Universidade Federal de Uberlândia
Faculdade de Computação
Programação Orientada a Objetos II
Prof. Fabiano Dorça

Padrão Observer (Observador)

Padrão Observer (Observador)

Problema:

→ Definir uma dependência *um-para-muitos* entre objetos, de forma quando o estado de um objeto em particular (observado) se altera, seus dependentes (observadores) são notificados.

→ Os objetos observadores podem, assim, desencadear as ações necessárias.

Padrão Observer (Observador)

Vantagem:

→ Evita que um ou mais objeto precise monitorar constantemente o estado de outro(s) objeto(s).

Padrão Observer (Observador)

Funcionamento:

→ Quando é criado, cada observador regista-se junto do observado, que mantém uma lista de observadores na sua estrutura interna.

→ Assim, quando o estado do observado é alterado, este envia uma mensagem para cada observador, informando-o do ocorrido. Este então pode reagir em conformidade.

Padrão Observer (Observador)

Participantes:

→ As classes e/ou objetos que participam do padrão são:

Subject

→ Representa uma abstração do sujeito a ser observado.
→ Fornece uma interface para adicionar e remover objetos *Observer*.
→ Conhece os seus observadores. Qualquer número de objetos *Observer* pode observar um *Subject*.

ConcreteSubject

→ Implementa uma classe de sujeitos concretos.
→ Armazena os estados que interessam ao *ConcreteObserver*.
→ Envia uma notificação aos *Observers*, quando o seu estado se altera.

Padrão Observer (Observador)

Observer

→ Define uma interface para os objetos observadores, que devem ser notificados de alterações no *Subject*.

