# JAVA I

Introdução aos componentes GUI

***Graphical User Interfaces***

Aula 14
Prof. Roque Junior
Elaborata Informática

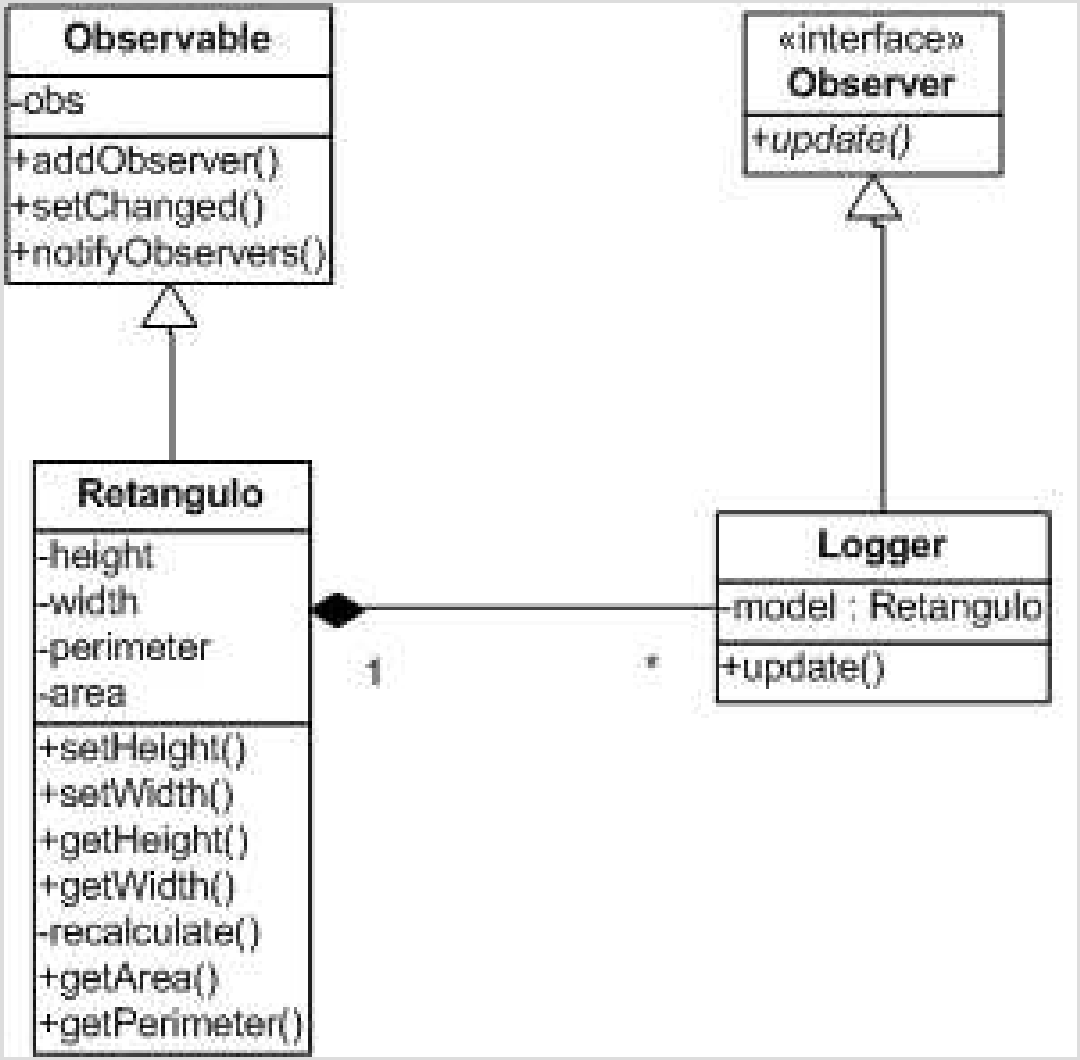
# Observable/Observer

Aqui apresentaremos o Observable/Observer. Este padrão pode ser usado para controlar a interação entre a interface do usuário e o modelo de dados. Esse padrão assegura que a visão conheça o modelo, mas o modelo não conheça sobre a visão.

