UNIVERSIDADE FEDERAL DO PAMPA
UNIPAMPA

CURSO DE ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA DE ENGENHARIA DE SOFTWARE

Luciano Brum e Thiago Dantas

Prof. Carlos Michel Betemps

SEMESTRE 2013.01

O que são padrões de projeto?

-Um Padrão de Projeto de Software ou Padrão de Desenho de Software descreve uma solução geral reutilizável para um problema comum no desenvolvimento de sistemas de software orientado a objetos.

- Os padrões não são criados e sim descobertos, pois a todo momento existem diversas formas de descobrir problemas na hora de implementar algum sistema, e a solução desse problema, é a forma geral de um padrão de projeto.

Elementos essenciais dos padrões de projeto.

-Nome. Uma referência para descrever um problema de projeto, suas soluções e consequências.

-Problema. Descreve em que situação aplicar o padrão.

-Solução. Descrição abstrata de um problema de projeto e de como um arranjo geral de elementos o resolve.

-Conseqüência. São críticas para a avaliação de alternativas de projetos e para a compreensão dos custos e benefícios da aplicação do padrão

Razões para usar padrões de projeto

-Aprender com a experiência dos outros.

- Identificar problemas comuns na estrutura e sintaxe do código.

- Aprender boas práticas de programação.

-Uso eficiente da Herança.

-Compreensão e documentação.

-Ter um caminho e um alvo para refatorações.

Padrões de projeto GOF

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Padrões que serão aplicados

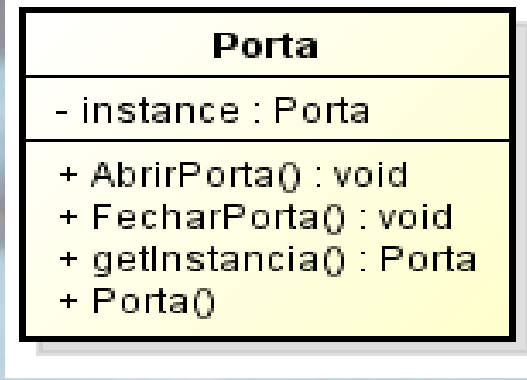
- Singleton: Uma única instância e um ponto de acesso global a essa única instância.

- Decorator: Adicionar responsabilidades a objetos, sem alterar outros objetos e seu Componente.

- Interpreter: Representação para uma gramática junto com um interpretador que usa a representação para interpretar sentenças dessa linguagem.

Singleton: classe Porta.

```
public class Porta{
        private static Porta instance = null;
         private porta{
         …
         }
        public static synchronyzed Porta getInstancia(){
           if(instance==null){
           instance = new Porta();
           …
           }
        return instance;
        }
        public void AbrirPorta(){
        System.out.print(“Porta foi aberta.”);
        }
        public void FecharPorta(){
        System.out.print(“Porta foi fechada.”);
        }
 }
```



Singleton: com uso de subclasses.

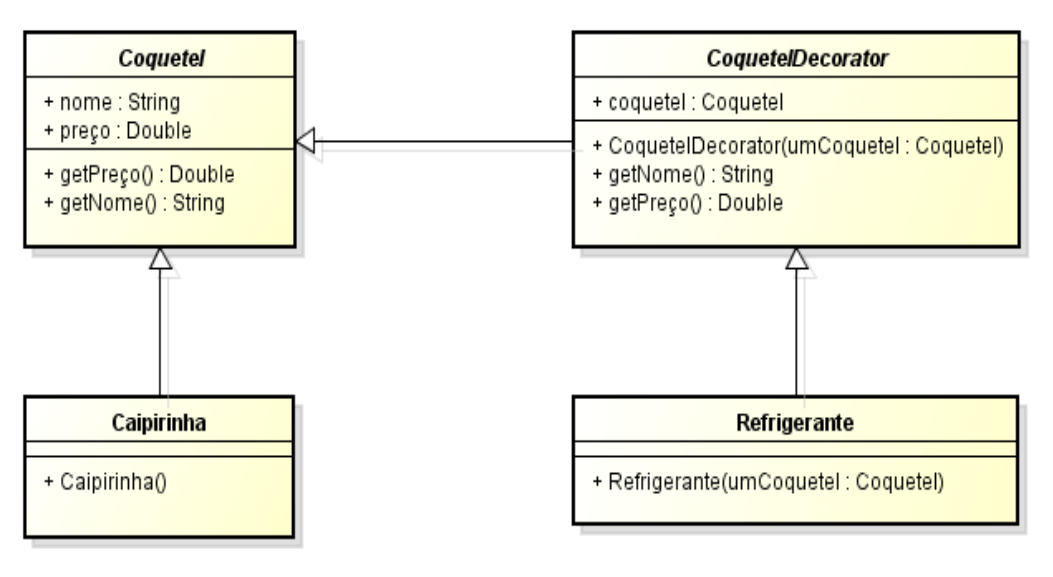
```
public class Porta{
        private static Porta instance = null;
        public String tipoDePorta;
        private porta{
        …
        }
        public static synchronyzed Porta getInstancia(){
           if(instance==null){
               if(tipoDePorta.contains(”ferro”)){
                  instance = new PortaFerro();
                  return instance;
                }
               if(tipoDePorta.contains(“madeira”)){
                  instance = new PortaMadeira();
                  return instance;
                }
                else{
                 instance = new Porta();
                }
            }
         return instancia;
         }
```

Singleton: vantagens e desvantagens.

