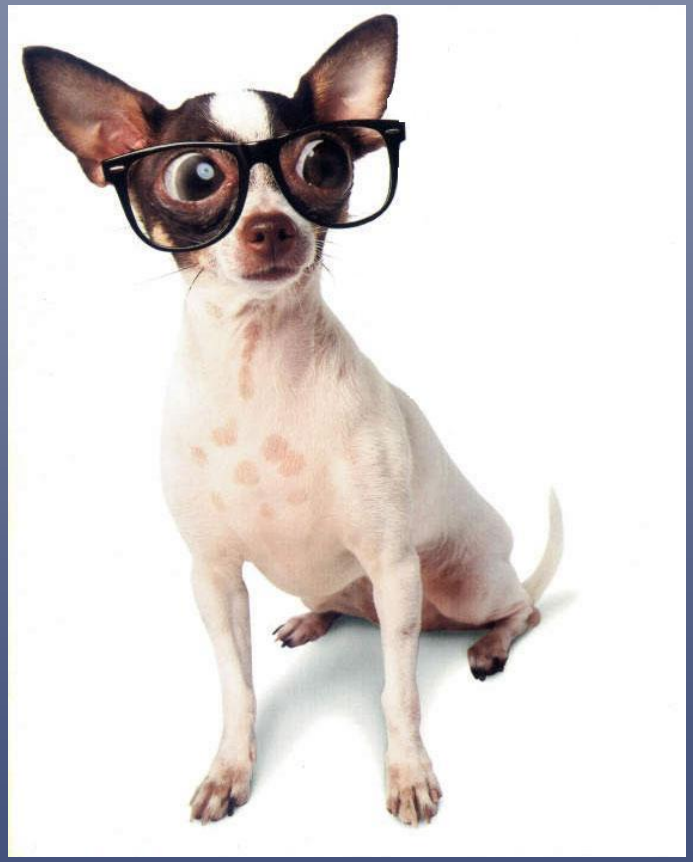
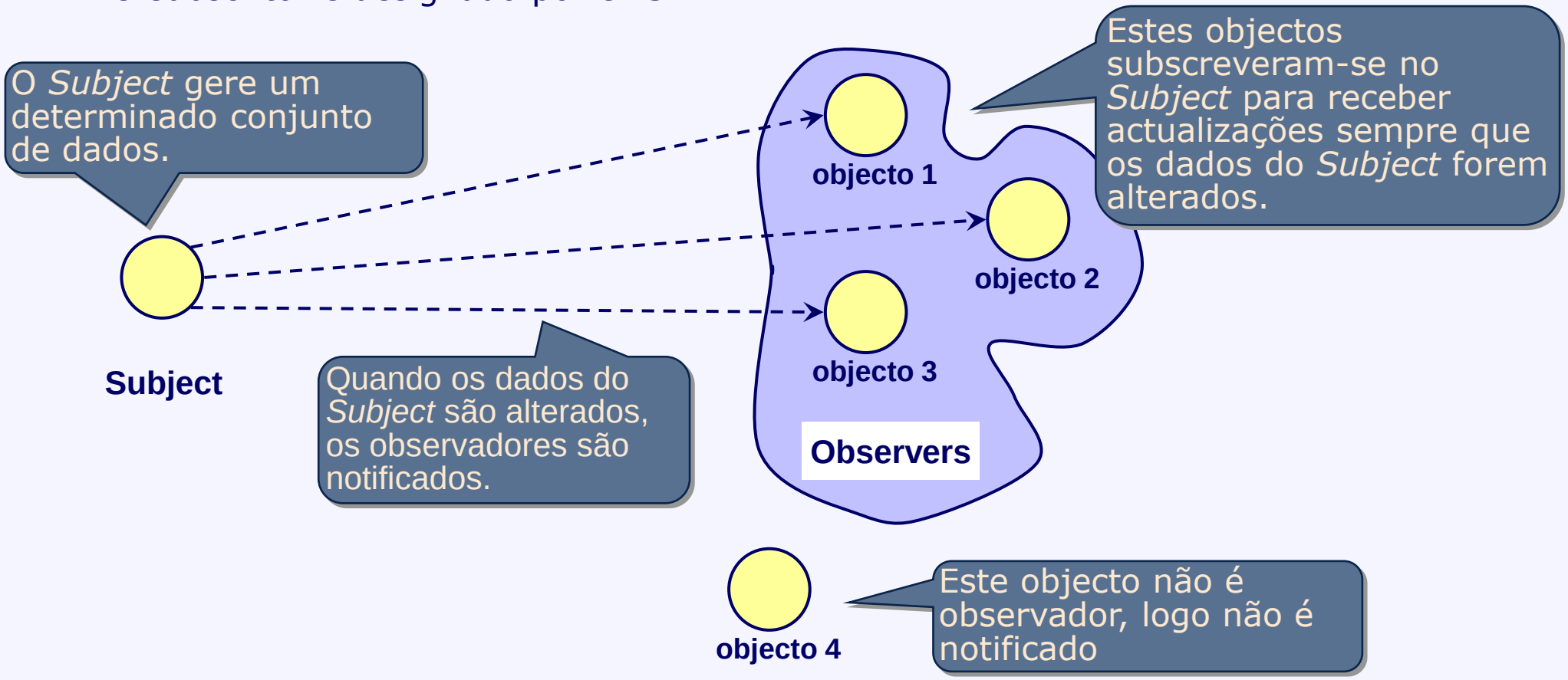
Padrão *Observer*