Usaremos um exemplo simples onde um retângulo pode calcular sua área e seu perímetro. A visão somente escreve uma linha no console toda vez que os dados do retângulo é alterado.

Aula 14
Prof. Roque Junior
Elaborata Informática

# Observable/Observer




Aula 14
Prof. Roque Junior
Elaborata Informática

# Observable/Observer

**Criando o Modelo (Observable)**

1. O primeiro passo será implementar a classe Retangulo. Veja no diagrama que Retangulo herda a classe Observable, então devemos importar essa classe do pacote java.util.

```
package com.elaborata.java1.aula13.observable;

import java.util.Observable;
public class Rectangulo extends Observable {
```

2. Criar os atributos da classe:

```
private int altura;
private int comprimento;
private int area;
private int perimetro;
```

# Observable/Observer

**Criando o Modelo (Observable)**

3. Criar os métodos gettter/setter. Pois assim quando o estado do objeto for modificado saberemos para notificar os observer, além de estar encapsulando os dados.

```
public int getAltura() {
return altura;
}
public void setAltura(int altura) {
this.altura = altura;
recalcular();
}
public int getComprimento() {
return comprimento ;
}
public void setComprimento(int comprimento ) {
this.comprimento = comprimento ;
recalcular();
}
```

# Observable/Observer

**Criando o Modelo (Observable)**

```
public int getArea() {
return area;
}
public int getPerimetro() {
return perimetro;
}
```

Note que os atributos área e perímetro não possuem métodos setter. Isso porque essas informações serão calculadas somente quando os atributos largura e comprimento for alterado.

# Observable/Observer

**Criando o Modelo (Observable)**

4. O método recalcular(), calcula a área e o perímetro do retângulo baseado nos novos dados, e notifica os observers:

```
public void recalcular() {
  perimetro=2*(altura+comprimento); //Calcula o perímetro
  area=altura*comprimento; //Calcula a área
  setChanged(); //Notifica os observers
  notifyObservers(); // esses métodos foram herdados de
  Observable
}
```

5. O construtor abaixo permite que um objeto seja criado baseado nos argumentos largura e comprimento:

```
public Rectangulo(int altura, int comprimento) {
  super();
  this.altura=altura;
  this.comprimento=comprimento;
  recalcular();
}
```

# Observable/Observer

**Criando a Visão(Observable)**

Agora que criamos a classe Retangulo, que é o modelo, devemos criar a classe Logger, que será a visão. Como pode perceber através do padrão Observable / Observer estamos também caminhando para o aprendizado do modelo MVC.

1. Logger deverá implementar Observer:

```
package com.elaborata.java1.aula13.observable;

import java.util.Observable;
import java.util.Observer;
public class Logger implements Observer{
```

# Observable/Observer

**Criando a Visão (Observable)**

2. A visão deve conhecer o modelo, mas o modelo não deverá conhecer a visão. Não esqueça essa regra:

```
private Rectangulo modelo;
public Logger(Rectangulo r) {
    modelo=r; //A visão conhece o modelo
    modelo.addObserver(this); //A visão observará
  notificações do modelo
}
```

# Observable/Observer

**Criando a Visão (Observable)**

3. Através do método update() será notificada quando o modelo sofrer alguma alteração, e neste momento as informações do retangulo será impressa no console. O método update() é a implementação do método update() da interface Observer:

```
public void imprimir() {
  Console.imprimirMensagem("Log:: Area: " + modelo.getArea()
  + " Perimetro: " + modelo.getPerimetro(), false);
}
```

