Padrões de Projeto

Mediator and Observer

Prof. Alexandre Vidal
DEINF-UFMA
Março de 2007

Tópicos Avançados em Linguagem de Programação

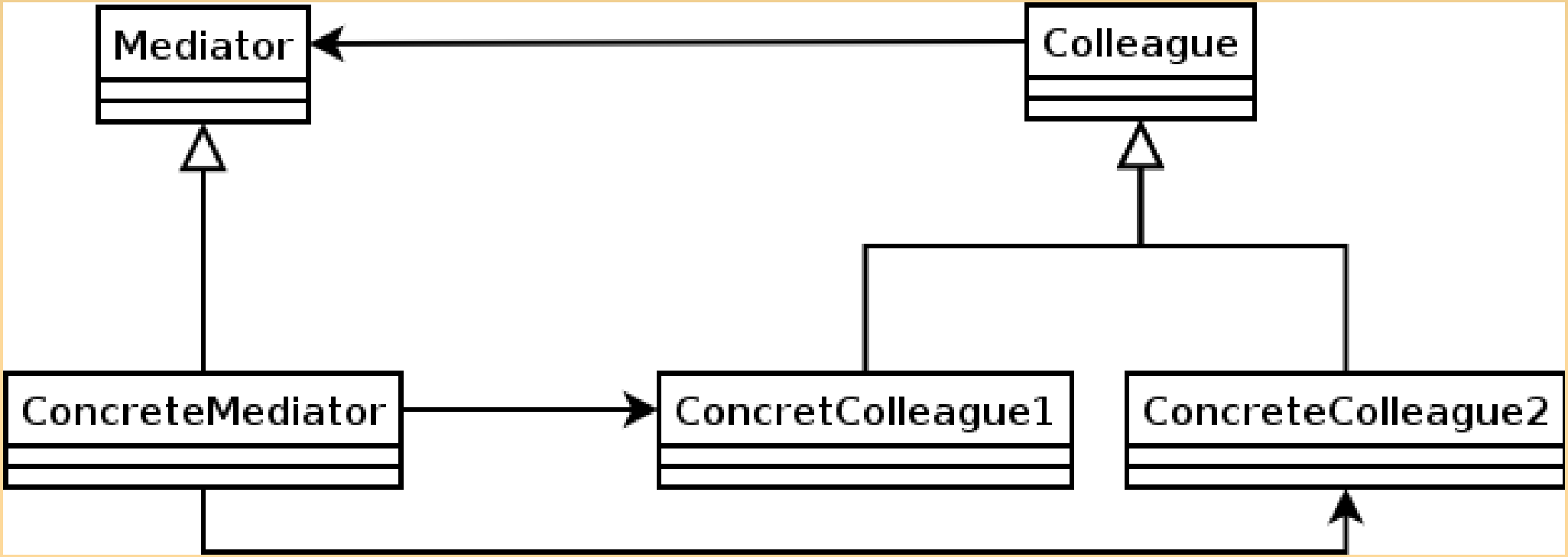
- *Mediator* (padrão comportamental)
  - intenção:
    - definir um objeto que encapsula como um conjunto de objetos que interage;
    - manter objetos fracamante acoplados impedindo-os de se referirem entre si explicitamente.

- *Mediator* (padrão comportamental)
  - motivação:
    - OO design encoraja a distribuição de responsabilidades (aumenta reúso);
    - isso pode resultar em um excessivo número de conexões entre objetos (atrapalha reúso);
    - e.g.: diferentes caixas de diálogos podem ter diferentes dependências entre os mesmos ***widgets***;
    - o comportamento coletivo pode ser encapsulado em um objeto ***mediator*** separado.

Tópicos Avançados em Linguagem de Programação

- *Mediator* (padrão comportamental)
  - aplicabilidade (quando usar o padrão):
    - quando um conjunto de objetos se comunica de modo bem definido, mas complexo;
    - é difícil reusar um objeto porque ele se comunica com muitos outros objetos;
    - o comportamento distribuído entre diversas classes deve ser personalizado sem excesso de subclasses.

