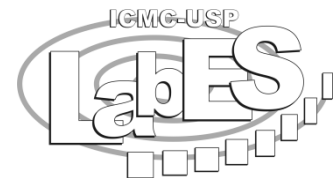




Design Patterns

Lucas Bueno Ruas de Oliveira
Profa. Elisa Yumi Nakagawa

SSC 122 – Engenharia de Software II
1. Semestre 2010



Classificação - Revisão

- Padrões vêm sendo utilizados há muito tempo; porém, não havia uma documentação formal sobre eles.
- Gamma et al. (1994) definiu 23 padrões que possuem aplicação em diversas áreas.
- Os padrões são classificados em três grupos:
 - Padrões de Estrutura
 - Padrões de Comportamento
 - Padrões de Criação

Padrões de Estrutura - Revisão

- Auxiliam o programador na composição dos objetos.
- Proporcionam economia de código.
- Torna o entendimento do código mais fácil.

Padrões de Estrutura - Revisão

- *Adapter* (Adaptador)
- *Bridge* (Ponte)
- *Composite* (Compositor)
- *Decorator* (Decorador)
- *Facade* (Fachada)
- *Flyweight*
- *Proxy* (Procurador)

Padrões de Criação - Revisão

- Isolar do cliente os detalhes do processo de instanciação de um objeto
- Permite que o sistema se torne menos dependente de como os objetos são criados.

Padrões de Criação - Revisão

- *Factory Method* (Método Fábrica)
- *Abstract Factory* (Fábrica Abstrata)
- *Builder* (Construtor)
- *Prototype* (Protótipo)
- *Singleton* (Objeto Único)

Padrões de Comportamento

- São padrões associados aos algoritmos e às responsabilidades relacionadas a cada objeto.
- Por meio dos padrões de comportamento, podemos alterar a maneira como sistema comporta-se.
- É possível adicionarmos novas funcionalidades a um objeto existente, por meio da inclusão de novos objetos.

Padrões de Comportamento

- *Chain of Responsibility* (Cadeia de Responsabilidades)
- *Command* (Comando)
- *Interpreter* (Interpretador)
- *Iterator*
- *Mediator* (Mediador)
- *Memento* (Recordador)

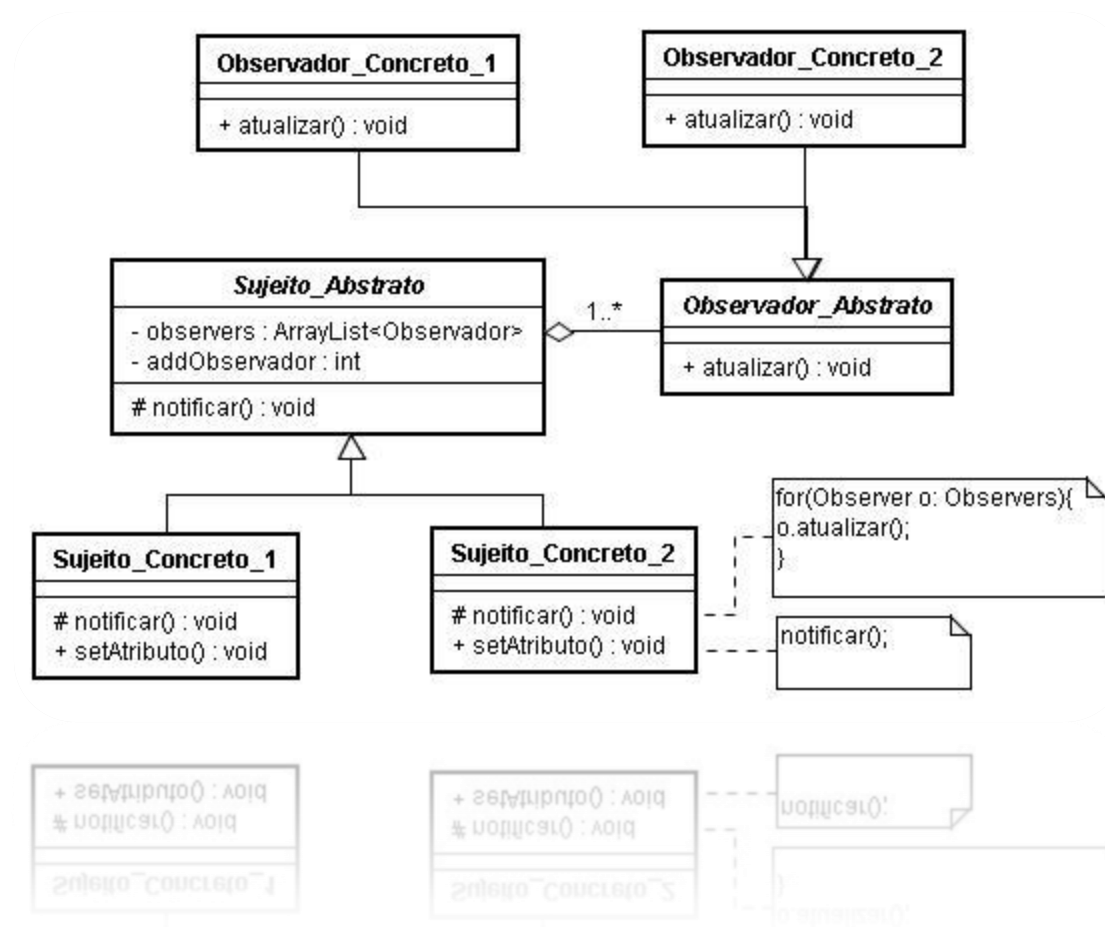
Padrões de Comportamento

- *Observer* (Observador)
- *State* (Estado)
- *Strategy* (Estratégia)
- *Template Method* (Método Template)
- *Visitor* (Visitante)

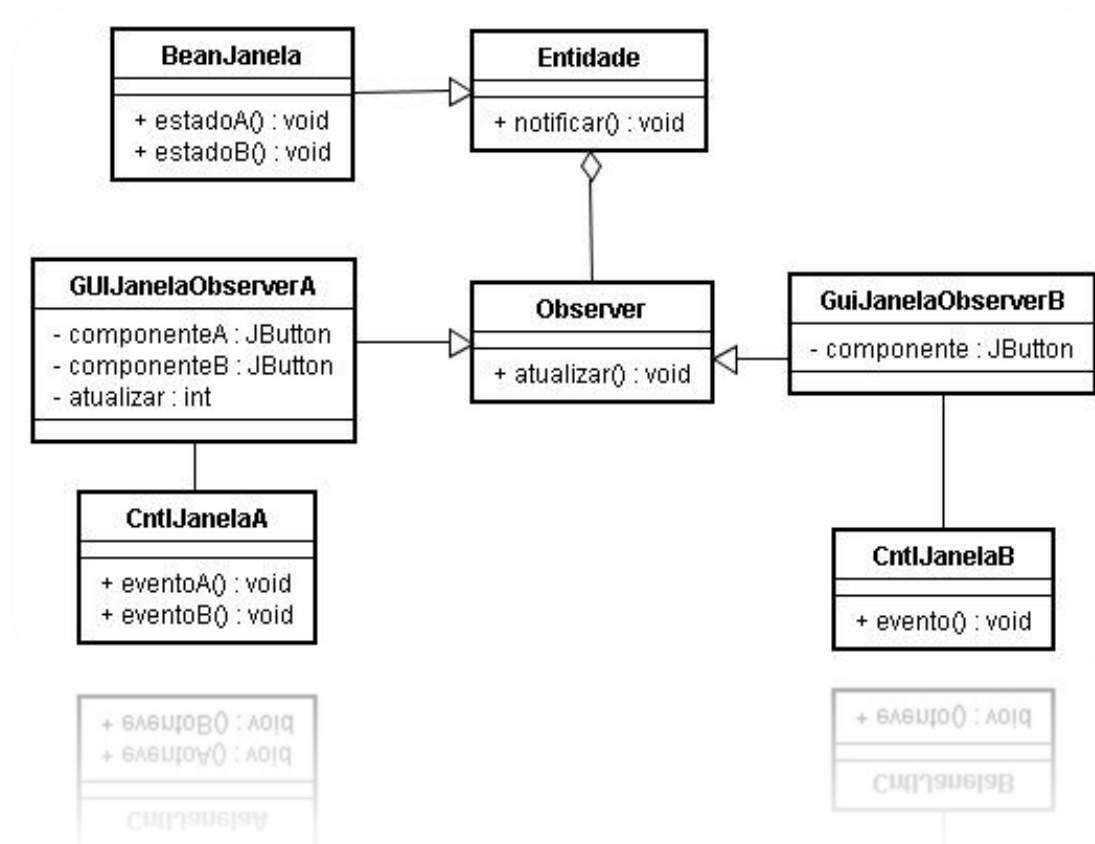
Observer(Observador)-Problema

- Garantir a consistência da informação
- Garantir a consistência de uma representação gráfica complexa, com muitos componentes.
- Transformar um grande problema em diversos pequenos problemas.