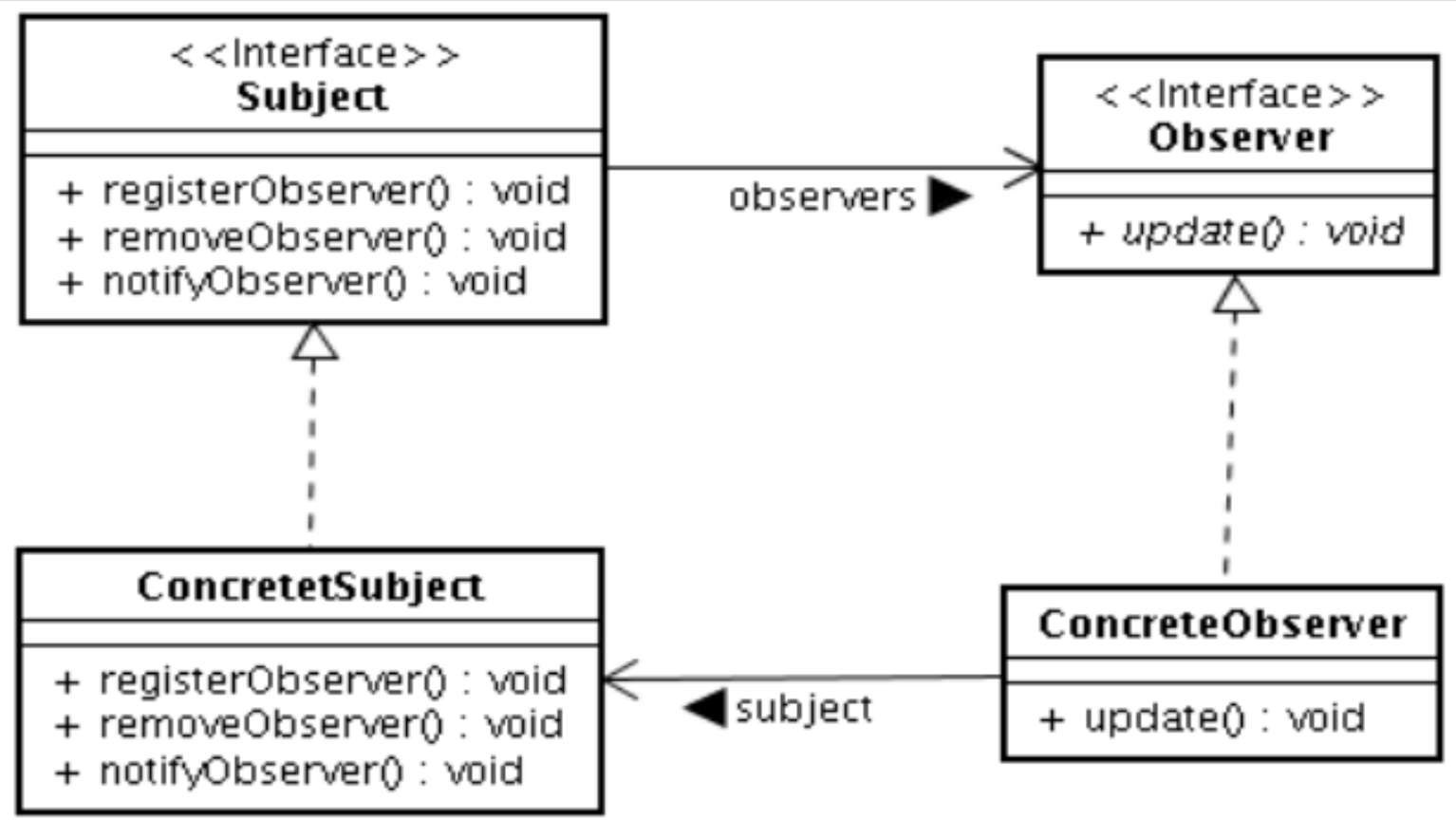
ConcreteObserver

→ Mantém uma referência para o objeto *ConcreteSubject*.

→ Implementa a interface de atualização definida em *Observer*, de forma a executar ações necessária quando notificado.

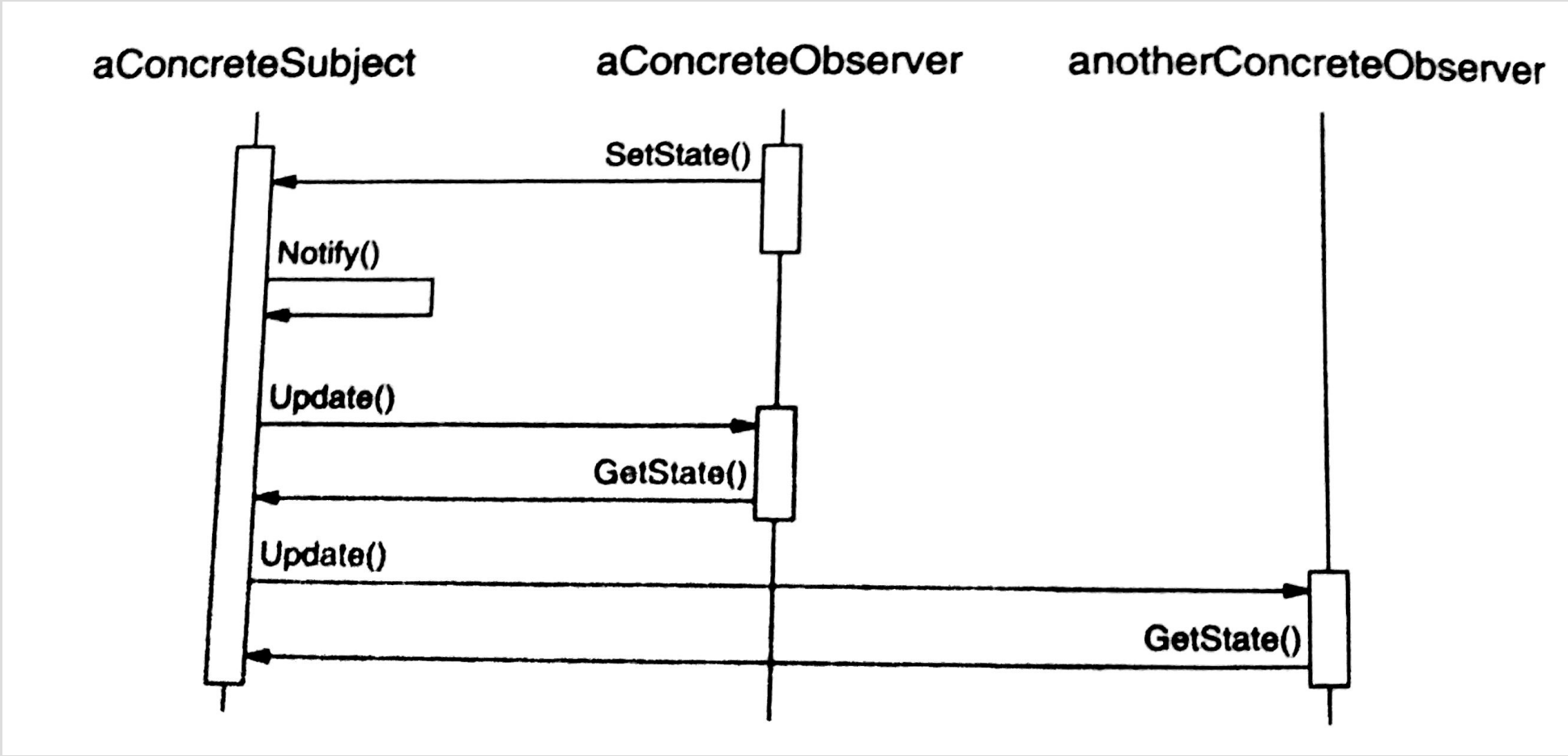
Padrão Observer (Observador)