# Observable/Observer

Código da classe console. A classe console é usada para emitir avisos no console, e mensagem com liberação pelo pressionamento da tecla enter.

```
package com.elaborata.java1.aula13.observable;

public class Console {
	public static void imprimirMensagem(String mensagem, boolean
                                                    parar){
	int tecla=0;
	boolean continuar=false;
	System.out.print(mensagem + " ");
	if (!parar)
	    System.out.println();
	else //Espera o pressionamento da tecla ENTER
	    while (!continuar) {
	        try {
	            tecla = System.in.read();
	            if (tecla<0 || (char)tecla=='\n')
	                continuar = true;
	        } catch(java.io.IOException e) {
	                continuar = true;
	        }
	    }
	}
}
```

# Observable/Observer

**Testando o Padrão Observable/Observer**

Para podemos testar a visão Logger obsevar o modelo Retangulo, devemos criar uma outra classe que instanciará as duas e fará alterações no modelo.

1. Criar a classe TesteObservableObserver:

```
public class TesteObservableObserver {
```

2. Criar o método startup:

```
public static void main(String[] args) {
```

3. Instanciar o Retangulo:

```
Console.imprimirMensagem("Instanciando o
                         retangulo...",false);
Rectangulo r = new Rectangulo(5,10);
```

# Observable/Observer

**Testando o Padrão Observable/Observer**

4. Instanciar o Logger. O objeto Retângulo é passado como argumento para o Logger, assim a visão conhecerá o modelo que estará observando:

```
Console.imprimirMensagem("Instanciando o
                          logger...",false);
Logger l = new Logger(r);
```

5. Alterar a Algura do retangulo. Perceba que uma mensagem do logger será impressa logo após o pressionamento da tecla Enter. Neste momento o modelo tendo seu estado modificado notifica as visões (observers)

```
Console.imprimirMensagem("Agora vamos alterar a altura do
          retangulo. " + "Pressione uma tecla!", true);
r.setAltura(500);
```