Notificador + Subscritor = Padrão *Observer*

Padrão Observer

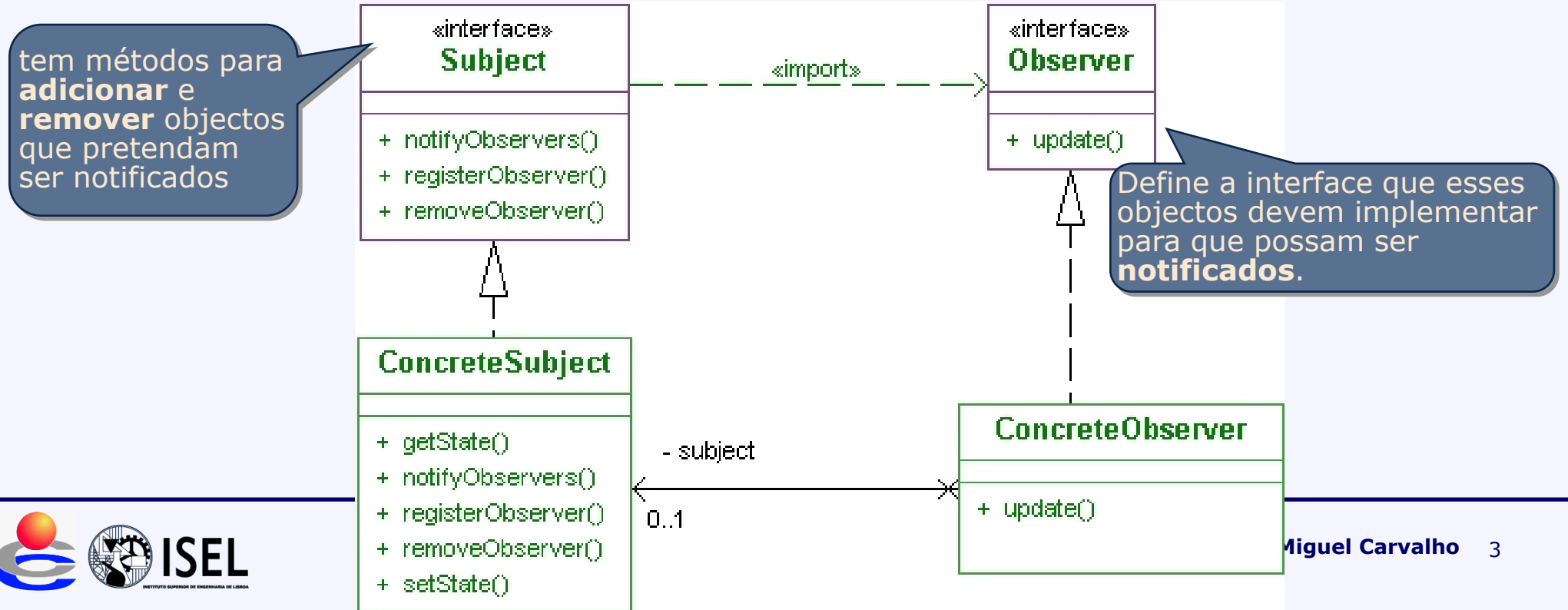
- O notificador é designado por *SUBJECT*;
- O subscritor é designado por *OBSERVER*.



Padrão Observer…

O padrão `Observer` define uma **dependência de um para muitos** entre objectos tal que, **quando um objecto muda de estado, todos os seus dependentes são notificados e actualizados automaticamente**.