- Diagrama de Classes



Padrão Observer (Observador)

- Colaboração entre os participantes [GAMMA, Padrões de Projeto – Soluções Reutilizáveis de Software OO.]

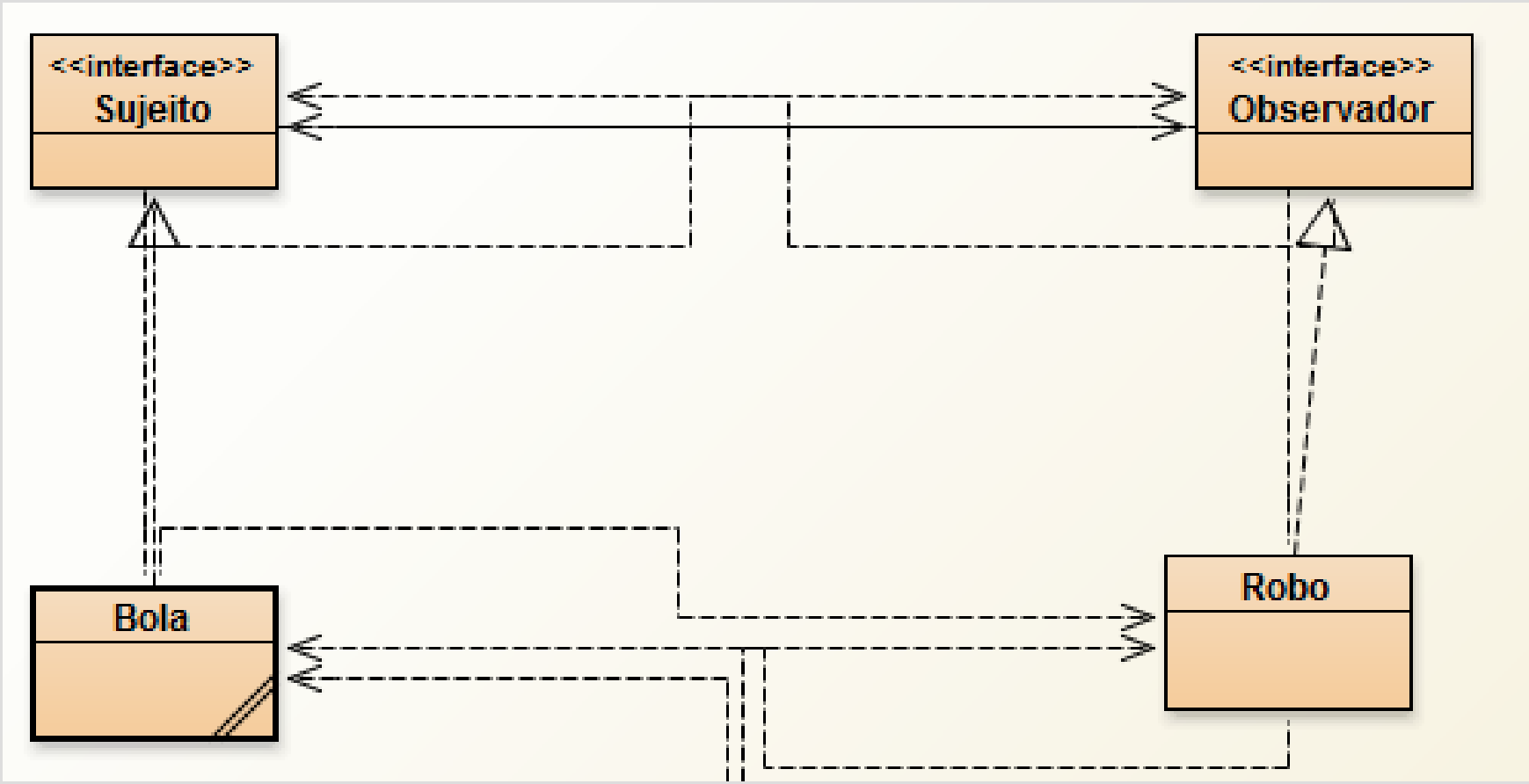


Exemplo:

- Um simulador de futebol de robôs
  - sempre que a bola muda de posição na tela (coordenadas x,y), os jogadores são notificados de sua nova posição, para que possam se movimentar na direção correta,