# Observable/Observer

**Testando o Padrão Observable/Observer**

6. Alterar o comprimento do retangulo.

```
        Console.imprimirMensagem("Agora vamos alterar o
comprimentodo " + " retangulo. Pressione uma tecla!", true);
        r.setComprimento(1000);
}
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

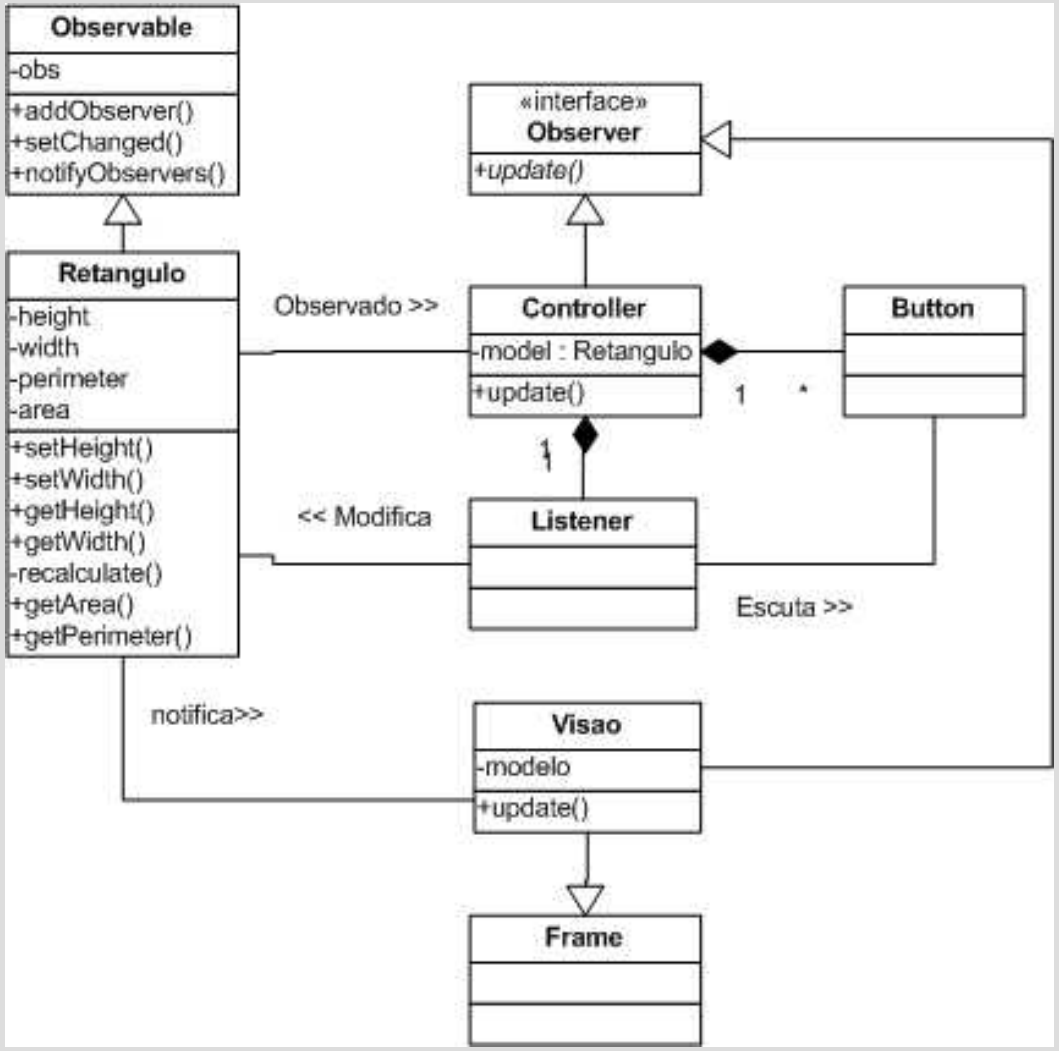
Os exemplos anteriores definiram o modelo e a visão através da classe Observable e da interface Observer e dessa forma implementamos o padrão Observable/Observe. Agora precisamos implementar o padrão MVC, e para isso resta−nos implementar o controlador do padrão.

O controlador será o objeto responsável por realizar as mudanças de estado do modelo. O controlador conhece o modelo e/ou a visão. O controlador também pode ser notificado sobre alterações no estado do modelo e também pode não ser. Sendo notificado, deverá implementar o método update() da interface Observer.

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

Observe o diagrama abaixo:




Para implementarmos o controlador usaremos uma interface gráfica com botões para aumentar e diminuir os valores de altura e comprimento de um retângulo.

Aula 14
Prof. Roque Junior
Elaborata Informática

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

1. Para criar o Controle devemos importar as seguintes classes:

```
package com.elaborata.java1.aula13.observable.controle;

import java.awt.Frame; //Janela
import java.awt.Button; //Botões
import java.util.Observer; //Observer, para o Controle ser
notificado de alterações no modelo
import java.util.Observable;
import java.awt.event.ActionListener; //Ouvinte de eventos
import java.awt.event.ActionEvent; //Evento gerado
```

2. Criar a classe Controle:

```
public class Controle extends Frame implements Observer {
Rectangulo modelo;
Button incr;
Button decr;
Listener c;
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

## 3.Criando o construtor da classe

```
public Controle(Rectangulo modelo){
    super("Controlador");
    this.setLayout(new GridLayout(0,2));
    this.modelo = modelo;
    modelo.addObserver(this);
    incr = new Button("+");
    add(incr);
    decr = new Button("-");
    add(decr);
    c = new Listener();
    update(null, null);
    addWindowListener(new WindowCloser());
    setSize(200,100);
}
```

Aula 14
Prof. Roque Junior
Elaborata Informática

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

O controlador deve conhecer o modelo, então no momento da instanciação do objeto Controle o modelo Retângulo é passado como parâmetro.

