Objetivos da aula

- Conhecer e aplicar o padrão
  - Observer

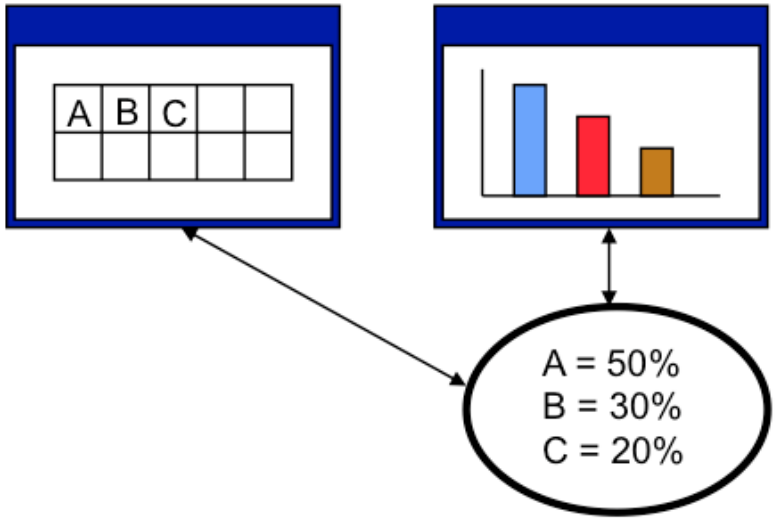


Problema (1)

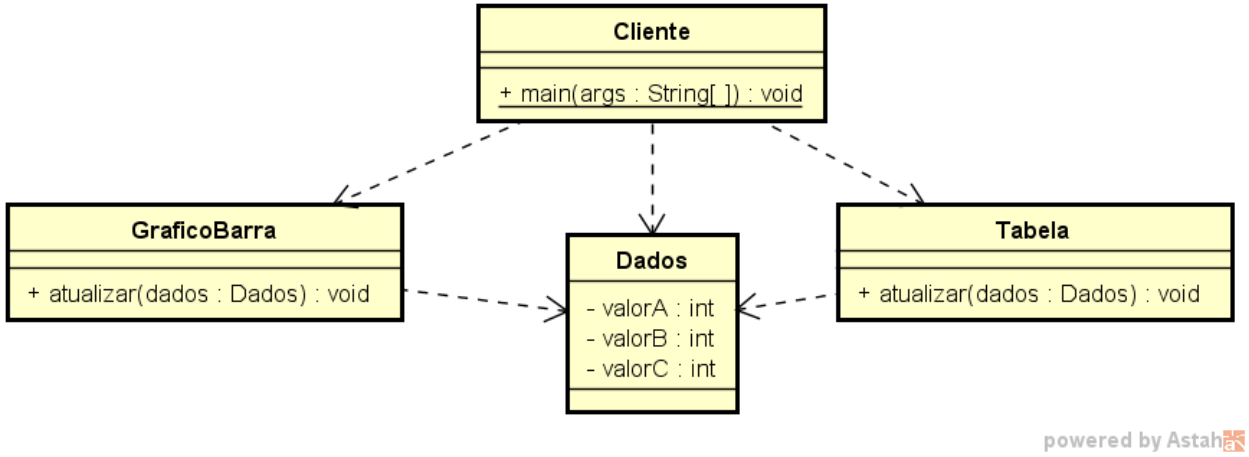


Problema (2)

- Sempre que os dados forem alterados, é preciso atualizar cada uma de suas representações
- O cliente pode exigir novas representações (requisitos mudam !!!)



Problema (3)

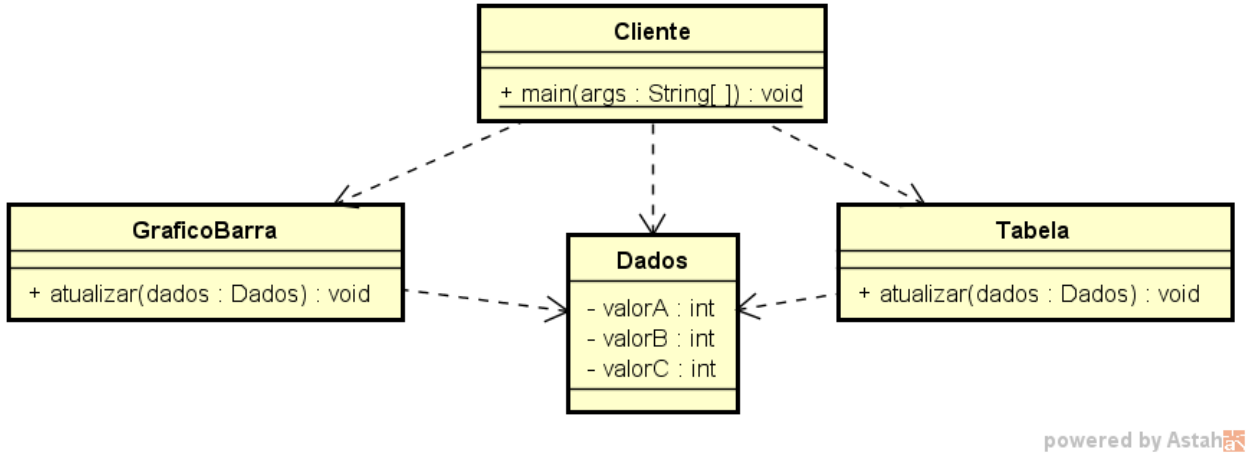



```
Dados d = new Dados(7, 3, 1);
GraficoBarra gb = new GraficoBarra();
Tabela tb = new Tabela();

gb.atualizar(d);
tb.atualizar(d);

d.setValorA(100);
gb.atualizar(d);
tb.atualizar(d);
```

Objetivos de Qualquer Solução OO

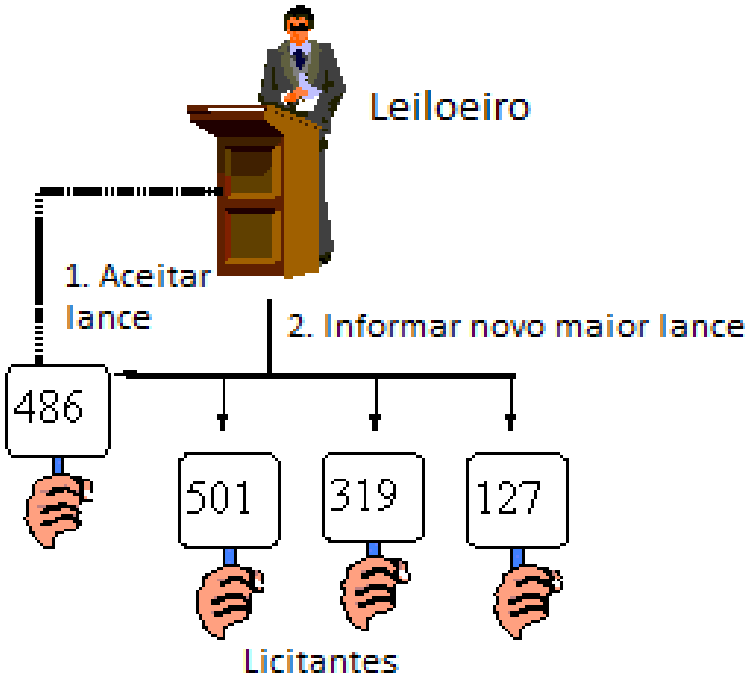
... úteis para aplicação de Observer

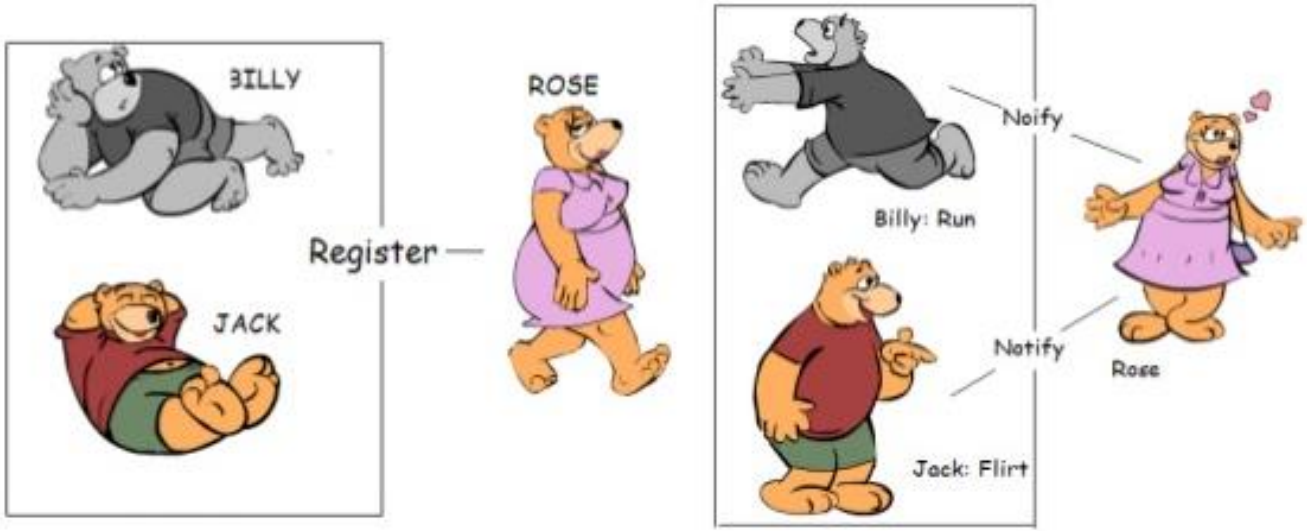
- Programe para uma interface, não para uma implementação
- Baixar o **acoplamento**



