Compartilhar o seu link com: luciorocha @ professores.utfpr.edu.br
Alexandre Stoll Calisto: Aula 27

Guilherme C.Ramalho: Copia de Aula 27 - POCO4A - Guilherme Ramalho
Lucas Ribeiro P. Maroja: POCO4A - 22/11/2022
Matheus H. A. Pereira: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Andrei Fernandes Zani e Ruan Perondi:
Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Maria Eduarda: Pedroso - Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Plinio e Fernando: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Vitor Batista: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Arnald: Aula 27 - 22/11
Michael Pariz: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Iago Macarini: Aula 27 - POCO4A
Lucas Santana: - POCO4A - 22/11/2022 - Lucas Santana
Mariana Pedroso Naves: Aula 27 - POO
Gabriel Takeshi: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Leonardo G. Fagote Aula 27 - Padrões de Projeto
Erik Noda: Aula27
Vinicius Letroche felix: Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
João Pedro de Paula: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Raphael Uematsu: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Gustavo Naoki Jodai Kurozawa: Aula 27 - POCO4A - 22/11/2022
William Eizo Hatakeyama:
Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Camila Costa: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Gabriel Candelária: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Gustavo Nunes : Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Felipe Antonio Magro: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Rodrigo Leandro Benedito:
Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos
Julio Cesar Farias: Cópia de Aula 27 - POCO4A - 22/11/2022 - Exercícios propostos

1) Padrão de Projeto: Singleton.

   Motivação: manter uma única instância ativa de um objeto de determinada classe.

   Passo 1: Criar classe com construtor private e variável de classe private.

```
public class Prateleira {

	private static Prateleira estoque;

	private Prateleira (){
		System.out.println("Singleton iniciado.");
	}

	public static Prateleira iniciar(){
		if(estoque==null)
			estoque = new Prateleira();
		return estoque;
	}

}
```

Passo 2: Instanciar o singleton em outra classe.

```
public class Principal {

	public void iniciar(){

		Prateleira adm = Prateleira.iniciar();
		Prateleira adm2 = Prateleira.iniciar();

	}

	public static void main(String[] args) {
		new Principal().iniciar();

	}
}
```

1.1) (Online) Exercício: Acesse o link da atividade (Aula25prog5): https://codeboard.io/projects/359841

```
/**
 * TODO1: Classe Estoque: implemente a classe para que ela seja um Singleton.
 *
 * TODO2: Classe Principal: faca a tentativa de instanciar 2 (dois) objetos da
 *        classe Estoque.
 *
 * TODO3: Estoque: Modifique a classe Estoque para que ela adicione a lista de
 *        MaterialEscolar.
 *
```

```
 * TODO4: Classe Principal: exiba todos os itens da lista do Estoque.
 */

import java.util.ArrayList;

public class Principal {

   public class MaterialEscolar {
   private String nome;
   public MaterialEscolar(String nome){
      this.nome=nome;
   }
   public String toString(){
      return this.nome;
   }

   }
   ArrayList<MaterialEscolar> lista;

   public interface Iterator {
      public boolean temAnterior();
      public Object anterior();
   }

public class ItemIterator implements Iterator {

         private ArrayList< MaterialEscolar > lista;
         private int pos=0;

         public ItemIterator(ArrayList<MaterialEscolar> lista){
                  this.lista = lista;

                  //pos=this.lista.size()-1;
         }

         public boolean temAnterior() {
                  boolean result=false;
                  if ( pos >= 0 )
                           result=true;
                  return result;
         }
         public Object anterior() {
                  MaterialEscolar item  = this.lista.get(pos);
                  pos--;
                  return item;
         }

         public boolean temProximo() {
```

```
                boolean result=false;
                if ( pos < this.lista.size() )
                        result=true;
                return result;
        }

        public Object proximo() {
                MaterialEscolar item  = this.lista.get(pos);
                pos++;
                return item;
        }
}

   public void iniciar(){
      lista = new ArrayList<>();
      lista.add(new MaterialEscolar("Lapis"));
      lista.add(new MaterialEscolar("Borracha"));



      //TODO2
      Estoque estoque = Estoque.iniciar( lista );

      Estoque estoque2 = Estoque.iniciar( lista );

      //TODO4
      ItemIterator i = new ItemIterator( estoque.getLista() );
      while( i.temProximo() )
        System.out.println( (MaterialEscolar) i.proximo() );

   }

   public static void main(String[] args) {
      Principal p = new Principal();
      p.iniciar();
   }
}

//
//TODO1
import java.util.ArrayList;
public class Estoque {

   private static Estoque objeto;
   private ArrayList<Principal.MaterialEscolar> lista;

   private Estoque(ArrayList<Principal.MaterialEscolar> lista){
      System.out.println("CRIADO");
```

```
        this.lista = lista;
    }
    public ArrayList<Principal.MaterialEscolar> getLista(){
        return this.lista;
    }
    public static Estoque iniciar(
        ArrayList<Principal.MaterialEscolar> lista){

        if( objeto == null )
            objeto = new Estoque( lista );

        return objeto;
    }

}
```

1.2) (Online) Exercício: Acesse o link da atividade (Aula25prog6): https://codeboard.io/projects/359856

2) Padrão de Projeto Visitor:

Motivação: filtrar de maneiras diferentes objetos de classes diferentes mantendo as mesmas assinaturas.