- *Mediator* (padrão comportamental)
  - Estrutura



Tópicos Avançados em Linguagem de Programação

- *Mediator* (padrão comportamental)
  - Participantes
    - Mediator:
      - define uma interface para comunicação com os objetos ***Colleague*.**
    - ConcreteMediator:
      - implementa comportamento cooperativo através da coordenação de objetos ***Colleague***;
      - conhece e mantém seus objetos ***Colleagues***.
    - Colleague classes:
      - cada classe ***Colleague*** conhece seu objeto ***Mediator***;
      - cada objeto ***Colleague*** comunica-se com o seu obejto ***Mediator*** em situaçoes nas quais teria de comunicar-se com outro objeto ***Colleague***.

- *Mediator* (padrão comportamental)
  - Colaborações
    - ***Colleagues*** enviam e recebem requisições do objeto ***Mediator***;
    - por sua vez, o ***Mediator*** implementa o comportamento cooperativo ao redirecionar as requisições aos objetos ***Colleague*** apropriados.

- *Mediator* (padrão comportamental)
  - Conseqüências (*Class Adapter*)
    - ✔ limita hierarquia de subclasses apenas à classe ***Mediator***;
    - ✔ promove desapotamento entre ***Colleagues***;
    - ✔ simplifica protocolos entre objetos: muitos-para-muitos substituido com um-para-muitos;
    - ✔ abstrai como os objetos cooperam (ajuda a entender o funcionamento dos objetos);
    - ✘ centraliza comportamento (objeto monolítico) .

- *Mediator* (padrão comportamental)
  - Implementação
    - omissão da classe ***Mediator*** abstrata:
      - quando ***Colleagues*** interagem com um único ***Mediator***;
    - comunicação ***Colleague-Mediator***:
      - ***Colleagues*** se comunicam com ***Mediator*** quando ocorre um evento de interesse. (ver padrão ***Observer***, como uma opção).

Tópicos Avançados em Linguagem de Programação

- *Mediator* (padrão comportamental) (from GoF)

```
class DialogDirector {
public:
virtual ~DialogDirector();
virtual void ShowDialog();
virtual void WidgetChanged(Widget*) = 0;
protected:
DialogDirector();
virtual void CreateWidgets() = 0; };
```

Tópicos Avançados em Linguagem de Programação

- *Mediator* (padrão comportamental) (from GoF)

```
class Widget {
public:
Widget(DialogDirector*);
virtual void Changed();
virtual void HandleMouse(MouseEvent& event); // ...
private:
DialogDirector* _director;};
void Widget::Changed ()
{       _director->WidgetChanged(this); }
```

- *Mediator* (padrão comportamental) (from GoF)

  A classe ***FontDialogDirector*** faz a mediação entre widgets no dialog box

```
class FontDialogDirector : public DialogDirector {
public:  FontDialogDirector();
         virtual ~FontDialogDirector();
         virtual void WidgetChanged(Widget *);
protected:     virtual void CreateWidgets();
private:       Button* _ok;
               Button* _cancel;
               ListBox* _fontList;
               EntryField* _fontName; };
```

- *Mediator* (padrão comportamental) (from GoF)

FontDialogDirector mantém os widgets que contém. Ela redefine ***CreateWidgets()*** para criar os widgets e inicializar suas referências para eles:

```
void FontDialogDirector::CreateWidgets () {
    _ok = new Button(this);
    _cancel = new Button(this);
    _fontList = new ListBox(this);
    _fontName = new EntryField(this);
    // fill the listBox with the available font names
    // assemble the widgets in the dialog }
```

- *Mediator* (padrão comportamental) (from GoF)

WidgetChanged assegura que widgets funcionam adequadamente juntos:

```
void FontDialogDirector::WidgetChanged ( Widget*
   theChangedWidget ) {
      if (theChangedWidget == _fontList) {
           _fontName->SetText(_fontList->GetSelection());
      } else if (theChangedWidget == _ok) {
           // apply font change and dismiss dialog
      } else if (theChangedWidget == _cancel) {
           // dismiss dialog
      }
}
```

- *Mediator* (padrão comportamental)
  - usos conhecidos
    - ver página xxx no GoF (traduzido);
  - Padrões Relacionados
    - *Facade*
      - *encapsula um subsistema de objetos e é unidirecional;*
      - *Mediator encapsula um conjunto de objetos que cooperam e interagem entre si e como o Mediator.*
    - *Comunicação no Mediator pode ser implementada com o Observer.*

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - intenção:
    - define uma dependência um-para-muito entre objetos, de modo que, se um objeto sofre uma modificação de estado, todos os seus dependentes são notificados e atualizados automaticamente.
  - a.k.a.: publisher*/subscribe, dependents.*