Observer (Observador)- Estrutura



Observer (Observador) - Exemplo



Observer (Observador) - Exemplo

```
public class Entidade implements Serializable{

    //lista contendo os objetos observadores
    protected ArrayList<Observer> listObs
    = new ArrayList<Observer>();

    protected void notificar(){
        for(Observer o : listObs)o.atualizar();
    }
    public void addObserver(Observer o){
        listObs.add(o);
    }
}
```

```
public interface Observer {

    public void atualizar();

}
```

Observer (Observador) - Exemplo

```
public class BeanJanela extends Entidade{

    private boolean estadoA;
    private boolean estadoB;

    public BeanJanela() {}

    //false representa a cor verde, true a amarela
    public boolean isEstadoA(){return estadoA;}
    public void setEstadoA(boolean a){
        estadoA = a;
        notificar();
    }
    //false representa desabilitação de funções, true a habilitação
    public boolean isEstadoB(){return estadoB;}
    public void setEstadoB(boolean b){
        estadoB = b;
        notificar();
    }
}
```

Observer (Observador) - Exemplo

```
public class GuiJanelaObserverA extends JFrame implements Observer{

    private JButton btnAmarelo;
    private JButton btnVerde;
    private CntrJanelaA cntl;
    private BeanJanela bean;

    public GuiJanelaObserverA() {
        cntl = new CntrJanelaA(this);
        //cria a interface gráfica
        initComponents();
    }

    public void atualizar() {
        if(bean.isEstadoB()){
            btnAmarelo.setEnabled(false);
            btnVerde.setEnabled(false);
        }else{
            btnAmarelo.setEnabled(true);
            btnVerde.setEnabled(true);
        }
    }
}
```

Observer (Observador) - Exemplo

```
public class GuiJanelaObserverB extends JFrame implements Observer{

    private JButton btnCor;
    private CntrJanelaB cntl;
    private BeanJanela bean;

    public GuiJanelaObserverB() {
        cntl = new CntrJanelaB(this);
        initComponents();
    }

    public void atualizar() {
        if(bean.isEstadoA()){
            btnCor.setForeground(Color.yellow);
            btnCor.setText("AMARELO");
        }else{
            btnCor.setForeground(Color.GREEN);
            btnCor.setText("VERDE");
        }
    }

    private void initComponents() {
```


Observer (Observador) - Exemplo

```
public class CntrJanelaA implements ActionListener{

    private GuiJanelaObserverA janela;
    //controlador de GuiJanelaObserverA
    public CntrJanelaA(GuiJanelaObserverA observerA) {
        janela = observerA;
    }

    public void actionPerformed(ActionEvent e) {
        if(((JButton)e.getSource()).getText().equals("VERDE")){
            janela.getBean().setEstadoA(false);
        }else if(((JButton)e.getSource()).getText().equals("AMARELO")){
            janela.getBean().setEstadoA(true);
        }
    }
}
```

Observer (Observador) - Exemplo

```
public class CntrJanelaB implements ActionListener {

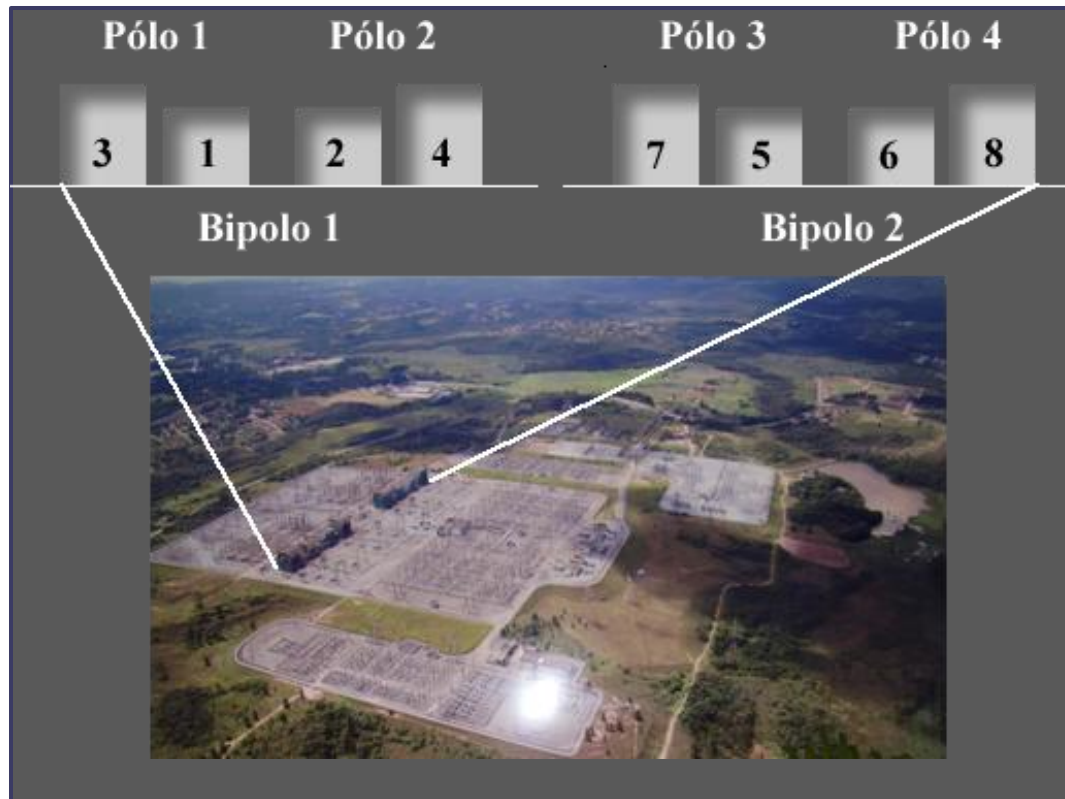
    private GuiJanelaObserverB janela;
    //controlador de GuiJanelaObserverB
    public CntrJanelaB(GuiJanelaObserverB observerB){
        janela = observerB;
    }

    public void actionPerformed(ActionEvent e){
        if(janela.getBean().isEstadoB()){
            janela.getBean().setEstadoB(false);
        }else{
            janela.getBean().setEstadoB(true);
        }
    }
}
```

Observer (Observador) - Exemplo

```
public class App {  
  
    public static void main(String args[]){  
        GuiJanelaObserverA janObsA = new GuiJanelaObserverA();  
        GuiJanelaObserverB janObsB = new GuiJanelaObserverB();  
  
        BeanJanela entidade = new BeanJanela();  
        //adiciona a entidade aos observadores  
        janObsA.setBean(entidade);  
        janObsB.setBean(entidade);  
        //adiciona os observadores a lista da entidade  
        entidade.addObserver(janObsA);  
        entidade.addObserver(janObsB);  
    }  
}
```

