DCC – UFMG – Especialização em Engenharia de Software

Trabalho Prático 2 : Hierarquia e Polimorfismo – Livro de Receitas

Ambientes de Programação – Prof. Roberto S. Bigonha

Charles Wellington de Oliveira Fortes

Belo Horizonte
Março de 2011

Descrição

Neste trabalho devem ser praticados os conceitos de hierarquia, herança e polimorfismo em Java para a criação de um programa de livro de receitas.

Detalhes da implementação

Arquitetura

O sistema foi implementado conforme o modelo de classes exibido abaixo (imagem1):

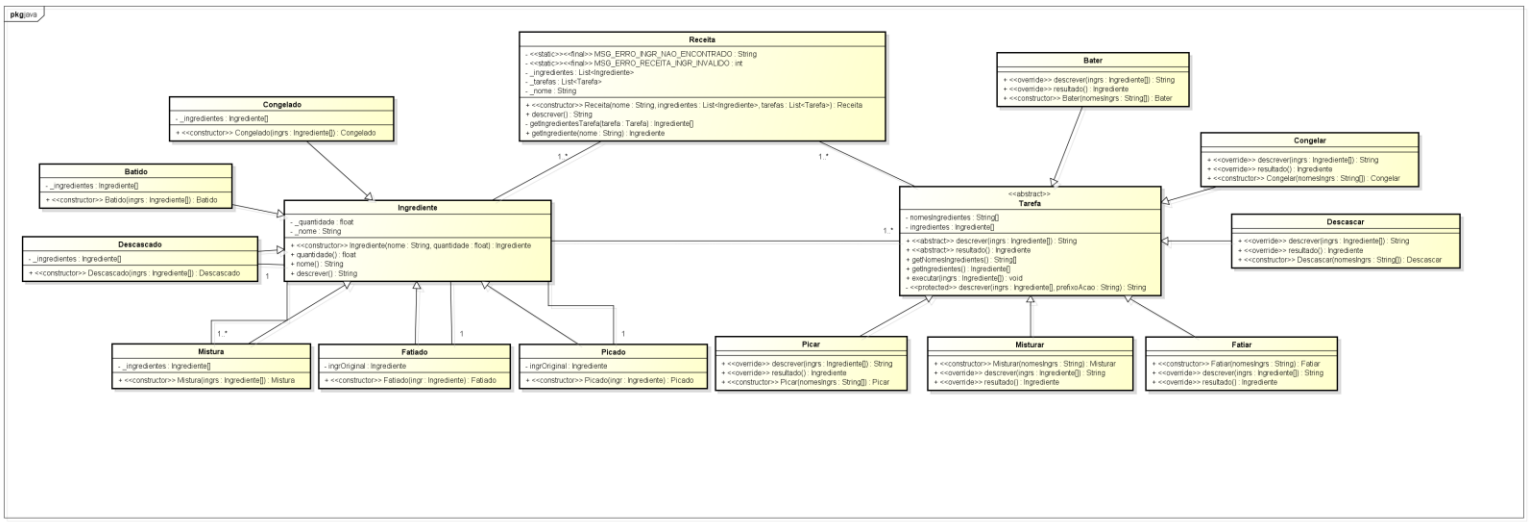

Imagem1 – Diagrama de classes do sistema

Classe Receita

A classe receita é responsável pela representação da receita no sistema, ela é composta por uma lista de ingredientes, uma lista de tarefas e métodos auxiliares para se descrever uma receita.

Foram criados dois atributos static final para representar as mensagens de erro exibidas pelo sistema, que são exibidas quando um ingrediente não é localizado ou quando um ingrediente necessário a uma tarefa não está presente no sistema.

> **Método descrever**: retorna a descrição completa da receita → ingredientes necessários, passos a serem executados e resultado final da receita.
>
> **Método getIngredientesTarefa**: método privado que recebe uma tarefa e retorna uma lista contendo os ingredientes necessários para sua execução. Estes ingredientes devem estar presentes na lista de ingredientes da receita. Caso algum ingrediente esteja faltando, será levantada uma exceção informando a falta do ingrediente.

Método getIngrediente: recebe a descrição de um ingrediente e retorna o ingrediente correspondente. Este ingrediente deve estar na lista de ingredientes da receita, caso contrário uma exceção será levantada.