Tópicos Avançados em Linguagem de Programação

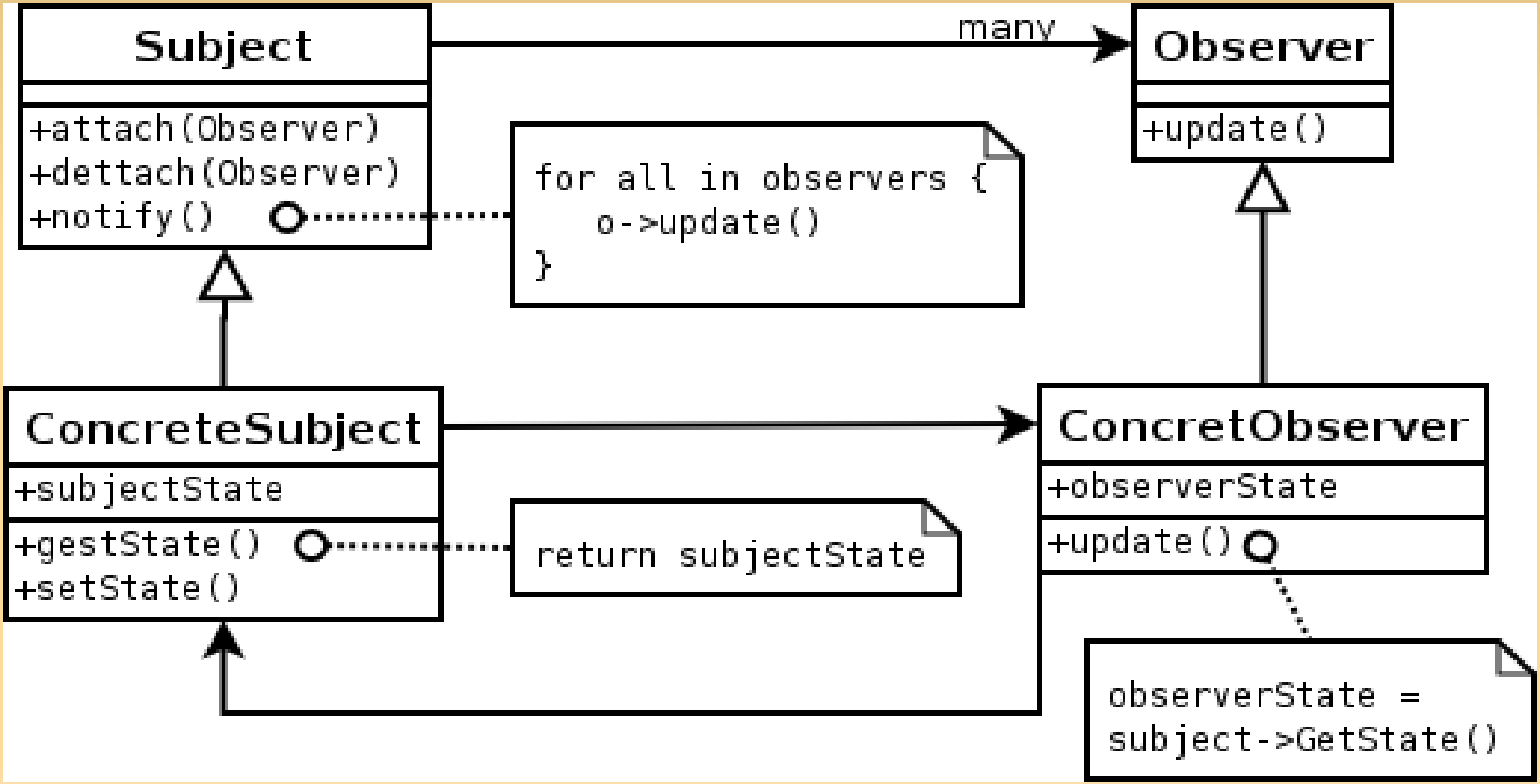
- *Observer* (padrão comportamental)
  - motivação:
    - um sistema formado por classes cooperantes separadas precisa manter sua consistência;
    - ferramentas GUI separam aspectos da representação gráfica dos dados subjacentes da aplicação (e.g., planilhas e gráficos):
      - classes representando representação gráfica e dados podem então ser usadas de modo independente.
    - Observer descreve como obter esse tipo de relacionamento entre, por exemplo, um conjunto de dados e vários diferentes interfaces gráficas.