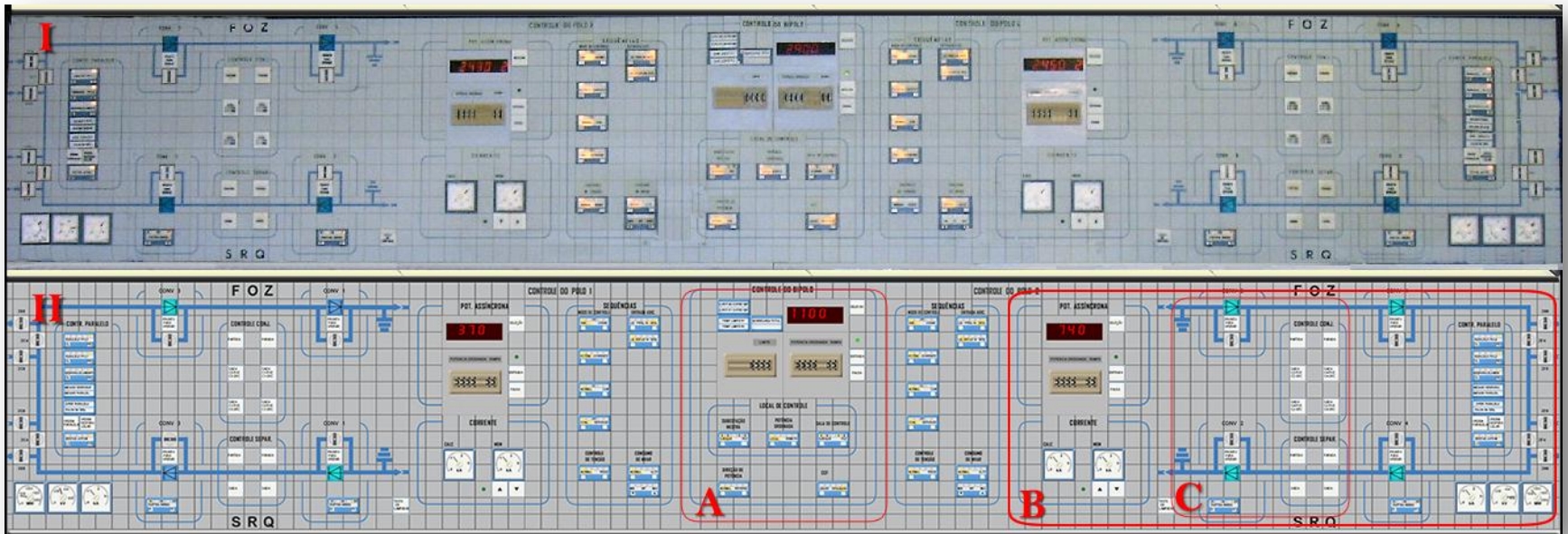
Observer (Observador) - Exemplo



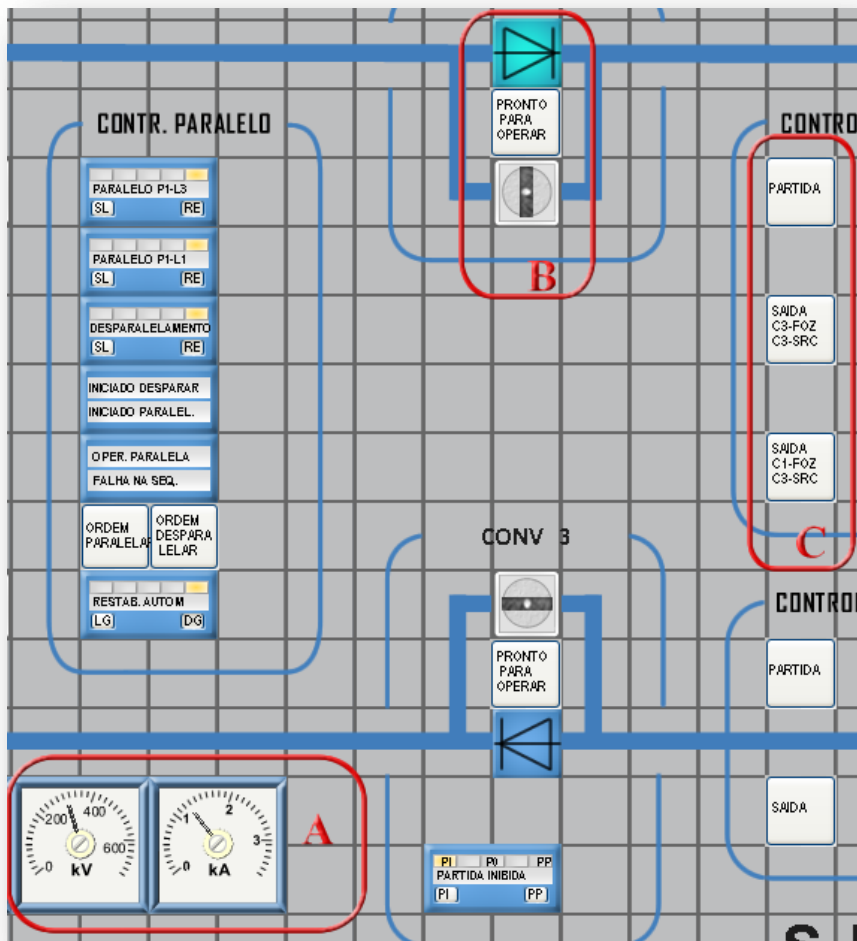
Observer (Observador) - Exemplo



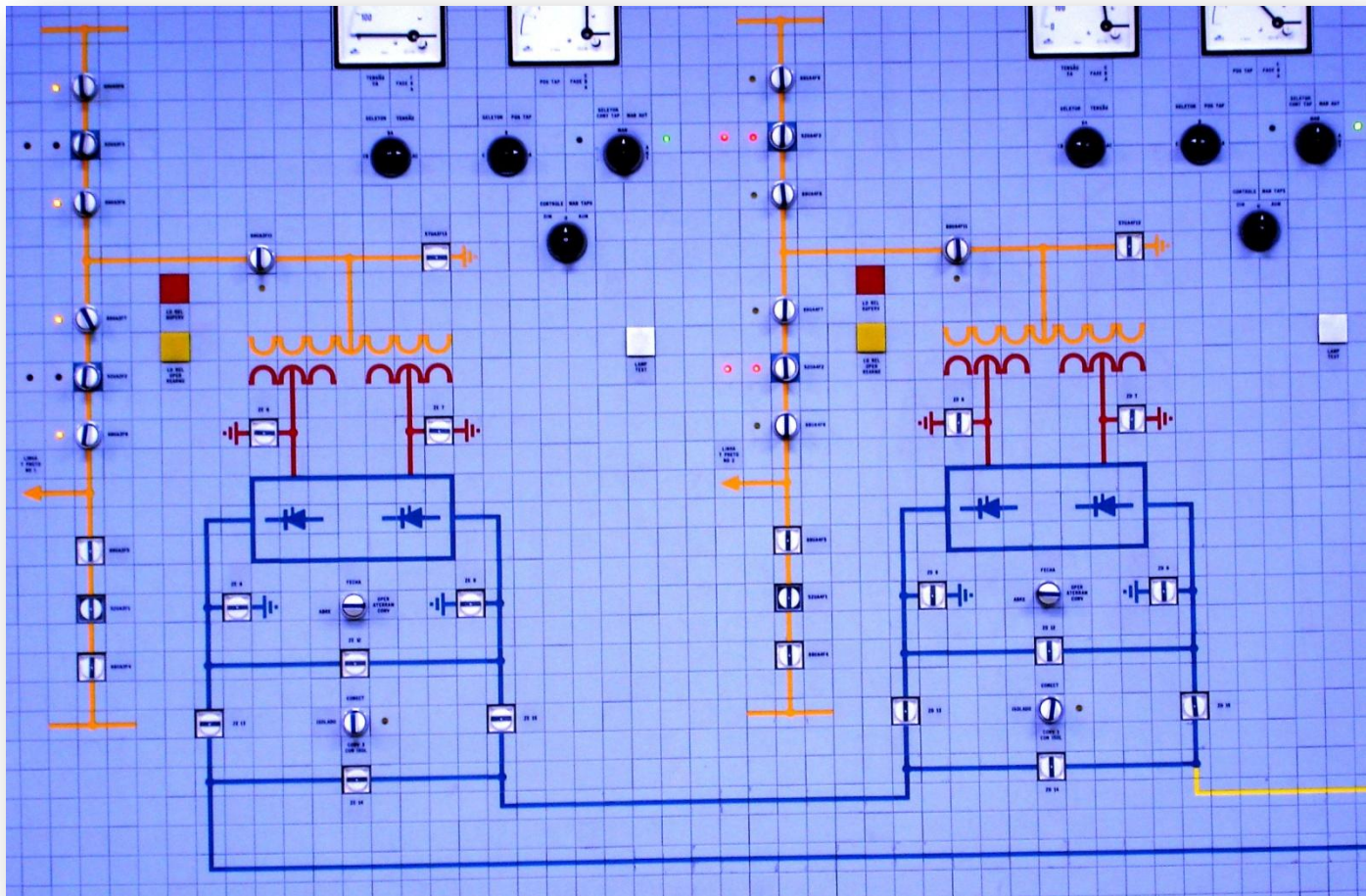
Observer (Observador) - Exemplo



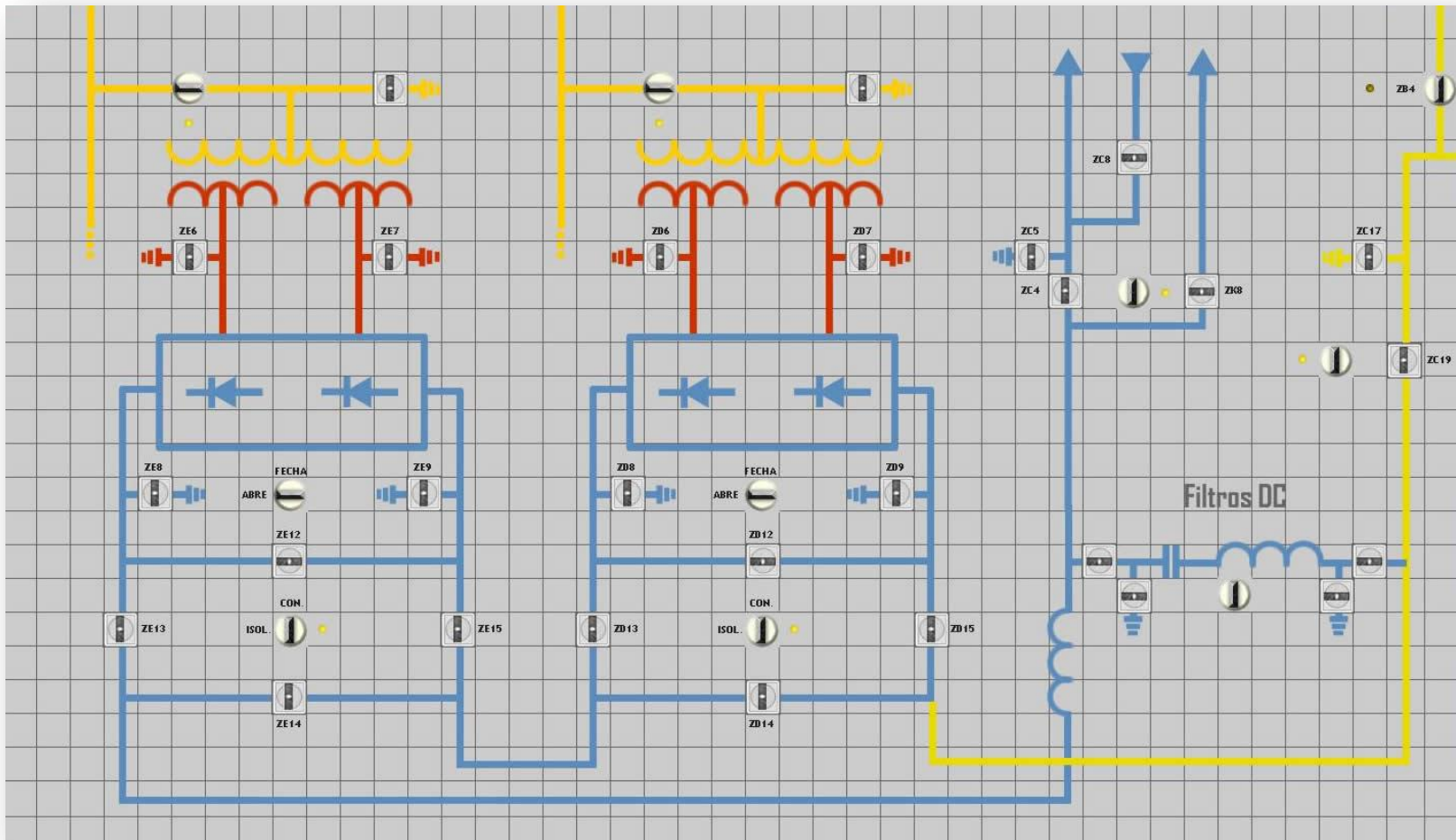
Observer (Observador) - Exemplo



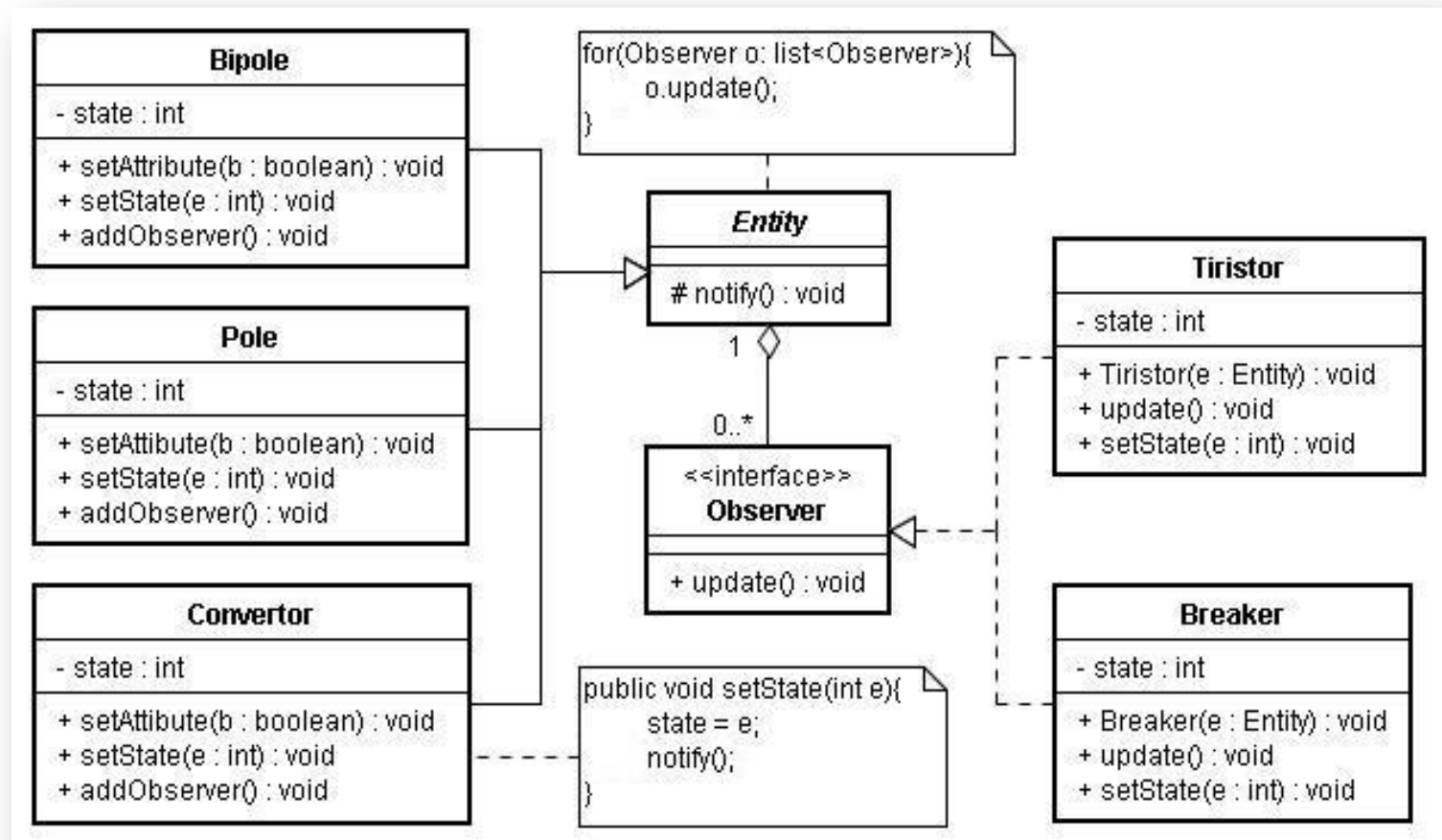
Observer (Observador) - Exemplo



Observer (Observador) - Exemplo



Observer (Observador) - Exemplo



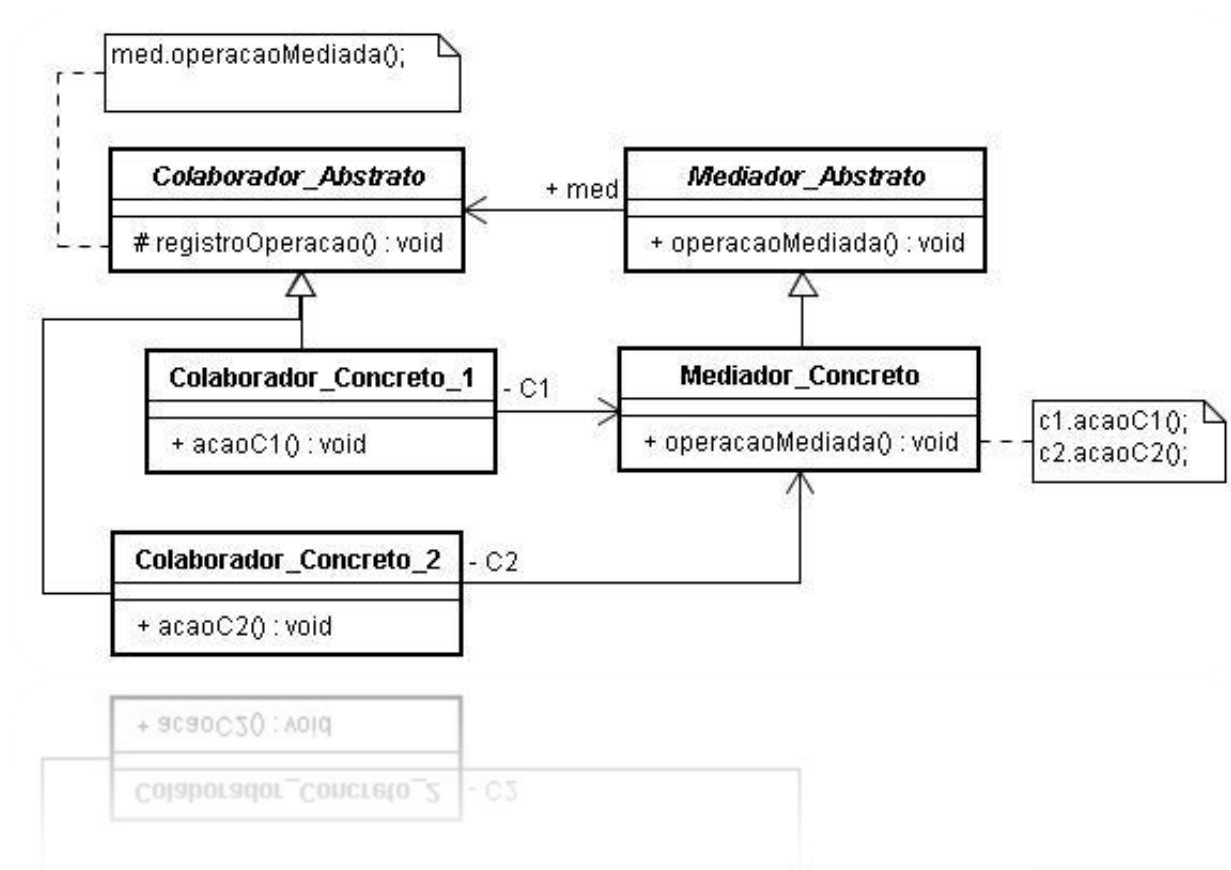
Observer - Consequências

- As informações inter-classes estão sempre consistentes
- Toda mudança de estado é automaticamente sinalizada aos observadores
- Um grande problema é dividido em diversos pequenos problemas
- Cada classe preocupa-se apenas com suas próprias informações.

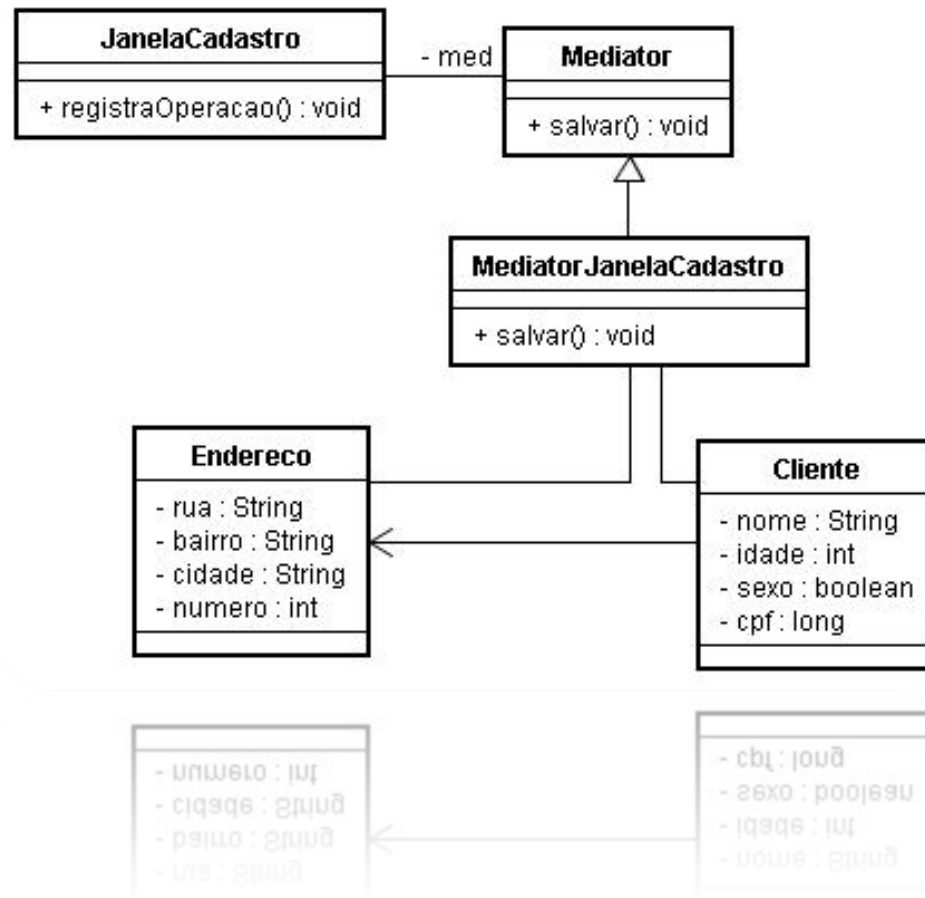
Mediator (Mediador)- Problema

- Para a construção de sistemas de qualidade, tanto grandes quanto pequenos, um importante requisito é a organização do código.
- Separação do código, Modelo-Visão-Controle.