  - e ao atingirem distância de 0,0 da bola possam chutá-la.

Padrão Observer (Observador)

- Implementação - Exemplo 01



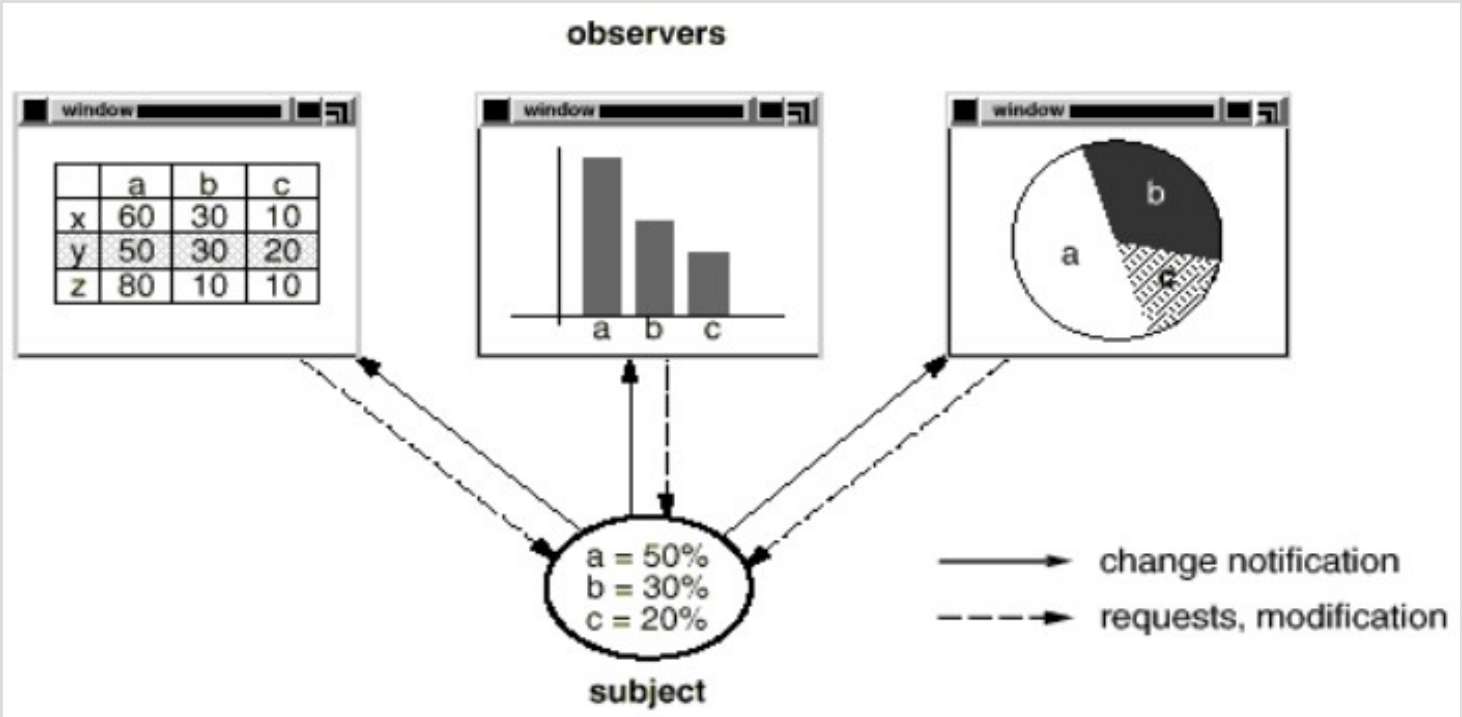
Padrão Observer (Observador)

Observação:

→ É possível dar nomes aos observadores criando um atributo *String* e passando seu nome para o construtor como segundo parâmetro, para ficar mais fácil identificá-los.