Classe Ingrediente

A classe ingrediente é responsável pela representação de um ingrediente da receita, possuindo um nome e uma quantidade.

Método descrever: retorna a descrição do ingrediente → <quantidade> + " " + <nome>

Classe Tarefa

A classe abstrata tarefa representa uma ação a ser executada com um ou mais ingredientes da receita, o que gera um novo ingrediente que é um subproduto desta ação e que passa a ser parte dos ingredientes da receita, sendo utilizados posteriormente em novo passos do sistema.

Uma tarefa não pode ser diretamente instanciada, por isto é definida como estática, e descreve as assinaturas "descrever" e "resultado" para serem implementadas em suas especializações.

Método abstrato descrever: recebe uma lista com os ingredientes do sistema e gera uma descrição da ação a ser executada (não executa a ação, por tanto não gera um resultado a ação – para a criação do subproduto da ação deve ser executado o método executar) → ex.: para tarefa "picar" relacionada ao "ingrediente" tomate → "picar tomate".

Método abstrato resultado: retorna o ingrediente que representa o subproduto de uma ação

Método executar: recebe uma lista de ingredientes para a execução de uma ação e armazena o subproduto desta ação, um ingrediente, para ser retornado no método resultado.

A opção pela criação do método "executar" foi tida para separação de responsabilidades, por exemplo: quando perguntamos a uma dona de casa que descreva o processo de se fazer um bolo ela não precisa ir fazer o bolo e em seguida te contar como foi, basta ela contar como será feito, e se você quiser o bolo ai sim, peça que ela o faça.

Método getNomesIngredientes: retorna uma lista com o nome de todos os ingredientes que fazem parte desta tarefa.

Método getIngredientes: retorna a lista de ingredientes instanciados para a execução da tarefa.

Método protegido descrever: método criado para encapsular diversos aspectos comuns a todas as descrições de tarefas, tornando o código mais reaproveitável.

Interface IResultado

Foi criada uma interface IResultado utilizada como recurso de marcação especial para as classes do tipo Ingrediente porém que são subprodutos de uma execução de determinada tarefa, por isto, nenhuma assinatura é definida nesta interface. Através desta marcação o sistema discerne o que deve ser impresso como sendo um ingrediente necessário para uma tarefa e o que será utilizado apenas para a composição de uma nova ação futura.

Classes de Tarefas:

As classes de tarefas do sistema são: *Picar, Misturar, Fatiar, Descascar, Congelar e Bater*, sendo que *Congelar e Bater* foram criadas para atenderem a receita criada para o teste2 do programa.

Todas as classes de tarefas implementas os mesmos métodos, possuindo estes os mesmos propósitos, sendos eles:

Método descrever: recebe uma lista de ingredientes e retorna a descrição da tarefa

Método resultado: retorna o subproduto da execução da tarefa

Classes de resultado:

As classes de resultado são ingredientes subprodutos da execução de uma tarefa, sendo que elas são: *Picado, Fatiado, Mistura, Descascado, Batido e Congelado*, sendo que as classes *Batido e Congelado* foram criadas para atenderem as necessidades do teste2 do programa.

Além de extenderem as classes de ingredientes, as classes de resultado implementam a interface IResultado como forma de marcação para diferenciação destas dos demais ingredientes da receita.

Todas estas classes implementam apenas seu construtor, que recebe como parâmetro o ingrediente original que o gerou, com exceção a casse Mistura que implementa uma lista de ingredientes originais.

Saídas

1) **Teste 1 : Molho a campanha (vinagrete)**

```
run:
 Molho a campanha (Vinagrete)
Ingredientes:

    2.0 Tomate

    1.0 Cebola

    0.5 Pimentão

    40.0 Vinagre branco

    20.0 Azeite


Modo de preparo:

    Picar Tomate
    Descascar Cebola
    Picar Cebola descascado
```

Picar Pimentão
Misturar Cebola descascado picado e Pimentão picado e Tomate picado
Misturar Vinagre branco e Mistura de (Cebola descascado picado e Pimentão picado e Tomate picado)
Misturar Azeite e Mistura de (Vinagre branco e Mistura de (Cebola descascado picado e Pimentão picado e Tomate picado))