Vantagens:
Acesso controlado a única instância da classe Porta.
Evita múltiplas instâncias não utilizadas ou utilizadas por muito pouco tempo.

Desvantagens: Synchronyzed não permite acessos simultâneos à instância, gerando certa lentidão no sistema, mas nesse caso é necessário.

Decorator: classe Coquetel.



Decorator: classe Coquetel.

```
Public abstract class Coquetel {
        String nome;
        double preco;
        public String getNome() {
        return nome;}
        public double getPreco() {
        return preco;}
 }
        public class Caipirinha extends Coquetel {
        public Caipirinha() {
        nome = "Caipirinha";
        preco = 3.5;}
        }
        public abstract class CoquetelDecorator extends Coquetel{
        Coquetel coquetel;
        public CoquetelDecorator(Coquetel umCoquetel){
        coquetel = umCoquetel;}
        public String getNome(){
        return coquetel.getNome()  + " + " + nome;}
        public double getPreco() {
        return coquetel.getPreco() + preco;}
        }
```

Decorator: classe Coquetel.

```
public class Refrigerante extends CoquetelDecorator {
        public Refrigerante(Coquetel umCoquetel) {
        super(umCoquetel);
        nome = "Refrigerante";
        preco = 1.0;}
        }
        public class Teste{
        public static void main(String[] args) {
        Coquetel meuCoquetel = new Caipirinha();
        System.out.println(meuCoquetel.getNome() + " =
"+meuCoquetel.getPreco());
        meuCoquetel = new Refrigerante(meuCoquetel);
        System.out.println(meuCoquetel.getNome() + " = "+
meuCoquetel.getPreco()); } }
```

Decorator: vantagens e desvantagens.

Vantagens:

-Adição de funcionalidades aos objetos em tempo de execução, sem alterar outros objetos e componente.

-Flexibilidade na criação de bebidas, que não seria possível com uso de subclasses .

-Permitem acrescentar a mesma funcionalidade 2 vezes, e com subclasses, é impossível estender uma classe 2 vezes.

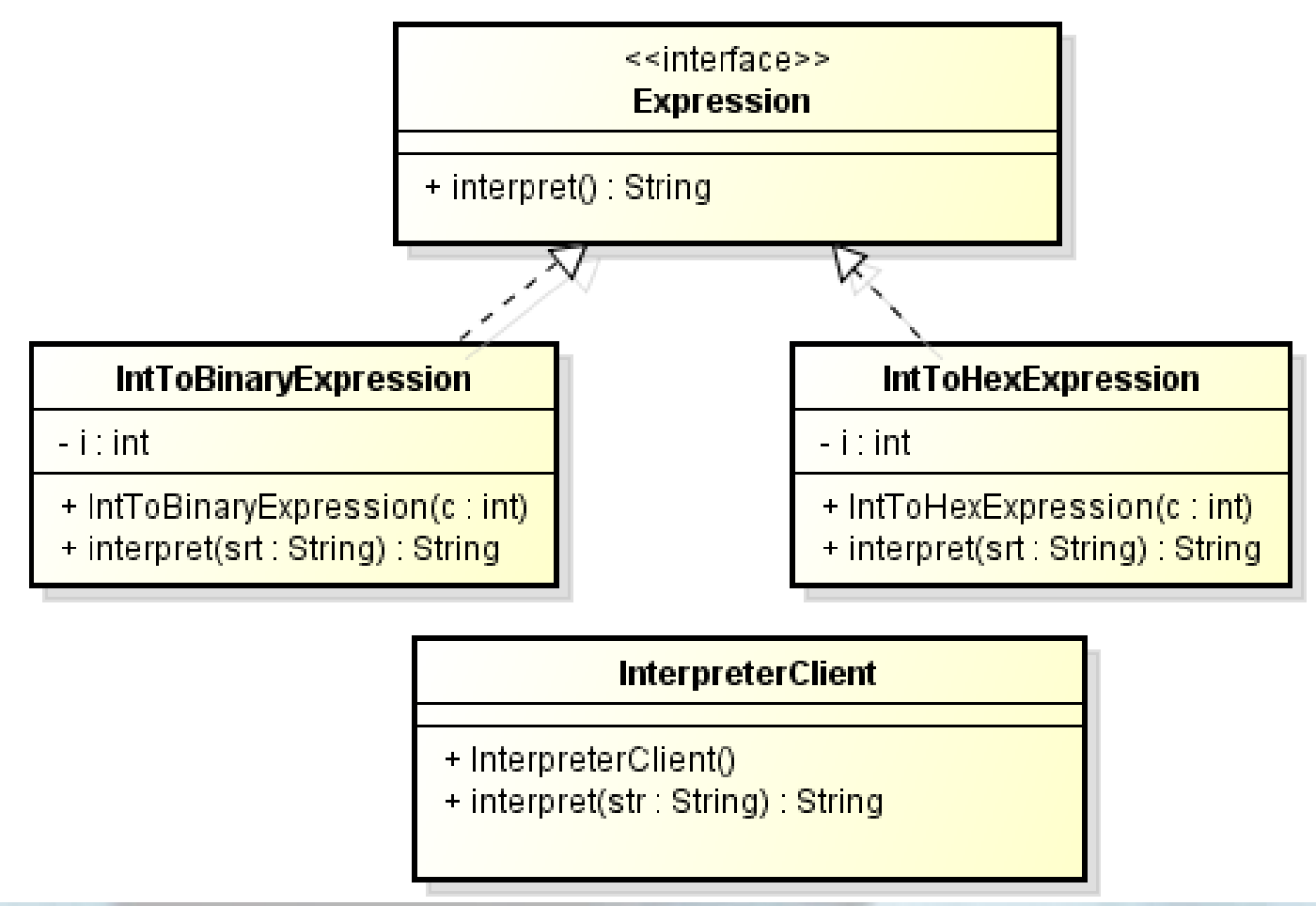
-Evita sobrecarga de funcionalidades na camada superior da hierarquia de classes.

