

DCC – UFMG – Especialização em Engenharia de Software

**Trabalho Prático 3 : Tratamento de Exceções e Interfaces Gráficas – Livro de Receitas
Ambientes de Programação – Prof. Roberto S. Bigonha**

Charles Wellington de Oliveira Fortes

Descrição

Implementação de tratamentos de exceções e interface gráfica para o Livro de Receitas gerado pelo TP2.

Serão listados abaixo, apenas as modificações realizadas nas classes do sistema para adaptar seu comportamento conforme solicitado no enunciado do trabalho.

As interfaces gráficas foram geradas manualmente (sem o auxílio de qualquer ferramenta) utilizando a biblioteca Javax.Swing.

As exceções foram propagadas para a interface, sendo exibidas para o usuário na forma de caixas de diálogo, usando o JOptionPane.

Detalhes da implementação

Arquitetura

O sistema está separado em dois pacotes, sendo o primeiro receitas.DomainObjects, que armazena todas as classes que representam os objetos do domínio da aplicação e sua lógica de funcionamento. O segundo pacote denominado receitas.LivroReceitas contém a interface gráfica para interação com o usuário e classes para testes dos resultados da camada de domínio.

Janelas

As janelas do sistema foram criadas herdando da classe JFrame, estando as telas de cadastro do sistema programadas para serem exibidas no modo “Always On Top”, para dar o efeito de uma janela “Modal”.

LayoutManagers das Janelas

Para a melhor adequação do layout de exibição das janelas, as mesmas foram organizadas das seguintes formas:

LivroDeReceitas (principal): Layout do tipo BorderLayout → Dentro do BorderLayout, foram adicionados JPanels utilizando layout do tipo Grid, possibilitando que fossem criados sub-componentes visuais que organizassem melhor as informações conforme sua função.

CadastroIngrediente: Layout do tipo GridLayout → Devido a simplicidade da tela, foi utilizado o esquema de Grid, fazendo com que os componentes da tela se organizassem de forma organizada e simétrica;

CadastroReceita: Layout do tipo BorderLayout → Utilizado o esquema de BorderLayout para melhor organizar a informação conforme sua função e importância. Em cada um dos “cantos” do layout, foram inseridos JPanels com layout do tipo GridLayout, FlowLayout ou BoxLayout, de forma que as informações pudessem se dispor da melhor forma possível;

Para os campos inseridos dinamicamente representando as tarefas a serem executadas, o painel teve seu layout definido como BoxLayout utilizando a orientação em Y_AXIS para melhor organização da informação.

Eventos

Os tratamentos de eventos foram feitos adicionando objetos do tipo “Listener” conforme cada situação, sendo estes implementados como Anonymous sempre que sua implementação foi simples e curta, nos demais casos, para melhor entendimento e clareza do fonte, foram criadas privadas classes aninhadas.

Para receber o evento das telas de cadastro, foram criadas duas Interfaces: CadReceitasModalListener e CadIngredienteModalListener, onde ambas implementam um método que será executada quando se clicar no botão Salvar das telas de cadastro.

Para isto, o form principal é passado como parâmento “parent” no construtor da tela de cadastro.

Comportamentos

- Para se cadastrar uma receita o sistema deve possuir pelo menos um ingrediente e ele(s) deve(m) estar selecionado(s);
- Ao entrar na tela de cadastro de receitas, todos os ingredientes selecionados estarão representados na interface de forma a receberem a quantidade necessária para cada um;
- Ao selecionar uma tarefa que necessita de mais de um ingrediente (misturar), a caixa de texto de nome do ingrediente é escondida e ao clicar no botão adicionar, dois InputsBoxes são exibidos solicitando o nome de cada ingrediente que compõe a mistura
- Quando o nome do ingrediente para a tarefa não está na lista de ingrediente, o sistema exibe uma caixa de diálogo informando ao usuário

Alterações

Classe Receita

- Sem alterações

Classe Ingrediente

- Incluído o atributo “Tipo” e constantes simbólicas para representar os tipos possíveis
- Incluído validações no construtor para “quantidade <= 0” e “tipo inválido”

Classe Tarefa

- Incluídos métodos estáticos para filtrar entre uma lista de ingredientes, quais pertencem a uma tarefa
- Alterações nos métodos “Executar” e “Descrever” para que validem se a tarefa é compatível com o tipo de ingrediente enviado
- Incluídos métodos abstratos “getTiposCompatíveis()” → [Retorna todos os tipos de ingredientes compatíveis com a tarefa] e “getMsgIncompatibilidade” → [Retorna a mensagem da tarefa informando a incompatibilidade com o tipo do ingrediente]
- Incluído o método “ValidaTipos”, que recebe uma lista que será preenchida com todos os erros encontrados e retorna um booleano informando se a tarefa foi validada com sucesso ou não

Interface IResultado

- Sem alterações

Classes de Tarefas:

- Implementações dos métodos abstratos criados em “Tarefa”

Classes de resultado:

- Incluídos os tipos em seus construtores

Inclusões

Classe IngredienteException

- Herda de “Exception” → Representa uma exceção levantada por um ingrediente

Classe TarefaException

- Herda de “Exception” → Representa uma exceção levantada por uma tarefa

Classe StringHelper

- Classe criada para conter métodos auxiliares para trabalhar com string
- **Métodos:** Join → Recebe uma lista ou arranjo de strings e uma sequência de caracteres que serão unidos em uma única string;

Classe VisualApp

- Classe inicial da aplicação visual para o livro de receitas

Classe LivroDeReceitas

- Classe que representa a tela principal do sistema

Classe CadastroReceita

- Classe que representa a tela de cadastro de receitas

Classe CadastroIngrediente

- Classe que representa a tela de cadastro de ingrediente

Classe CadReceitasModalListener

- Interface que representa uma classe que receberá o retorno da ação da janela de cadastro de receitas

Classe CadIngredienteModalListener

- Interface que representa uma classe que receberá o retorno da ação da janela de cadastro de ingredientes

Saídas

1) Teste 1 : Exceções

```
package receitas.LivroReceitas;

import receitas.DomainObjects.Ingrediente;
import receitas.DomainObjects.IngredienteException;

/**
 *
 * @author Charles.Fortes
 */
public class TestesExcessoos {
    public static void main(String[] args) {
        //Quantidade Negativa
        try {
            Ingrediente tomate = new Ingrediente("Tomate", -1f, Ingrediente.TIPO_SOLIDO);
        }
        catch (IngredienteException ex)
        {
            System.out.println("Teste executado com sucesso: Erro obtido: " + ex.getMessage());
        }
    }
}
```

Saída:

run:

Teste executado com sucesso: Erro obtido: A quantidade informada para o ingrediente não é válida:

A quantidade deve ser maior do que zero.