Observer (1)

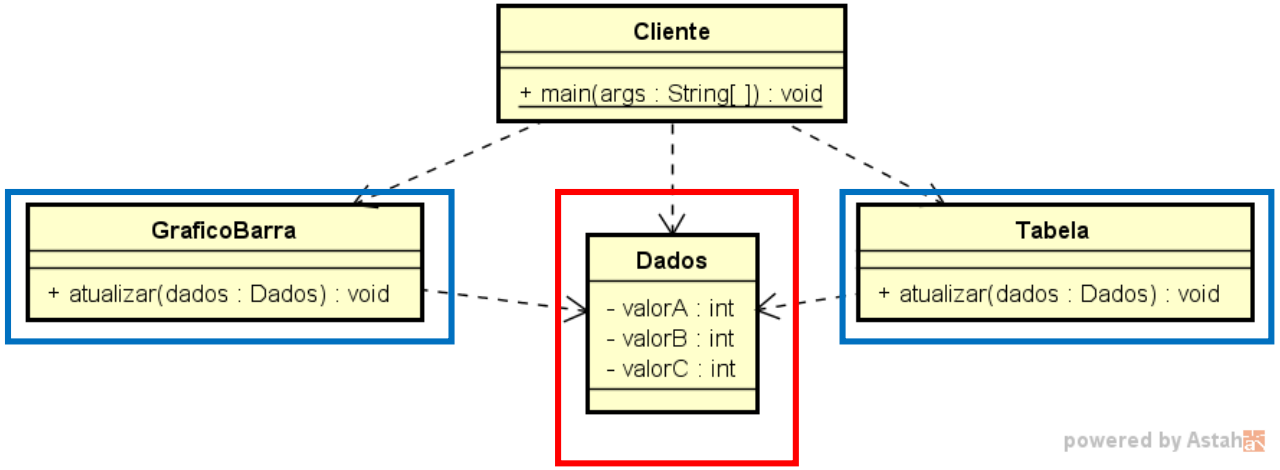
- **Observer:** definir uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente.
- Os dependentes podem pertencer a diferentes tipos.



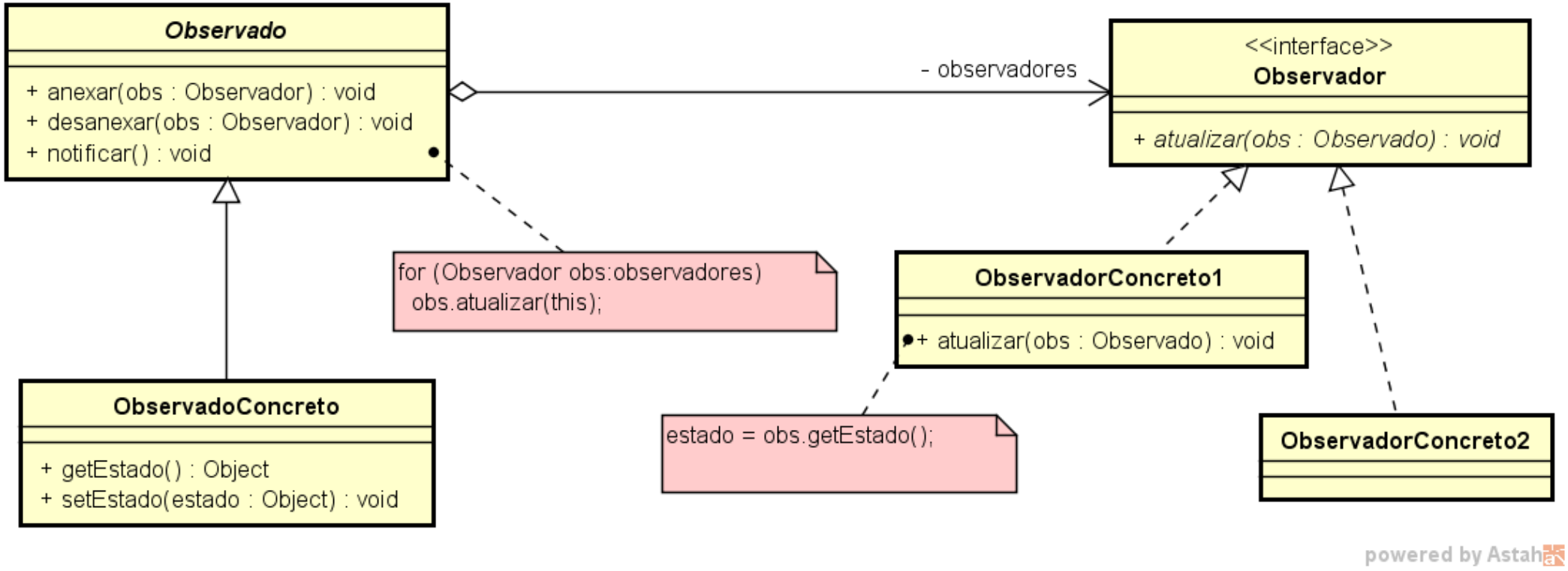


Observer (2)

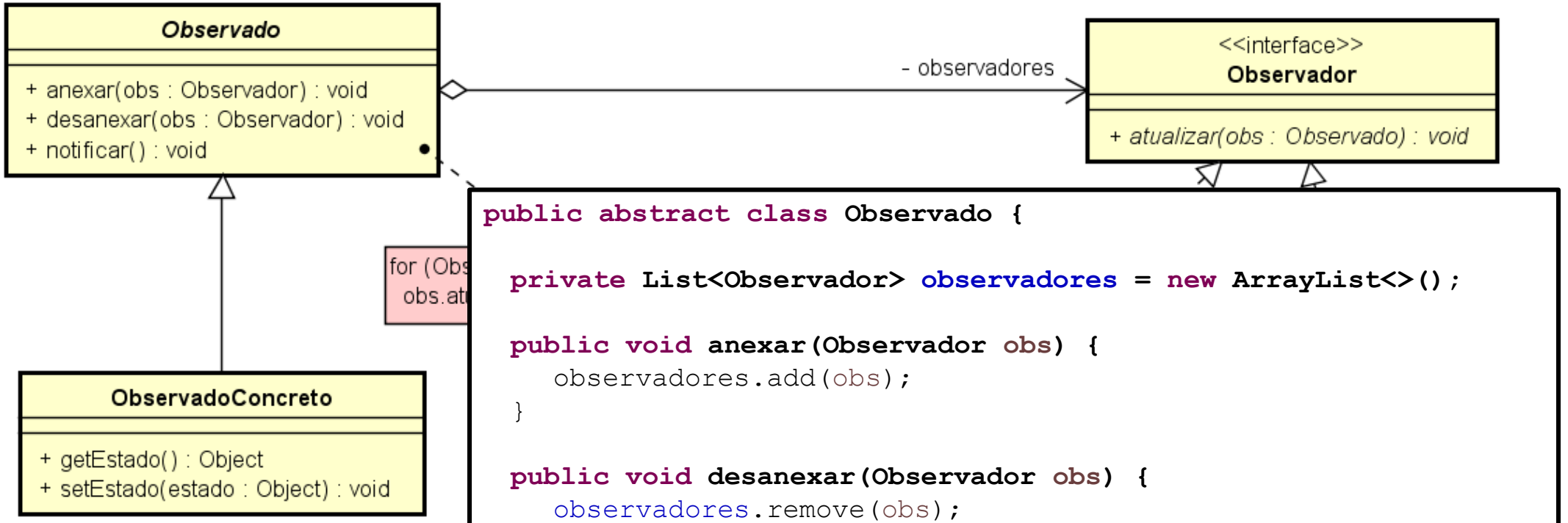
- **Observer:** definir uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente.
- Os dependentes podem pertencer a diferentes tipos.



