Padrão Publish/Subscribe e Eventos

Centro de Cálculo
Instituto Superior de Engenharia de Lisboa

F. Miguel Carvalho (mcarvalho@cc.isel.ipl.pt)

Enquadramento

O conceito de **Evento** surge:

Eventos

Padrão *Publish/Subscribe* (ou padrão ***Observer***)

Padrões de desenho do Gamma et al. (1995)

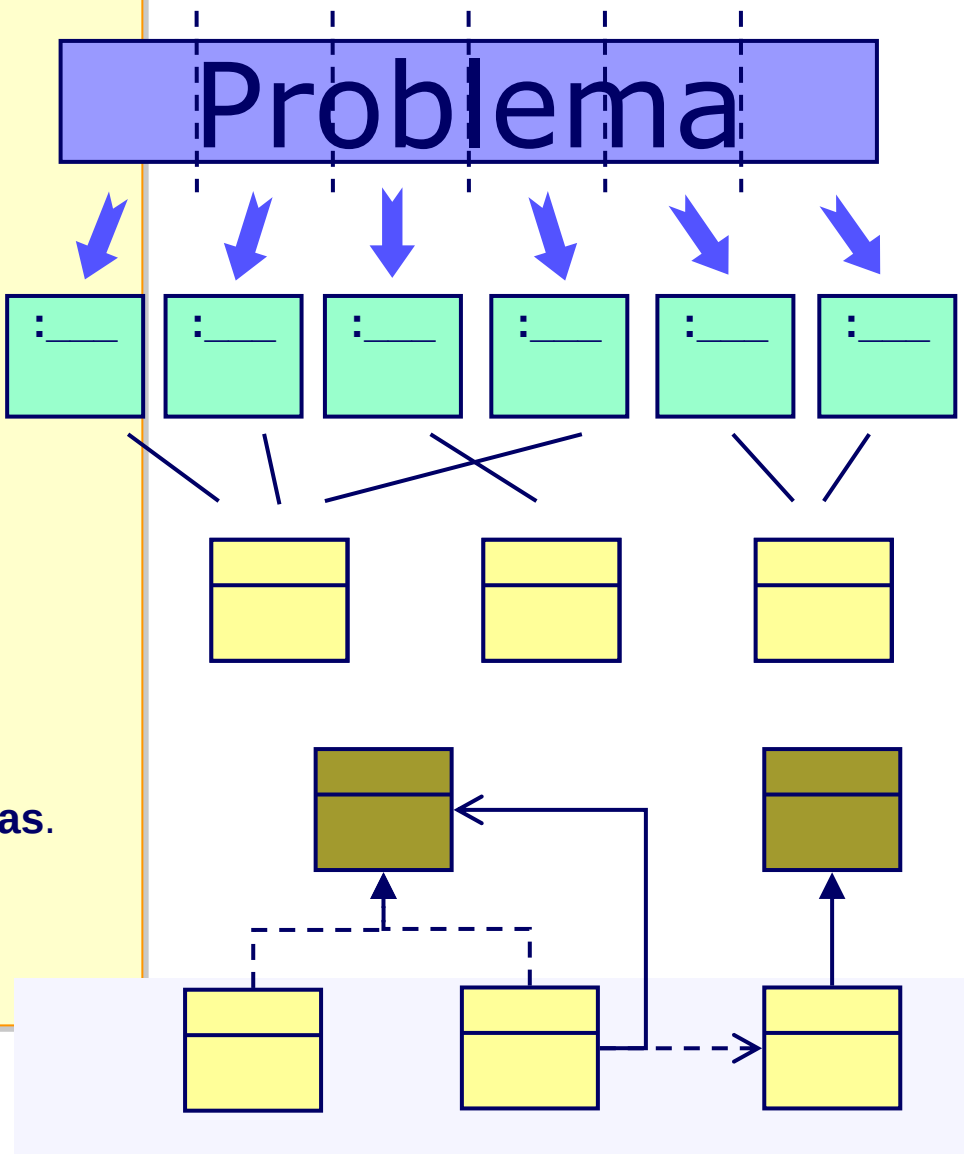
Agenda

- Padrões de Desenho
- Padrão *Observer* ou *Publish/Subscribe*
- Padrão *Observer* com *Delegates*
- Eventos

Desenvolvimento de SW *Object Oriented*

O que fazem os programadores **experientes**?

- Não resolvem todos os problemas de raiz;
- Identificam os objectos pertinentes;
- Decompõem os objectos em tipos, na granularidade **correcta**;
- Desenham hierarquias e relações **correctas**.



Desenvolvimento de SW *Object Oriented*

Como reutilizar a experiência no desenvolvimento de SW?

Pretende-se que os padrões de desenho ajudem a alcançar este objectivo:

- **Registando** soluções de software (desenho) reconhecidas na resolução de problemas recorrentes;
- **Divulgando** dessas soluções e problemas;
- **Evitando** soluções “comprometedoras”;
- Oferecendo melhor **documentação**.

Padrão de Desenho... Exemplo

contexto:
- passagens longas e estreitas;

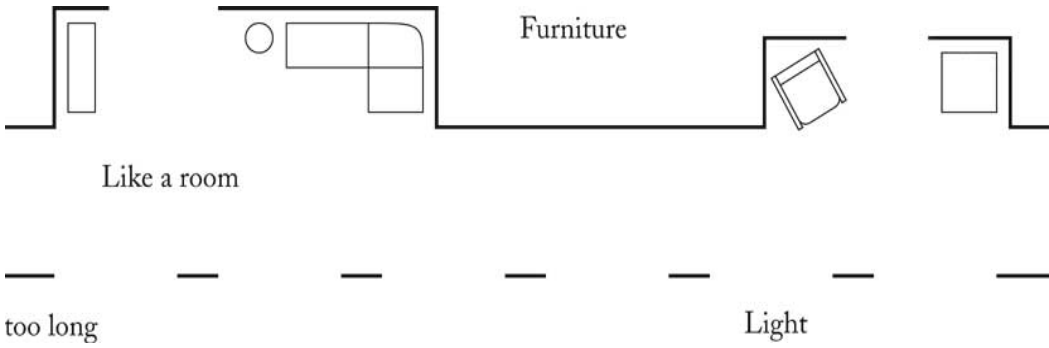
problema:
- corredores compridos são deprimentes e desconfortáveis;

constrangimentos:
- comprimento;
- falta de luz.

objectivo: evitar a ansiedade.

Desenho:
- evitar áreas estreitas;
- intercalar espaço amplos;
- aparência de uma sala;
- aplicar janelas.