Padrão Observer (Observador)

Exemplo:



Padrão Observer (Observador)

- Exercício: Utilizando o padrão Observer, implemente a separação entre modelo (dados) e visão (interface com usuário), de forma que quando o modelo sofre alteração, visões do tipo texto e gráfica exibem ao usuário a informação.

  - Para visão em modo texto, utilize
    - System.out.println(INFO);
  - Para visão em modo gráfico, utilize
    - JoptionPane.showMessageDialog(null, INFO); do pacote javax.swing

Padrão Observer (Observador)

Exercício:

Em um sistema de monitoramento hospitalar, a atualização seria um dos pontos importantes.
A intenção desse sistema é receber informações de aparelhos ligados a um paciente e informar quaisquer alterações de seu estado automaticamente para um médico ou responsável.
Utilizando o padrão *observer*, desenvolva um sistema que alerte o médico quando a temperatura dos seus pacientes se alterar para um valor menor do que 36 ou maior do que 39, e quando os batimentos por minuto ficarem maior que 120 ou menor que 50.

Padrão Observer (Observador) usando a API do JAVA

→ Utilizando a classe *Observable* e a interface *Observer* do Java para implementação do padrão.

→ Existe suporte para este padrão na API do JAVA: java.util.Observer e java.util.Observable.

→ O objeto que será observado deve ser de uma subclasse de *Observable (sujeito)*.

→ E os objetos observadores devem implementar a interface *Observer (observador)*.

Padrão Observer (Observador) usando a API do JAVA

Métodos da classe Observable:

→ addObserver(Observer o) – adiciona um objeto à sua lista de observadores

→ removeObserver(Observer o) – remove um objeto da sua lista de observadores

→ setChanged() - marca o sujeito como “alterado”. O método hasChanged() vai passar a retornar *true*.

→ clearChanged() - desfaz o setChanged()

→ notifyObservers() e notifyObservers(Object o) – chama o método update(Observable obs, Object o) dos observadores. É necessário executar setChanged() antes.
Chama o clearChanged() automaticamente.

Padrão Observer (Observador) usando a API do JAVA

→ O segundo argumento do método update() será null caso o método notifyObservers() seja invocado.

Métodos da interface Observer:

→ update(Observable o, Object o);

→ Cada objeto observador deve implementar essa interface.