Observer (3)


Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado( );
ObservadorConcreto2
powered by Astah

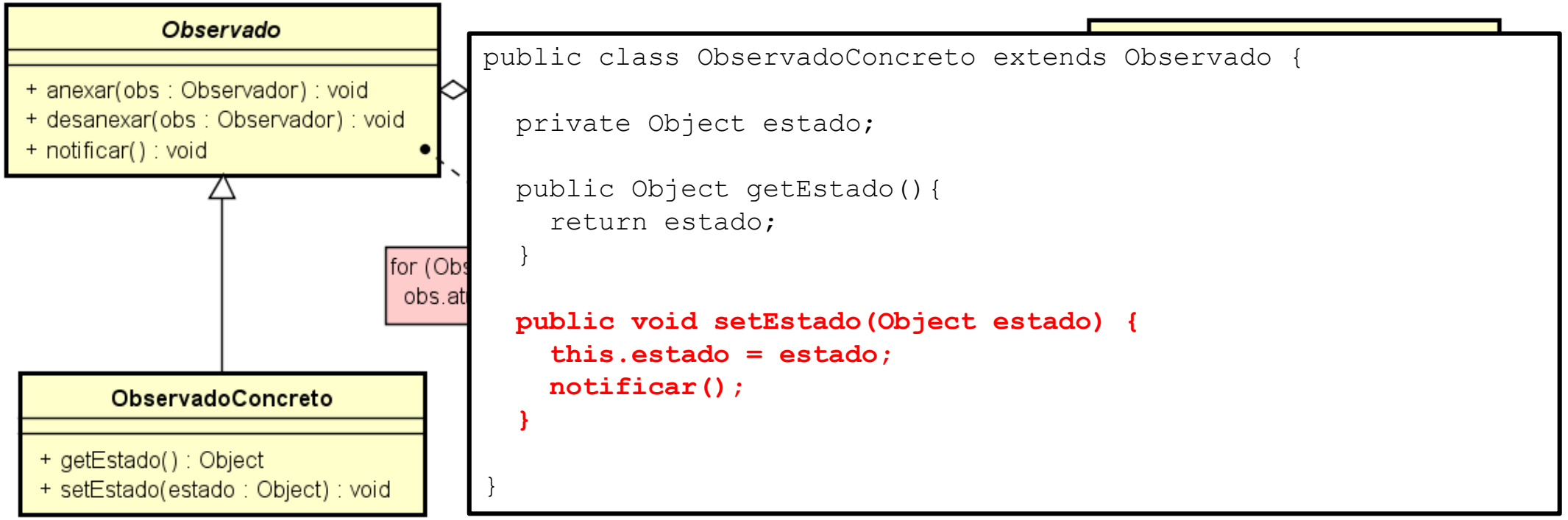
Observer (4)



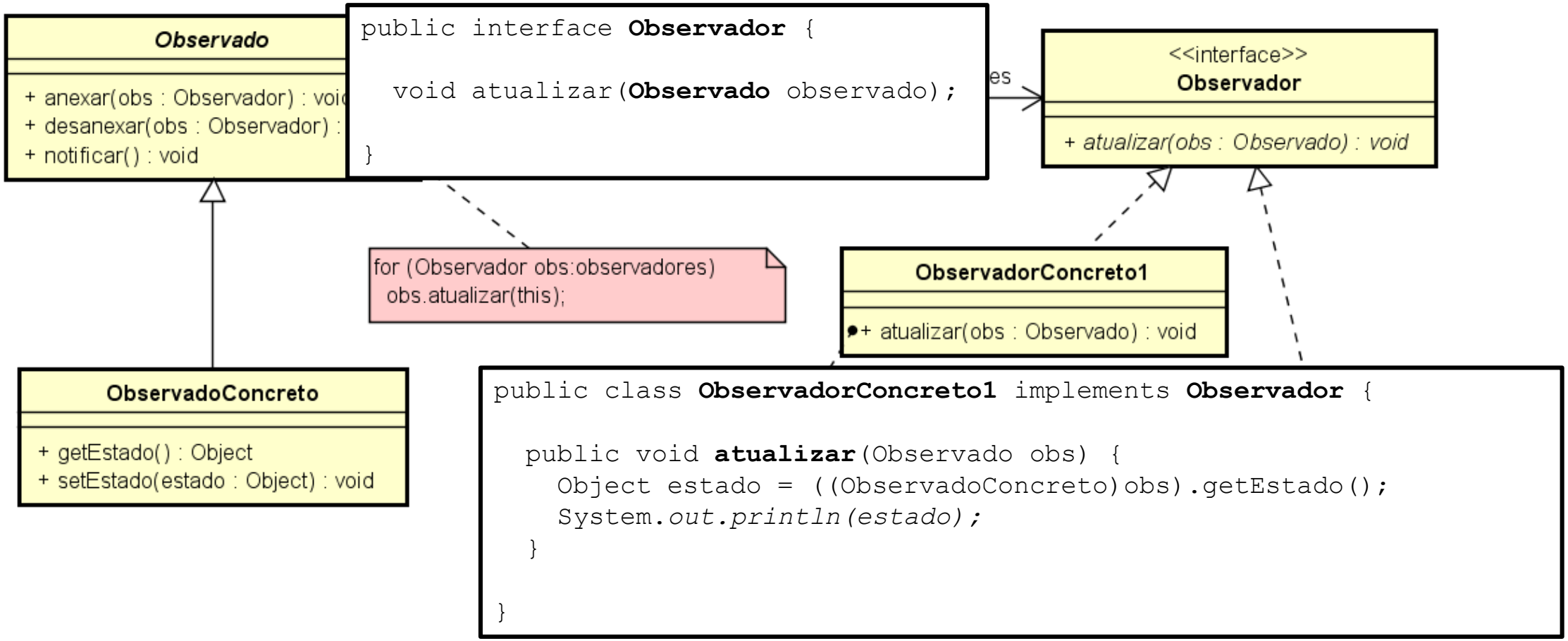

```
public abstract class Observado {

  private List<Observador> observadores = new ArrayList<>();

  public void anexar(Observador obs) {
     observadores.add(obs);
  }

  public void desanexar(Observador obs) {
     observadores.remove(obs);
  }

  public void notificar() {
     for (Observador obs:observadores)
       obs.atualizar(this);
  }

}
```

ick

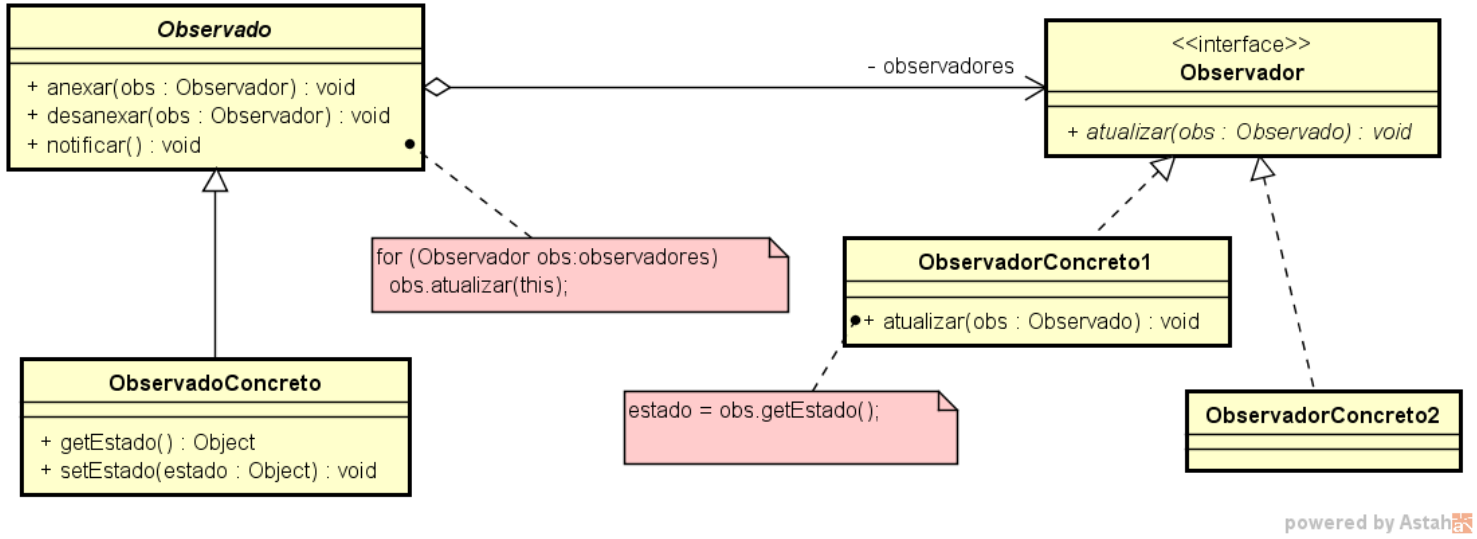
Observer (5)


Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
for (Obs
obs.at
public class ObservadoConcreto extends Observado {
private Object estado;
public Object getEstado(){
return estado;
}
public void setEstado(Object estado) {
this.estado = estado;
notificar();
}
}
powered by Astah

Observer (6)