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - aplicabilidade (usar o padrão):
    - quando uma aplicação tem dois aspectos, um dependente do outro, tornando-os mais independentes e reusáveis;
    - quando mudar um objeto exige mudar outros e não se sabe quantos objetos precisam ser alterados;
    - quando um objeto precisa notificar outros objetos sem ter que conhecê-los.

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - Estrutura



Tópicos Avançados em Linguagem de Programação

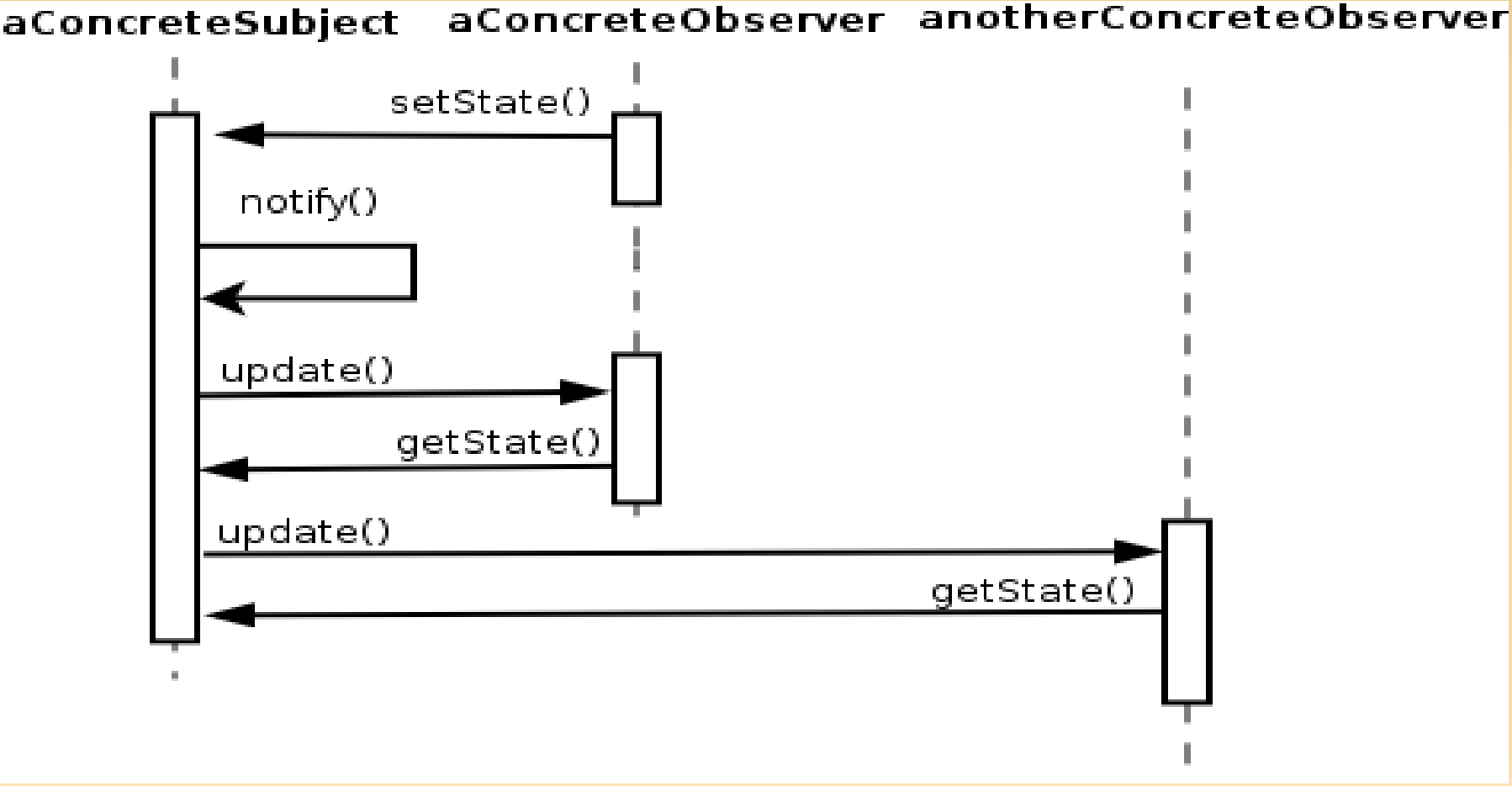
- *Observer* (padrão comportamental)
  - Participantes
    - Subject:
      - conhece seus observadores;
    - Observer:
      - define uma interface para objetos que devem ser notificados de mudanças em um ***Subject***;