Resultado:

Mistura de (Azeite e Mistura de (Vinagre branco e Mistura de (Cebola descascado picado e Pimentão picado e Tomate picado)))
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

2) Teste 2 : Mousse de Maracujá

run:
Mousse de Maracujá
Ingredientes:

200.0 Creme de Leite

300.0 Leite Condensado

1.0 Suco de Maracujá

Modo de preparo:

Bater Creme de Leite e Leite Condensado
Misturar Suco de Maracujá e Mistura de (Creme de Leite batido e Leite Condensado batido)
Bater Mistura de (Mistura de (Creme de Leite batido e Leite Condensado batido) e Suco de Maracujá)
Congelar Mistura de (Mistura de (Creme de Leite batido e Leite Condensado batido) e Suco de Maracujá) batido

Resultado:

Mistura de (Mistura de (Creme de Leite batido e Leite Condensado batido) e Suco de Maracujá) batido congelado
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Implementação:

```
package receitas.DomainObjects;

import java.util.*;

/**
 *
 * @author Charles.Fortes
 */
public class Bater extends Tarefa {
    public Bater(String[] nomesIngrs)
    {
        super(nomesIngrs);
    }

    @Override
    public String descrever(Ingrediente[] ingrs) {
        return super.descrever("Bater", ingrs);
    }

    @Override
    public Ingrediente resultado() {
        if (getIngredientes().length > 1){
            List<Batido> f = new ArrayList<Batido>();
            for (Ingrediente ingr : getIngredientes())
                f.add(new Batido(ingr));
            return new Mistura(f.toArray(new Batido[f.size()]));
        }
        else
            return new Batido(getIngredientes()[0]);

    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Batido extends Ingrediente implements IResultado {
    Ingrediente ingrOriginal;

    Batido(Ingrediente ingr) {
        super(ingr.nome() + " batido", ingr.quantidade());
        ingrOriginal = ingr;
    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Congelado extends Ingrediente implements IResultado {
    Ingrediente ingrOriginal;

    Congelado(Ingrediente ingr) {
        super(ingr.nome() + " congelado", ingr.quantidade());
```

```
        ingrOriginal = ingr;
    }
}
```

```
package receitas.DomainObjects;

import java.util.*;

/**
 *
 * @author Charles.Fortes
 */
public class Congelar extends Tarefa {
    public Congelar(String[] nomesIngrs)
    {
        super(nomesIngrs);
    }

    @Override
    public String descrever(Ingrediente[] ingrs) {
        return super.descrever("Congelar", ingrs);
    }

    @Override
    public Ingrediente resultado() {
        if (getIngredientes().length > 1){
            List<Congelado> f = new ArrayList<Congelado>();
            for (Ingrediente ingr : getIngredientes())
                f.add(new Congelado(ingr));
            return new Mistura(f.toArray(new Congelado[f.size()]));
        }
        else
            return new Congelado(getIngredientes()[0]);

    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Descascado extends Ingrediente implements IResultado {
    Ingrediente ingrOriginal;

    Descascado(Ingrediente ingr) {
        super(ingr.nome() + " descascado", ingr.quantidade());
        ingrOriginal = ingr;
    }
}
```

```
package receitas.DomainObjects;

import java.util.*;

/**
 *
 * @author Charles.Fortes
 */
public class Descascar extends Tarefa {
    public Descascar(String[] nomesIngrs)
```

```
    {
        super(nomesIngrs);
    }

    @Override
    public String descrever(Ingrediente[] ingrs) {
        return super.descrever("Descascar", ingrs);
    }

    @Override
    public Ingrediente resultado() {
        if (getIngredientes().length > 1){
            List<Descascado> f = new ArrayList<Descascado>();
            for (Ingrediente ingr : getIngredientes())
                f.add(new Descascado(ingr));
            return new Mistura(f.toArray(new Descascado[f.size()]));
        }
        else
            return new Descascado(getIngredientes()[0]);

    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Fatiado extends Ingrediente implements IResultado {