Observado
+ anexar(obs : Observador) : voi
+ desanexar(obs : Observador) :
+ notificar() : void
public interface Observador {
void atualizar(Observado observado);
}
es
<<interface>>
Observador
+ atualizar(obs : Observado) : void
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado() : Object
+ setEstado(estado : Object) : void
public class ObservadorConcreto1 implements Observador {
public void atualizar(Observado obs) {
Object estado = ((ObservadoConcreto)obs).getEstado();
System.out.println(estado);
}
}

Observer (7)



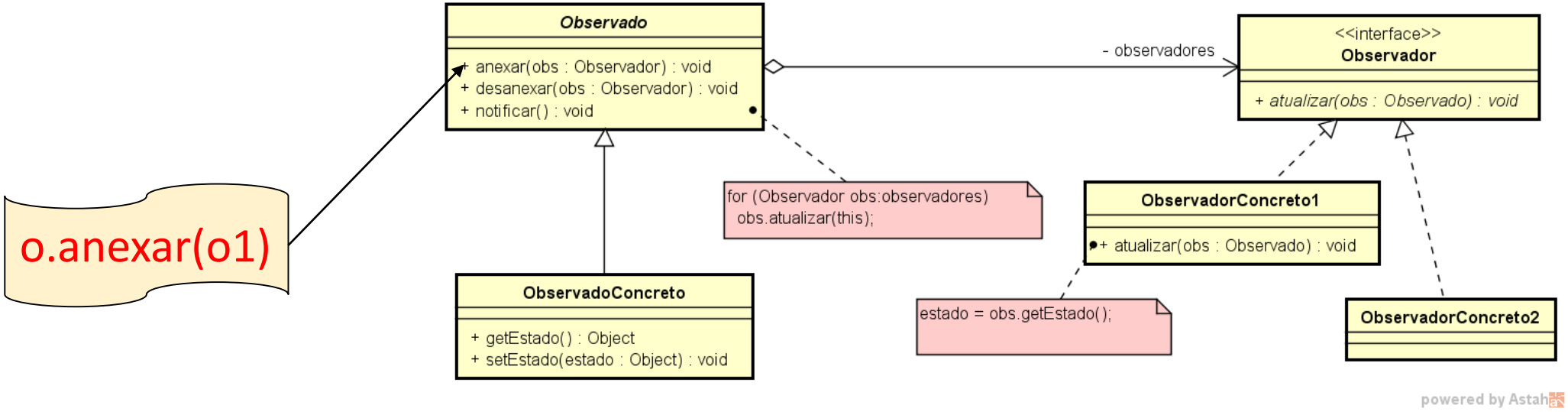
o:ObservadoConcreto

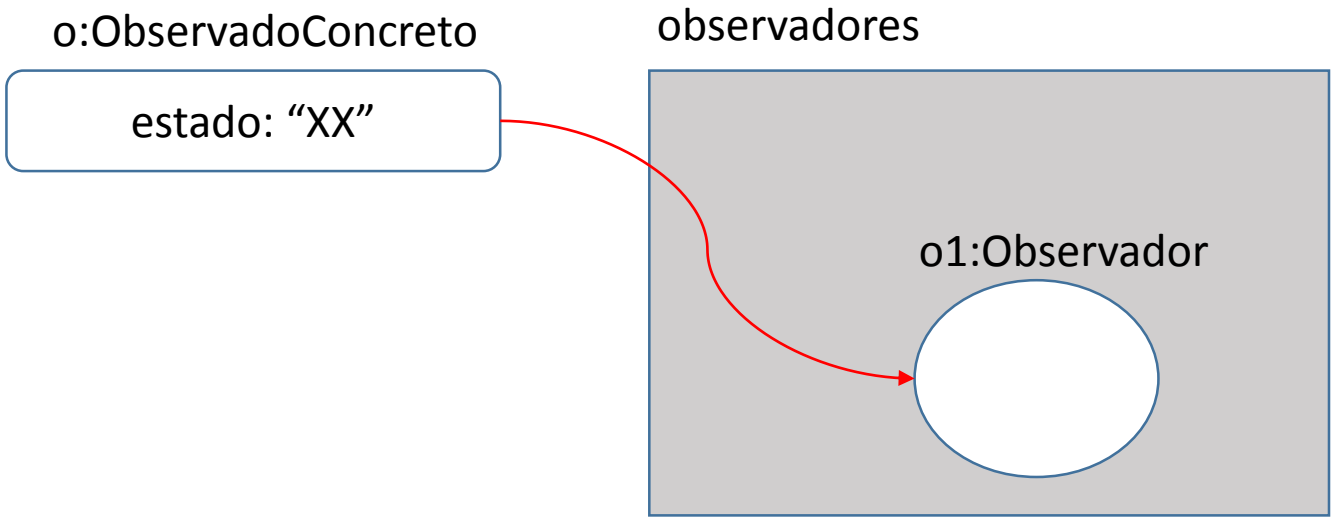
estado: “XX”

o2:Observador

o1:Observador

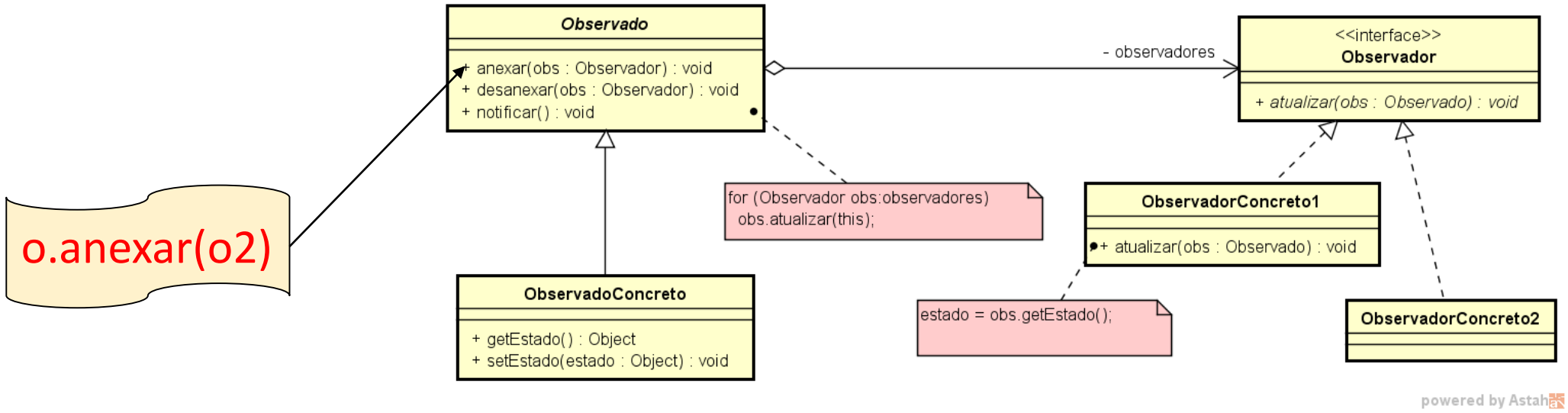
Observer (8)

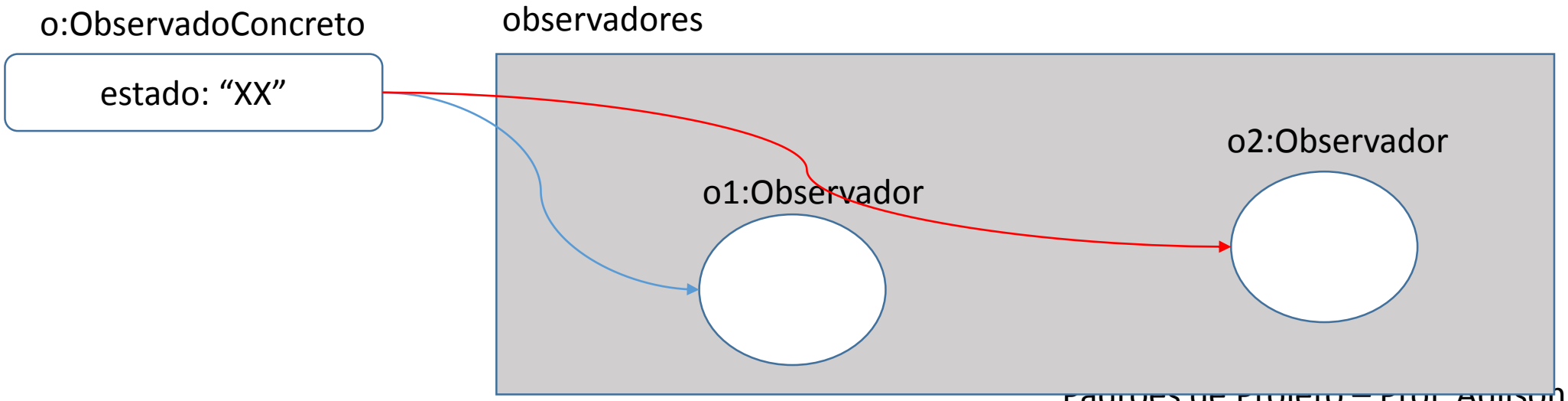

Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
o.anexar(o1)
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado();
ObservadorConcreto2
powered by Astah