Solução



Padrão de Desenho…

Conceito:

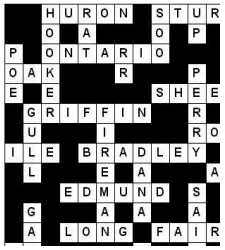
➔ Um padrão é uma **solução** para um **problema** num dado **contexto**.

- **Contexto**:
  - Situação a que se aplica o padrão.
  - Deve ser uma **situação recorrente**.

Exemplo:
Passagens longas e estreitas.

- **Problema**:
  - **Objectivo** a atingir nesse contexto.
  - Conjunto de **constrangimentos** que ocorrem nesse contexto.

Exemplo:
Num hospital os corredores compridos aumentam a ansiedade dos pacientes.



- **Solução**:
  - **Desenho genérico** que qualquer um possa aplicar para resolver um **problema** com um conjunto de **constrangimentos** e um **objectivo** comum.

Exemplo:
Intercalar espaço amplos dando a aparência de uma sala.

Padrão de Desenho... Requisitos

Nem todas as solução podem-se tornar num padrão!

1. Uma solução para se tornar num padrão necessita de se aplicar a **problemas recorrentes**.

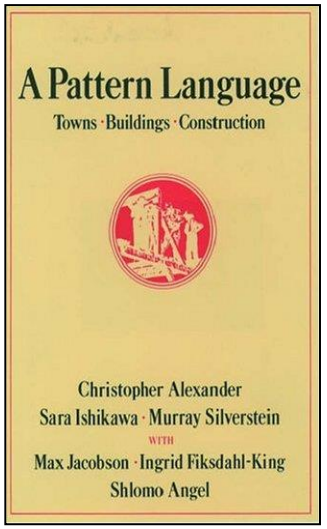
2. Tem de ter uma descrição suficientemente **genérica** e **precisa**, de maneira a que possa ser **adaptado a outros problemas com as mesmas características**.

- Os padrões não são leis, nem regras estáticas.
- **Os padrões são linhas orientadoras que podem ser adaptadas para servir determinadas necessidades**.

Padrões de Desenho… 1ª Definição

- *Christopher Alexander*, professor de arquitectura em Berkeley, foi o primeiro autor a descrever padrões para a arquitectura de áreas habitáveis como casas, edifícios e cidades no livro: "***A Pattern Language: Towns Buildings Constructions***" (Alexander et al. 1977).

"Cada padrão descreve um **problema que ocorre sistematicamente** num **determinado contexto** e descreve os **aspectos chave** da solução, de forma a que esta possa ser reutilizada um número indeterminado de vezes sem que em nenhuma delas seja implementado da mesma forma."



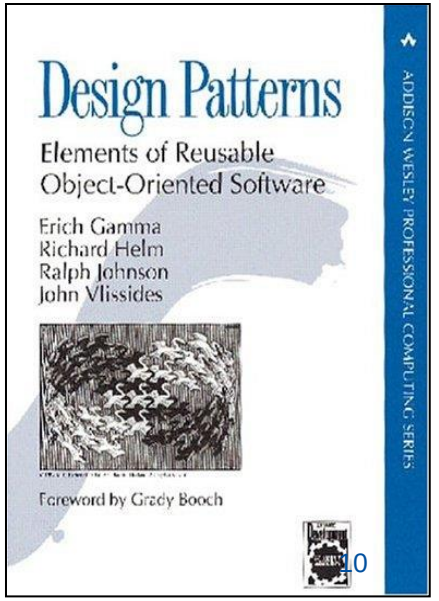
Padrão de desenho em OO

- O livro "***Design Patterns (Gamma et al. 1995)***" foi o primeiro a catalogar de forma sistemática e consistente 23 dos padrões de desenho mais utilizados (com sucesso) em programação OO.

- Este livro tornou-se a principal referência dos **padrões de desenho fundamentais** em programação OO e também é identificado de modo informal na comunidade científica por **GoF**, ou ***Gang-of-Four*** (por terem sido quatro os autores do livro).

- Desde então surgiram diversas propostas para introdução de novos padrões de desenho, mas nenhum foi universalmente aceite.

- Apenas o catálogo do GoF continua a ser, ainda hoje, consensualmente aceite como referência no ceio da comunidade científica.



Padrões de Desenho… descrição do GoF

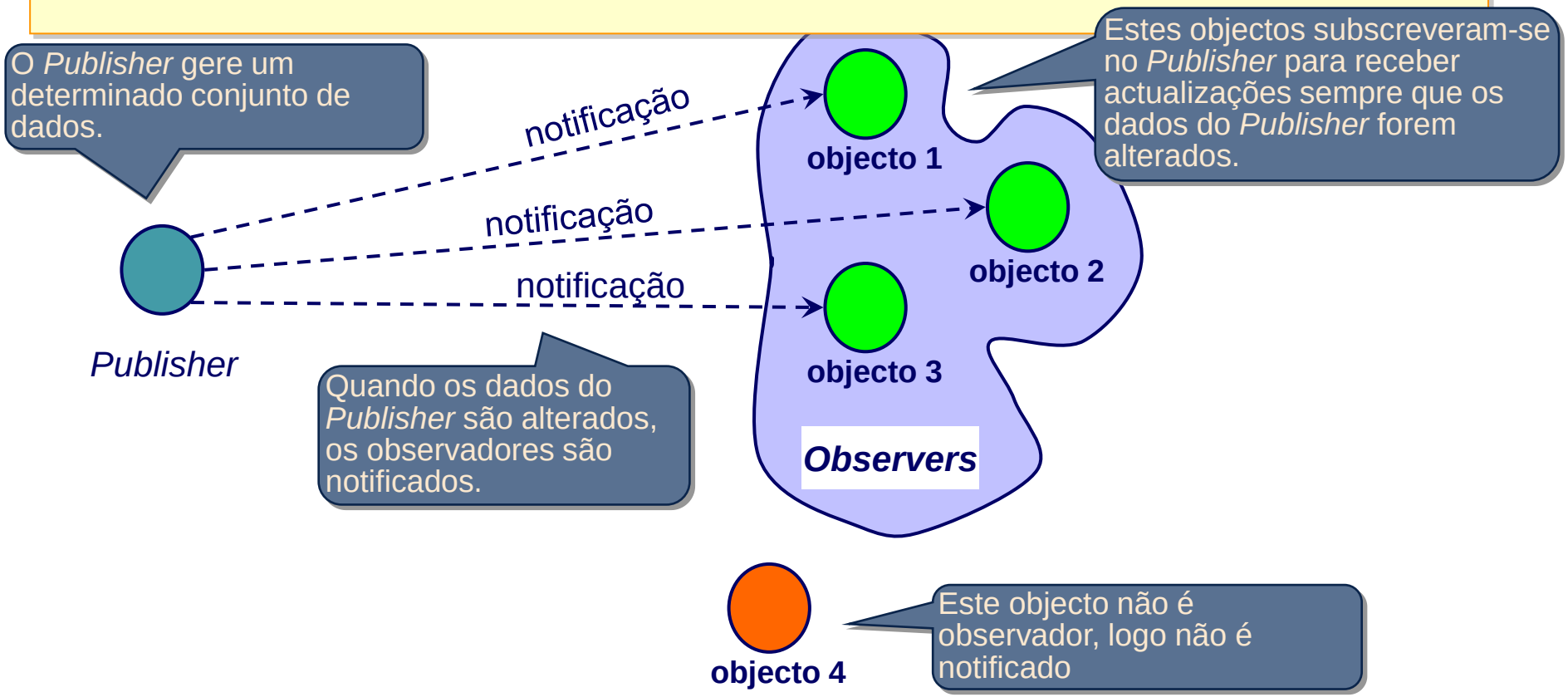
- Nome
- Classificação
- Objectivo
- Outros nomes do padrão
- Motivação
- Aplicabilidade
- Estrutura
- Participantes
- Colaboração
- Consequências
- Implementação
- Código exemplo
- Utilizações conhecidas
- Padrões relacionados