    Ingrediente ingrOriginal;

    Fatiado(Ingrediente ingr) {
        super(ingr.nome() + " fatiado", ingr.quantidade());
        ingrOriginal = ingr;
    }
}
```

```
package receitas.DomainObjects;

import java.util.*;

/**
 *
 * @author Charles.Fortes
 */
public class Fatiar extends Tarefa {
    public Fatiar(String[] nomesIngrs)
    {
        super(nomesIngrs);
    }

    @Override
    public String descrever(Ingrediente[] ingrs) {
        return super.descrever("Fatiar", ingrs);
    }

    @Override
    public Ingrediente resultado() {
        if (getIngredientes().length > 1){
            List<Fatiado> f = new ArrayList<Fatiado>();
            for (Ingrediente ingr : getIngredientes())
                f.add(new Fatiado(ingr));
            return new Mistura(f.toArray(new Fatiado[f.size()]));
```

```
        }
        else
            return new Fatiado(getIngredientes()[0]);
    }
}
```

```
package receitas.DomainObjects;
```

```
/**
 *
 * @author Charles.Fortes
 */
public interface IResultado {

}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Ingrediente {
    private float _quantidade;
    private String _nome;

    public Ingrediente(String nome, float quantidade)
    {
        _quantidade = quantidade; _nome = nome;
    }

    public float quantidade(){ return _quantidade; }

    public String nome() { return _nome; }

    public String descrever()
    {
        return quantidade() + " " + nome();
    }

    protected void setNome(String nome)
    {
        _nome = nome;
    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Mistura extends Ingrediente implements IResultado{
    private Ingrediente[] _ingredientes;

    public Mistura(Ingrediente[] ingrs)
    {
        super("Mistura_" + ingrs.length, ingrs.length);
        _ingredientes = ingrs;
        setNome(this.descrever());
    }
```

```
    @Override
    public String descrever()
    {
        StringBuilder str = new StringBuilder();

        str.append("Mistura de (");

        for (int i = 0; i < _ingredientes.length; i++)
        {
            str.append(_ingredientes[i].nome()).append((i < _ingredientes.length - 1) ? " e " : "");
        }

        str.append(")");

        return str.toString();
    }
}
```

```
package receitas.DomainObjects;

import java.util.*;

/**
 *
 * @author Charles.Fortes
 */
public class Misturar extends Tarefa {
    public Misturar(String[] nomesIngrs)
    {
        super(nomesIngrs);
    }

    @Override
    public String descrever(Ingrediente[] ingrs) {
        return super.descrever("Misturar", ingrs);
    }

    @Override
    public Ingrediente resultado() {
        return new Mistura(getIngredientes());
    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public class Picado extends Ingrediente implements IResultado{
    Ingrediente ingrOriginal;

    Picado(Ingrediente ingr) {
        super(ingr.nome() + " picado", ingr.quantidade());
        ingrOriginal = ingr;
    }
}
```

```
package receitas.DomainObjects;

import java.util.*;
```

```
/**
 *
 * @author Charles.Fortes
 */
public class Picar extends Tarefa {
    public Picar(String[] nomesIngrs)
    {
        super(nomesIngrs);
    }

    @Override
    public String descrever(Ingrediente[] ingrs) {
        return super.descrever("Picar", ingrs);
    }

    @Override
    public Ingrediente resultado() {
        if (getIngredientes().length > 1){
            List<Picado> f = new ArrayList<Picado>();
            for (Ingrediente ingr : getIngredientes())
                f.add(new Picado(ingr));
            return new Mistura(f.toArray(new Picado[f.size()]));
        }
        else
            return new Picado(getIngredientes()[0]);

    }
}
```

```
package receitas.DomainObjects;

import java.util.*;

/**
 *
 * @author Charles.Fortes
 */
public class Receita {
    private static final String MSG_ERRO_INGR_NAO_ENCONTRADO =
            "Ingrediente não encontrado na receita.";
    private static final String MSG_ERRO_RECEITA_INGR_INVALIDO =