- Os **observadores** são dependentes do **subject**, tal que quando o *subject* muda de estado os observadores são notificados.
- Dependendo do tipo de notificação os observadores também poderão ser actualizados com novos valores.



Fracamente acoplados

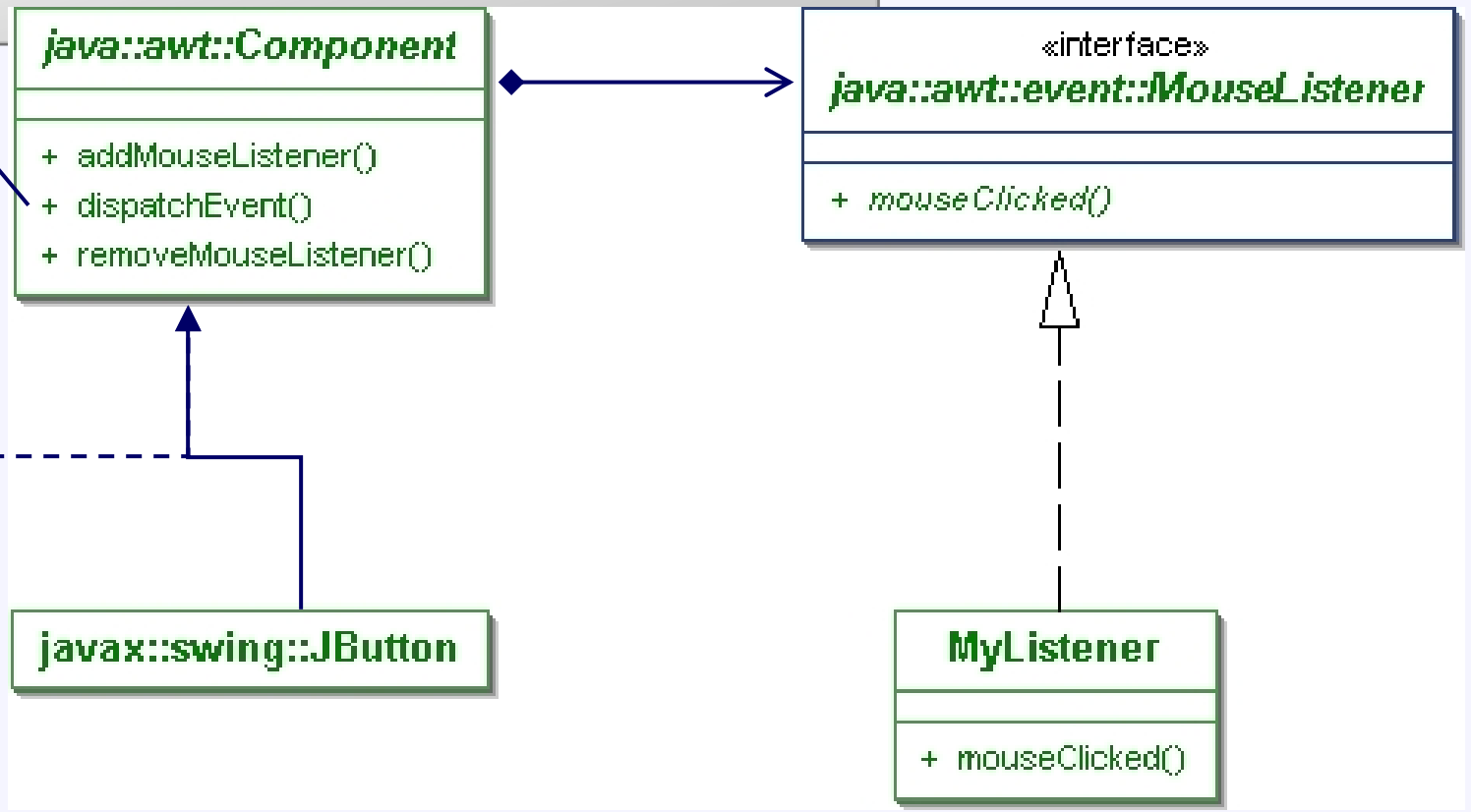
Quando dois objectos são fracamente acoplados, estes podem interactuar tendo muito pouco conhecimento um do outro.

O padrão *Observer* oferece um desenho OO onde o *subject* e os **observadores** são fracamente acoplados, porque:

- A única coisa que o *subject* conhece acerca do seu observador é que implementa uma **determinada interface**.
- Podem ser adicionados **novos observadores** a qualquer momento.
- **Não é necessário modificar** o *subject* para adicionar novos tipos de observadores.
- Alterações ao *subject* ou ao observador, **não afectarão** a outra parte.