Agenda

- Padrões de Desenho
- Padrão *Observer* ou *Publish/Subscribe*
- Padrão *Observer* com *Delegates*
- Eventos

Publisher + Subscriber = Padrão *Observer*

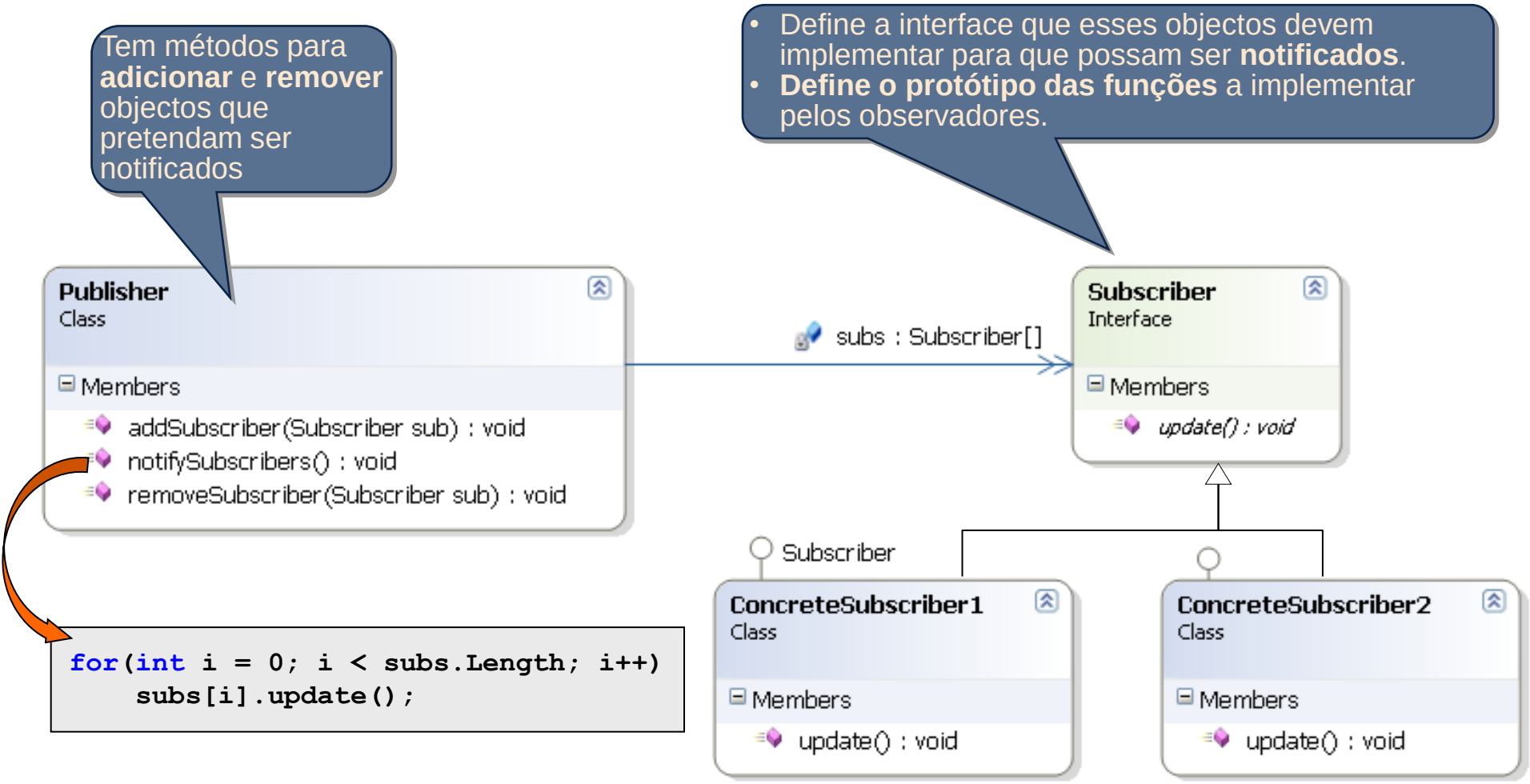
- Padrão ***Observer***
  - O notificador é designado por *PUBLISHER*;
  - O subscritor é designado por *SUBSCRIBER* ou *OBSERVER*.



