Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – Observer (19)

Prof. Osmar de Oliveira Braz Junior

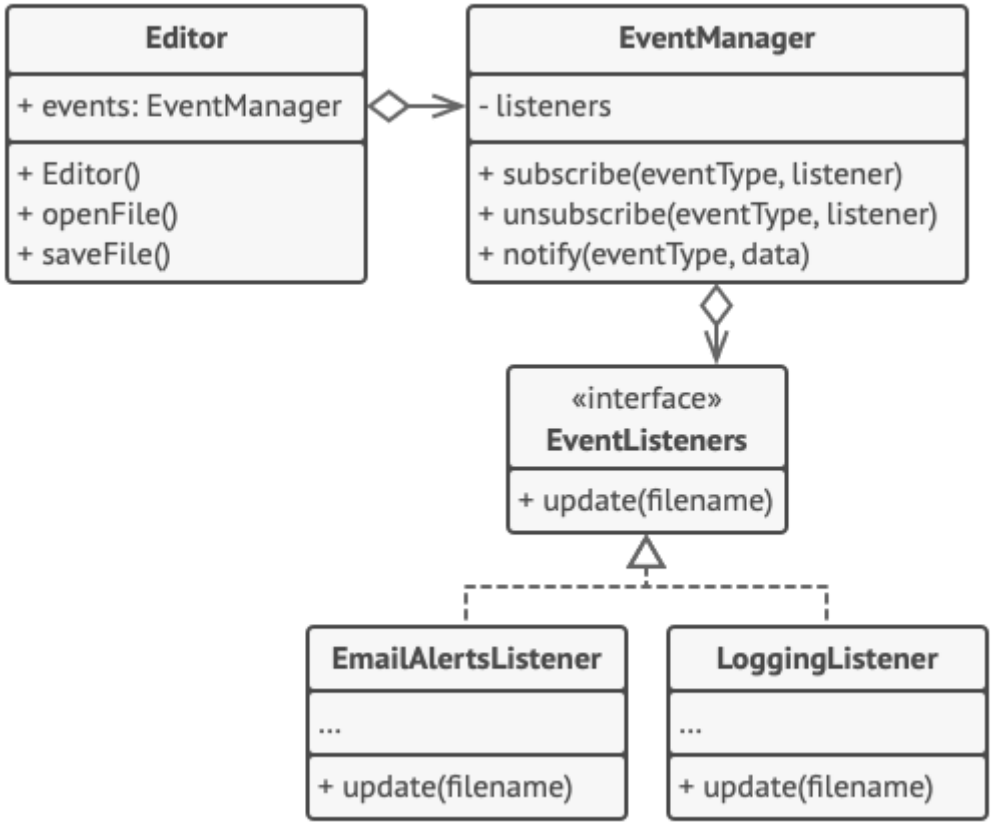
Prof. Richard Henrique de Souza

Objetivos

- Aplicar padrão comportamental ***Memento*** em situação problema.

19. Observer

Pseudocódigo: Neste exemplo o padrão Observer permite que um objeto editor de texto notifique outros objetos de serviço sobre mudanças em seu estado.

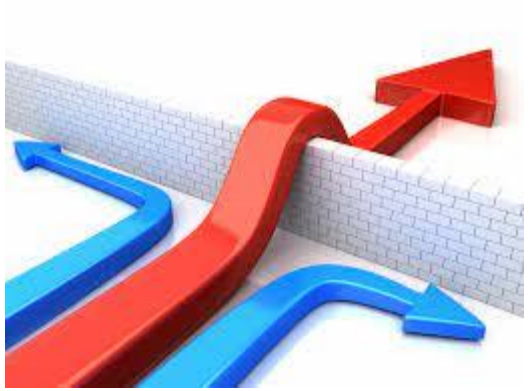


https://refactoring.guru/pt-br/design-patterns/observer

https://github.com/osmarbraz/observer

Importante

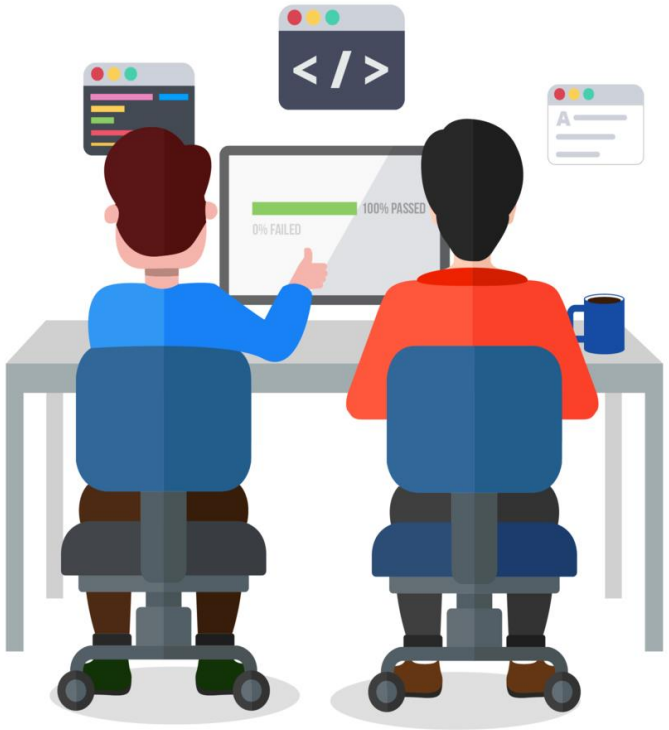
Siga os ROTEIROS !!!