Também é criado dois botões: um para incrementar a altura e comprimento e outro para decrementar os mesmo atributos.

Um listener é definido para escutar a interação do usuário e alterar o modelo.

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

4. O Controle também saberá sobre alterações no modelo. E instancia this do Controle também é passado para o modelo através do método addObserver. O método update deverá ser implementado para que o Controle seja notificado.

```
public void update(Observable o, Object arg){
    this.checkStatus();
}
public void checkStatus() {
    if (modelo.getAltura()>50 || modelo.getComprimento()> 50)
        incr.setEnabled(false);
    else
        incr.setEnabled(true);
    if (modelo.getAltura()<=0 || modelo.getComprimento()<= 0)
        decr.setEnabled(false);
    else
        decr.setEnabled(true);
}
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

O método checkStatus verifica os valores do Retangulo não permitindo assim valores menores que 0 ou maiores que 50.

5. O Listener é implementado através de uma classe interna. Somente a classe Controle poderá instanciar este Listener. Através do Listener o modelo sofrerá alterações, pois é o responsável por capturar as interações do usuário.

# Observable/Observer

## Criando o controlador (Concluindo o padrão MVC)

```
class Listener implements ActionListener {
    public Listener(){
        incr.addActionListener(this);
        decr.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
        if (e.getSource()==incr){
            modelo.setAltura(modelo.getAltura()+10);
            modelo.setComprimento(modelo.getComprimento()+10);
        }
        if (e.getSource()==decr){
            modelo.setAltura(modelo.getAltura()-10);
            modelo.setComprimento(modelo.getComprimento()-10);
        }
    }
}
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

6. A classe WindowCloser herda de um adaptador o WindowAdapter, que implementa todos os métodos da interface WindowListener, e assim podemos somente implementar o método que satisfaça a necessidade, que é o windowClosing, para saber quando o usuário quer fechar a janela

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class WindowCloser extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        e.getWindow().dispose();
    }
}
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

7. Agora devemos criar uma classe que junte os pedaços (o modelo, a visão e o controlador) do padrão MVC. Essa classe simplesmente terá um método estático main para instanciar as partes de montar o MVC.
Veja:

```
public class TesteMVC {
    public static void main(String[] args) {
        //Instancia o modelo
        Rectangulo r = new Rectangulo(10,20);
        //Instancia a visão que deve conhecer o modelo
        new Logger(r);
        //Instancia o controle que deve também conhecer o modelo
        new Controle(r).show();
    }
}
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

8. Podemos também definir diversas visões para o mesmo modelo, a janela abaixo mostrará o Retangulo de forma mais real. Através de um Frame essa visão desenhará o retangulo.

```
import java.awt.Frame;
import java.awt.Graphics;
import java.util.Observable;
import java.util.Observer;
import java.awt.Color;

public class VisaoGrafica extends Frame implements Observer {
    private Retangulo modelo;

    public VisaoGrafica(Retangulo r) {
        super("Visão Gráfica do Modelo");
```

# Observable/Observer

## Criando o controlador (Concluindo o padrão MVC)

```
        this.modelo = modelo;
        modelo.addObserver(this);
        addWindowListener(new WindowCloser());
        setSize(300,300);
        setLocation(250,150);
        update(null, null);
    }
    public void update(Observable o, Object arg) {
        repaint();
    }
    public void paint(Graphics g) {
        g.setColor(Color.green);
        g.fillRect(10,10,modelo.getAltura(),
                         modelo.getComprimento());
    }
```

# Observable/Observer

**Criando o controlador (Concluindo o padrão MVC)**

9.Agora é só instanciar a classe VisaoGrafica passando o modelo como parametro no construtor, tudo isso no método main da classe TesteMVC. Veja:

```
Logger l = new Logger(r);
new VisaoGrafica(r).show(); //nova linha.
new Controle(r).show();
```