- Usado quando necessita-se prover ao sistema um fraco acoplamento.

Mediator (Mediador)- Solução



Mediator (Mediador)- Exemplo



Mediator (Mediador)- Exemplo

```
public class Cliente {  
  
    private String nome;  
    private int idade;  
    private boolean sexo;//false = femino, true = masculino  
    private Long cpf;  
    private Endereco endereco;  
  
    public Cliente(){}  
  
    public Endereco getEndereco() {return endereco;}  
    public void setEndereco(Endereco end){endereco = end;}  
  
    public Long getCpf() {return cpf;}  
    public void setCpf(Long cpf) {this.cpf = cpf;}  
  
    public int getIdade() {return idade;}  
    public void setIdade(int idade) {this.idade = idade;}  
  
    public String getNome() {return nome;}  
    public void setNome(String nome) {this.nome = nome;}  
  
    public boolean isSexo() {return sexo;}  
    public void setSexo(boolean sexo) {this.sexo = sexo;}  
  
}
```

Mediator (Mediador)- Exemplo

```
public class Endereco {  
  
    private String rua;  
    private String bairro;  
    private String cidade;  
    private int numero;  
  
    public Endereco() {}  
  
    public String getBairro() {return bairro;}  
    public void setBairro(String bairro) {this.bairro = bairro;}  
  
    public String getCidade() {return cidade;}  
    public void setCidade(String cidade) {this.cidade = cidade;}  
  
    public int getNumero() {return numero;}  
    public void setNumero(int numero) {this.numero = numero;}  
  
    public String getRua() {return rua;}  
    public void setRua(String rua) {this.rua = rua;}  
}
```


Mediator (Mediador)- Exemplo

```
public class JanelaCadastro extends JFrame{

    private Mediator med;

    public JanelaCadastro() {
        med = new MediatorJanelaCadastro(this);
        //cria a interface gráfica
        initComponents();
        registraOperacao();
    }

    public void registraOperacao() {
        btnSalvar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                med.salvar();
            }
        });
    }
}
```