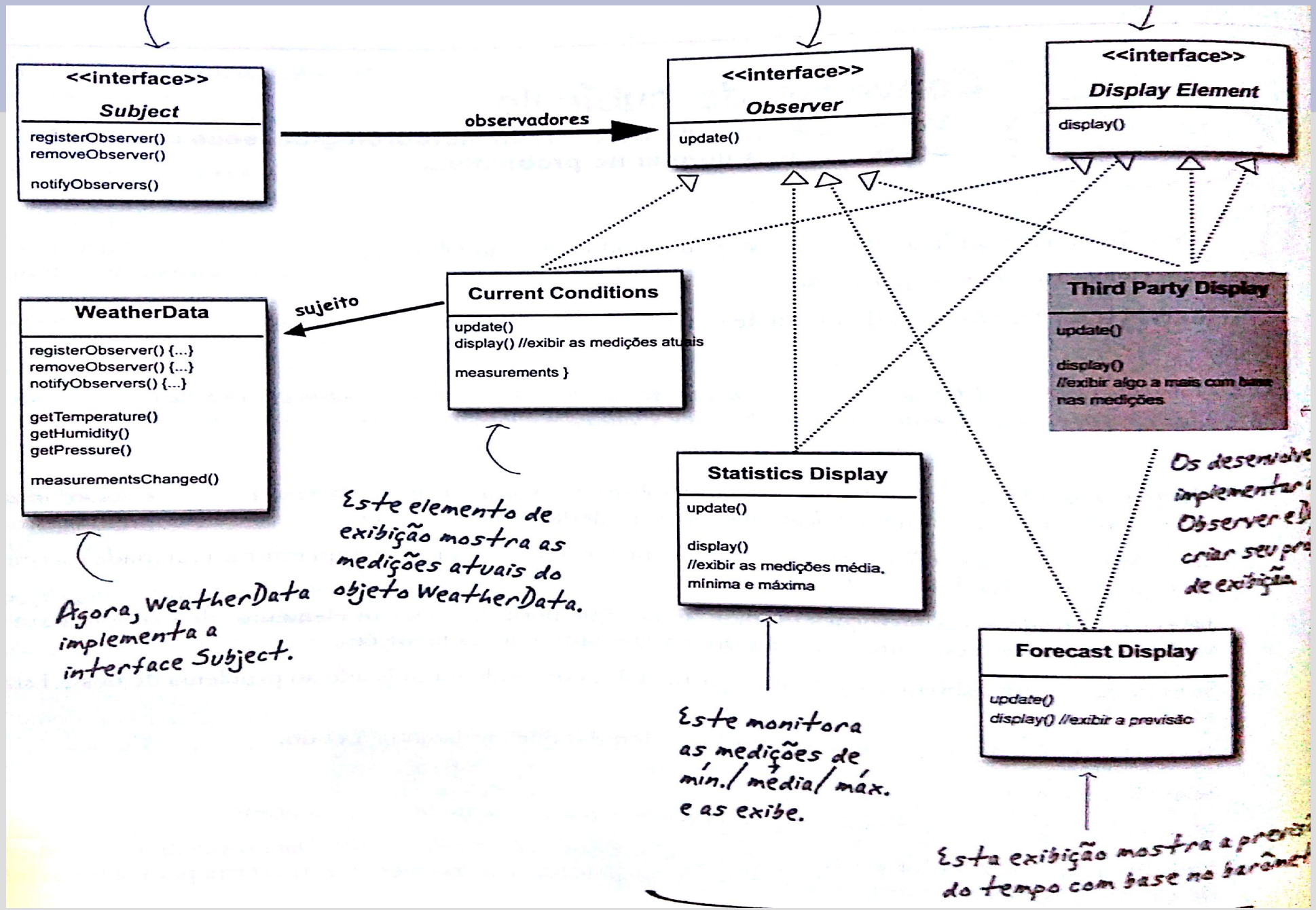
→ O primeiro argumento de update() é o objeto observado, e o segundo é null ou um argumento opcional que poderá ser incluído no método notifyObservers(), a ser enviado pelo sujeito.

Padrão Observer (Observador) usando a API do JAVA

Exemplo 2:

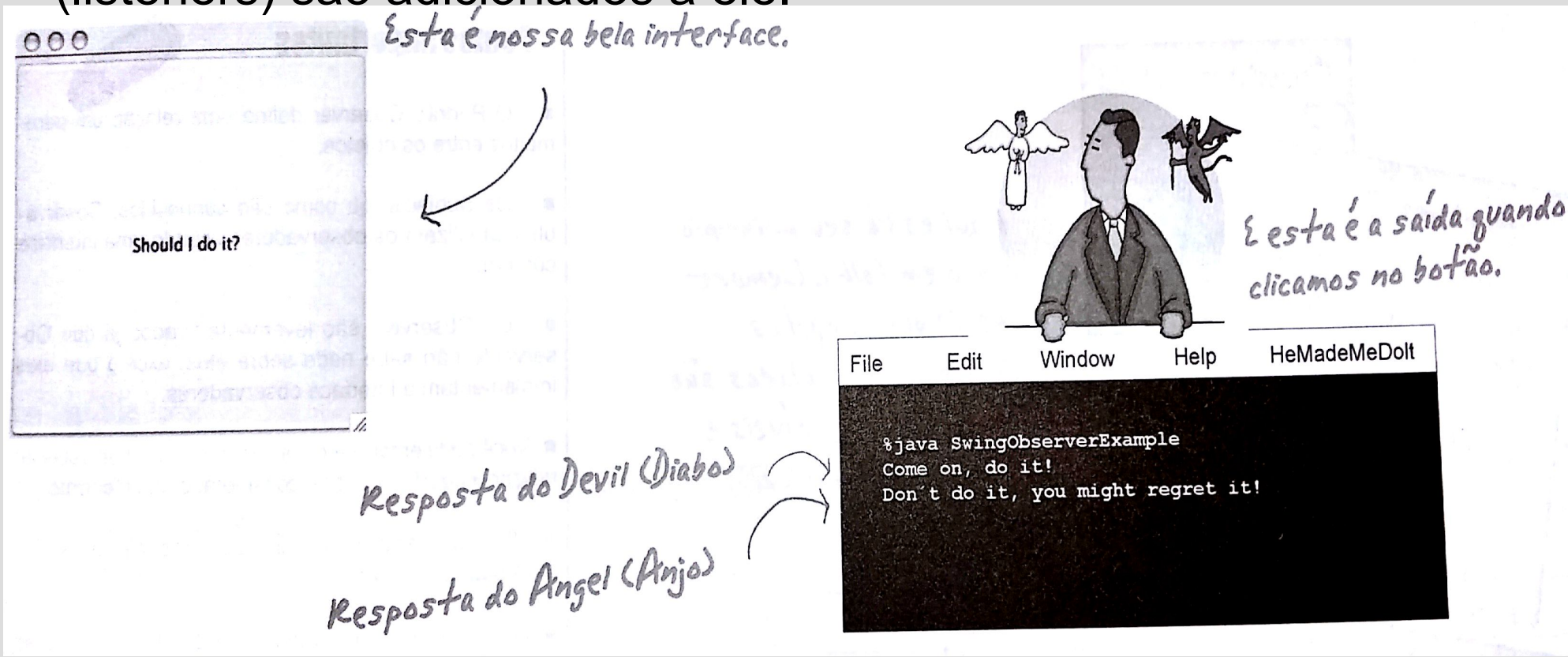
Implementação sim. Futebol de robôs usando a API do Java