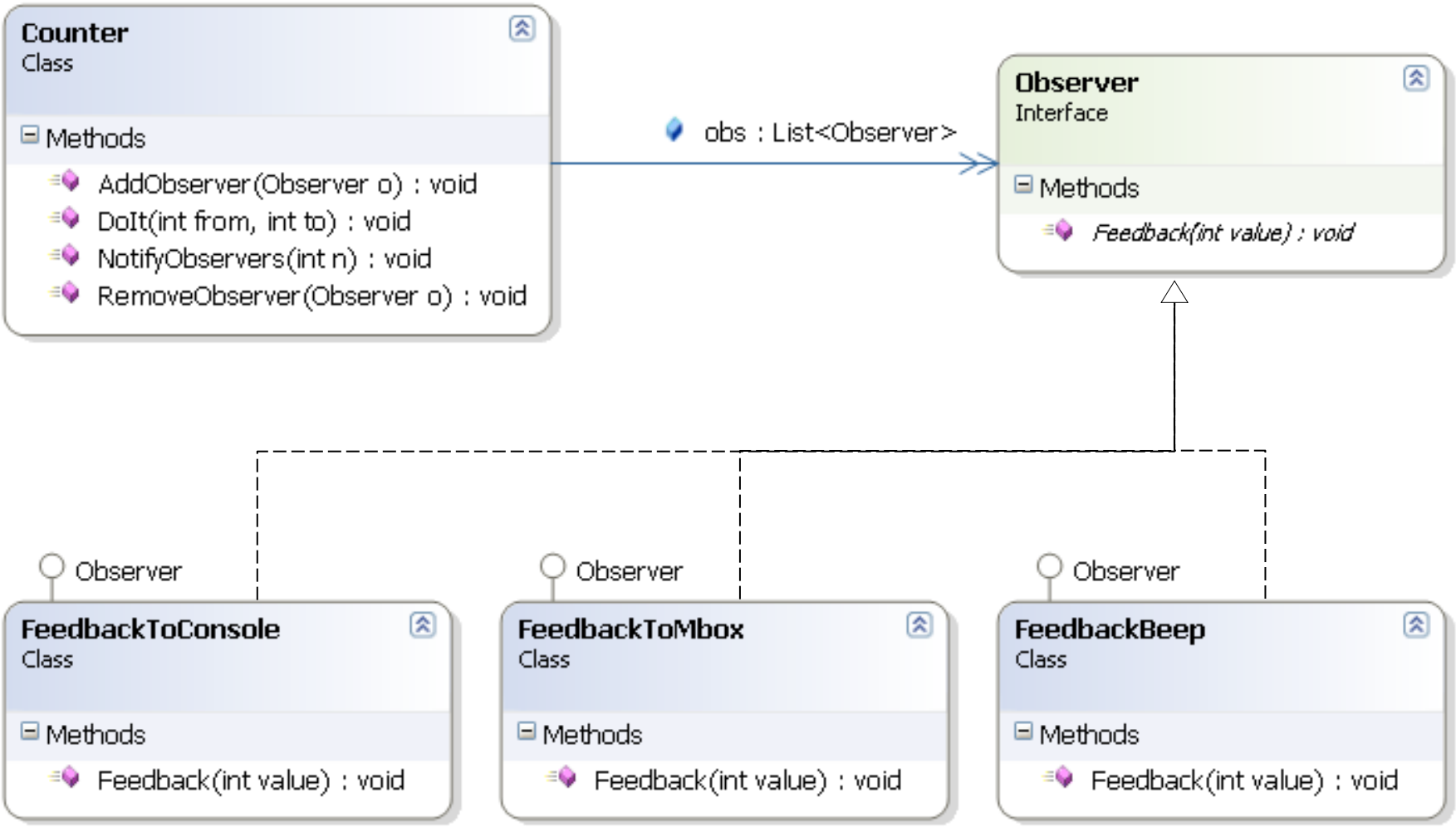
Padrão *Observer* segundo Gamma et al.

- Um dos desenhos de *software* mais usados para desacoplar a dependência entre um **produtor** (***publisher***) e vários **consumidores** (***subscribers*** ou ***observers***), de modo a que:
  - quando o estado do produtor muda todos os consumidores interessados são notificados;
  - novos tipos de consumidores sejam suportados sem necessidade de modificar a implementação do produtor;
  - o produtor apenas conhece a interface suportada pelo consumidor.

Padrão *Observer…* diagrama de tipos



Padrão *Observer*... exemplo... Demo11



demo

Agenda

- Padrões de Desenho
- Padrão *Observer* ou *Publish/Subscribe*
- Padrão *Observer com Delegates*
- Eventos

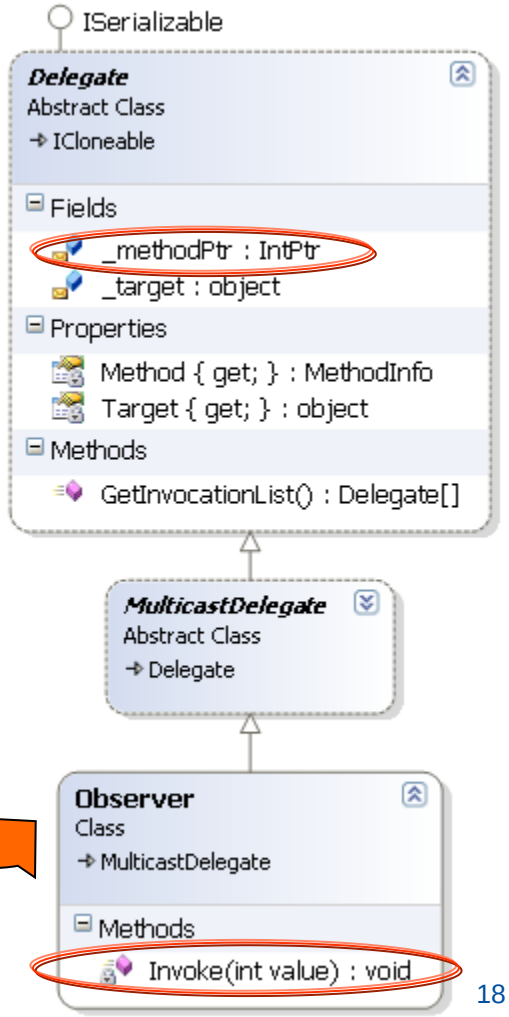
Exemplo… com delegates… Demo12

- A interface `Observer` é substituída pelo *delegate* `Observer`.

```
interface Observer {
  void Feedback(int value);
}
```

```
delegate void Observer(int value);
```

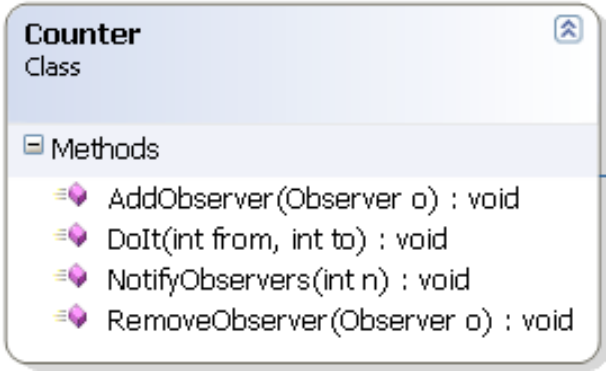
- Cria uma classe `Observer` derivada de `MulticastObserver`
- Define nessa classe um metodo `Invoke` com o mesmo protótipo definido na declaração do delegate.
- O campo `_methodPtr` armazena um ponteiro para a função do observador concreto.