Mediator (Mediador)- Exemplo

```
public class MediatorJanelaCadastro extends Mediator{

    public JanelaCadastro janela;
    public MediatorJanelaCadastro(JanelaCadastro cadastro) {
        janela = cadastro;
    }

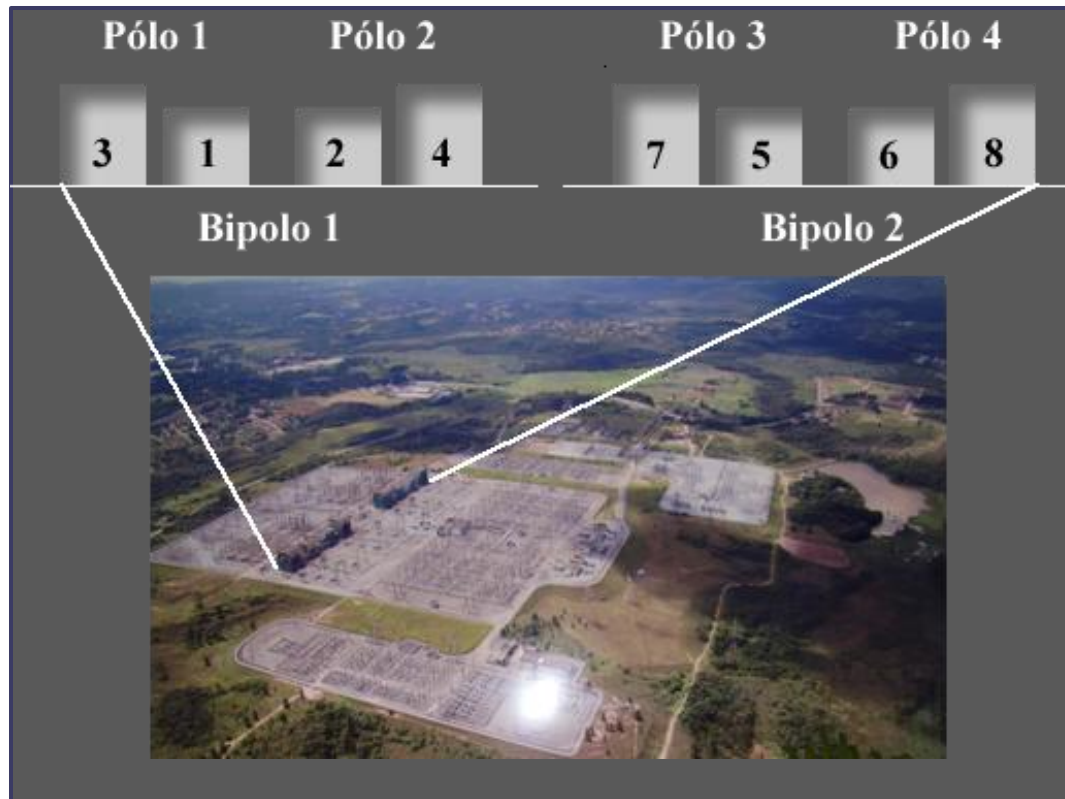
    //sobrescrita do método de MEDIATOR
    public void salvar(){
        //A GUI não conhece esta entidade
        Cliente cliente = new Cliente();
        cliente.setNome(janela.getTxtNome().getText());
        cliente.setIdade(Integer.parseInt(janela.getTxtIdade().getText()));
        cliente.setCpf(Long.parseLong(janela.getTxtCpf().getText()));
        cliente.setSexo(janela.getRadioFemino().isSelected()?false:true);
        //A GUI não conhece esta entidade
        Endereco endereco = new Endereco();
        endereco.setBairro(janela.getTxtBairro().getText());
        endereco.setCidade(janela.getTxtCidade().getText());
        endereco.setRua(janela.getTxtRua().getText());
        endereco.setNumero(Integer.parseInt((janela.getTxtNum().getText())));
        //adiciona o endereço a pessoa
        cliente.setEndereco(endereco);
        //Mostra as entidades preenchidas
        janela.dispose();
        JOptionPane.showMessageDialog(janela,"O endereço de "+cliente.getNome()+
        " é rua: "+ cliente.getEndereco().getRua()+" número: "+cliente.getEndereco().getNumero());
        //PODERIAMOS INCLUIR PERSISTENCIA A ESTE EXEMPLO E A GUI CONTINUARIA SER NECESSITAR
        //DE MUDANÇAS, O MEDIADOR CUIDA DE TUDO.
    }
}
```

Mediator (Mediador)- Exemplo

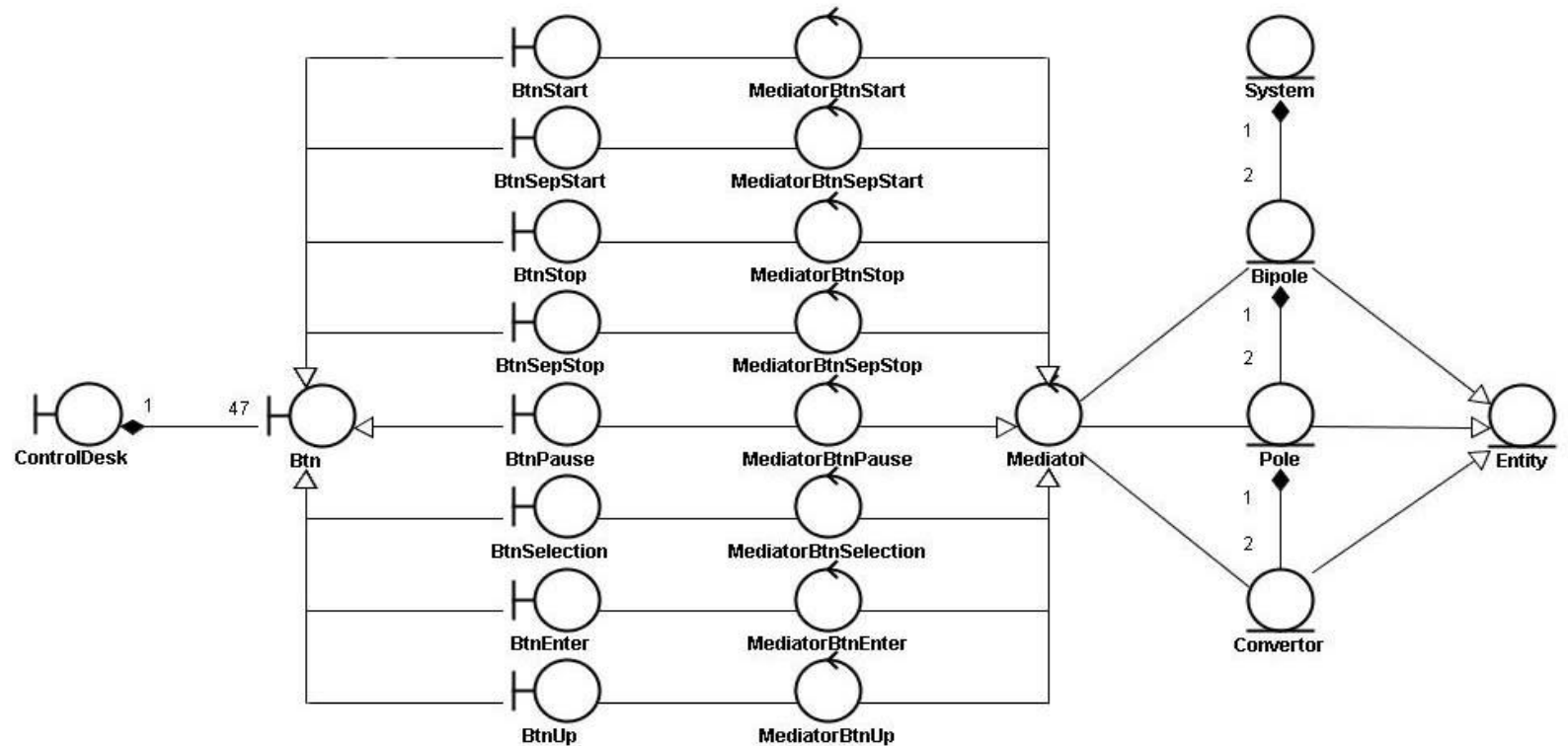
```
public abstract class Mediator {  
    //Exemplo de método a sobrescrever.Em grandes  
    //aplicações existem dezenas de métodos como este  
    public void salvar() {}  
}
```

```
public class App {  
  
    public static void main(String args[]){  
        new JanelaCadastro();  
    }  
}
```