            "Existem ingredientes inválidos na receita, favor verificar a lista "
            + "de ingredientes.";

    private List<Ingrediente> _ingredientes;
    private List<Tarefa> _tarefas;
    private String _nome;

    public Receita(String nome, List<Ingrediente> ingredientes, List<Tarefa> tarefas)
    {
        _nome = nome;
        _ingredientes = ingredientes;
        _tarefas = tarefas;
    }
    public String descrever()
    {
        StringBuilder str = new StringBuilder();

        str.append(" ").append(_nome);
        str.append("\nIngredientes:\n\n");
        for (Ingrediente ingr : _ingredientes)
        {
            if (!(ingr instanceof IResultado))
            str.append("\t").append(ingr.descrever()).append("\n\n");
        }
```

```
        str.append("\nModo de preparo:\n\n");
        try {
            for (Tarefa tarf : _tarefas)
            {
                //Verifica se todos os ingredientes estão presentes na receita
                //e retorna um arranjo com eles
                Ingrediente[] ingrs = getIngredientesTarefa(tarf);
                str.append("\t").append(tarf.descrever(ingrs)).append("\n");
            }
        } catch (Exception ex)
        {
            return ex.getMessage();
        }

        str.append("\nResultado:\n\n");

        str.append("\t").append(_tarefas.get(_tarefas.size() - 1).resultado().descrever());
        return str.toString();
    }

    private Ingrediente[] getIngredientesTarefa(Tarefa tarefa) throws Exception
    {
        List<Ingrediente> ingrs = new ArrayList<Ingrediente>();
        try {
            for (String nomeIngr : tarefa.getNomesIngredientes())
                ingrs.add(getIngrediente(nomeIngr));
        } catch (Exception ex)
        {
           if (ex.getMessage().equals(MSG_ERRO_INGR_NAO_ENCONTRADO))
               throw new Exception(MSG_ERRO_RECEITA_INGR_INVALIDO);
           else
               throw ex;
        }
        return ingrs.toArray(new Ingrediente[ingrs.size()]);
    }

    public Ingrediente getIngrediente(String nomeIngrediente) throws Exception
    {
        Ingrediente ingrRetorno = null;

        for (Ingrediente ingr : _ingredientes)
        {
            if (ingr.nome().intern() == nomeIngrediente.intern())
            {
                ingrRetorno = ingr;
                break;
            }
        }
        if (ingrRetorno == null)
            throw new Exception(MSG_ERRO_INGR_NAO_ENCONTRADO);

        return ingrRetorno;
    }
}
```

```
package receitas.DomainObjects;

/**
 *
 * @author Charles.Fortes
 */
public abstract class Tarefa {
    private String [] nomesIngredientes;
    private Ingrediente[] ingredientes;
```

```
    public Tarefa(String[] nomesIngrs){ nomesIngredientes = nomesIngrs; }

    /**
     * descrição da tarefa a partir de um conjunto de ingredientes
     * @param ingrs: Ingredientes da tarefa
     * @return descrição da tarefa
     */
    public abstract String descrever(Ingrediente[] ingrs);

    /**
     * Retorna o ingrediente resultante da tarefa
     * @return ingrediente resultante da tarefa
     */
    public abstract Ingrediente resultado();

    public String[] getNomesIngredientes() { return nomesIngredientes; }
    public Ingrediente[] getIngredientes() { return ingredientes; }


    /**
     * Executa uma tarefa
     * @param ingrs: Ingredientes para a execução de uma tarefa
     */
    public void executar(Ingrediente[] ingrs) {
        ingredientes = new Ingrediente[ingrs.length];
        for (int i = 0; i < ingrs.length; i++)
            ingredientes[i] = new Ingrediente(ingrs[i].nome(),
                    ingrs[i].quantidade());
    }

    protected String descrever(String prefixoAcao, Ingrediente[] ingrs) {
        StringBuilder str = new StringBuilder();
        str.append(prefixoAcao + " ");

        for (int i = 0; i < ingrs.length; i++)
        {
            str.append(ingrs[i].nome());
            if (i < ingrs.length -1)
                str.append(" e ");
        }

        return str.toString();
    }

}