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```
package receitas.LivroReceitas;

import receitas.DomainObjects.Bater;
import receitas.DomainObjects.Ingrediente;
import receitas.DomainObjects.IngredienteException;
import receitas.DomainObjects.Picar;
import receitas.DomainObjects.Tarefa;
import receitas.DomainObjects.TarefaException;

/**
 *
 * @author Charles.Fortes
 */
public class TestesExcessoos {
    public static void main(String[] args) {

        //Tipo incompatível
        try {
            Ingrediente leite = new Ingrediente("leite", 2f, Ingrediente.TIPO_LIQUIDO);
            Tarefa picar = new Picar(new String[]{"leite"});
            picar.executar(new Ingrediente[]{leite});
            System.out.println("resultado da tarefa: " + picar.resultado().descrever());
        }
    }
}
```

```

        System.out.println("descrição: " + picar.descrever(new Ingrediente[]{leite}));
    }
    catch (IngredienteException ex)
    {
        System.out.println("Erro inesperado obtido: " + ex.getMessage());
    }
    catch (TarefaException ex)
    {
        System.out.println("Teste executado com sucesso: Erro obtido: " + ex.getMessage());
    }
}
}
}

```

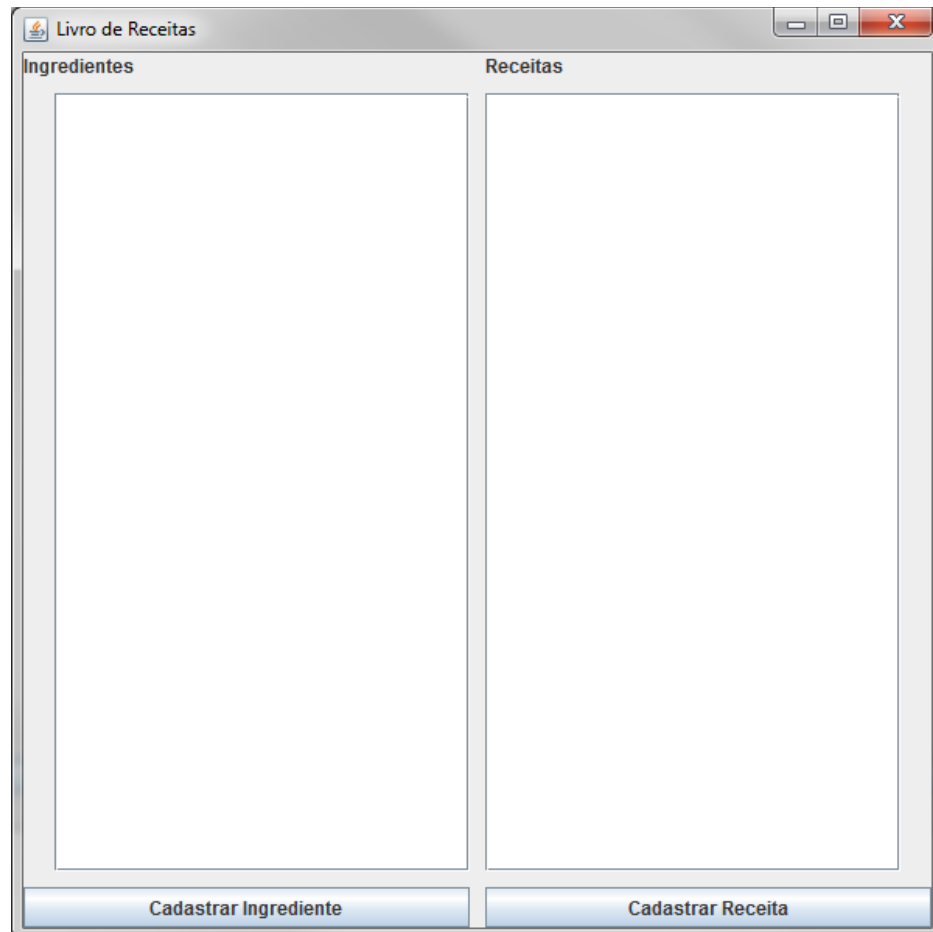
Saída:


run:

Teste executado com sucesso: Erro obtido: Ingredientes liquidos não podem ser picados

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

2) Teste 2 : Interface Gráfica



 Cadastro de ingrediente


Nome do ingrediente

Tomate

Tipo do ingrediente

Sólido

Salvar Cancelar

 Cadastro de ingrediente


Nome do ingrediente

Alface

Tipo do ingrediente

Sólido

Salvar Cancelar

 Cadastro de ingrediente


Nome do ingrediente

Azeite

Tipo do ingrediente

Líquido

Salvar Cancelar

 Livro de Receitas

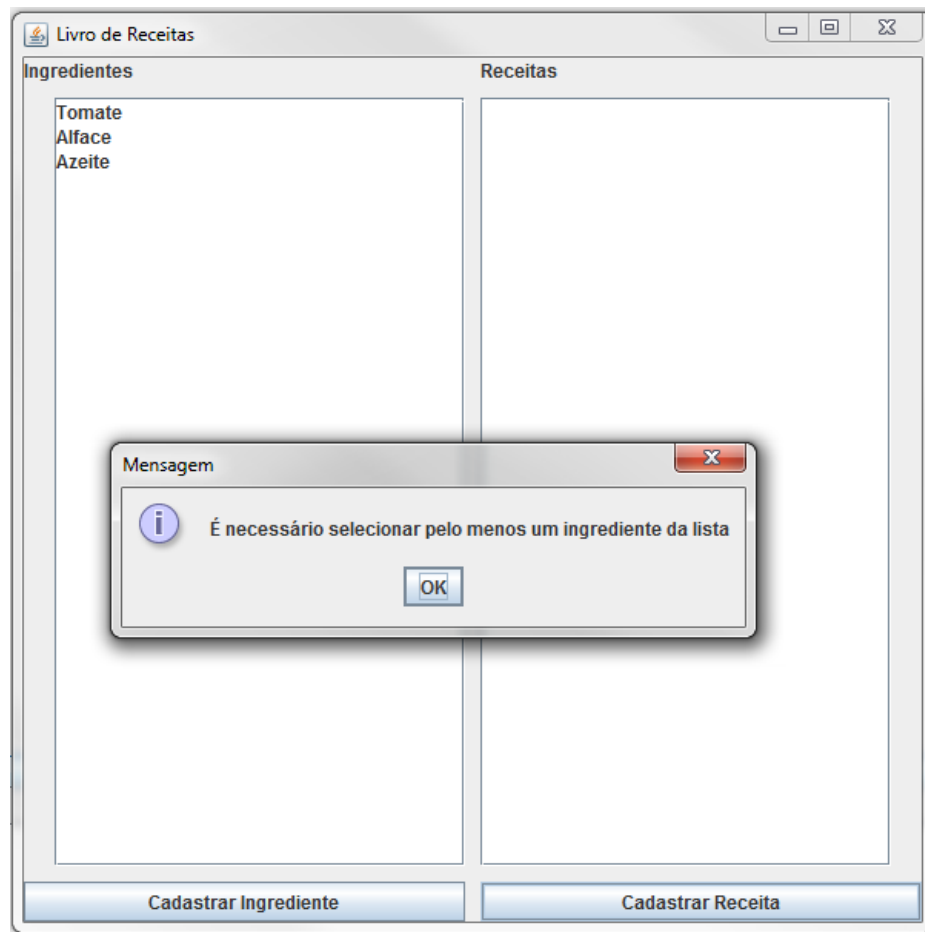
Ingredientes

Receitas

Tomate
Alface
Azeite

Cadastrar Ingrediente

Cadastrar Receita



Livro de Receitas

Ingredientes

Tomate
Alface
Azeite

Receitas

Cadastro de receitas

Nome da Receita

Ingredientes

Tomate
Quantidade 1.0

Alface
Quantidade 1.0

Azeite
Quantidade 1.0

Tarefa Bater

Ingredinete

+

Cadastrar Ingrediente

Salvar Cancelar

Cadastro de receitas