Exemplo… com delegates… Demo12

- Com interface `Observer`:

```
Counter c = new Counter();
c.AddObserver(new FeedbackToConsole());
c.DoIt();
```

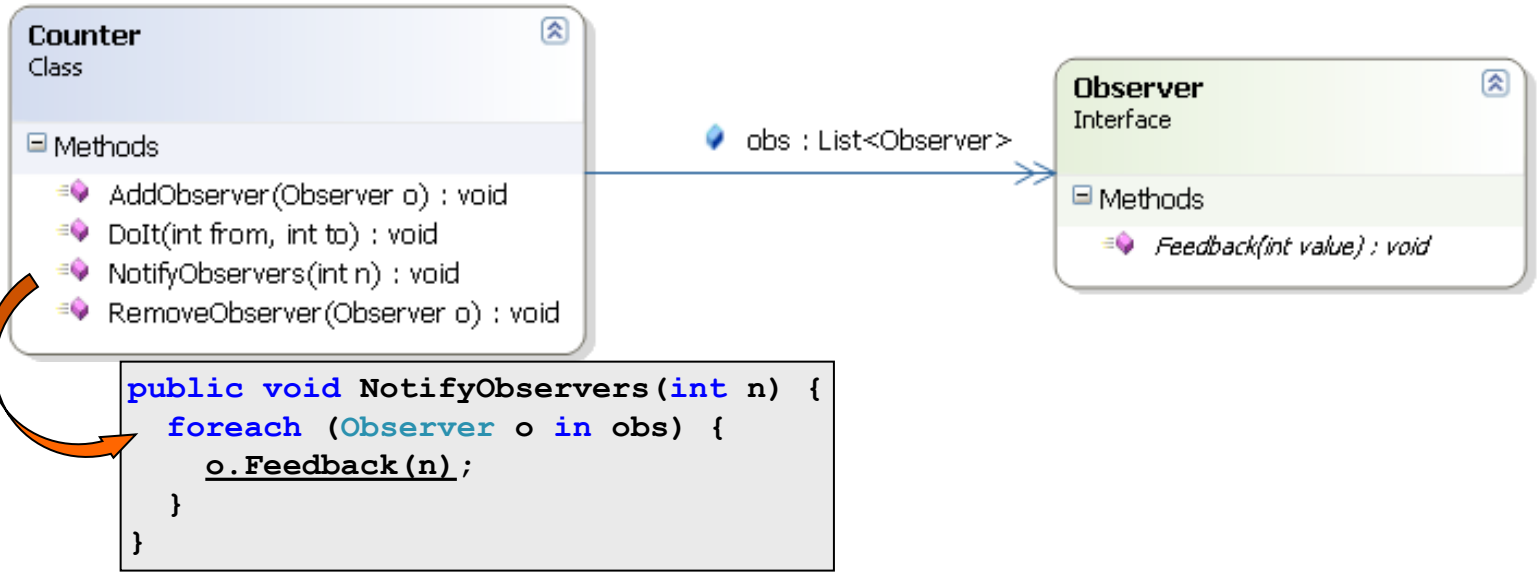


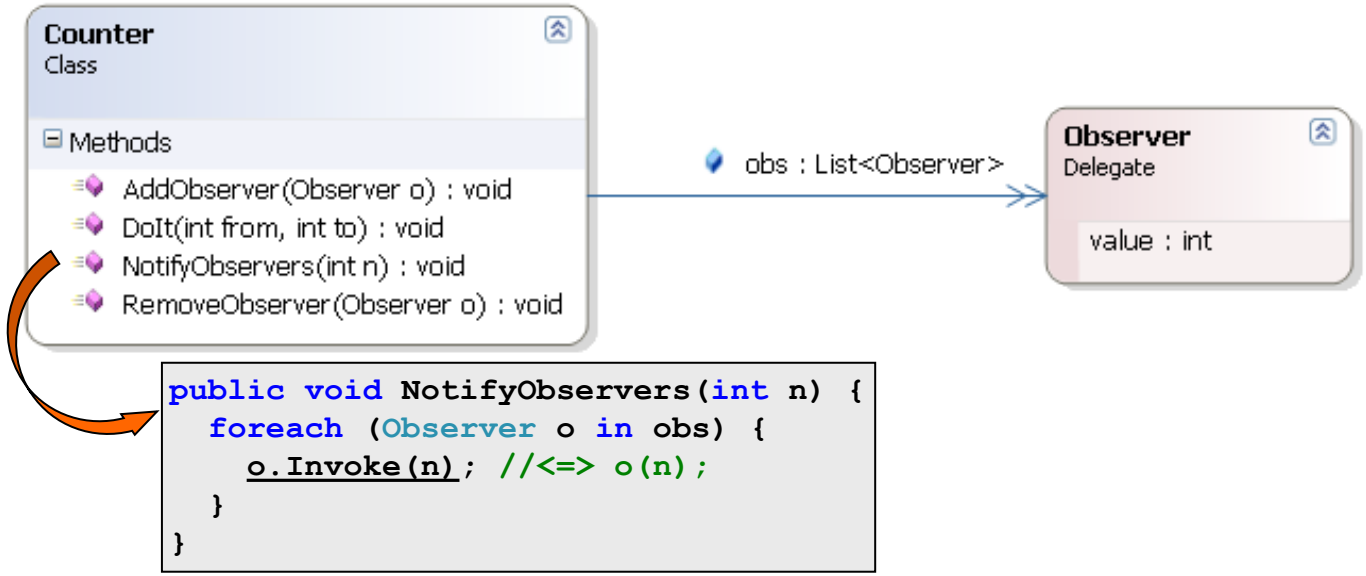
- Com delegate `Observer`:

```
Counter c = new Counter();
FeedbackToConsole fc = new FeedbackToConsole();
c.AddObserver(new Observer(fc.Feedback));
c.DoIt(5, 7);
```

- Em tempo de execução é resolvido o endereço da função que será armazenado no campo `_methodPtr` do objecto `Observer`.

Exemplo… com delegates … Demo12





Comparação entre os dois modelos

- Vantagens dos Delegates sobre as Interfaces como forma de especificação do protótipo de uma função.
  - Cada tipo de observador não precisa de ser uma classe distinta;
  - A função *handler* não tem que ter acessibilidade publica;
  - A função *handler* pode ser um membro de instância ou estático.