o:ObservadoConcreto
estado: “XX”
observadores
o1:Observador

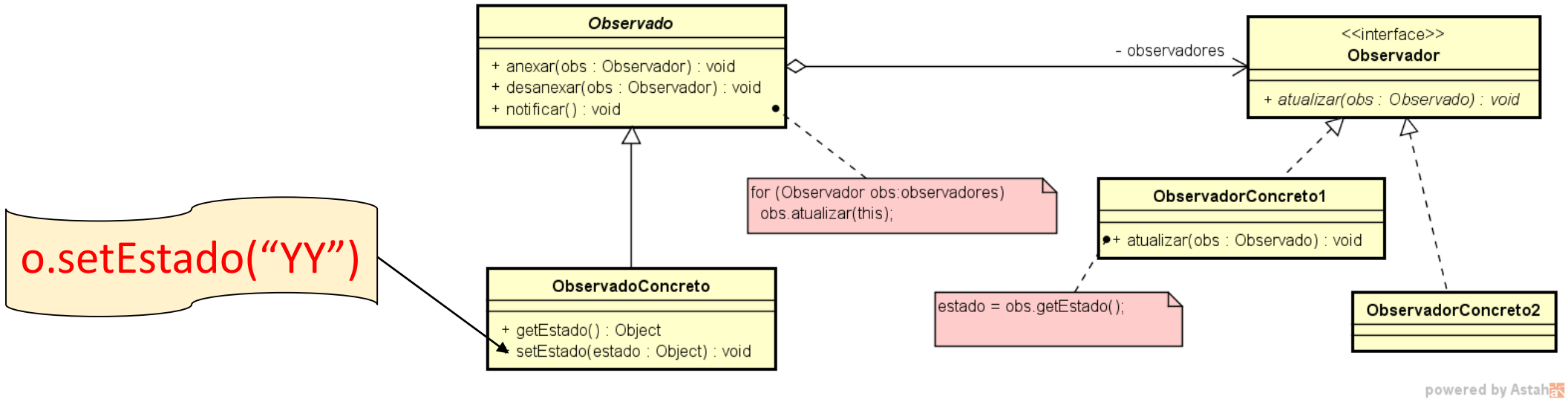

o2:Observador

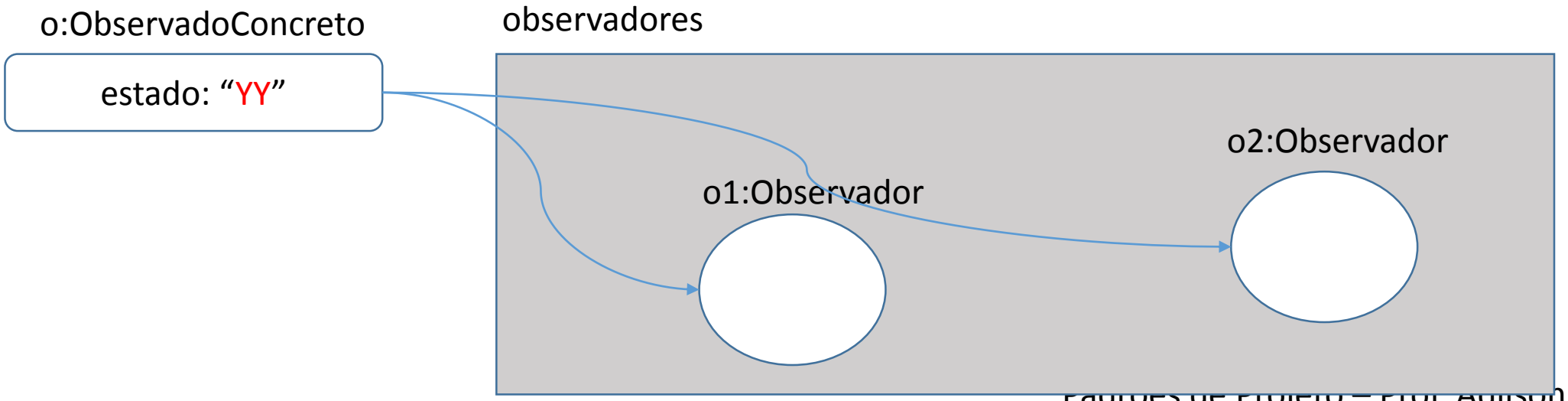
Observer (8)


Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
o.anexar(o2)
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado();
ObservadorConcreto2
powered by Astah

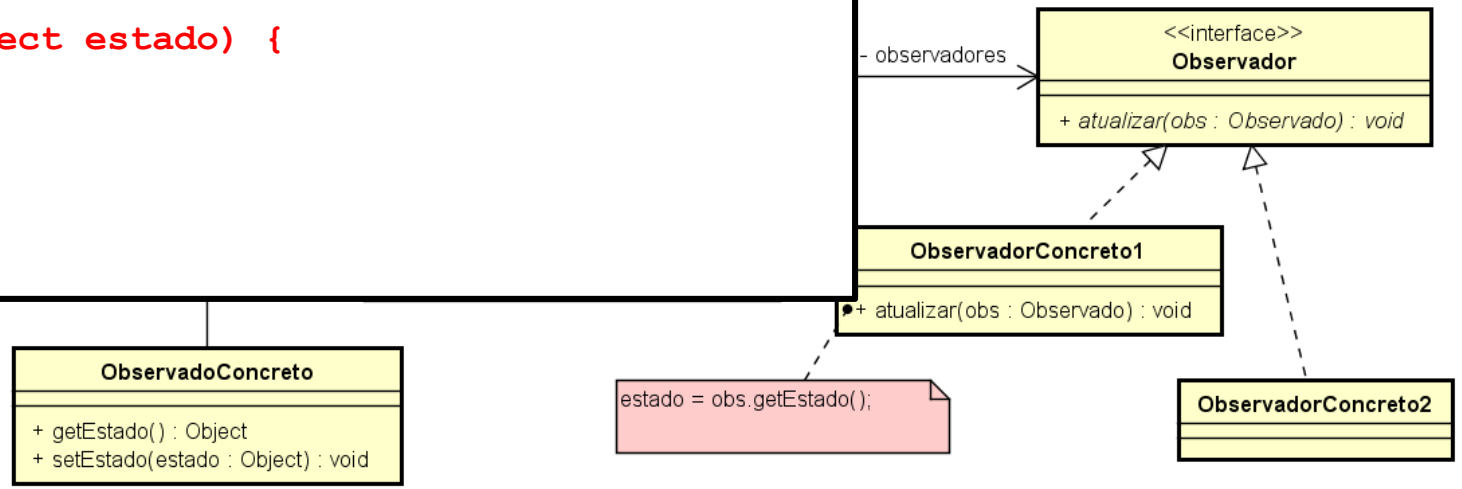

o:ObservadoConcreto
estado: “XX”
observadores
o1:Observador
o2:Observador

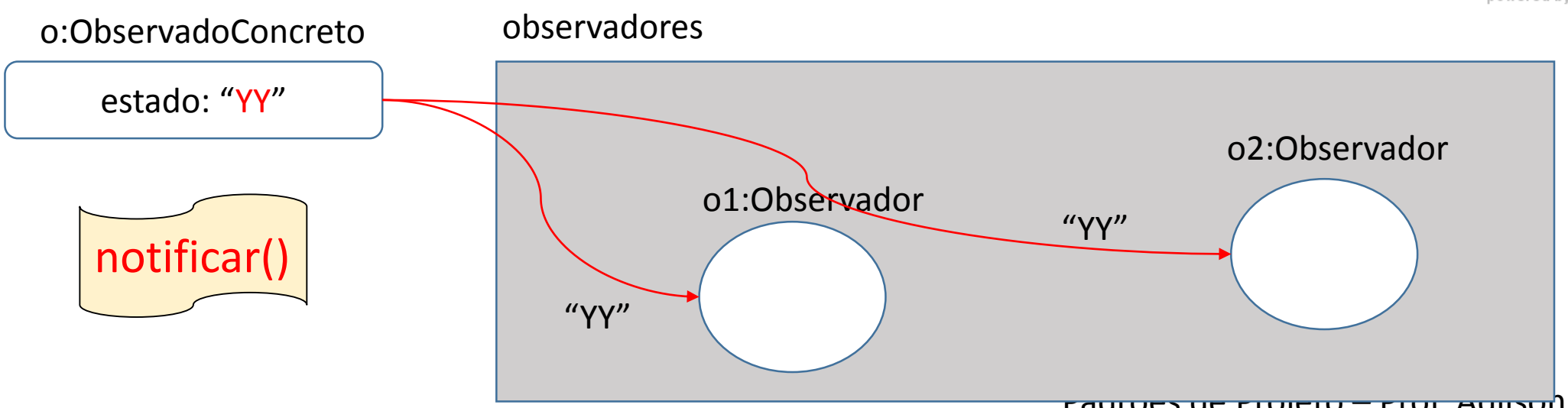
Observer (9)


Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
o.setEstado("YY")
ObservadoConcreto
+ getEstado( ) : Object
setEstado(estado : Object) : void
estado = obs.getEstado( );
ObservadorConcreto2
powered by Astah

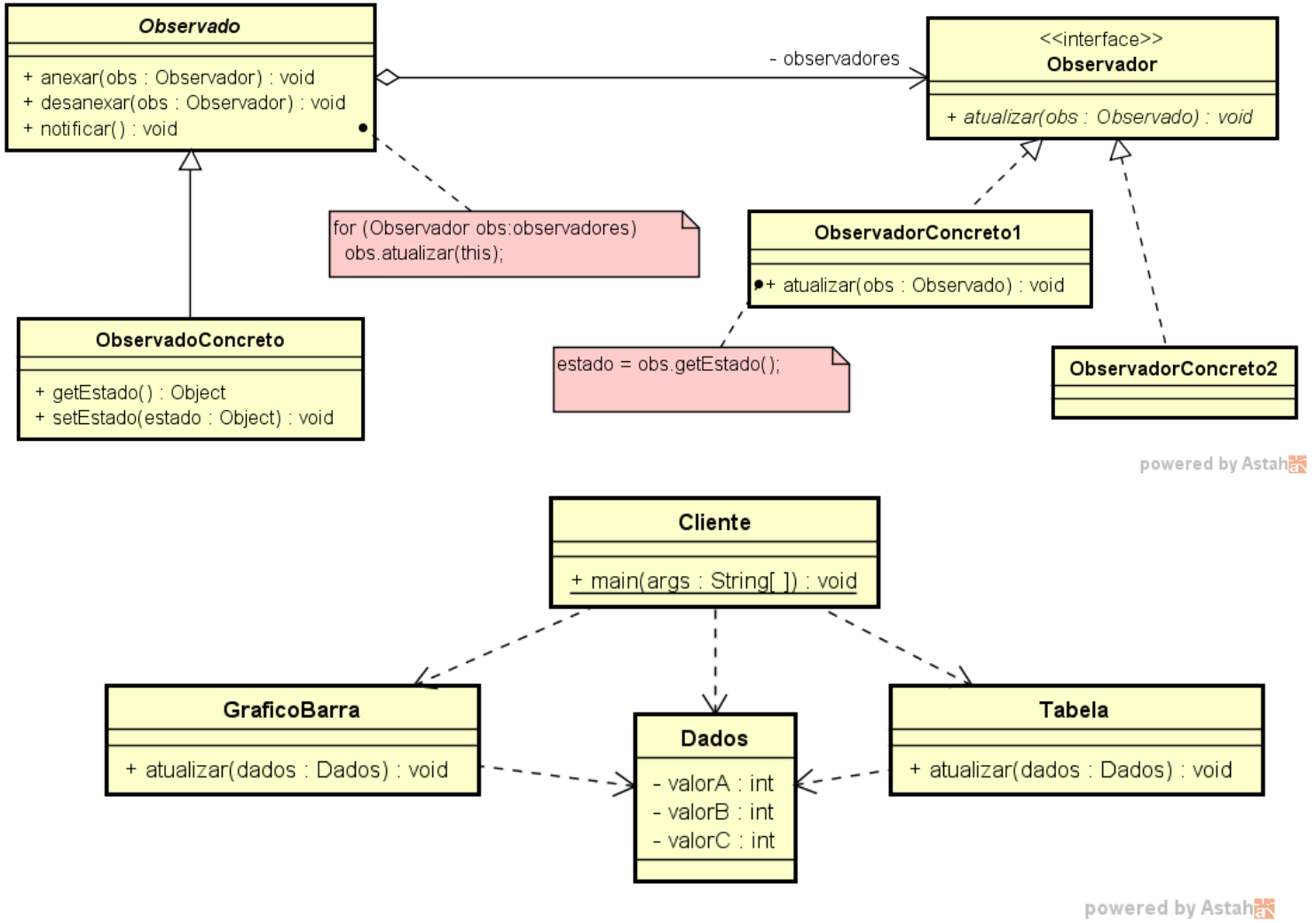

o:ObservadoConcreto
estado: "YY"
observadores
o1:Observador
o2:Observador

```
public class ObservadoConcreto extends Observado {

...

  public void setEstado(Object estado) {
    this.estado = estado;
    notificar();
  }

}
```

