UFRPE UAST Unidade Acadêmica de Serra Talhada - PE Desde 2006

BSi Bacharelado em Sistemas de Informação

MPOO

Site: https://sigs.ufrpe.br/sigaa/ava/index.jsf

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS XII

Leia atentamente as instruções gerais:

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaXII.SeuNomeSobrenome**, o qual deverá ter a **pasta de pacotes "**sistemaClinica".
- A lista envolve questões práticas que deverão ser entregues no AVA todos os códigos-fonte (projeto completo).
- A entrega da lista compõe sua frequência e avaliação na disciplina.

Fique atento!

Observer é um padrão comportamental para definir uma dependência *1..n* (um para muitos) entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente (GAMMA et al. 2000, p. 274[1]).

Esse conceito pode refletir nas aplicações gráficas em que apresentam dados de um objeto e quando este for modificado, seu conteúdo seja atualizado nas aplicações, sem estar atrelado necessariamente a um comportamento de interação do usuário. Uma das situações mais conhecidas deste comportamento aparece em MVC, em que model contém as definições dos objetos a serem utilizados e view contém GUI para exibição desse model.

Você Sabia?

Uma curiosidade é que em Java esse padrão comportamental existe desde a JDK 1.0, através da interface `java.util.Observer` e a classe `java.util.Observable`. A interface `Observer` possui o método abstrato `update()` que recebe como parâmetro a instância de `Observable` que gerou o evento e mais um objeto que é o parâmetro passado para o método `notifyObservers()`.

A ideia da classe `Observable` é que ela seja estendida por classes que queiram emitir notificações sobre a mudança de seu estado. Ela possui métodos implementados para gerenciar e notificar as instâncias de `Observer`. Vale ressaltar que essa classe não é utilizada em nenhum lugar da API da JDK, mesmo o padrão Observer sendo utilizado em diversos contextos. A questão principal é que essas classes consideram o Observer como uma implementação pronta e não como uma solução que pode ser adaptada para diversos contextos. Sendo assim, é possível implementar Observer em Java sem a utilização da interface `Observer` ou da classe `Observable`, mas para isso é preciso estudar mais a fundo o padrão.

Vejamos um exemplo de observador e observável a partir da utilização de `java.util` e MVC (não modificado):

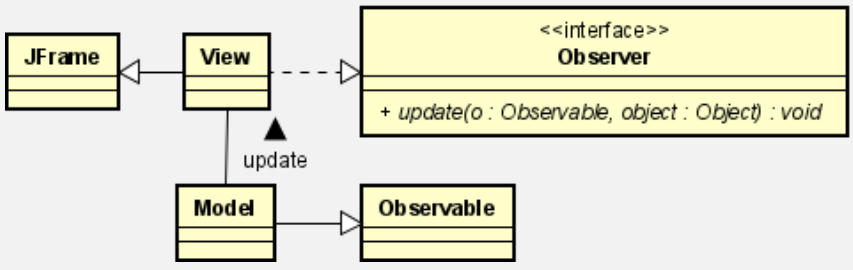
```
//Model.Java
public class Model extends Observable {
	int atr;
	ArrayList<View> observadores = new ArrayList<View>();

	public void modificarModel(int valor){
		atr=valor;
		//informa aos observadores a modificação realizada:
		setChanged();
		notifyObservers(atr);
	}



	//continua....
```

[1] GAMMA, E. et al. ***Padrões de Projeto:*** *Soluções reutilizáveis de software Orientado a Objetos.* Porto Alegre: Bookman, 2000.

```
        /**
         * Adiciona Observadores ao observável
         */
        public void addObservadores(View view){
                observadores.add(view);
                addObserver(observadores.get(observadores.size()-1));
        }
}//fim Model.Java

//View.Java
public class View extends JFrame implements Observer{

        JLabel informacaoModel;
        Model model;
        public View (Model model){
                setSize(100,100);
                model.addObservadores(this); //informa ao model ser um observador

                this.model=model;
                informacaoModel = new JLabel(model.atr+"", SwingConstants.CENTER);

                add(informacaoModel);
                setVisible(true);
        }

        /**
         * update atualiza a gui a partir da modiciação do model observado
         */
        @Override
        public void update(Observable o, Object object) {
                informacaoModel.setText(object+"");
        }
}

//App.Java
public class App {
        public static void main(String[] args) {
           //1 observável e 2 observadores
           Model model = new Model();

           View view = new View(model);//Relacionamento View-Model, em consonância com MVC (KRASNER; POPE, 1988)
           View view2 = new View(model);

           model.modificarModel(10);
        }
}
```