Cadeias de *delegates* … Demo13

```
class Counter {
  public List<Observer> obs = new List<Observer>();
  public void AddObserver(Observer o) {
    obs.Add(o);
  }
  public void RemoveObserver(Observer o) {
    obs.Remove(o);
  }
  public void NotifyObservers(int n) {
    //if any callbacks are specified, call them
    foreach (Observer o in obs) {
      o.Invoke(n);
    }
  }
```

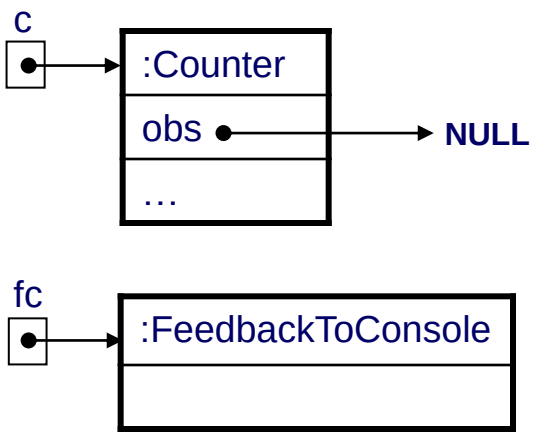
- Substituido por delegates em cadeia.

```
class Counter {
  Observer obs;
  public void AddObserver(Observer o) {
    obs = obs + o;
  }
  public void RemoveObserver(Observer o) {
    obs -= o;
  }
  public void NotifyObservers(int n) {
    obs(n);
  }
```

```
class Counter {
 Observer obs;
 public void AddObserver(Observer o) {
  obs = (Observer)Delegate.Combine(obs, o);
 }
 public void RemoveObserver(Observer o) {
  obs = (Observer)Delegate.Remove(obs, o);
 }
 public void NotifyObservers(int n) {
  foreach (Observer o in obs. GetInvocationList())
   o.Invoke(n);
 }
```

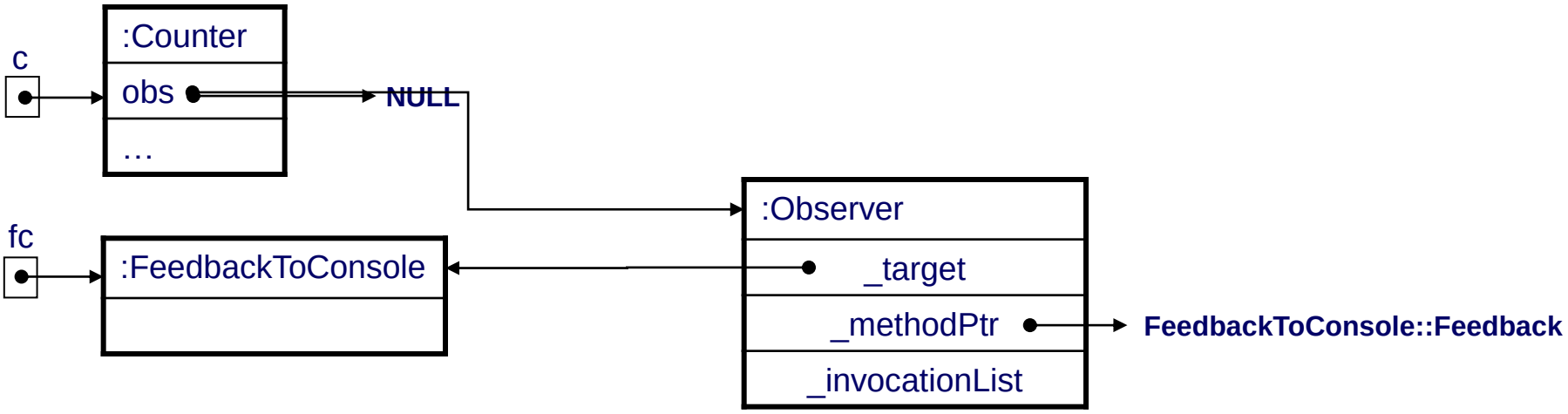
Cadeias de *delegates*… Demo13

```
Counter c = new Counter();
FeedbackToConsole fc = new FeedbackToConsole();
c.AddObserver(new Observer(fc.Feedback));
c.AddObserver(new Observer(Feedback));
c.AddObserver(new Observer(FeedbackBeep.Feedback));
c.DoIt(5, 7);
```



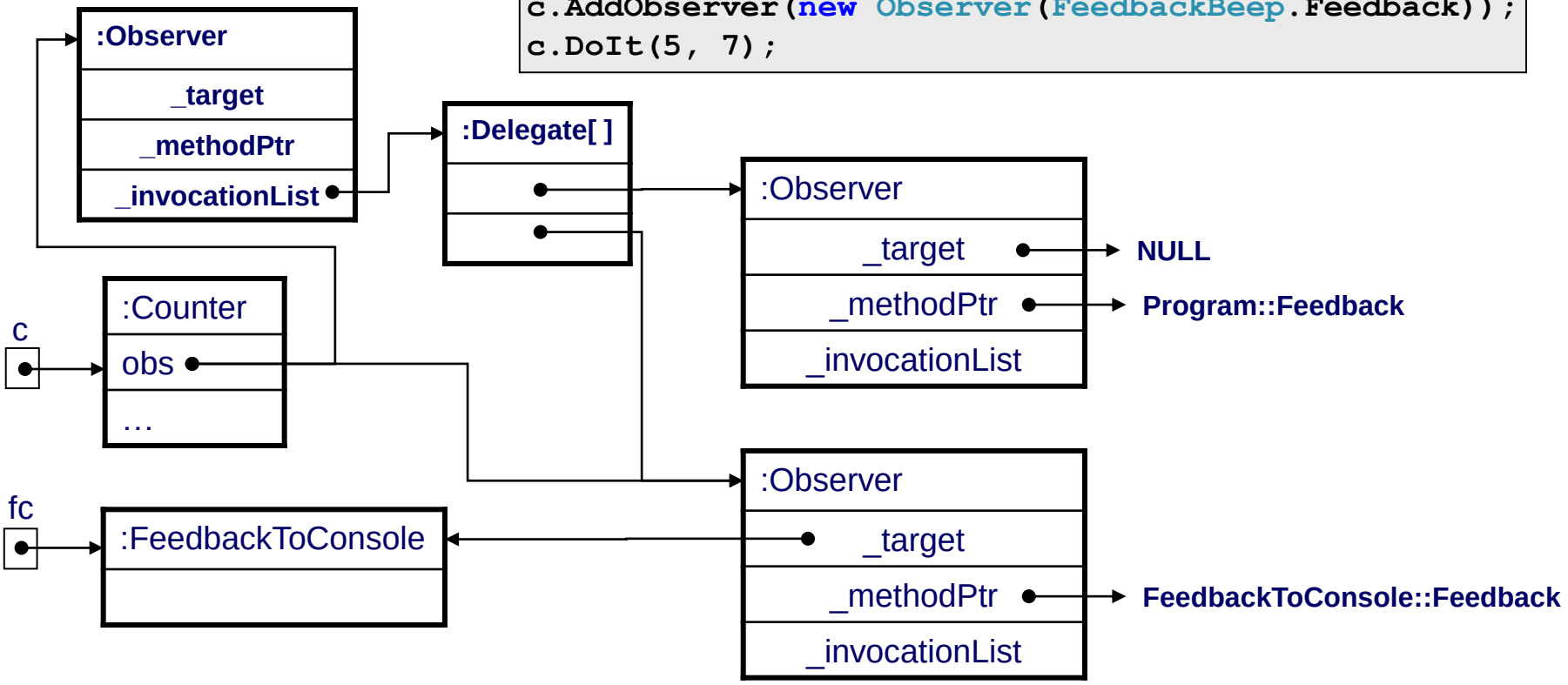
Cadeias de *delegates* … Demo13

```
Counter c = new Counter();
FeedbackToConsole fc = new FeedbackToConsole();
c.AddObserver(new Observer(fc.Feedback));
c.AddObserver(new Observer(Feedback));
c.AddObserver(new Observer(FeedbackBeep.Feedback));
c.DoIt(5, 7);
```