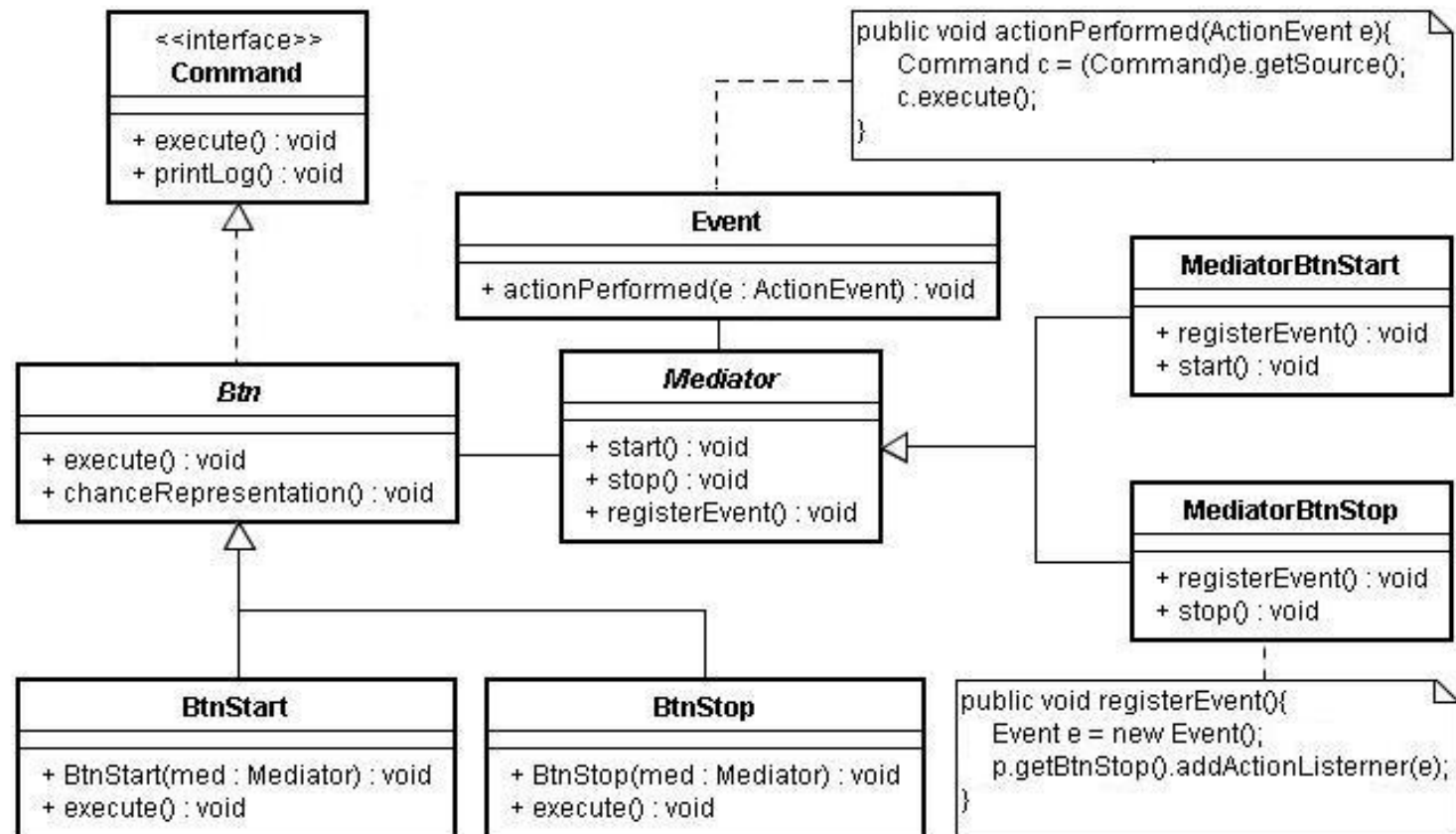
Mediator (Mediador)- Exemplo



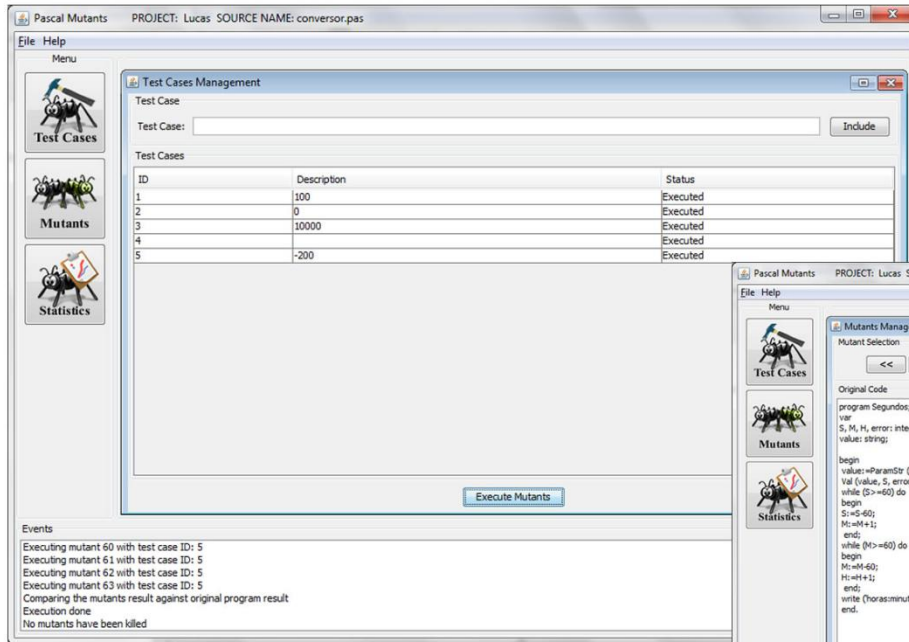
Mediator (Mediador)- Exemplo



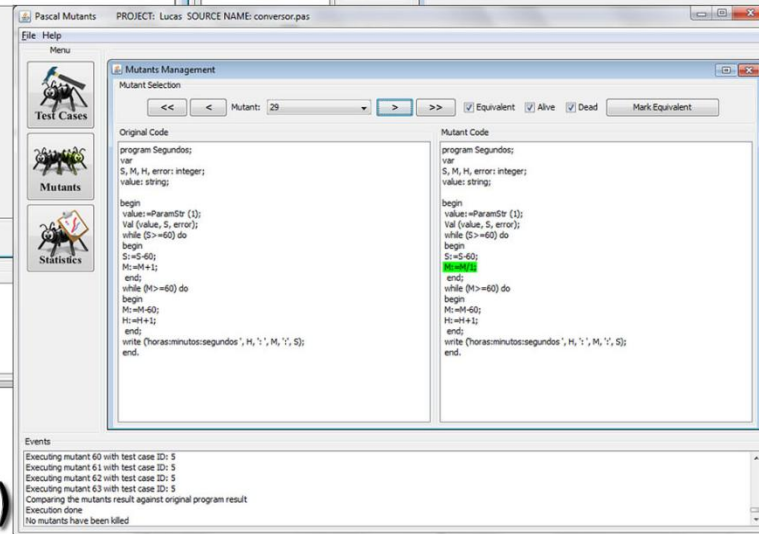
Mediator & Command



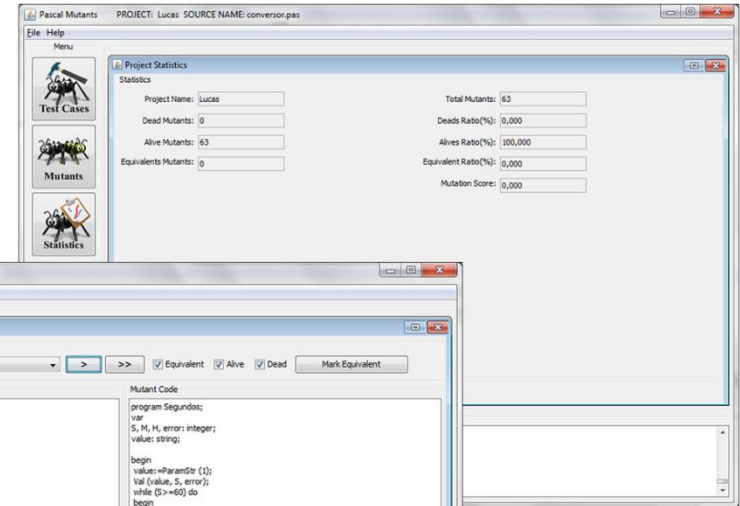
Mediator & Command (Exemplo)



(b)



(c)



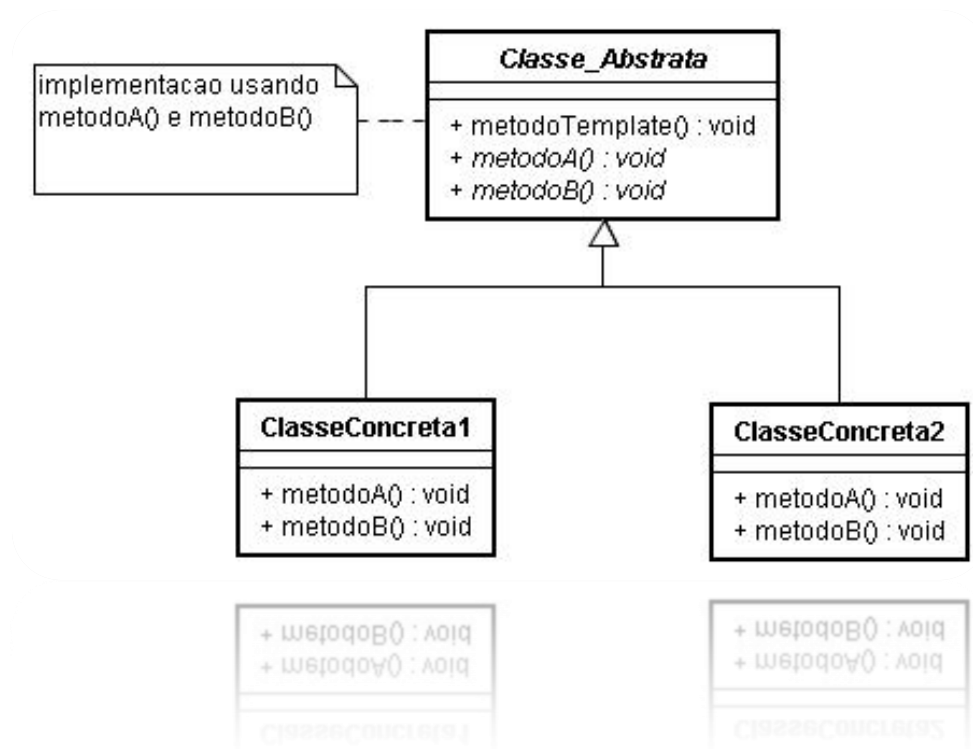
Mediator (Mediador) - Consequências

- É possível gerenciar eventos de uma classe por meio de outra classe.
- O objeto gerador do evento não precisa conhecer os objetos que ele influencia.
- É possível acrescentar novas funcionalidades a um evento sem ter de modificar o emissor do mesmo.
- *Experimente combinar esse padrão com o padrão *Command*.