Nome da Receita

Salada Simples (bem simples)

Ingredientes

Tomate
Quantidade 2

Alface
Quantidade 4

Azeite
Quantidade 40

Tarefa Picar

Ingredinete

Tomate

+

Salvar Cancelar

Cadastro de receitas

Nome da Receita

Salada Simples (bem simples)

Ingredientes

Tomate	Alface
Quantidade 2	Quantidade 4
Azeite	
Quantidade 40	

Tarefa **Misturar**

+

Picar Tomate(res: Tomate picado)

Picar Alface(res: Alface picado)

Salvar Cancelar

Entrada

? Informe o primeiro ingrediente

Tomate picado

OK Cancelar

Entrada

? Informe o segundo ingrediente

Alface picado

OK Cancelar

Cadastro de receitas

Nome da Receita

Salada Simples (bem simples)

Ingredientes

Tomate	Alface
Quantidade 2	Quantidade 4
Azeite	
Quantidade 40	

Tarefa: Misturar

+

Picar Tomate(res: Tomate picado)
Picar Alface(res: Alface picado)
Misturar Tomate picado e Alface picado(res: Mistu...)

Salvar Cancelar

Entrada

Informe o primeiro ingrediente

Mistura de (Tomate picado e Alface pica

OK Cancelar

Entrada

Informe o segundo ingrediente

Azeite

OK Cancelar

Cadastro de receitas

Nome da Receita

Salada Simples (bem simples)

Ingredientes

Tomate	Alface
Quantidade 2	Quantidade 4
Azeite	
Quantidade 40	

Tarefa

Misturar

+

Picar Tomate(res: Tomate picado)

Picar Alface(res: Alface picado)

Misturar Tomate picado e Alface picado(res: Mistu...

Misturar Mistura de (Tomate picado e Alface picad...

Salvar Cancelar

Livro de Receitas

Ingredientes

Tomate

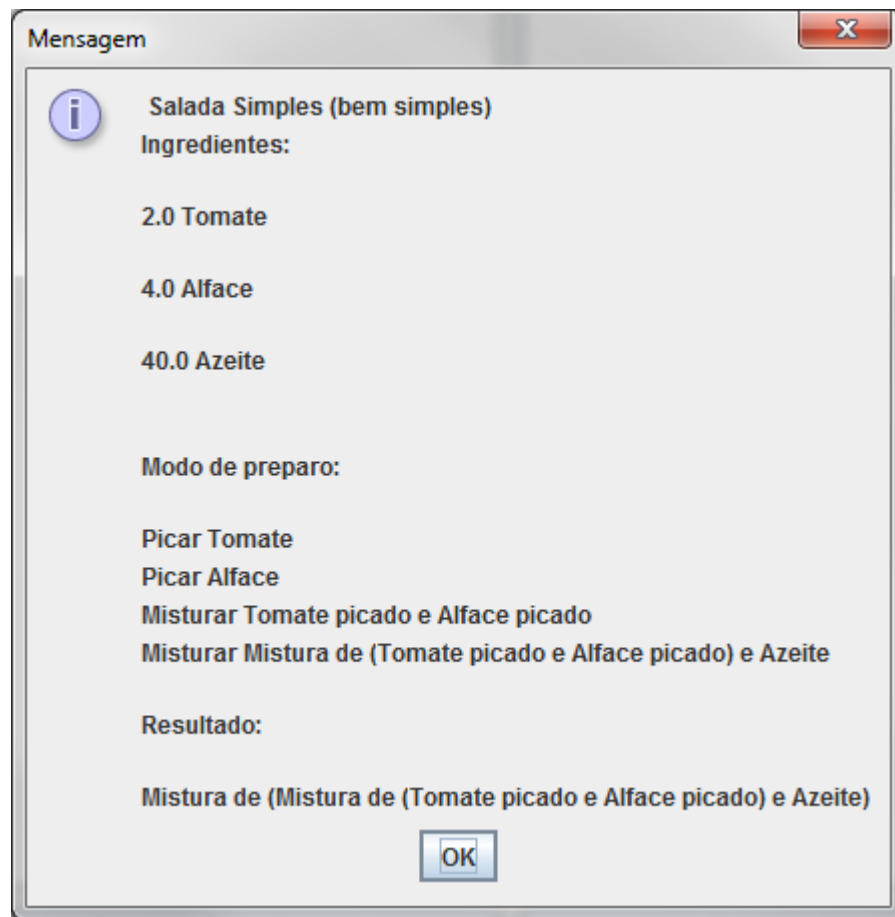
Alface

Azeite

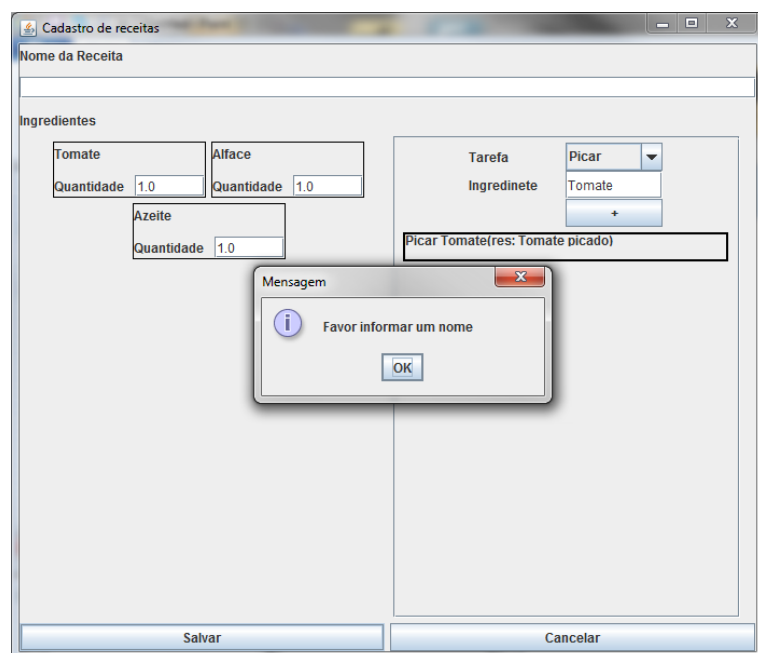
Receitas

Salada Simples (bem simples)

Cadastrar Ingrediente Cadastrar Receita



Erros



Cadastro de receitas

Nome da Receita

Teste

Ingredientes

Tomate	Alface	Tarefa	Picar
Quantidade -1	Quantidade 1.0	Ingredinete	Tomate
Azeite			
Quantidade			

Mensagem

A quantidade informada para o ingrediente não é válida:
A quantidade deve ser maior do que zero.

OK

Salvar Cancelar

Cadastro de receitas

Nome da Receita

Teste

Ingredientes

Azeite

Quantidade1.0

Tarefa


Ingredinete

Picar

Azeite

+

Mensagem



Ingredientes liquidos não podem ser picados

OK

Salvar

Cancelar

Implementação:

```
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");
        try {
            str.append("\t").append(_tarefas.get(_tarefas.size() - 1).resultado().descrever());
        } catch (IngredienteException ex) {
            str.append(ex.getMessage());
        }
        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 String nome() { return _nome; }
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;

/**
 * Representa um ingrediente
 * @author Charles.Fortes
 */
public class Ingrediente {
    // -- Constantes simbólicas para tipos de ingredientes
    public static final int TIPO_LIQUIDO = 0;
    public static final int TIPO_SOLIDO = 1;
    public static final int TIPO_PO = 2;
    public static final int TIPO_MISTO = 3;
    // -----

    private static final String ERRO_QUANTIDADE_NEGATIVA =
        "A quantidade informada para o ingrediente não é válida:\n"
        + "\tA quantidade deve ser maior do que zero.";
    private static final String ERRO_TIPO_INVALIDO = "O tipo do ingrediente informado"
        + " não é válido: \n\tO tipo do ingrediente deve ser Liquido, Solido ou Pó.";

    private float _quantidade;
    private String _nome;
    private int _tipo;

    public Ingrediente(String nome, float quantidade, int tipo)
        throws IngredienteException
    {
        if (quantidade <= 0)
            throw new IngredienteException(ERRO_QUANTIDADE_NEGATIVA);

        if (tipo < TIPO_LIQUIDO || tipo > TIPO_MISTO)
            throw new IngredienteException(ERRO_TIPO_INVALIDO);

        _quantidade = quantidade; _nome = nome; _tipo = tipo;
    }
}

```

```

    }

    public int tipo() { return _tipo; };
    public float quantidade(){ return _quantidade; }
    public String nome() { return _nome; }
    protected void setNome(String nome) { _nome = nome; }

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

```

```

package receitas.DomainObjects;

import java.util.*;

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

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

    /**
     * produz uma 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() throws IngredienteException;

    protected abstract String getMsgIncompatibilidade();
    protected abstract List<Integer> getTiposCompativeis();

    /**
     * Valida se há algum ingrediente inválido na lista de ingredientes passados a tarefa
     * @param ingrs: Ingredientes a serem validados
     * @param brokenRules: Regras inválidas entre as tarefas e os ingredientes
     * @return true se não há nenhum problema com os ingredientes, false caso um ou
     * mais ingredientes não possam ser executados pela tarefa
     */
    private boolean ValidaTipos(Ingrediente[] ingrs, List<String> brokenRules)
    {
        for (Ingrediente ingr: ingrs)
        {
            if (!getTiposCompativeis().contains(ingr.tipo()))
            {
                String msg = getMsgIncompatibilidade();
                if (!msg.contains("{0}"))
                    msg = msg + "\n\tIngredientes fo tipo: {0}";

                String descTipo;
                switch (ingr.tipo())
                {
                    case Ingrediente.TIPO_LIQUIDO:
                        descTipo = "liquidos";

```

```

        break;
    case Ingrediente.TIPO_SOLIDO:
        descTipo = "solidos";
        break;
    case Ingrediente.TIPO_PO:
        descTipo = "em pó";
        break;
    case Ingrediente.TIPO_MISTO:
        descTipo = "misturados";
        break;
    default:
        descTipo = "tipo não tratado";
        break;
    }
    brokenRules.add(String.format(msg, descTipo));
}
}
return brokenRules.isEmpty();
}

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) throws TarefaException {

    List<String> brokenRules = new ArrayList<String>();
    if (!ValidaTipos(ingrs, brokenRules))
        throw new TarefaException(StringHelper.join(brokenRules, "\n"));

    ingredientes = ingrs;
}

protected String descrever(String prefixoAcao, Ingrediente[] ingrs)
    throws TarefaException {

    List<String> brokenRules = new ArrayList<String>();
    if (!ValidaTipos(ingrs, brokenRules))
        throw new TarefaException(StringHelper.join(brokenRules, "\n"));