Exemplo – Freeman, Padrões de Projeto. Use a Cabeça.



Exemplo – Freeman, Padrões de Projeto. Use a Cabeça.

A classe JButton do Java Swing implementa o padrão observer. Neste exemplo um Botão (sujeito) é criado, e dois observadores (listeners) são adicionados a ele.



```
public class SwingObserverExample {
    JFrame frame;

    public static void main(String[] args) {
        SwingObserverExample example = new SwingObserverExample();
        example.go();
    }

    public void go() {
        frame = new JFrame();
        JButton button = new JButton("Should I do it?");
        button.addActionListener(new AngelListener());
        button.addActionListener(new DevilListener());
        frame.getContentPane().add(BorderLayout.CENTER, button);
        // Ajuste as propriedades do frame aqui
    }

    class AngelListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            System.out.println("Don't do it, you might regret it!");
        }
    }

    class DevilListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            System.out.println("Come on, do it!");
        }
    }
}
```

Aplicativo que apenas cria um quadro e coloca um botão nele.

Torna os objetos devil e angel ouvintes (observadores) do botão.

Estas são as definições de classe para os observadores, definidas como classes internas (mas não precisam ser).

Em vez de update(), o método actionPerformed() é chamado quando o estado no sujeito (neste caso, o botão) é alterado.

Padrão Observer (Observador)

Desafio do Princípio de Projeto

Princípio de projeto

Identifique os aspectos de seu aplicativo que variam e separe-os do que continua igual.

Princípio de projeto

Programe para uma interface, e não para uma implementação.

Princípio de projeto

Dê prioridade à composição em relação à herança.