Exemplo

```
for(MouseListener l : listeners){
  switch(e.getID()){
    case MouseEvent.MOUSE_CLICKED: l.mouseClicked();
    case ...
  }
}
```

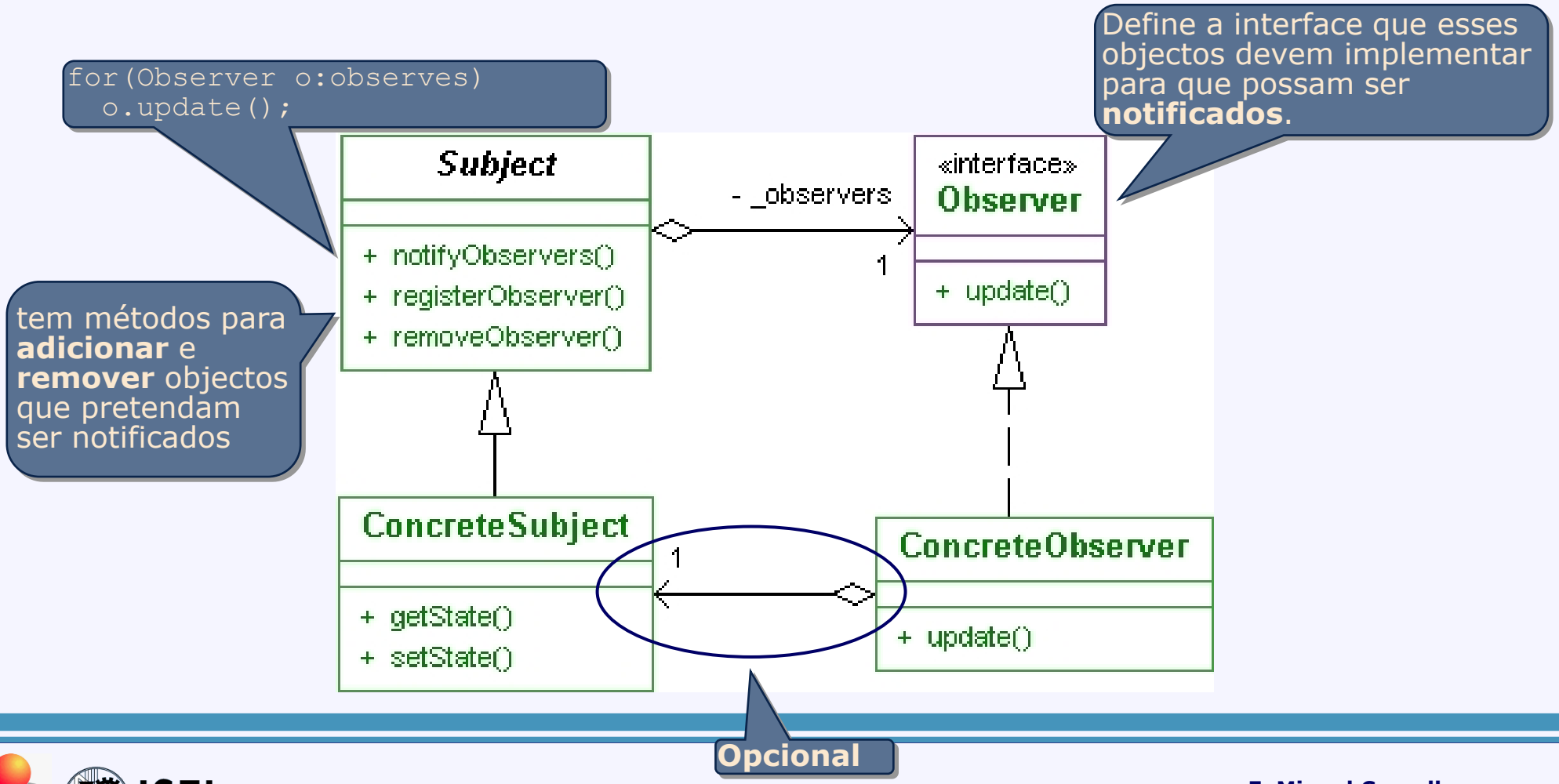


Padrão *Observer* - Participantes

Nome do Participante	Descrição
`Subject` `(Component)`	• Conhece os seus *observers*. • Disponibiliza interface para registo e remoção de registo de *observers*
`Observer` `(MouseListener)`	Interface para notificação de alterações, para ser implementada por quem pretende ser notificado.
`ConcreteSubject` `(JButton, JTextField, etc)`	Notifica os *observers* quando o seu estado muda.
`ConcreteObserver` `(MyListener)`	Implementa a interface `Observer` de modo a manter o seu estado consistente com o `Subject`

Padrão Observer…

O padrão Observer define uma **dependência de um para muitos** entre objectos tal que, **quando um objecto muda de estado, todos os seus dependentes são notificados e actualizados automaticamente**.