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - Participantes
    - ConcreteSubject:
      - armazena estados de interesse dos objetos ***ConcreteObserver***;
      - envia notificações para seus ***Observers*** quando esses estados mudam.
    - ConcreteObserver:
      - mjantém uma referência para um objeto ***ConcreteSubject***;
      - armazena estado que deve ficar consistente com o do ***subject***;
      - implementa a interface de atualização do ***Observer*** para manter-se consistente com o do ***Subject***.

- *Observer* (padrão comportamental)
  - Colaborações
    - ConcreteSubject notifica seus observadores sempre que ocorre uma mudança que poderia fazer com que o estado dos observadores inconsistenstes com o seu próprio estado;
    - ao ser informado de mudanças em ConcretSubject um objeto ConcreteObserver pode consultar o ConcreteSubject para obter informações e usá-las para reconciliar seu estado com o subject.

- *Observer* (padrão comportamental)
  - Colaborações



Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - Conseqüências
    - permite reusar ***Subjects*** sem reusar seus ***Observers*** e vice-versa;
    - suporte para comunicação em broadcast (a notificação é enviada pelo ***Subject*** automaticamente para todos os ***Observers*** subscritos);
    - atualizações inesperadas (cuidar dos critérios de dependência).

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - Implementação (aspectos)
    - mapeamento entre subjects e observers;
    - observando mais de um Subject (e.g., planilha com diferentes fontes);
    - quem inicia a atualização ?
    - ..... veja outras questões no GoF

- *Observer* (padrão comportamental) (from GoF)

  Uma classe abstrata define a interface ***Observer***:

```
class Subject;
class Observer {
public:
        virtual ~ Observer();
        virtual void Update(Subject* theChangedSubject) = 0;
protected:
        Observer();
};
```

  Ela suporta múltiplos ***subjects*** for each ***observer***.
  O subject passado para a operação Update permite ao observer determinar qual subject mudou quando ele observa mis que um.

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental) (from GoF)
- Uma classe abstrata define a interface ***Subject***:

```
class Subject {
public:
        virtual ~Subject();
        virtual void Attach(Observer*);
        virtual void Detach(Observer*);
        virtual void Notify();
protected:
        Subject();
private:
List<Observer*> *_observers;   };
```

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental) (from GoF)

```
void Subject::Attach (Observer* o) {
        _observers->Append(o); }
void Subject::Detach (Observer* o) {
        _observers->Remove(o); }
void Subject::Notify () {
      ListIterator<Observer*> i(_observers);
      for (i.First(); !i.IsDone(); i.Next()) {
            i.CurrentItem()->Update(this);
      }
}
```

- *Observer* (padrão comportamental) (from GoF)

ClockTimer is a concrete subject for storing andmaintaining the time of day.

```
class ClockTimer : public Subject {
public:
    ClockTimer();
    virtual int GetHour();
    virtual int GetMinute();
    virtual int GetSecond();
    void Tick();
};
```

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental) (from GoF)

Tickupdates the ClockTimer's internal state and callsNotify to inform observers of the change:

```
void ClockTimer::Tick () {
        // update internal time-keeping state
        // ...
        Notify();
}
```

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental) (from GoF)

```
class DigitalClock: public Widget, public Observer {
public:
        DigitalClock(ClockTimer*);
        virtual ~DigitalClock();
        virtual void Update(Subject*);
                // overrides Observer operation
        virtual void Draw();
                // overrides Widget operation;
                // defines how to draw the digital clock
private:
        ClockTimer* _subject; };
```

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental) (from GoF)

O código abaixo cria um ***AnalogClock*** e um ***DigitalClock*** que sempre nostram a mesma hora:

```
ClockTimer* timer = new ClockTimer;
AnalogClock* analogClock = new AnalogClock(timer);
DigitalClock* digitalClock = new DigitalClock(timer);
```

Sempre que o timer “ticks”, os dois relógios serão atualizados e e reexibidos apropriadamente.

Tópicos Avançados em Linguagem de Programação

- *Observer* (padrão comportamental)
  - usos conhecidos
    - ver página xxx no GoF (traduzido);
  - Padrões Relacionados
    -
    -