    StringBuilder str = new StringBuilder();
    str.append(prefixoAcao).append(" ");

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

    return str.toString();
}

public static Ingrediente[] getIngredientesTarefa(Tarefa tarefa,
    List<Ingrediente> lstIngrs) throws TarefaException
{
    List<Ingrediente> ingrs = new ArrayList<Ingrediente>();
    try {
        for (String nomeIngr : tarefa.getNomesIngredientes())
            ingrs.add(getIngrediente(nomeIngr, lstIngrs));
    } catch (TarefaException ex)
    {

```

```

        if (ex.getMessage().equals(MSG_ERRO_INGR_NAO_ENCONTRADO))
            throw new TarefaException(MSG_ERRO_RECEITA_INGR_INVALIDO);
        else
            throw ex;
    }
    return ingr.toArray(new Ingrediente[ingrs.size()]);
}

private static Ingrediente getIngrediente(String nomeIngrediente,
    List<Ingrediente> lstIngrs) throws TarefaException
{
    Ingrediente ingrRetorno = null;

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

    return ingrRetorno;
}
}

```

```
package receitas.DomainObjects;
```

```

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

```

```

}
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[] ingr) {
        String ret = "";
        try {
            ret = super.descrever("Picar", ingr);
        } catch (TarefaException ex) {
            ret = ex.getMessage();
        }
        return ret;
    }

    @Override
    public Ingrediente resultado() throws IngredienteException {
        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]);

}

@Override
protected String getMsgIncompatibilidade() {
    return "Ingredientes {0} não podem ser picados";
}

@Override
protected List<Integer> getTiposCompativeis() {
    return new ArrayList<Integer>(Arrays.asList(new Integer[]{
        Ingrediente.TIPO_SOLIDO}
    ));
}
}

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) {
        String ret = "";
        try {
            ret = super.descrever("Misturar", ingrs);
        } catch (TarefaException ex) {
            ret = ex.getMessage();
        }
        return ret;
    }

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

    @Override
    protected String getMsgIncompatibilidade() {
        return "Ingredientes {0} não podem ser misturados";
    }

    @Override
    protected List<Integer> getTiposCompativeis() {
        return new ArrayList<Integer>(Arrays.asList(new Integer[]{
            Ingrediente.TIPO_LIQUIDO,
            Ingrediente.TIPO_PO,
            Ingrediente.TIPO_SOLIDO,
            Ingrediente.TIPO_MISTO}
        ));
    }
}

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) {
        String ret = "";
        try {
            ret = super.descrever("Fatiar", ingrs);
        } catch (TarefaException ex) {
            ret = ex.getMessage();
        }
        return ret;
    }

    @Override
    public Ingrediente resultado() throws IngredienteException {
        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]);
    }

    @Override
    protected String getMsgIncompatibilidade() {
        return "Ingredientes {0} não podem ser fatiados";
    }

    @Override
    protected List<Integer> getTiposCompatíveis() {
        return new ArrayList<Integer>(Arrays.asList(new Integer[]{
            Ingrediente.TIPO_SOLIDO}
        ));
    }
}

```

```

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) {
        String ret = "";
        try {

```

```

        ret = super.descrever("Descascar", ingr);
    } catch (TarefaException ex) {
        ret = ex.getMessage();
    }
    return ret;
}

@Override
public Ingrediente resultado() throws IngredienteException {
    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]);
}

@Override
protected String getMsgIncompatibilidade() {
    return "Ingredientes {0} não podem ser descascados";
}

@Override
protected List<Integer> getTiposCompativeis() {
    return new ArrayList<Integer>(Arrays.asList(new Integer[]{
        Ingrediente.TIPO_SOLIDO}
    ));
}
}

```

```

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[] ingr) {
        String ret = "";
        try {
            ret = super.descrever("Congelar", ingr);
        } catch (TarefaException ex) {
            ret = ex.getMessage();
        }
        return ret;
    }

    @Override
    public Ingrediente resultado()throws IngredienteException {
        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]);
    }

    @Override
    protected String getMsgIncompatibilidade() {
        return "Ingredientes {0} não podem ser congelados";
    }

    @Override
    protected List<Integer> getTiposCompativeis() {
        return new ArrayList<Integer>(Arrays.asList(new Integer[]{
            Ingrediente.TIPO_LIQUIDO,
            Ingrediente.TIPO_SOLIDO,
            Ingrediente.TIPO_MISTO}
        ));
    }
}
}

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) {
        String ret = "";
        try {
            ret = super.descrever("Bater", ingrs);
        } catch (TarefaException ex) {
            ret = ex.getMessage();
        }
        return ret;
    }

    @Override
    public Ingrediente resultado() throws IngredienteException {
        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]);
    }

    @Override
    protected String getMsgIncompatibilidade() {
        return "Ingredientes {0} não podem ser batidos";
    }

    @Override
    protected List<Integer> getTiposCompativeis() {
        return new ArrayList<Integer>(Arrays.asList(new Integer[]{
            Ingrediente.TIPO_LIQUIDO,
            Ingrediente.TIPO_PO,

```

```

        Ingrediente.TIPO_SOLIDO,
        Ingrediente.TIPO_MISTO}
    });
}
}

```

```

package receitas.DomainObjects;

```

```

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

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

```

```

package receitas.DomainObjects;

```

```

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

    public Mistura(Ingrediente[] ingrs) throws IngredienteException
    {
        super("Mistura_" + ingrs.length, ingrs.length, TIPO_MISTO);
        _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;

```

```

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

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

```

```

package receitas.DomainObjects;

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

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

```

```

package receitas.DomainObjects;

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

    Congelado(Ingrediente ingr) throws IngredienteException {
        super(ingr.nome() + " congelado", ingr.quantidade(), TIPO_SOLIDO);
        ingrOriginal = ingr;
    }
}

```

```

package receitas.DomainObjects;

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

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

```

```

package receitas.DomainObjects;

/**
 * @author Charles.Fortes
 */
public class IngredienteException extends Exception {
    public IngredienteException(String mensagem) { super(mensagem);}
}

```

```

package receitas.DomainObjects;

import java.util.*;

/**
 * @author Charles.Fortes
 */
public class TarefaException extends Exception{
    public TarefaException(String mensagem) { super(mensagem); }
}

```

```

package receitas.DomainObjects;

import java.util.List;