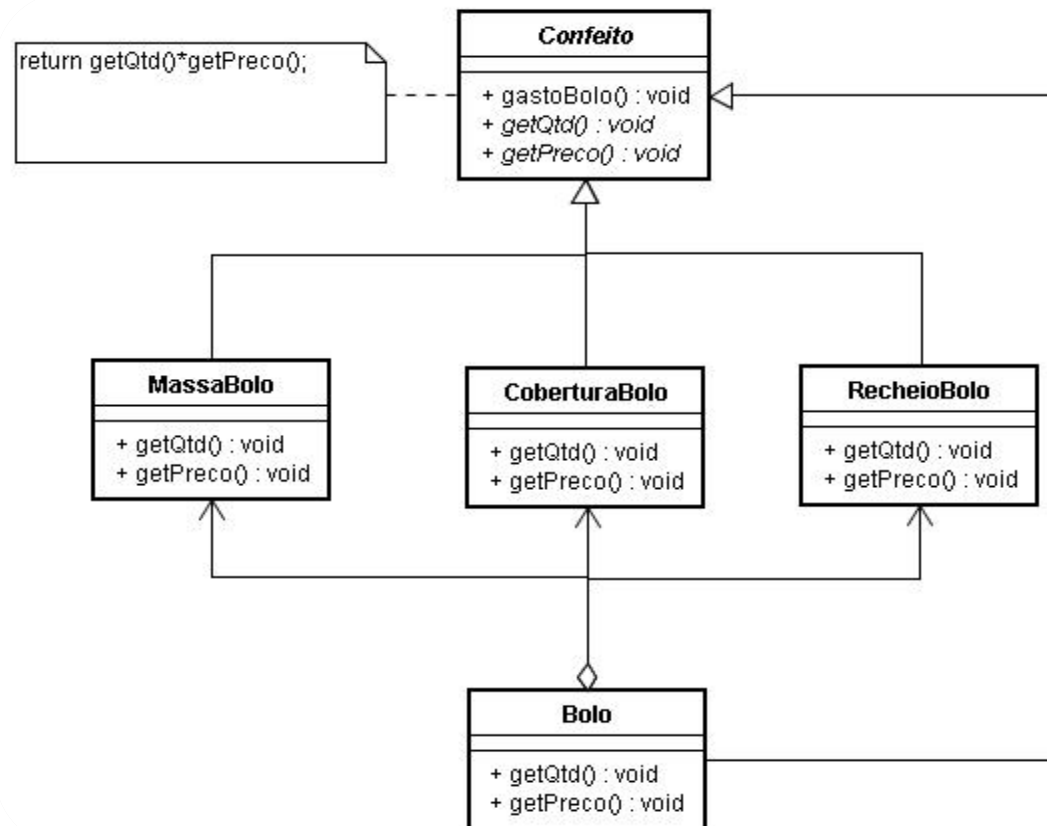
Template Method - Problema

- Quando existe a necessidade de garantir que um determinado método forneça uma lógica segundo um modelo definido (*Template*).
- Criação de um esqueleto de implementação que varia quanto ao conteúdo das partes mas não do todo.

Template Method - Solução



Template Method- Exemplo



Template Method- Exemplo

```
public abstract class Confeito {  
  
    //Método TEMPLATE, sempre FIXO  
    public double gastoConfeito(){ return getQuantidade() * getPreco();}  
  
    public abstract double getQuantidade();  
    public void setQuantidade(double qtd) {}  
  
    public abstract double getPreco();  
    public void setPreco(double preco) {}  
  
}
```

Template Method- Exemplo

```
public class CoberturaBolo extends Confeito{

    public CoberturaBolo(){};

    public double gastoConfeito()
    { return super.gastoConfeito();}

    @Override
    public double getQuantidade() {
        return 0.7;
    }

    @Override
    public double getPreco() {
        return 6.30;
    }
}
```

Template Method- Exemplo

```
public class MassaBolo extends Confeito{

    public MassaBolo(){};

    public double gastoConfeito()
    { return super.gastoConfeito();}

    @Override
    public double getQuantidade() {
        return 2.0;//bolo duplo
    }

    @Override
    public double getPreco() {
        return 5.1;
    }
}
```

Template Method- Exemplo

```
public class RecheioBolo extends Confeito {  
  
    public RecheioBolo() {};  
  
    public double gastoConfeito()  
    { return super.gastoConfeito();}  
  
    @Override  
    public double getQuantidade() {  
        return 2.5;//muito recheio  
    }  
  
    @Override  
    public double getPreco() {  
        return 4.3;  
    }  
  
}
```

Template Method- Exemplo

```
public class Bolo extends Confeito{

    private CoberturaBolo cobertura;
    private RecheioBolo recheio;
    private MassaBolo massa;
    private int quantidade;

    public Bolo(){}

    public double gastoConfeito(){ return super.gastoConfeito();}
    @Override
    public double getQuantidade() {
        return quantidade;//numero de bolos
    }
    @Override
    public double getPreco() {
        return recheio.gastoConfeito()+massa.gastoConfeito()+cobertura.gastoConfeito();
        //preco da soma das partes
    }
    public void setQuantidade(int qtd){quantidade = qtd;}
    public CoberturaBolo getCobertura() {return cobertura;}
    public void setCobertura(CoberturaBolo c) {cobertura = c;}

    public MassaBolo getMassa() {return massa;}
    public void setMassa(MassaBolo m) {massa = m;}

    public RecheioBolo getRecheio() {return recheio;}
    public void setRecheio(RecheioBolo r) {recheio = r;}
}
```


Template Method- Exemplo

```
public class App {  
  
    public static void main(String args[]){  
        //cria as partes do bolo  
        CoberturaBolo cobertura = new CoberturaBolo();  
        MassaBolo massa = new MassaBolo();  
        RecheioBolo recheio = new RecheioBolo();  
        //monta o bolo  
        Bolo bolo = new Bolo();  
        bolo.setRecheio(recheio);  
        bolo.setCobertura(cobertura);  
        bolo.setMassa(massa);  
        bolo.setQuantidade(2);  
        //independente de quanto custa um confeito ou como o método  
        //o método getPreco é implementado o gasto é sempre preco*peso,  
        //esse é o template de confeito.  
        System.out.println("O gasto com a massa é: " + massa.gastoConfeito());  
        System.out.println("O gasto com a cobertura é: " + cobertura.gastoConfeito());  
        System.out.println("O gasto com o recheio é: " + recheio.gastoConfeito());  
        System.out.println("O gasto com a bolo é: " + bolo.gastoConfeito());  
    }  
}
```

Template - Consequências

- É possível controlar o quanto a subclasse pode variar a implementação.
- A implementação de uma ação pode variar (de acordo com quem a implementa)

Pesquisar

- *Chain of Responsibility* (Cadeia de Respons.)
- *Command* (Comando)
- *Interpreter* (Interpretador)
- *Iterator*
- *Memento* (Recordador)
- *State* (Estado)
- *Strategy* (Estratégia)
- *Visitor* (Visitante)

Exercício

- Você deseja sempre ter a mão o cálculo atualizado das taxas de seu carro.
- Desenvolva uma aplicação que calcula o valor do *Seguro Obrigatório* (5% do valor do carro) e também o *IPVA* (4% do valor do carro).
- *IPVA* e *Seguro Obrigatório* devem ser classes sempre atualizadas com relação às mudanças no valor de mercado de seu carro.
- Utilize um padrão de projeto para solucionar esse problema

Bibliografia

- ALEXANDER, Christopher. *A Pattern Language: Towns, Buildings, Construction*. Ed. USA: Oxford University Press, 1977.
- GAMMA, Erich et al, *A Design Patterns: Elements of Reusable Object-Oriented Software*, Construction. 36. Ed. Portland: Addison-Wesley, 1994.
- SERAPHIM, Enzo. *Tópicos Especiais de Programação*. Notas de aula, Universidade Federal de Itajubá-UNIFEI, Itajubá-MG, Brasil, 2008.