```

```
/*
 * Testes: Molho a campanha (Vinagrete)
 */


package receitas.LivroReceitas;

import java.util.*;
import receitas.DomainObjects.*;
/**
 *
 * @author Charles.Fortes
 */
public class AppTeste1 {

    /**
```

```
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        //ingredientes
        Ingrediente tomate = new Ingrediente("Tomate", 2);
        Ingrediente cebola = new Ingrediente("Cebola", 1);
        Ingrediente pimentao = new Ingrediente("Pimentão", 0.5f);
        Ingrediente vinagreBranco = new Ingrediente("Vinagre branco", 40);
        Ingrediente azeite = new Ingrediente("Azeite", 20);
        //------

        Tarefa t1 = new Picar(new String[]{tomate.nome()});
        t1.executar(new Ingrediente[]{tomate});
        Ingrediente r1 = t1.resultado();

        Tarefa t2 = new Descascar(new String[]{cebola.nome()});
        t2.executar(new Ingrediente[]{cebola});
        Ingrediente r2 = t2.resultado();

        Tarefa t3 = new Picar(new String[]{r2.nome()});
        t3.executar(new Ingrediente[]{r2});
        Ingrediente r3 = t3.resultado();

        Tarefa t4 = new Picar(new String[]{pimentao.nome()});
        t4.executar(new Ingrediente[]{pimentao});
        Ingrediente r4 = t4.resultado();

        Tarefa t5 = new Misturar(new String[]{r3.nome(), r4.nome(),
                r1.nome()});
        t5.executar(new Ingrediente[]{r3, r4, r1});
        Ingrediente r5 = t5.resultado();

        Tarefa t6 = new Misturar(new String[]{vinagreBranco.nome(), r5.nome()});
        t6.executar(new Ingrediente[]{vinagreBranco, r5});
        Ingrediente r6 = t6.resultado();

        Tarefa t7 = new Misturar(new String[]{azeite.nome(), r6.nome()});
        t7.executar(new Ingrediente[]{azeite, r6});
        Ingrediente r7 = t7.resultado();

        List<Ingrediente> ingrs = new ArrayList<Ingrediente>();
        ingrs.addAll(Arrays.asList(new Ingrediente[]
                        { tomate, cebola, pimentao, vinagreBranco, azeite,
                          r1, r2, r3, r4, r5, r6, r7 } ));

        List<Tarefa> tarefas = new ArrayList<Tarefa>();
        tarefas.addAll(Arrays.asList( new Tarefa[] { t1, t2, t3, t4, t5, t6, t7 } ));

        Receita receita = new Receita("Molho a campanha (Vinagrete)", ingrs, tarefas);

        System.out.println(receita.descrever());
    }

}
```

```
/*
 * Testes: Mousse de Maracujá
 */


package receitas.LivroReceitas;

import java.util.*;
import receitas.DomainObjects.*;
```

```
/**
 *
 * @author Charles.Fortes
 */
public class AppTeste2 {
    public static void main(String[] args) {
            //ingredientes
        Ingrediente cremeDeLeite = new Ingrediente("Creme de Leite", 200);
        Ingrediente leiteCondencado = new Ingrediente("Leite Condensado", 300);
        Ingrediente sucoMaracuja = new Ingrediente("Suco de Maracujá", 1);
        //------

        Tarefa t1 = new Bater(new String[]{cremeDeLeite.nome(), leiteCondencado.nome()});
        t1.executar(new Ingrediente[]{cremeDeLeite, leiteCondencado});
        Ingrediente r1 = t1.resultado();

        Tarefa t2 = new Misturar(new String[]{sucoMaracuja.nome(), r1.nome()});
        t2.executar(new Ingrediente[]{r1, sucoMaracuja});
        Ingrediente r2 = t2.resultado();

        Tarefa t3 = new Bater(new String[]{r2.nome()});
        t3.executar(new Ingrediente[]{r2});
        Ingrediente r3 = t3.resultado();

        Tarefa t4 = new Congelar(new String[]{r3.nome()});
        t4.executar(new Ingrediente[]{r3});
        Ingrediente r4 = t4.resultado();


        List<Ingrediente> ingrs = new ArrayList<Ingrediente>();
        ingrs.addAll(Arrays.asList(new Ingrediente[]
                        { cremeDeLeite, leiteCondencado, sucoMaracuja,
                          r1, r2, r3, r4 } ));

        List<Tarefa> tarefas = new ArrayList<Tarefa>();
        tarefas.addAll(Arrays.asList( new Tarefa[] { t1, t2, t3, t4 } ));

        Receita receita = new Receita("Mousse de Maracujá", ingrs, tarefas);

        System.out.println(receita.descrever());
    }
}
```