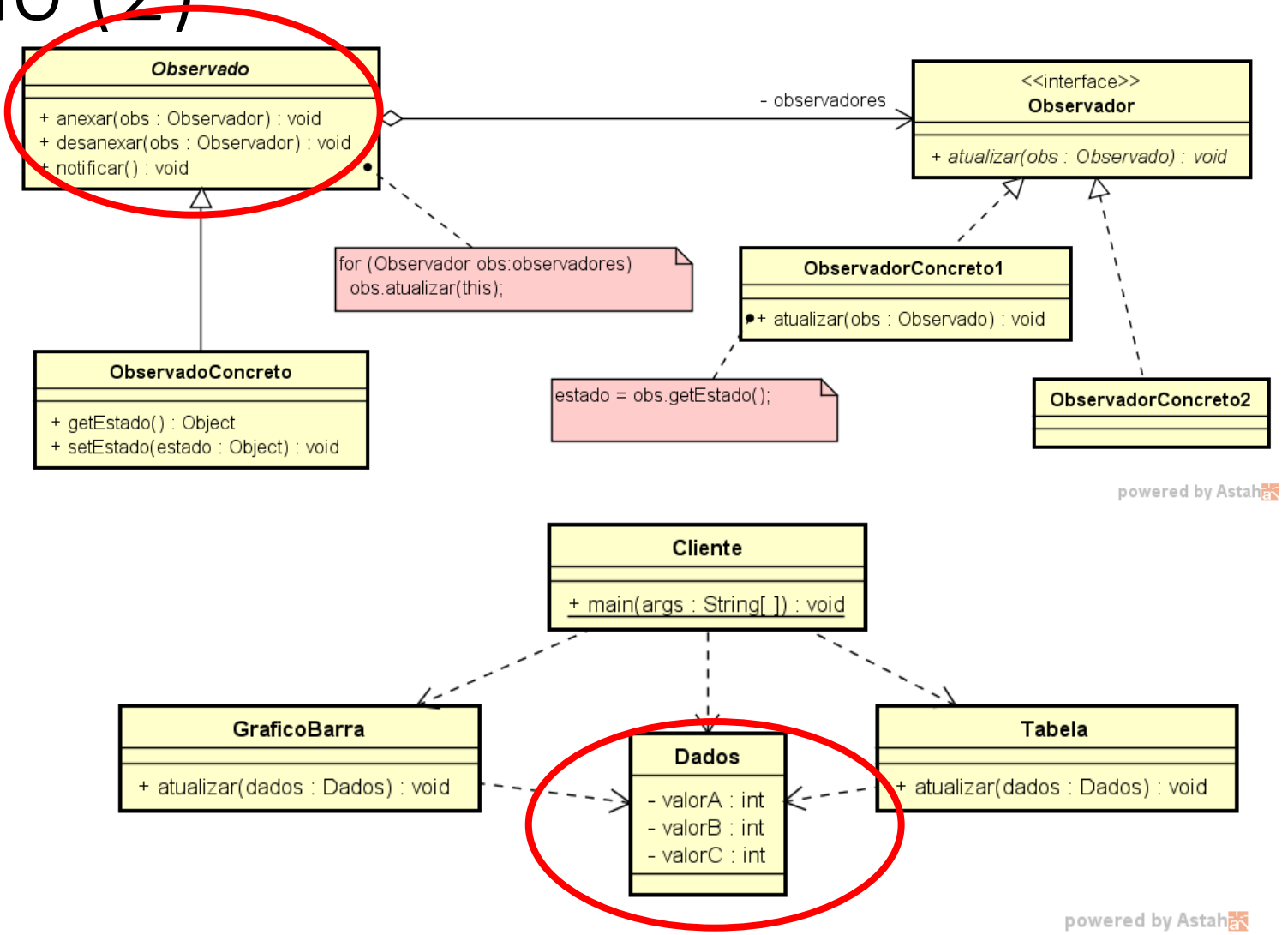




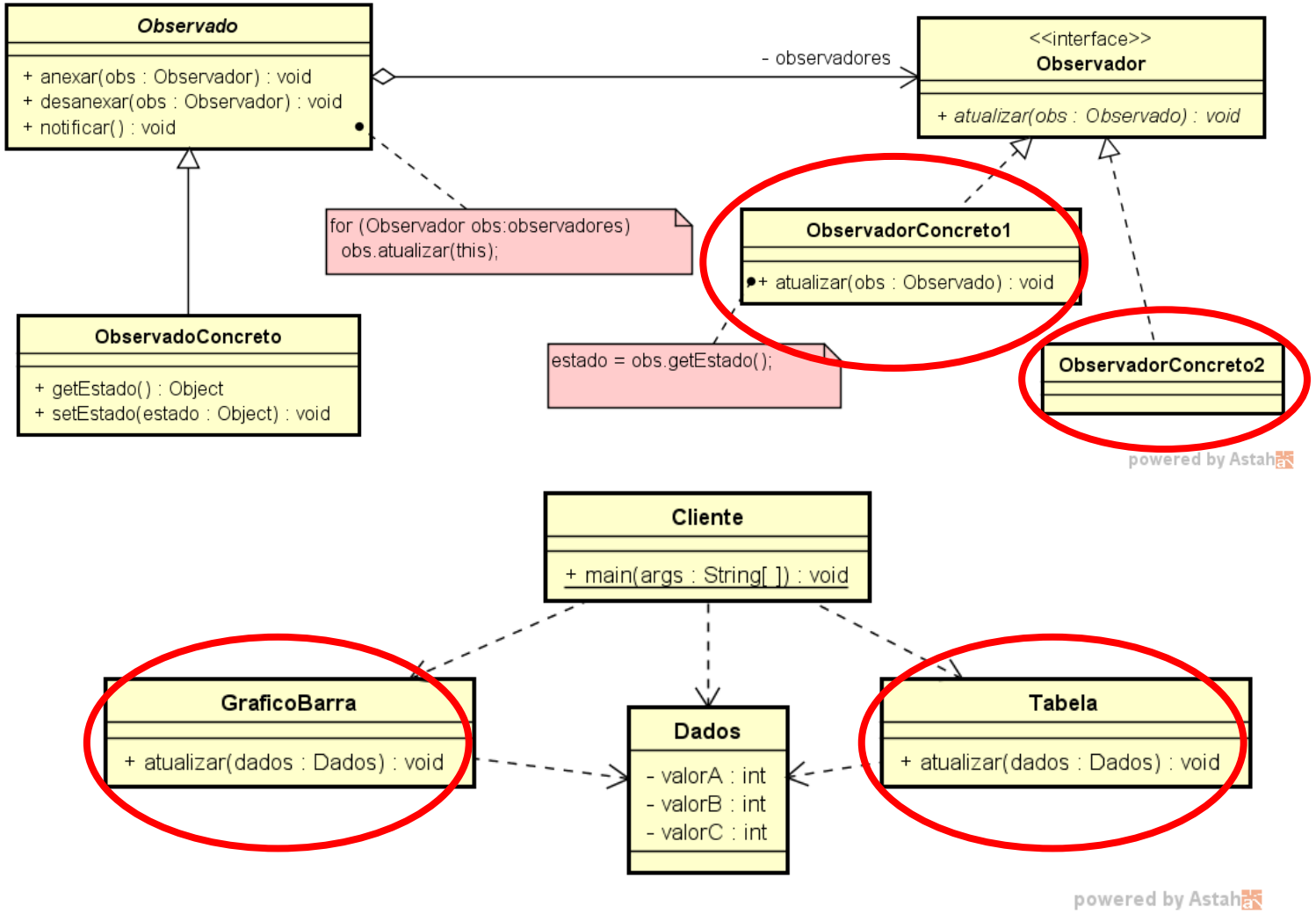
Solução (1)


Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado();
ObservadorConcreto2
powered by Astah
Cliente
+ main(args : String[ ]) : void
GraficoBarra
+ atualizar(dados : Dados) : void
Dados
- valorA : int
- valorB : int
- valorC : int
Tabela
+ atualizar(dados : Dados) : void
powered by Astah

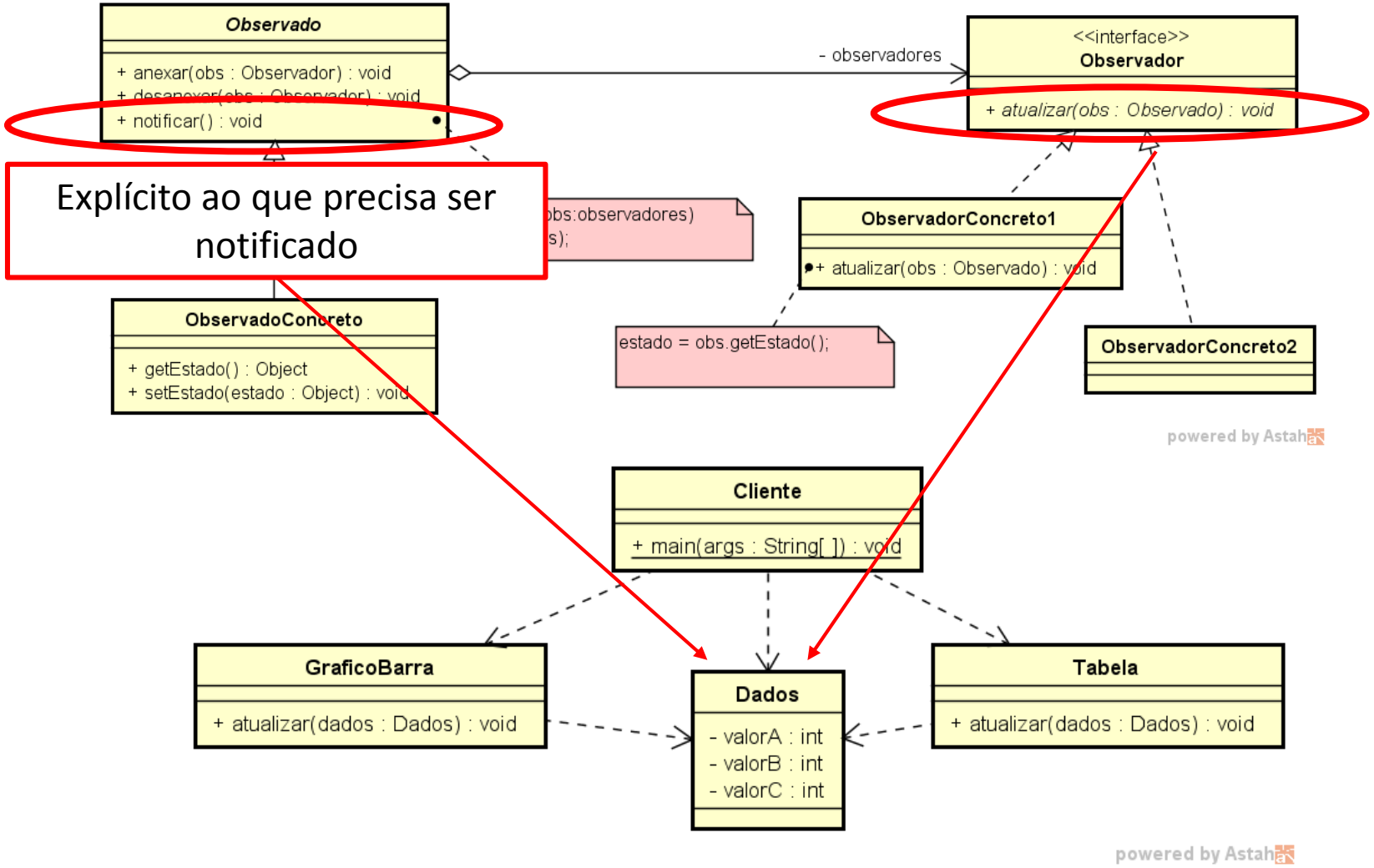
Solução (2)


Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado();
ObservadorConcreto2
powered by Astah
Cliente
+ main(args : String[ ]) : void
GraficoBarra
+ atualizar(dados : Dados) : void
Dados
- valorA : int
- valorB : int
- valorC : int
Tabela
+ atualizar(dados : Dados) : void
powered by Astah

Solução (3)

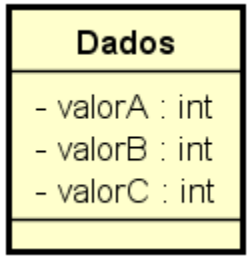

Observado
+ anexar(obs : Observador) : void
+ desanexar(obs : Observador) : void
+ notificar( ) : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
for (Observador obs:observadores)
obs.atualizar(this);
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado( ) : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado();
ObservadorConcreto2
powered by Astah
Cliente
+ main(args : String[ ]) : void
GraficoBarra
+ atualizar(dados : Dados) : void
Dados
- valorA : int
- valorB : int
- valorC : int
Tabela
+ atualizar(dados : Dados) : void
powered by Astah

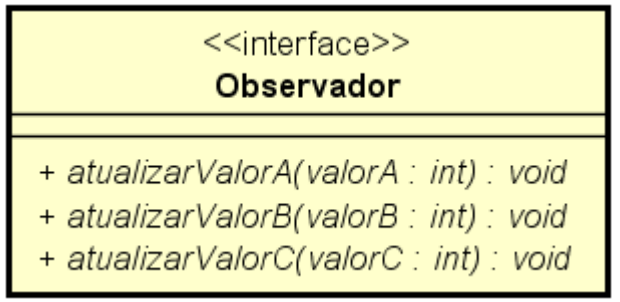
Solução (4)