/**

```

```

    * @author Charles.Fortes
    */
    public class StringHelper {
        public static String join(List<String> lst, CharSequence s)
        {
            StringBuilder msg = new StringBuilder();
            for (String item : lst)
                msg.append(item).append(s);

            return msg.toString();
        }

        public static String join(String[] lst, CharSequence s) {
            StringBuilder msg = new StringBuilder();
            for (String item : lst)
                msg.append(item).append(s);

            return msg.toString();
        }
    }
}

package receitas.LivroReceitas;

import javax.swing.*;

/**
 * @author Charles.Fortes
 */
public class VisualApp {
    public static void main(String[] args) {
        JFrame app = new LivroDeReceitas();
        app.setVisible(true);
    }
}

package receitas.LivroReceitas;

import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import javax.swing.*;
import receitas.DomainObjects.*;

/**
 *
 * @author Charles.Fortes
 */
public class LivroDeReceitas extends JFrame implements CadIngredienteModalListener, CadReceitasModalListener {
    private JList lstIngredientes;
    private JList lstReceitas;
    private JLabel lblIngrs;
    private JLabel lblReceitas;
    private JButton btnCadIngrs;
    private JButton btnCadRec;
    private JPanel topPane;
    private JPanel bottomPane;
    private JPanel centerPane;
    private JPanel leftPane; //apenas para dar um espaço na lateral
    private JPanel rightPane; //apenas para dar um espaço na lateral
    private java.util.List<Ingrediente> ingredientes = new ArrayList<Ingrediente>();
    private java.util.List<Receita> receitas = new ArrayList<Receita>();

    public LivroDeReceitas()
    {
        super("Livro de Receitas");
        initialize();
    }
}

```

```

private void initialize()
{
    LayoutManager layout = new BorderLayout(10, 10);
    this.getContentPane().setLayout(layout);

    lstIngredientes = new JList();
    lstReceitas = new JList();
    lblIngrs = new JLabel("Ingredientes");
    lblReceitas = new JLabel("Receitas");
    btnCadIngrs = new JButton("Cadastrar Ingrediente");
    btnCadRec = new JButton("Cadastrar Receita");
    topPane = new JPanel(new GridLayout(1, 2, 10, 10));
    bottomPane = new JPanel(new GridLayout(1, 2, 10, 10));
    centerPane = new JPanel(new GridLayout(1, 2, 10, 10));
    rightPane = new JPanel();
    leftPane = new JPanel();

    //lstIngredientes
    lstIngredientes.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
    lstIngredientes.setLayoutOrientation(JList.VERTICAL);
    lstIngredientes.setCellRenderer(new ingrCellRender());
    //---

    //lstReceitas
    lstReceitas.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    lstReceitas.setLayoutOrientation(JList.VERTICAL);
    lstReceitas.setCellRenderer(new recCellRender());
    lstReceitas.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e)
        {
            if (e.getClickCount() == 2)
                JOptionPane.showMessageDialog(null, ((Receita)lstReceitas.getSelectedValue()).descrever());
        }
    });
    // ---

    //btnCadIngrs
    btnCadIngrs.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnCadIngrs_action(e);
        }
    });
    // ---

    //btnCadRec
    btnCadRec.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnCadRec_action(e);
        }
    });
    // ---

    topPane.add(lblIngrs);
    topPane.add(lblReceitas);

    centerPane.add(new JScrollPane(lstIngredientes));
    centerPane.add(new JScrollPane(lstReceitas));

    bottomPane.add(btnCadIngrs);
    bottomPane.add(btnCadRec);

    this.getContentPane().add(topPane, BorderLayout.NORTH);
    this.getContentPane().add(centerPane, BorderLayout.CENTER);
    this.getContentPane().add(bottomPane, BorderLayout.SOUTH);

    this.getContentPane().add(rightPane, BorderLayout.EAST);
    this.getContentPane().add(leftPane, BorderLayout.WEST);
}

```

```

        this.setSize(600, 600);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private void btnCadIngrs_action(ActionEvent e)
    {
        JFrame cadIng = new CadastroIngrediente(this);
        cadIng.setVisible(true);
    }

    private void btnCadRec_action(ActionEvent e)
    {
        if (lstIngredientes.getSelectedValues().length <= 0)
        {
            JOptionPane.showMessageDialog(null, "É necessário selecionar pelo menos "
                + "um ingrediente da lista");
            return;
        }

        java.util.List<Ingrediente> ingr = new ArrayList<Ingrediente>();
        for (Object o : lstIngredientes.getSelectedValues())
            ingr.add((Ingrediente)o);

        JFrame cadRec = new CadastroReceita(this, ingr);
        cadRec.setVisible(true);
    }

    public void modalResult(Ingrediente result)
    {
        ingredientes.add(result);
        lstIngredientes.setListData(ingredientes.toArray());
        repaint();
    }

    public void modalResult(Receita result)
    {
        receitas.add(result);
        lstReceitas.setListData(receitas.toArray());
        repaint();
    }

    private class ingrCellRender extends JLabel implements ListCellRenderer
    {
        ingrCellRender()
        {
            setOpaque(true);
        }

        public Component getListCellRendererComponent(JList list, Object value, int index, boolean isSelected,
            boolean cellHasFocus) {
            Ingrediente _ingr = (Ingrediente)value;
            setText(_ingr.nome());
            if (isSelected) {
                setBackground(list.getSelectionBackground());
                setForeground(list.getSelectionForeground());
            } else {
                setBackground(list.getBackground());
                setForeground(list.getForeground());
            }

            return this;
        }
    }

    private class recCellRender extends JLabel implements ListCellRenderer

```

```

    {
        recCellRender() {
            setOpaque(true);
        }

        public Component getListCellRendererComponent(JList list, Object value, int index, boolean isSelected,
boolean cellHasFocus) {
            Receita _rec = (Receita)value;
            setText(_rec.nome());
            if (isSelected) {
                setBackground(list.getSelectionBackground());
                setForeground(list.getSelectionForeground());
            } else
            {
                setBackground(list.getBackground());
                setForeground(list.getForeground());
            }
            return this;
        }
    }
}

```

```

package receitas.LivroReceitas;

```

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import receitas.DomainObjects.*;