Padrão *Observer*

Característica	Descrição
Nome	***Observer***
Categoria	Comportamento - Objectos
Objectivo	Definir uma dependência entre um objecto e outros dependentes deste, de modo a quando o estado do objecto muda todos os dependentes sejam notificados
Aplicabilidade	• Uma abstracção tem dois aspectos, um dependente de outro. Encapsular cada aspecto num objecto permite alterar e reutilizar cada um independentemente; • Alterações num objecto provocam alterações noutros cujo número não é conhecido; • Quando um objecto necessita de notificar outros, sem ter qualquer conhecimento de que tipo são (e não se pretende que exista dependência forte).
Nome alternativo	***Dependents***, ***Publish-Subscribe***

MVC

Adapter

		Propósito		
		Criação	**Estrutura**	**Comportamento**
Âmbito de acção	**Classes**	**Factory Method**	**Adapter** (class)	**Interpreter** Template Method
	Objectos	Abstract Factory **Builder** Prototype **Singleton**	Adapter (object) **Bridge** Composite Decorator **Facade** **Flyweight** **Proxy**	**Chain of Responsibility** **Command** **Iterator** **Mediator** **Memento** Observer State Strategy **Visitor**

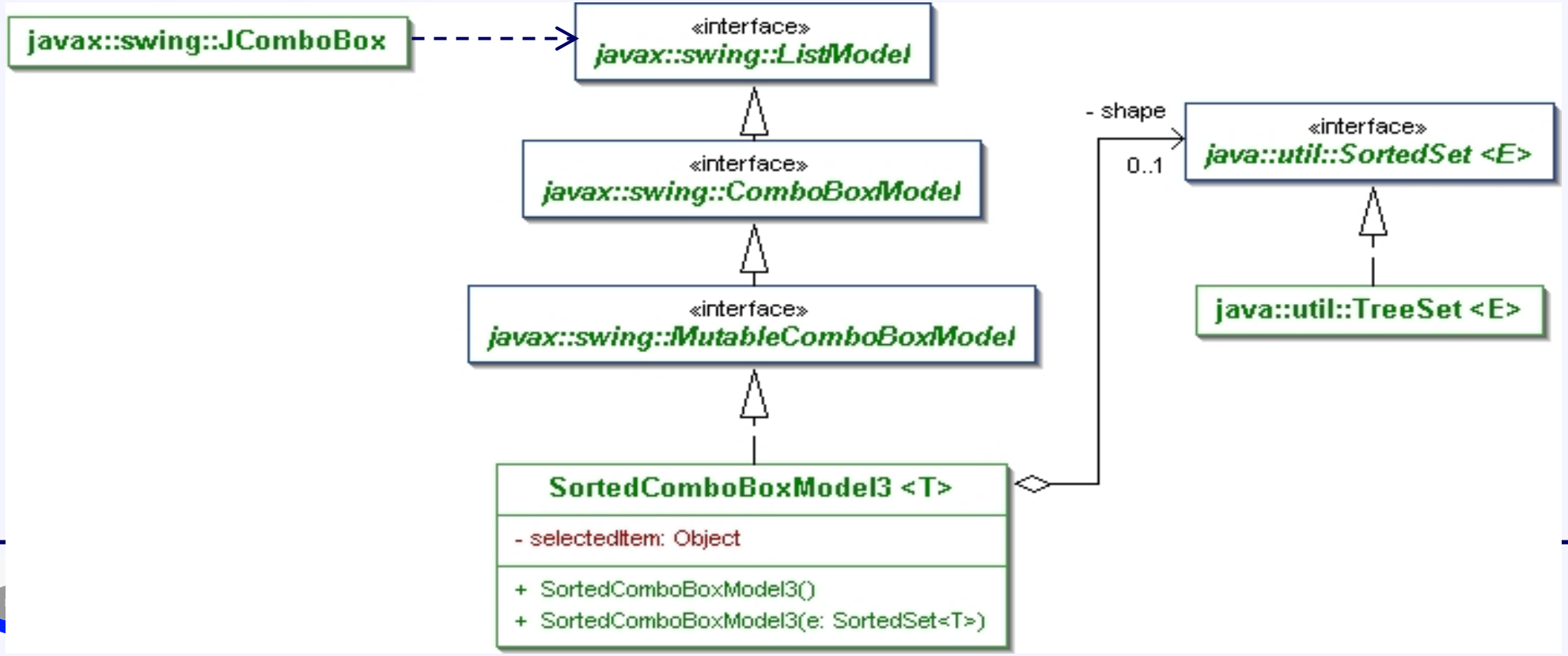
Exemplo

Contexto: `JComboBox` usa como modelo de dados uma instância de uma classe compatível com `ListModel`.

Objectivo: Apresentar os itens de uma `JComboBox` de forma ordenada.

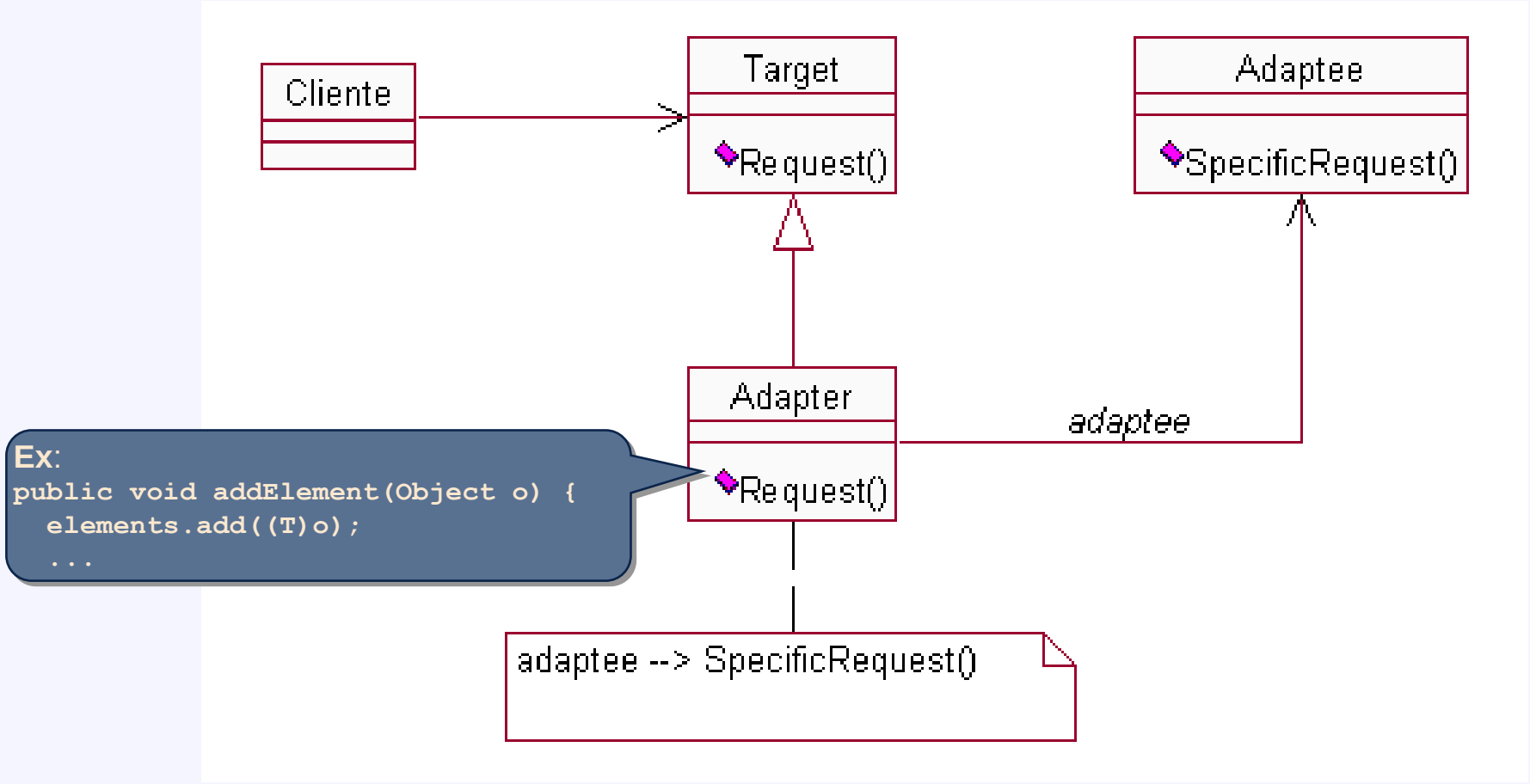
Solução: Implementar um modelo de dados compatível com `ListModel` que mantenha os elementos ordenados.