Desvantagens:

-O sistema não pode depender da identidade dos objetos quando usa o padrão Decorator.

-O sistema resulta em uma grande quantidade de objetos pequenos e parecidos que implica em um sistema complexo e de difícil depuração para quem não projetou o sistema.

Interpreter: interface Expression.



Interpreter: interface Expression.

```
public interface Expression {
        String interpret(InterpreterContext ic);}

        public class IntToBinaryExpression implements Expression {
        private int i;
        public IntToBinaryExpression(int c){
        this.i=c;}
        public String interpret(){
        return Integer.toBinaryString(this.i);}
        }

        public class IntToHexExpression implements Expression {
          private int i;
          public IntToHexExpression(int c){
          this.i=c;}
        public String interpret(){
        return Integer.toHexString(this.i);}
        }
```

Interpreter: interface Expression.

```
public class InterpreterClient {
        public String interpret(String str){
        Expression exp=null;
        if(str.contains("Hexadecimal")){
        exp=new IntToHexExpression(Integer.parseInt (str.substring(0,str.indexOf(" "))));
        }else if(str.contains("Binario")){
          exp=new IntToBinaryExpression(Integer.parseInt(str.substring(0,str.indexOf(" "))));
        }else return str;
        return exp.interpret();
        }
 }

public class Teste{
        public static void main(String args[]){
        String binario = "28 em Binario";
        String hexadecimal = "28 em Hexadecimal";
        InterpreterClient cliente = new InterpreterClient();
        System.out.println(binario+"= "+cliente.interpret(binario));
        System.out.println(hexadecimal+"= "+cliente.interpret(hexadecimal));
        }
```

Interpreter: vantagens e desvantagens

Vantagens:

-É fácil estender a gramática.

-Implementar a gramática também é fácil, pois apenas é avaliada uma String antes do interpretador traduzir o significado da expressão.

Desvantagens:

-Para cada regra de gramática é criada uma classe que interpreta a expressão, e para um número grande de regras, induz em um número grande de classes, causando muita complexidade no sistema e difícil manutenção.

-Mudanças na forma de interpretar, ou novas formas de interpretar uma expressão é muito exaustivo e dispendioso para um sistema que já está em funcionamento.

Referências bibliográficas:

Disponível em http://www.inf.ufsc.br/~bosco/extensao/NovosTalentos2012/D:/additional/addnlApps/jhtp6_appM_design_patterns.pdf> Visitado em 12/09/13 ás 17h.

Disponível em <http://www.inf.ufg.br/~fabrizzio/web/java/aula8.pdf> Visitado em 12/09/13 ás 18h.

[GHJV 2000] GAMMA, Eric; Helm, R.; Johnson, R.; Vlissides, J. Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. Porto Alegre: Bookman, 2000.

Disponível em <http://brizeno.wordpress.com/2011/12/12/classificacao-dos- padroes-de-projeto-gof/> Visi em 20/09/13 ás 20h30min.

Disponível em <http://www.devmedia.com.br/trabalhando-com-singleton-java/23632> Visitado em 20/09/ ás 23h10min.

Disponível em <http://www.tutorialspoint.com/design_pattern/interpreter_pattern.htm> Visitado em 20/0 ás 3h.

Disponível em <http://www.ufpa.br/cdesouza/teaching/es/8-patterns.pdf> Visitado em 20/09/13 ás 5h30m