```

```

/**
 * @author Charles.Fortes
 */
public class CadastroIngrediente extends JFrame {
    private JLabel lblTipo;
    private JLabel lblNome;
    private JTextField txtNome;
    private JComboBox cboTipo;
    private JButton btnSalvar;
    private JButton btnCancelar;
    private JPanel buttonsPane;
    private CadIngredienteModalListener parent;

    public CadastroIngrediente(CadIngredienteModalListener parent) {
        super("Cadastro de ingrediente");
        this.parent = parent;
        intialize();
    }

    private void intialize() {
        this.getContentPane().setLayout(new GridLayout(5, 1, 5, 5));

        lblNome = new JLabel("Nome do ingrediente");
        lblTipo = new JLabel("Tipo do ingrediente");
        txtNome = new JTextField();
        cboTipo = new JComboBox(new String[]{"Líquido", "Sólido", "Em Pó", "Misto"});
        btnSalvar = new JButton("Salvar");
        btnCancelar = new JButton("Cancelar");
        buttonsPane = new JPanel(new GridLayout(1, 2, 5, 5));

        // btnSalvar
        btnSalvar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                btnSalvar_action(e);
            }
        });
        // ---
    }

```

```

        // btnCancelar
        btnCancelar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                btnCancelar_action(e);
            }
        });
        // ---
        buttonsPane.add(btnSalvar);
        buttonsPane.add(btnCancelar);

        this.getContentPane().add(lblNome);
        this.getContentPane().add(txtNome);
        this.getContentPane().add(lblTipo);
        this.getContentPane().add(cboTipo);
        this.getContentPane().add(buttonsPane);

        this.setSize(320, 240);
        this.setAlwaysOnTop(true);
    }

    private void btnSalvar_action(ActionEvent e)
    {
        try{
            parent.modalResult(new Ingrediente(txtNome.getText(), 1, cboTipo.getSelectedIndex()));
        } catch (IngredienteException ex)
        {
            JOptionPane.showMessageDialog(null, "Falha ao criar novo ingrediente\n\n" + ex.getMessage());
        } finally
        {
            btnCancelar.doClick();
        }
    }

    private void btnCancelar_action(ActionEvent e)
    {
        this.setVisible(false);
        this.dispose();
    }
}

```

```

package receitas.LivroReceitas;

```

```

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.LineBorder;
import receitas.DomainObjects.*;

/**
 * @author Charles.Fortes
 */
public class CadastroReceita extends JFrame {
    private JLabel lblIngrs;
    private JLabel lblNome;
    private JTextField txtNome;
    private JButton btnSalvar;
    private JButton btnCancelar;
    private JPanel buttonsPane;
    private CadReceitasModalListener parent;
    private JPanel topPane;
    private JPanel bottomPane;
    private JPanel centerPane;
    private JPanel leftPane; //apenas para dar um espaço na lateral
    private JPanel rightPane; //apenas para dar um espaço na lateral
    private java.util.List<Ingrediente> ingredientes = new ArrayList<Ingrediente>();
    private java.util.List<Ingrediente> ingrSalvos = new ArrayList<Ingrediente>();
    private java.util.List<Tarefa> tarefas = new ArrayList<Tarefa>();
}

```



```

private JPanel ingredientesPane;
private JPanel cTarefasPane;
private JPanel tarefasPane;
private JFrame self;

public CadastroReceita(CadReceitasModalListener parent, java.util.List<Ingrediente> ingr) {
    super("Cadastro de receitas");
    this.parent = parent;
    this.ingredientes = ingr;
    intialize();
}

private void intialize()
{
    this.getContentPane().setLayout(new BorderLayout(5, 5));

    lblNome = new JLabel("Nome da Receita");
    txtNome = new JTextField();
    btnSalvar = new JButton("Salvar");
    btnCancelar = new JButton("Cancelar");
    buttonsPane = new JPanel(new GridLayout(1, 2, 5, 5));
    topPane = new JPanel(new GridLayout(3, 1, 10, 10));
    bottomPane = new JPanel(new GridLayout(1, 1, 10, 10));
    centerPane = new JPanel(new GridLayout(1, 2, 10, 10));
    rightPane = new JPanel();
    leftPane = new JPanel();
    lblIngrs = new JLabel("Ingredientes");
    ingredientesPane = new JPanel(new FlowLayout());
    cTarefasPane = new JPanel(new FlowLayout());
    tarefasPane = new JPanel();
    BoxLayout b = new BoxLayout(tarefasPane, BoxLayout.Y_AXIS);
    tarefasPane.setLayout(b);

    // btnSalvar
    btnSalvar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnSalvar_action(e);
        }
    });
    // ---

    // btnCancelar
    btnCancelar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnCancelar_action(e);
        }
    });
    // ---
    buttonsPane.add(btnSalvar);
    buttonsPane.add(btnCancelar);

    // INgredientPane
    for (Ingrediente ingr : ingredientes)
        ingredientesPane.add(new ingrPane(ingr));
    // ---

    topPane.add(lblNome);
    topPane.add(txtNome);
    topPane.add(lblIngrs);

    // TarefasPane
    tarefasPane.setBorder(new LineBorder(Color.BLACK));
    cTarefasPane.setPreferredSize(new Dimension(200, 330));
    cTarefasPane.add(new cadTarefaPane());
    cTarefasPane.add(tarefasPane);
    // ---

```

```

centerPane.add(ingredientesPane);
centerPane.add(new JScrollPane(cTarefasPane));

bottomPane.add(buttonsPane);

this.getContentPane().add(topPane, BorderLayout.NORTH);
this.getContentPane().add(centerPane, BorderLayout.CENTER);
this.getContentPane().add(bottomPane, BorderLayout.SOUTH);

this.getContentPane().add(rightPane, BorderLayout.EAST);
this.getContentPane().add(leftPane, BorderLayout.WEST);

this.setSize(700, 600);
this.setAlwaysOnTop(true);

self = this;

for (Component c: ingredientesPane.getComponents())
{
    ingrPane i = (ingrPane)c;
    ingrsSalvos.add(i.save());
}

}

/**
 * Action listener do botão de salvar
 * @param e
 */
private void btnSalvar_action(ActionEvent e)
{
    if (txtNome.getText().isEmpty()){
        JOptionPane.showMessageDialog(self, "Favor informar um nome");
        return;
    }

    for (Component c: ingredientesPane.getComponents())
    {
        ingrPane i = (ingrPane)c;
        i.update();
    }

    for (Component c: tarefasPane.getComponents())
    {
        tarefaPane<Tarefa> t = (tarefaPane<Tarefa>)c;
        if (t.lblDesc != null){
            Tarefa tr = t.save();
            tarefas.add(tr);
        }
    }

    parent.modalResult(new Receita(txtNome.getText(), ingrsSalvos, tarefas));
    btnCancelar.doClick();
}

/**
 * Action listener do botão de cancelar
 * @param e
 */
private void btnCancelar_action(ActionEvent e)
{
    this.setVisible(false);
    this.dispose();
}