Curiosidade

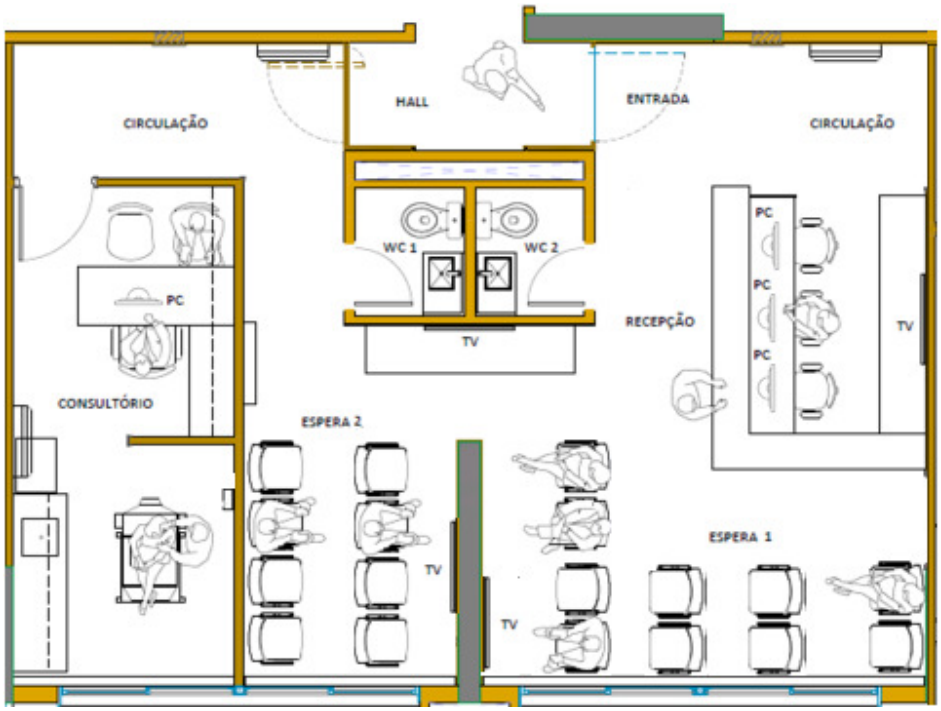
Mas porque em Java Observer é uma interface? Provavelmente pelo fato de em Java GUI's comumente estenderem de JFrame e como não há herança múltipla na linguagem e essas serem justamente as observadoras, então como solução tem-se o uso de interface, obrigando sobrescrever o método abstrato para atualização de uma GUI quando houver mudança na classe que esta observa!



Mão na Massa!

1) Pela primeira vez o programador *de codinome mustela putórius furo – "O Furão"* – conseguiu solucionar uma demanda de sua empresa, na qual o proprietário de uma clínica Dr. MPOO solicitou um sistema simples apenas para exibir nas telas do seu consultório a ficha de atendimento de seus pacientes e estes saberem quando se dirigir ao consultório. Inicialmente o consultório só possuía 1 *All in One* na sala do médico, no qual este controlava a chamada do próximo atendimento e duas TVs cada uma em uma sala de espera, essas conectadas como monitores estendidos do computador do médico. Abaixo estão as janelas da solução de O Furão:

Entretanto a clínica expandiu e Dr. MPOO solicitou uma nova demanda a empresa, tendo em vista sua nova infraestrutura:



Observe agora a existência de, além das TV's das salas de espera e o computador do consultório, uma TV na Recepção e outra entre as salas de espera e três *All in One* para as Recepcionistas. É demanda de Dr. MPOO:

- Em todas as TV's deve aparecer o número da Ficha de Atendimento e o nome do Paciente;
- Os computadores das recepcionistas deve possuir um sistema para cadastro do paciente e a possibilidade de configurar o número da Ficha de Atendimento que aparecerá em todos os monitores/TV's. As telas devem ser acessadas por opções dispostas em JMenuBar. Nesses computadores não é possível ver os dados da consultada do paciente;
- A tela de Dr. MPOO deve possuir:
  - Possibilidade de gerenciar os dados de um paciente; e
  - Configurar o número da ficha de atendimento e botão para chamar o próximo paciente.

(continua na próxima página...)

Mais uma vez *O Furão* falhou (*por não ter estudado design pattern*) e não saber como atender uma solução proposta pelo seu *Product Owner*, a qual deve estar em conformidade com o diagrama de classes abaixo. Aproveite a oportunidade e mostre que você tem a capacidade de ocupar a vaga de programador sênior na empresa. Observe a possibilidade de utilização de componentes gráficos para facilitar o formulário:

JComboBox para estado;

- JRadioButton para casa ou apartamento;
- Uso de máscara para cpf, telefone e cep;
- E outros que possam facilitar a interação com o usuário.

Também observe a utilização de Herança para evitar a duplicidade de código e o uso do padrão comportamental Observer no padrão arquitetural MVC, possibilitando que futuras telas possam ser atualizadas diretamente pela notificação do model!