Exemplo2:

```
package javaapplication1;


public class Principal {

    public interface IFiltro {
        public abstract void imprimir(Fruta fruta);
    }

    public class FiltroPreco implements IFiltro {
        public void imprimir(Fruta fruta){
            System.out.println(fruta.getPreco());
        }
```

```
    }
    public class FiltroNome implements IFiltro {
        public void imprimir(Fruta fruta){
            System.out.println(fruta.getNome());
        }
    }

    public interface Fruta {
        public float getPreco();
        public String getNome();
    }

    public class Abacate implements Fruta {
        private String nome;
        private float preco;
        public Abacate(String nome, float preco){
            this.nome=nome;
            this.preco=preco;
        }
        public float getPreco(){ return this.preco; }
        public String getNome(){ return this.nome; }
    }

    public void iniciar(){
        Fruta fruta = new Abacate("ABACATE", 1.5f);
        IFiltro filtro = new FiltroPreco();

        filtro.imprimir( fruta );

        filtro = new FiltroNome();

        filtro.imprimir( fruta );


    }


    public static void main(String[] args) {
        Principal principal =  new Principal();
        principal.iniciar();
    }

}
```

Exemplo1:

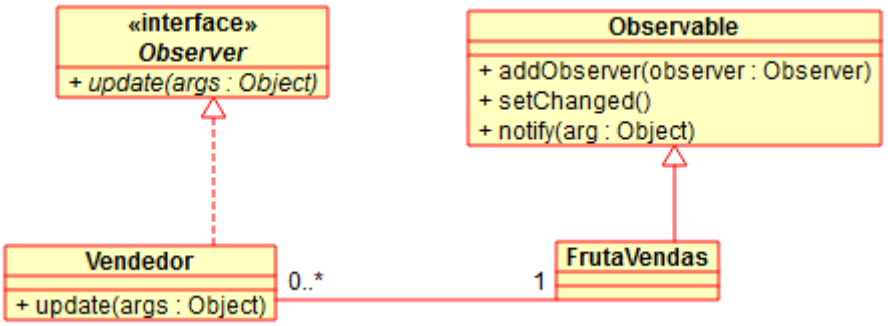
```
public class Principalb {

	public interface IFiltro {
		public abstract void filtrar(IFruta fruta);
	}

	public class FiltroMaca implements IFiltro {
		public void filtrar(IFruta fruta){
			System.out.println("Filtro 1");
		}
	}
	public class FiltroAbacate implements IFiltro {
		public void filtrar(IFruta fruta){
			System.out.println("Filtro 2");
		}
	}

	public interface IFruta {
		public abstract void imprimir();
	}
	public class Abacate implements IFruta {
		public void imprimir(){

		}
	}
	public void imprimir(IFiltro filtro, IFruta fruta){
		filtro.filtrar(fruta);
	}

	public Principalb(){

		IFiltro filtro = new FiltroMaca();
		IFruta fruta = new Abacate();
		imprimir(filtro, fruta);

		filtro = new FiltroAbacate();
		imprimir(filtro, fruta);
	}

	public static void main (String [] args){

		new Principalb();

	}//fim main

}//fim classe
```

2.1) (Online) Exercício: Acesse o link da atividade (Aula27prog1): https://codeboard.io/projects/360577

2.2) (Online) Exercício: Acesse o link da atividade (Aula27prog2): https://codeboard.io/projects/360580

3) Padrão de Projeto: Observer

Motivação: receber notificação de alteração de instâncias.



Passo 1: A classe observadora deve implementar a interface 'java.util.Observer':

```
public class Vendedor implements java.util.Observer {

		private String nome;
		public Vendedor(String nome){
			this.nome=nome;
		}

		@Override
		public void update(Observable remoto, Object atributoRemoto) {
			System.out.println(this +": Recebi atualizacao de [" + remoto + "]" +
					" que mudou o seguinte atributo: " +
atributoRemoto);
		}
		public String toString(){
			return this.nome;
		}

	}
```

Passo 2: A classe observável deve ser subclasse de ‘java.util.Observable’:

```
public class FrutaVendas extends java.util.Observable {
		private String nome;
		private int estoque;
		public FrutaVendas(String nome, int estoque){
			this.nome=nome;
			this.estoque=estoque;
		}
		public void venda(int qtde){
			if(estoque>0){
				estoque -= qtde;
				this.setChanged(); //Mudou o estado do objeto
				this.notifyObservers("estoque: " + this.estoque); //Notifique
todos os Observers
			}
		}//fim venda
		public String toString(){
			return this.getClass().getSimpleName();
		}
	}//fim classe
```

Passo 3: A instância da classe observável deve adicionar um observador:

```
public void iniciar(){
		FrutaVendas fruta = new FrutaVendas("Abacate",100); //nome,estoque

		Vendedor joao = new Vendedor("joao");
		Vendedor maria = new Vendedor("MARIA");

		fruta.addObserver(joao);
		fruta.addObserver(maria);

		fruta.venda(10);
	}
```

3.1) (Online) Exercício: Acesse o link da atividade (Aula27prog3): https://codeboard.io/projects/360597

3.2) (Online) Exercício: Acesse o link da atividade (Aula27prog4): https://codeboard.io/projects/360600

4) Padrão de Projeto: Template

Motivação: Invocar na superclasse métodos da subclasse.

```
public class Principalb {

	public interface ISuperClasse {
		//Visivel no package + subclasses
		public abstract String imprimir();
	}

	public abstract class SuperClasse implements ISuperClasse {

		public SuperClasse(){
			imprimirSubClasse();
		}

		public void imprimirSubClasse(){
			System.out.println("Estou na superclasse: " +
					this.imprimir()); //Invoca informacoes da subclasse
		}
	}

	public class SubClasse extends SuperClasse {

		public SubClasse(){
		}

		public final String imprimir(){
			return this.getClass().getSimpleName()+"";
		}
	}

	public Principalb(){
		SubClasse s = new SubClasse();
	}

	public static void main (String [] args){

		new Principalb();

	}//fim main

}//fim classe
```