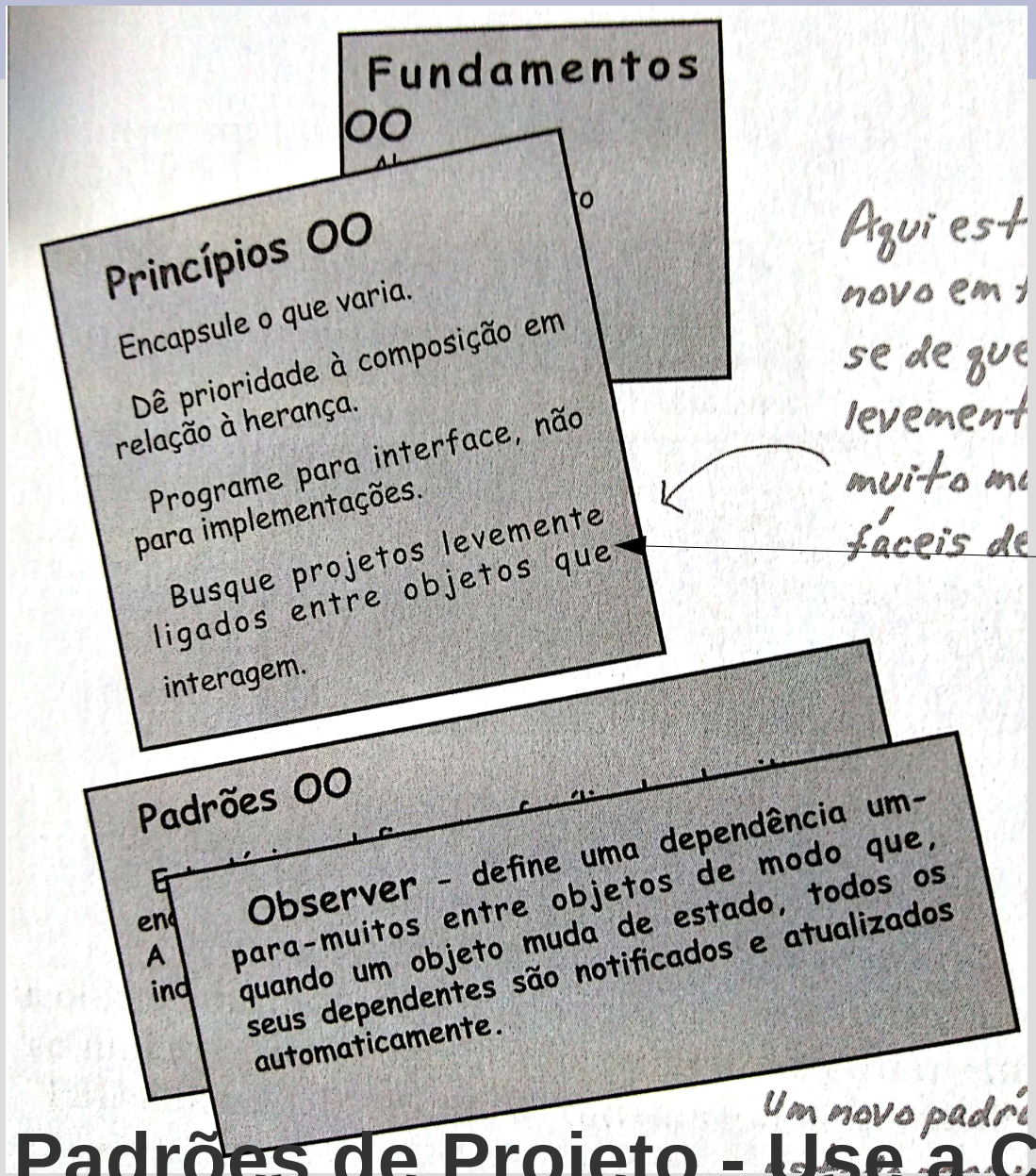
O que varia no Padrão Observer é o estado de sujeito e o número e os tipos de observadores. Com esse padrão, você pode variar os objetos que dependem do estado do sujeito sem ter que alterar esse sujeito. Isso se chama planejar o futuro!

Sujeito e objervador usam interfaces. O sujeito monitora os objetos implementando a interface Observer, enquanto os observadores registram e são notificados pela interface Subject. Conforme vimos, isso mantém as coisas bem e levemente ligadas.

O Padrão Observer utiliza a composição para compor qualquer número de observador com seus sujeito. Essas relações não são configuradas por algum tipo de hierarquia de herança. Não, elas são configuradas no tempo de execução pela composição

Freeman, Padrões de Projeto - Use a Cabeça.

Padrão Observer (Observador)



Acoplamento Fraco

Freeman, Padrões de Projeto - Use a Cabeça.

Conclusões

- O padrão Obsever fornece um design de objeto em que os sujeitos e os observadores são levemente ligados:
  - A única coisa que o sujeito sabe sobre um observador é que ele implementa uma certa interface (observer).
  - Pode-se adicionar novos observadores a qualquer momento.
  - Nunca será necessário modificar o sujeito para adicionar novos tipos de observadores.
  - Pode-se reutilizar sujeitos ou observadores independentemente uns dos outros.
  - Alterações no sujeito ou em algum observador não afetam o outro.

Padrão Observer (Observador) usando a API do JAVA

- Ok.