Observado
+ anexar(obs : Observador) : void
+ notificar() : void
- observadores
<<interface>>
Observador
+ atualizar(obs : Observado) : void
Explícito ao que precisa ser notificado
ObservadorConcreto1
+ atualizar(obs : Observado) : void
ObservadoConcreto
+ getEstado() : Object
+ setEstado(estado : Object) : void
estado = obs.getEstado();
ObservadorConcreto2
powered by Astah
Cliente
+ main(args : String[ ]) : void
GraficoBarra
+ atualizar(dados : Dados) : void
Dados
- valorA : int
- valorB : int
- valorC : int
Tabela
+ atualizar(dados : Dados) : void
powered by Astah

Solução (5)

1. Identifique o que precisa ser notificado do objeto observado. Crie uma interface com métodos suficientes para a notificação de cada mudança a ser observada

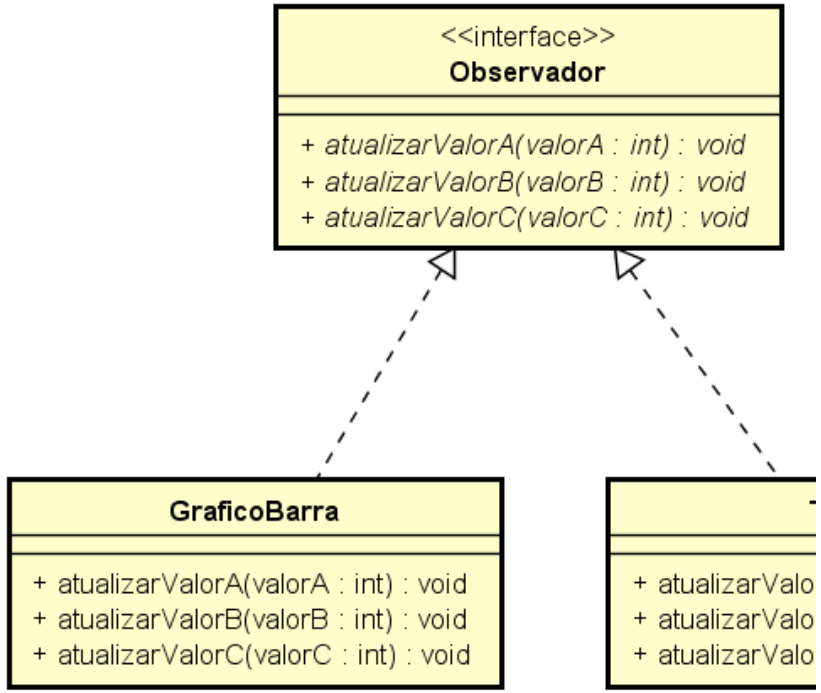





```
public interface Observador {

  void atualizarValorA(int valorA);
  void atualizarValorB(int valorB);
  void atualizarValorC(int valorC);

}
```

Solução (6)

2. Aplique a interface observadora em cada classe receptora.

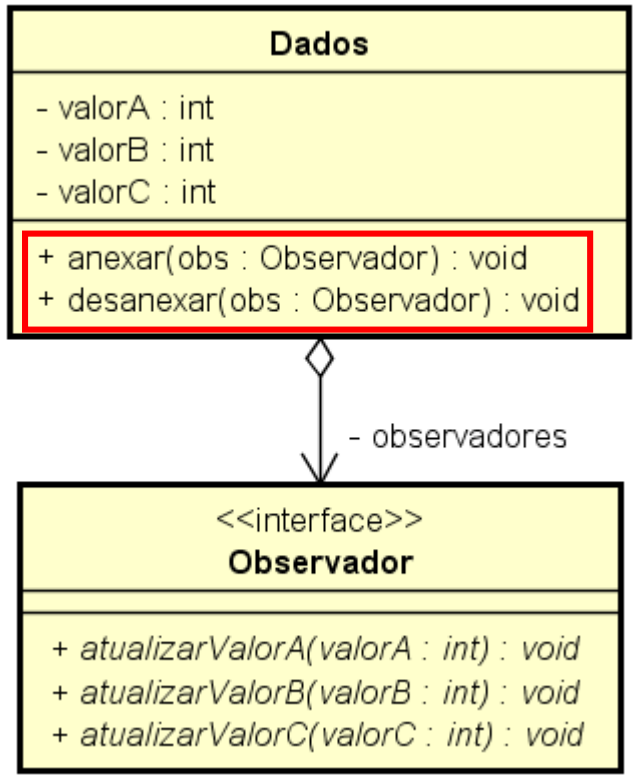



```
public class GraficoBarra implements Observador {

    private String barraA, barraB, barraC;

    private void desenhar() {
        System.out.println("Barras:\n Valor A: " + barraA + "\nValor B: "
                + barraB + "\nValor C: " + barraC);
    }

    @Override
    public void atualizarValorA(int valorA) {
      barraA = "";
      for (int i = 0; i < valorA; i++) {
            barraA += '=';
      }
      desenhar();
    }
    ...

}
```

Solução (7)

3. Refatore a classe observada para suportar as listas de observadores.

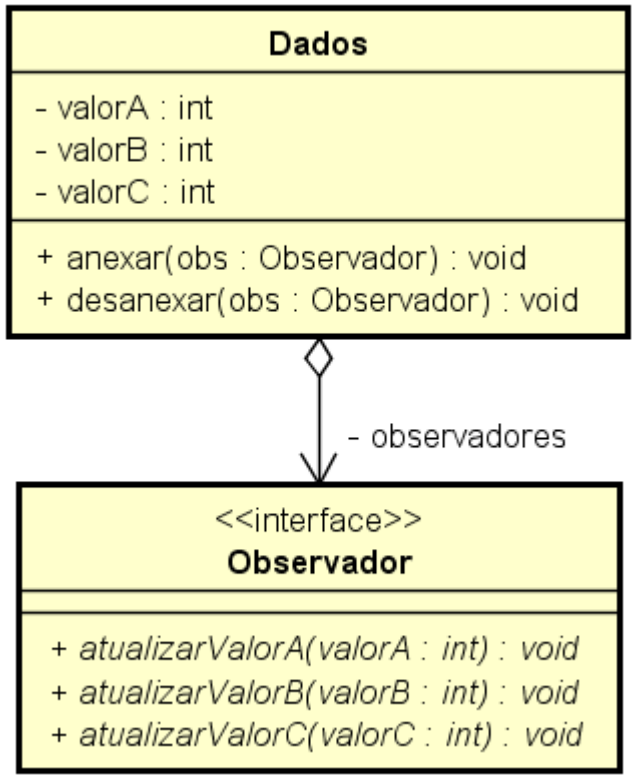
```
public class Dados {

   private int valorA, valorB, valorC;


   private List<Observador> observadores = new ArrayList<>();

   public void anexar(Observador obs) {  this.observadores.add(obs);    }

   public void desanexar(Observador obs) {
    this.observadores.remove(obs);
   }
   ...
}
```



Solução (8)

4. Na classe observada adicione os métodos para notificar os observadores.

```
public class Dados {

   private int valorA, valorB, valorC;

  public void setValorA(int valorA) {
     this.valorA = valorA;
     for (Observador obs:observadores)
        obs.atualizarValorA(valorA);
   }
  public void setValorB(int valorB) {
     this.valorB = valorB;
     for (Observador obs:observadores)
        obs.atualizarValorB(valorB);
  }
  public void setValorC(int valorC) {
     this.valorC = valorC;
     for (Observador obs:observadores)
        obs.atualizarValorC(valorC);
   }
...
}
```



Solução (9)

```
public static void main(String[] args) {
        Dados dados = new Dados(7, 3, 1);
        dados.anexar(new Tabela(dados));
        dados.anexar(new GraficoBarra(dados));

        dados.setValorA(10);
        dados.setValorB(20);
        dados.setValorC(15);
    }
```

Exercícios

- observador1 – completo
- observador2 – falta registrar os observadores
- observador3 – falta aplicar o padrão:
  - Identifique quem são os observadores e quem é o observado
- observador4 – desenvolver do zero (as classes de domínio estão no diagrama)
- observador5 – desenvolver as classes de domínio e aplicar o padrão (a GUI está pronta)

Referências

- FREEMAN, E.; FREEMAN, E. **Use a cabeça**: padrões de projetos. Rio de Janeiro: Alta Books, 2005. Cap 2.

- GAMMA, E. et al. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. pp 274-284.

- KERIEVSKY, J. **Refatoração para padrões**. Porto Alegre: Bookman, 2008. pp 268-278.

- SHALLOWAY, A.; TROTT, J. **Explicando padrões de projeto**: uma nova perspectiva em projeto orientado a objeto. Porto Alegre: Bookman, 2004. Cap 17.