Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 17 Padrões de Comportamento - Mediator

Intenção

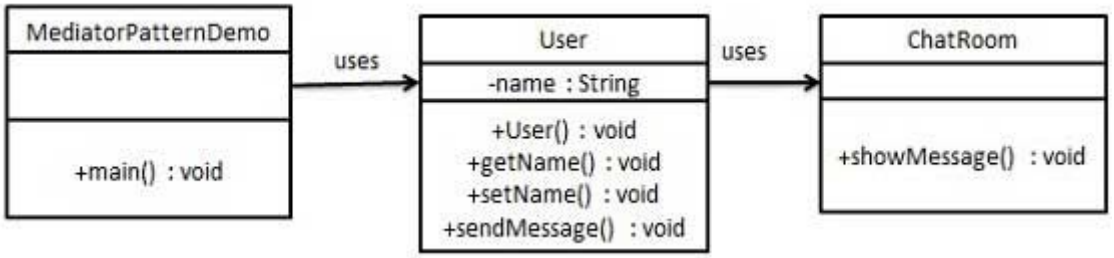
Definir um objeto que encapsula a informação de como um conjunto de outros objetos interagem entre si. Promove o acoplamento fraco, permitindo que você altere a forma de interação sem alterar os objetos que interagem.

Usar este padrão quando…

- Um conjunto de objetos se comunica de uma forma bem determinada, porém complexa;
- Reutilizar uma classe é difícil pois ela tem associação com muitas outras;
- Um comportamento que é distribuído entre várias classes deve ser extensível sem ter que criar muitas subclasses.

Vantagens e desvantagens

- Limita extensão por herança:
  - Para estender ou alterar o comportamento, basta criar uma subclasse do mediador.
- Desacopla objetos:
  - Desacoplamento promove o reuso.
- Simplifica o protocolo:
  - Relações Colleagues x Mediator são mais simples de manter do que muitas espalhadas;
  - Fica mais claro como os objetos interagem.
- Exagero pode levar a sistema monolítico.



Passo 1
Crie uma classe mediadora.

ChatRoom.java

```
import java.util.Date;

public class ChatRoom {
   public static void showMessage(User user, String message){
      System.out.println(new Date().toString() + " [" + user.getName() + "] : " + message);
   }
}
```

Passo 2
Criar classe de usuário

User.java

```
public class User {
   private String name;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public User(String name){
      this.name  = name;
   }

   public void sendMessage(String message){
      ChatRoom.showMessage(this,message);
   }
}
```

Passo 3
Use o objeto Usuário para mostrar as comunicações entre eles.

MediatorPatternDemo.java

```
public class MediatorPatternDemo {
   public static void main(String[] args) {
      User robert = new User("Robert");
      User john = new User("John");

      robert.sendMessage("Hi! John!");
      john.sendMessage("Hello! Robert!");
   }
}
```

Passo 4

Teste sua implementação!