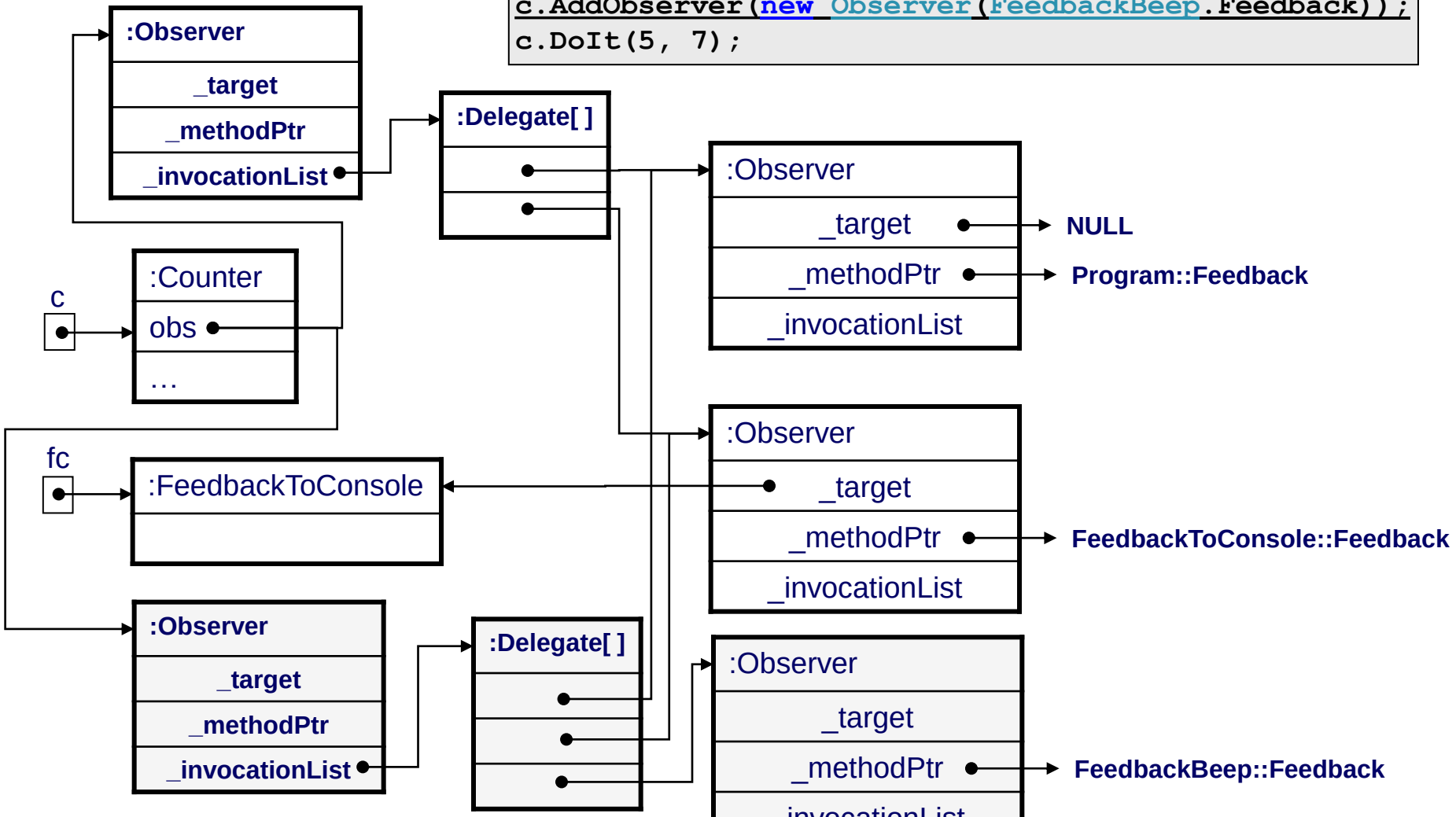
Cadeias de *delegates* … Demo13

```
Counter c = new Counter();
FeedbackToConsole fc = new FeedbackToConsole();
c.AddObserver(new Observer(fc.Feedback));
c.AddObserver(new Observer(Program.Feedback));
c.AddObserver(new Observer(FeedbackBeep.Feedback));
c.DoIt(5, 7);
```



Cadeias de *delegates* … Demo13

```
Counter c = new Counter();
FeedbackToConsole fc = new FeedbackToConsole();
c.AddObserver(new Observer(fc.Feedback));
c.AddObserver(new Observer(Feedback));
c.AddObserver(new Observer(FeedbackBeep.Feedback));
c.DoIt(5, 7);
```



Cadeias de *delegates*… Resumo

- *Delegates* suportam encadeamento:
  - método estático System.Delegate.Combine;
  - operador += em C# usa *Combine.*

- Resultado do encadeamento de duas instâncias de *delegate* é uma terceira instância *delegate:*
  - quando invocada, resulta na invocação de todas as instâncias de *delegate* da cadeia.

Delegates … Resumo

- Versão *Object-Oriented* e *type-safe* de ponteiro para função;
  - ponteiro para método de tipo ou de instância.
- Tipos *delegate:*
  - derivados de `System.Delegate` e de `System.MulticastDelegate`;
  - definem assinatura de método.
- Instâncias de tipos *delegate* referem:
  - um método com a assinatura definida no tipo *delegate* (`_methodPtr`);
  - um objecto do tipo que contém o método (ou *null*, no caso de métodos estáticos) (`_target`).
- Parametrização com comportamento:
  - Por exemplo: indicar critério de comparação a um algoritmo genérico de ordenação.
- Notificações
  - Por exemplo: indicar reacção a botão pressionado em aplicação visual.