Atividade em Grupo

Para esta atividade crie grupos de 2 alunos, para desenvolver a atividade segundo ***Pair Programming.***

Navegador



Piloto

Extreme Programming, metodologia ágil criada por Kent Beck em 1996.

Pair Programming

- Um é o **piloto**, responsável por escrever o código, o outro o navegador, acompanha a escrita de código e verificar se está de acordo com os **padrões do projeto** e de encontro à solução necessária.
- A intenção desta técnica é **evitar** erros de lógica, e ter um código mais confiável e melhor estruturado, utilizando-se para isso a máxima de que “**duas cabeças pensam melhor do que uma**”.

Preparação do ambiente

- Acesso a ferramenta **draw.io(https://app.diagrams.net/)** para realizar a modelagem.
- Escolha a sua linguagem de programação de preferência
- Escolha uma IDE ou o **git.dev**
- Crie um repositório no github(https://github.com/) para que todos os membros da equipe possam colaborar no desenvolvimento.

Atividade prática

1

19. Observer
Propósito

- Definir uma dependência um-para-muitos entre objetos de forma que quando um objeto muda de estado, os outros são notificados e se atualizam.
- Também conhecido como: Dependents, Publish-Subscribe, Observador, Assinante do evento, Event-Subscriber, Escutador ou Listener.