```

```

/**
 * representa o painel de ingredientes para o usuário digitar a quantidade
 */
private class ingrPane extends JPanel
{
    private Ingrediente _ingr;
    private JLabel lblIngr;
    private JTextField txtQtd;
    private JLabel lblQuantidade;

    private ingrPane(Ingrediente ingr) {
        _ingr = ingr;
        initialize();
    }

    private void initialize() {
        this.setLayout(new GridLayout(2,2,10,10));
        lblIngr = new JLabel(_ingr.nome());
        lblQuantidade = new JLabel("Quantidade");
        txtQtd = new JTextField(((Float)_ingr.quantidade()).toString());

        this.add(lblIngr);
        this.add(new JPanel());
        this.add(lblQuantidade);
        this.add(txtQtd);
        this.setBorder(new javax.swing.border.LineBorder(Color.BLACK));
    }

    public Ingrediente save()
    {
        Ingrediente ingr = null;
        try {
            ingr = new Ingrediente(_ingr.nome(), Float.parseFloat(txtQtd.getText()), _ingr.tipo());
            _ingr = ingr;
        } catch (IngredienteException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        } catch (Exception ex)
        {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        } finally {
            return ingr;
        }
    }

    public void update()
    {
        ingrSalvos.remove(_ingr);
        ingrSalvos.add(save());
    }
}

/**
 * representa o painel de tarefas incluídas no sistema
 * @param <T>: Tipo da tarefa
 */
private class tarefaPane<T extends Tarefa> extends JPanel
{
    private Tarefa _tarefa;
    private JLabel lblDesc;

    tarefaPane(T tarefa) {
        _tarefa = tarefa;
        initialize();
    }

    private void initialize() {

```

```

        this.setLayout(new GridLayout(2,2,0,0));
        try {
            _tarefa.executar(Tarefa.getIngredientesTarefa(_tarefa, ingrSalvos));
            ingrSalvos.add(_tarefa.resultado());
            lblDesc = new JLabel(
                _tarefa.descrever(Tarefa.getIngredientesTarefa(_tarefa, ingrSalvos))
                + "(res: " + _tarefa.resultado().nome() + ")");
        } catch (TarefaException ex) {
            JOptionPane.showMessageDialog(self, ex.getMessage());
            return;
        } catch (IngredienteException ex) {
            JOptionPane.showMessageDialog(self, ex.getMessage());
            return;
        }
    }

    this.add(lblDesc);
    this.setBorder(new javax.swing.border.LineBorder(Color.BLACK));
    this.setPreferredSize(new Dimension(300, 25));
}

public T save()
{
    return (T)_tarefa;
}

}

/**
 * representa o painel de cadastro de novas tarefas
 */
private class cadTarefaPane extends JPanel
{
    private JLabel lblNomeTarefa;
    private JComboBox cboTarefas;
    private JTextField txtNomeIngrediente;
    private JLabel lblNomeIngrediente;
    private JButton btnAddTarefa;

    private cadTarefaPane() {
        initialize();
    }

    private void initialize() {
        this.setLayout(new GridLayout(3,2, 0,0));
        lblNomeIngrediente = new JLabel("Ingrediente");
        lblNomeTarefa = new JLabel("Tarefa");
        txtNomeIngrediente = new JTextField();

        cboTarefas = new JComboBox(new String[]{"Bater", "Congelar",
            "Descascar", "Fatiar", "Misturar", "Picar"});
        cboTarefas.addItemListener(new ItemListenerImpl());
        ;
        btnAddTarefa = new JButton("+");

        this.add(lblNomeTarefa);
        this.add(cboTarefas);

        this.add(lblNomeIngrediente);
        this.add(txtNomeIngrediente);

        this.add(new JPanel());
        btnAddTarefa.addActionListener(new ActionListenerImpl());
        this.add(btnAddTarefa);
    }

    private class ItemListenerImpl implements ItemListener {

```

```

    public void itemStateChanged(ItemEvent e) {
        if (e.getItem().toString().equals("Misturar") &&
            e.getStateChange() == ItemEvent.SELECTED)
        {
            txtNomeIngrediente.setText("");
            txtNomeIngrediente.setVisible(false);
            lblNomeIngrediente.setVisible(false);
        }
        else{
            txtNomeIngrediente.setVisible(true);
            lblNomeIngrediente.setVisible(true);
        }
    }
}

private class ActionListenerImpl implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        switch (cboTarefas.getSelectedIndex()) {
            case 0:
                tarefasPane.add(new tarefaPane<Bater>(new Bater(
                    new String[]{txtNomeIngrediente.getText()})));
                break;
            case 1:
                tarefasPane.add(new tarefaPane<Congelar>(new Congelar(
                    new String[]{txtNomeIngrediente.getText()})));
                break;
            case 2:
                tarefasPane.add(new tarefaPane<Descascar>(new Descascar(
                    new String[]{txtNomeIngrediente.getText()})));
                break;
            case 3:
                tarefasPane.add(new tarefaPane<Fatiar>(new Fatiar(
                    new String[]{txtNomeIngrediente.getText()})));
                break;
            case 4:
                tarefasPane.add(new tarefaPane<Misturar>(new Misturar(
                    new String[]{
                        JOptionPane.showInputDialog(self, "Informe o primeiro ingrediente"),
                        JOptionPane.showInputDialog(self, "Informe o segundo ingrediente"),
                    })));
                break;
            case 5:
                tarefasPane.add(new tarefaPane<Picar>(new Picar(
                    new String[]{txtNomeIngrediente.getText()})));
                break;
        }
        self.setVisible(true); //Força a atualização visual da janela
    }
}

}

}

package receitas.LivroReceitas;

import receitas.DomainObjects.*;

/**
 *
 * @author Charles.Fortes
 */
public interface CadReceitasModalListener {
    void modalResult(Receita result);
}

package receitas.LivroReceitas;

```

```
import receitas.DomainObjects.Ingrediente;

/**
 *
 * @author Charles.Fortes
 */
public interface CadIngredienteModalListener {
    void modalResult(Ingrediente result);
}
```