Agenda

- Padrões de Desenho
- Padrão *Observer* ou *Publish/Subscribe*
- Padrão *Observer* com *Delegates*
- Eventos

Eventos

- Padrão comum em tipos que notificam acontecimentos (eventos)
  - campo de tipo *delegate*
  - método para registar instâncias de *delegate*
  - método para remover instâncias de *delegate*
  - notificação através da invocação dos *delegates* registados

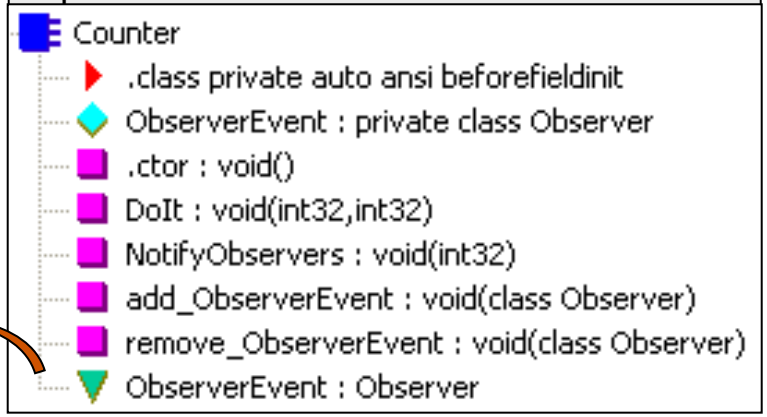
- Padrão concretizado através de membros *event*
  - implementação automática do padrão
  - informação explícita em *metadata* sobre existência de evento
    - definição explícita do conjunto de eventos de um tipo

Exemplo anterior com eventos … Demo14

```
class Counter {
  Observer obs;
  public void AddObserver(Observer o) {
    obs += o;
  }
  public void RemoveObserver(Observer o) {
    obs -= o;
  }
  public void NotifyObservers(int n) {
    Observer tmp = obs;
    if(tmp != null)
      tmp.Invoke(n);
  }
```

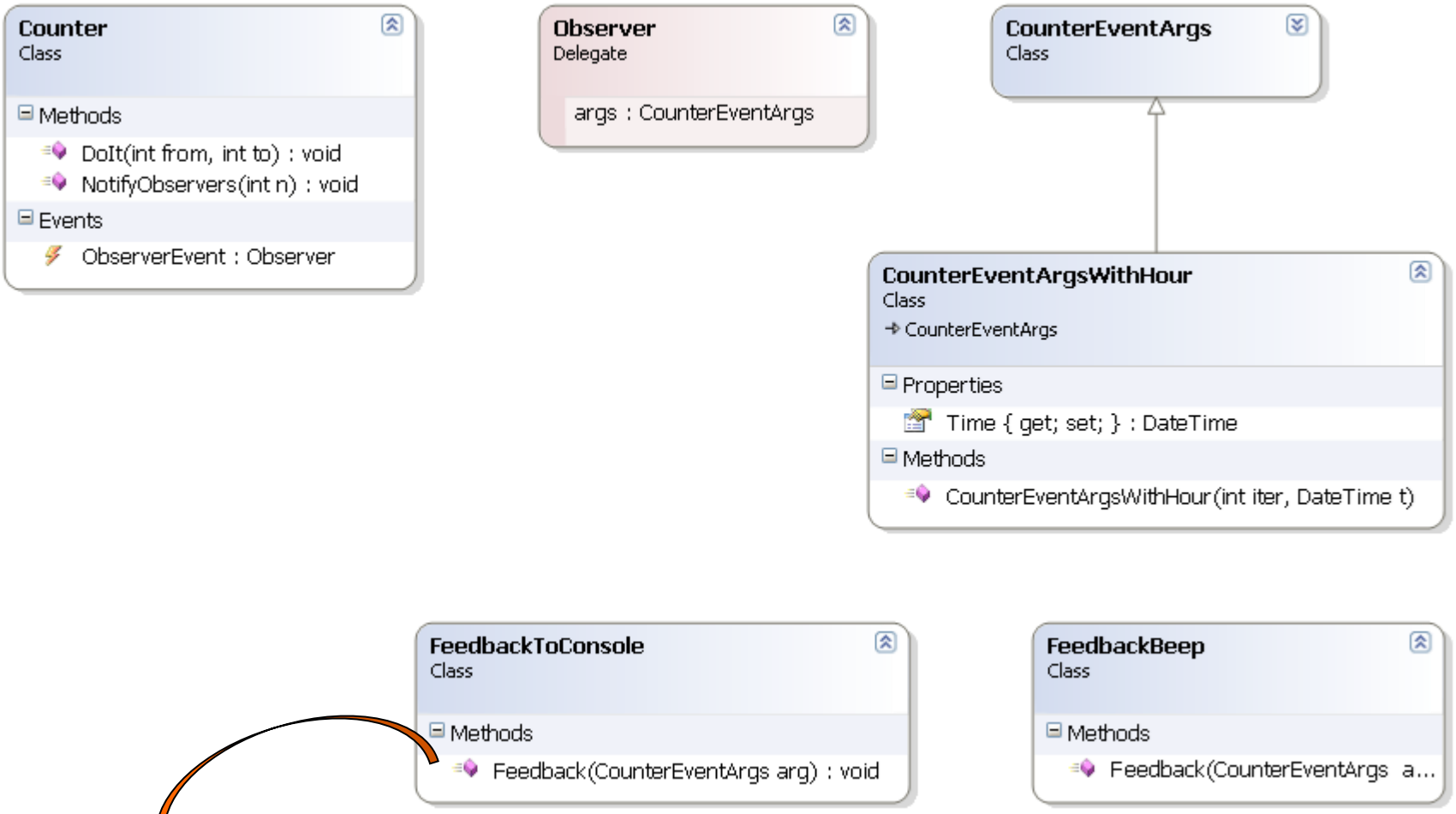
- Com eventos

```
class Counter {
  public event Observer ObserverEvent;
  public void NotifyObservers(int n) {
    Observer tmp = ObserverEvent;
    if(tmp != null)
      tmp.Invoke(n);
```




```
.event Observer ObserverEvent
{
  .addon instance void Counter::add_ObserverEvent(class Observer)
  .removeon instance void Counter::remove_ObserverEvent(class Observer)
} // end of event Counter::ObserverEvent
```

Passagem de parâmetros a *handlers* de Eventos … Demo15




```
CounterEventArgsWithHour argHour = arg as CounterEventArgsWithHour;
if(argHour == null)
  Console.WriteLine(argHour.Time + " ------- Item = " + argHour.Iter);
```

Implementação explícita de eventos … Demo17

```
class Counter {
  private List<EventHandler> obs = new List<EventHandler>();
  public event EventHandler ObserverEvent {
    add {
      obs.Add(value);
      Console.WriteLine("Observer adicionado: " + value.GetType());
    }
    remove {
      throw new NotSupportedException();
    }
  }
  public void NotifyObservers(int n) {
    foreach(EventHandler h in obs)
      h.Invoke(this, new CounterEventArgsWithHour(n, DateTime.Now));
  }
...
```