Desenho: Aproveitar a funcionalidade disponibilizada pelas classes derivadas de `SortedSet<E>`, na implementação de um `SortedListModel<E>`.



Padrão *adapter* (objectos)

Âmbito de acção	Propósito: Criação	Propósito: Estrutura	Propósito: Comportamento
Classes			
Objectos		■	



Padrão *adapter* (objectos)

Âmbito de acção	Propósito: Criação	Propósito: Estrutura	Propósito: Comportamento
Classes			
Objectos		■	

Característica	Descrição
Nome	*Adapter (objectos)*
Nome alternativo	*Wrapper*
Categoria	**Estrutura – Objectos**
Objectivo	Converte a interface de uma classe noutra interface conhecida pelo cliente. Permite que classes que não poderiam interagir devido a incompatibilidades entre as interfaces possam trabalhar em conjunto.
Aplicabilidade	• quando se quer usar uma classe já existente e a sua interface não é a desejada pelo cliente. • quando se quer criar uma classe reutilizável que coopera com classes não relacionadas, ou não previstas, isto é, classes que não tenham necessariamente interfaces compatíveis. • (apenas para o padrão *adapter* a objectos) quando é necessário usar várias subclasses existentes, mas é impraticável adaptar as suas interfaces fazendo um *subclassing* de cada uma. Um objecto *adapter* pode adaptar a interface da super classe.

Padrão *adapter* (objectos) - Participantes

Nome do Participante	Descrição
`Target` (`ListModel`)	Define a interface específica do domínio conhecida por `Client`.
`Client` (`JComboBox`)	Colabora com os objectos que estão em conformidade com a interface `Target`.
`Adaptee` (`SortedSet`)	Define a interface existente que necessita de ser adaptada.
`Adapter` (`SortedListModel`)	Adapta a interface de `Adaptee` à interface `Target`.

Exemplo 2

Contexto: Um simples ecrã para visualização de ficheiros de imagens, localizados numa determinada directoria.

Requisito:

- Cada ficheiro de imagem pode ser carregado na aplicação num objecto da classe `ImageIcon`.
- Um contentor UI aloja e faz o render de instâncias de subtipos de `JComponent`.

```
File directory = new File(_txtPath.getText());
for (File f : directory.listFiles())
  p2.add(new ImageIcon(f.getPath()));
```

`ImageIcon` não é compatível com `Component` logo não pode ser adicionado a um `JPanel`

Exemplo… Solução

```
File directory = new File(_txtPath.getText());
for (File f : directory.listFiles())
  p2.add(new IconAdapter(new ImageIcon(f.getPath())));
```

Encapsular a instância de `ImageIcon` num `ImageAdapter` que extende de `JComponent`

Adapta qualquer objecto de um subtipo de `Icon`.

```
public class IconAdapter extends JComponent{
  public IconAdapter(Icon i){
    _icon = i;
  }
  protected void paintComponent(Graphics g) {
    _icon.paintIcon(this, g, 0, 0);
  }
  public int getWidth() {
    return _icon.getIconWidth();
  }
  public int getHeight() {
    return _icon.getIconHeight();
  }
  public Dimension getPreferredSize() {
    return new Dimension(getWidth(), getHeight());
  }
  private Icon _icon;
}
```

Adapter

		Propósito		
		Criação	**Estrutura**	**Comportamento**
Âmbito de acção	**Classes**	**Factory Method**	**Adapter** (class)	**Interpreter** Template Method
	Objectos	Abstract Factory **Builder** Prototype **Singleton**	Adapter (object) **Bridge** Composite Decorator **Facade** **Flyweight** **Proxy**	**Chain of Responsibility** **Command** **Iterator** **Mediator** **Memento** Observer State Strategy **Visitor**

Adapter

		Propósito		
		Criação	**Estrutura**	**Comportamento**
Âmbito de acção	**Classes**	**Factory Method**	**Adapter** (class)	**Interpreter** Template Method
	Objectos	Abstract Factory **Builder** Prototype **Singleton**	Adapter (object) **Bridge** Composite Decorator **Facade** **Flyweight** **Proxy**	**Chain of Responsibility** **Command** **Iterator** **Mediator** **Memento** Observer State Strategy **Visitor**

Padrão *adapter* (classes)

		Propósito		
		Criação	Estrutura	Comportamento
Âmbito de acção	Classes			
	Objectos			

Uma variante do padrão *adapter* consiste na sua aplicação ao nível das relações entre classes, no lugar de objectos.