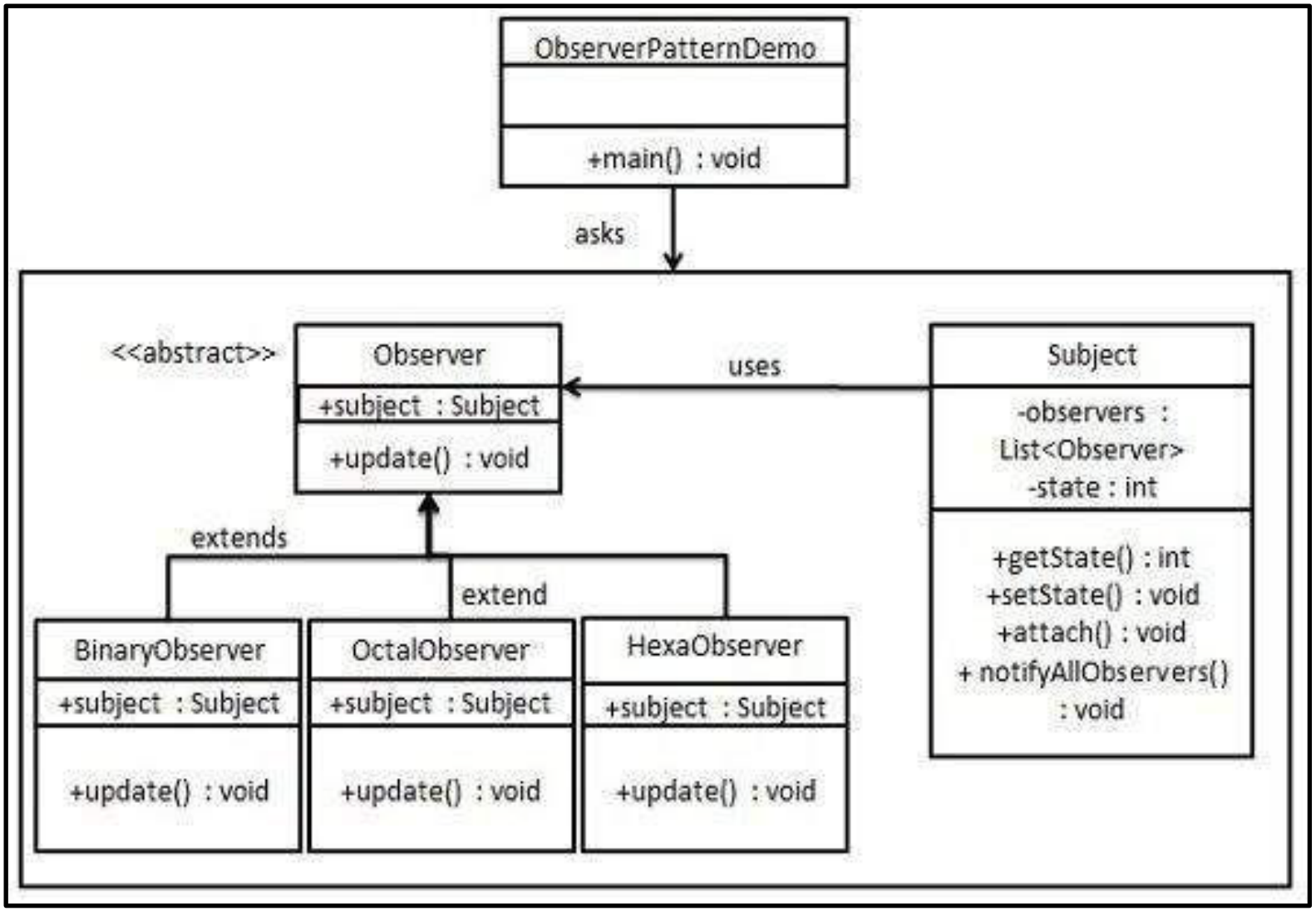
19. Observer

- Usar este padrão quando…
  - Uma abstração possui dois aspectos e é necessário separá-los em dois objetos para variá-los;
  - Alterações num objeto requerem atualizações em vários outros objetos não-determinados;
  - Um objeto precisa notificar sobre alterações em outros objetos que, a princípio, ele não conhece.

19. Observer

- Vantagens e desvantagens
  - Flexibilidade:
    - Observável e observadores podem ser quaisquer objetos;
    - Acoplamento fraco entre os objetos: não sabem a classe concreta uns dos outros;
    - É feito broadcast da notificação para todos, independente de quantos;
    - Observadores podem ser observáveis de outros, propagando em cascata.

19. Observer


ObserverPatternDemo
+main() : void
asks
<<abstract>>
Observer
+subject : Subject
+update() : void
uses
Subject
-observers : List<Observer>
-state : int
+getState() : int
+setState() : void
+attach() : void
+ notifyAllObservers() : void
extends
extend
BinaryObserver
+subject : Subject
+update() : void
OctalObserver
+subject : Subject
+update() : void
HexaObserver
+subject : Subject
+update() : void

19. Observer
Passo 1

Criar classe de Subject .

Subject.java

```
import java.util.ArrayList;
import java.util.List;

public class Subject {

   private List<Observer> observers = new ArrayList<Observer>();
   private int state;

   public int getState() {
      return state;
   }

   public void setState(int state) {
      this.state = state;
      notifyAllObservers();
   }

   public void attach(Observer observer){
      observers.add(observer);
   }

   public void notifyAllObservers(){
      for (Observer observer : observers) {
         observer.update();
      }
   }
}
```

19. Observer
Passo 2

Criar classe Observer.

Observer.java

```
public abstract class Observer {
    protected Subject subject;
    public abstract void update();
}
```

19. Observer
Passo 3

Crie classes concretas de observadores

BinaryObserver.java

```
public class BinaryObserver extends Observer{

   public BinaryObserver(Subject subject){
      this.subject = subject;
      this.subject.attach(this);
   }

   @Override
   public void update() {
      System.out.println( "Binary String: " + Integer.toBinaryString( subject.getState() ) );
   }
}
```

19. Observer
Passo 3 - Continuação

OctalObserver.java

```
public class OctalObserver extends Observer{

   public OctalObserver(Subject subject){
      this.subject = subject;
      this.subject.attach(this);
   }

   @Override
   public void update() {
     System.out.println( "Octal String: " + Integer.toOctalString( subject.getState() ) );
   }
}
```

19. Observer
Passo 3 - Continuação

HexaObserver.java

```
public class HexaObserver extends Observer{

   public HexaObserver(Subject subject){
      this.subject = subject;
      this.subject.attach(this);
   }

   @Override
   public void update() {
      System.out.println( "Hex String: " + Integer.toHexString( subject.getState() ).toUpperCase() );
   }
}
```

19. Observer
Passo 4

Use o Subject e objetos observadores concretos.

ObserverPatternDemo.java

```
public class ObserverPatternDemo {
   public static void main(String[] args) {
      Subject subject = new Subject();

      new HexaObserver(subject);
      new OctalObserver(subject);
      new BinaryObserver(subject);

      System.out.println("First state change: 15");
      subject.setState(15);
      System.out.println("Second state change: 10");
      subject.setState(10);
   }
}
```

19. Observer
Passo 5

- Terminamos
  - Teste sua implementação

Compile e **Mostre** o código para o professor
  - Pense, o que você fez aqui ?

Lembre de salvar no seu github

Conclusão

Os padrões comportamentais tem como principal função designar responsabilidades entre objetos.

Referências

- PRESSMAN, Roger; MAXIM, Bruce. Engenharia de software: uma abordagem profissional. 8.ed. Bookman, 2016. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788580555349

- SOMMERVILLE, Ian. Engenharia de software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. E-book. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/2613/epub/0

- LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788577800476

Fim