Desvantagens:

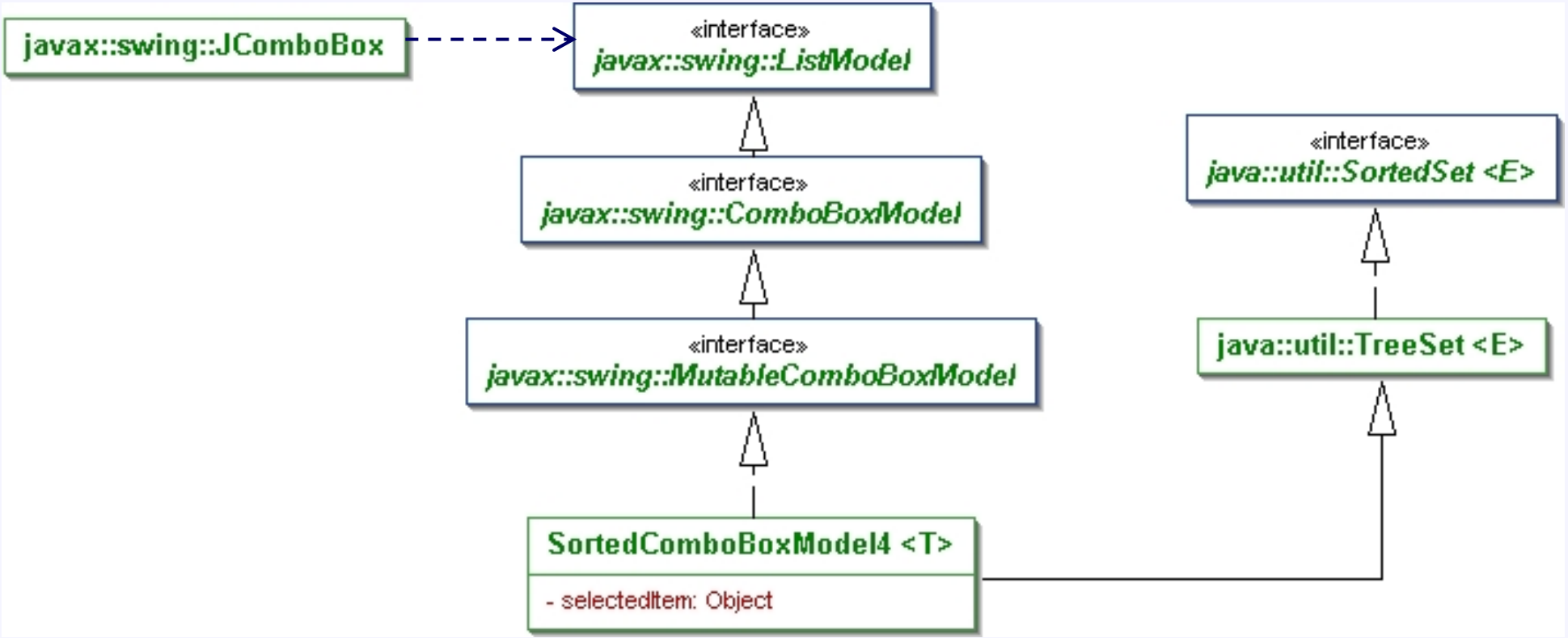
- A relação é estabelecida em tempo de compilação e obriga a ter um tipo concreto `Adapter` para cada tipo adaptado.

  ➔ Ao contrário do exemplo anterior em que `IconAdapter` pode adaptar qualquer subtipo de `Icon`, que pode ser actualizado em tempo de execução.

- Limitação da herança única de classes.

  ➔ Não seria possível implementar a classe `IconAdapter` que derivasse simultaneamente de `JComponet` e `ImageIcon`.

Exemplo


javax::swing::JComboBox
«interface»
javax::swing::ListModel
«interface»
javax::swing::ComboBoxModel
«interface»
javax::swing::MutableComboBoxModel
SortedComboBoxModel4 <T>
- selectedItem: Object
«interface»
java::util::SortedSet <E>
java::util::TreeSet <E>

Padrão *adapter* (classes)

Âmbito de acção	Propósito: Criação	Propósito: Estrutura	Propósito: Comportamento
Classes		■	
Objectos