

**E.E.M. ENGENHEIRO ANNES GUALBERTO**

**CURSO DE ENSINO MÉDIO INTEGRADO A EDUCAÇÃO  
PROFISSIONAL (EMIEP) – TÉCNICO EM INFORMÁTICA**

JAYSTHERSON DE SOUSA MONTEIRO

LUCAS MANOEL LOPES

PEDRO SOUZA PITTIGLIANI DE CARVALHO

**DESENVOLVIMENTO DE UM PROTÓTIPO DE CASO DE USO  
TRYFATUS**

Imbituba, Santa Catarina

2019

JAYSTHERSON DE SOUSA MONTEIRO  
LUCAS MANOEL LOPES  
PEDRO SOUZA PITTIGLIANI DE CARVALHO

**DESENVOLVIMENTO DE UM PROTÓTIPO DE CASO DE USO  
TRYFATUS**

Monografia de trabalho para conclusão do curso  
de ensino médio integrado ao técnico em  
informática apresentado a escola E.E.M. Engº  
Annes Gualberto.

Orientador: Marcos Roberto Sanchez Mena

Imbituba, Santa Catarina  
2019

JAYSTHERSON DE SOUSA MONTEIRO  
LUCAS MANOEL LOPES  
PEDRO SOUZA PITTIGLIANI DE CARVALHO

## **DESENVOLVIMENTO DE UM PROTÓTIPO DE CASO DE USO TRYFATUS**

Trabalho final, apresentado a E.E.M. Eng. Annes  
Gualberto, como parte das exigências para a obtenção  
do título de técnico em informática.

Imbituba, 20 de novembro de 2019.

### **BANCA EXAMINADORA**

---

Prof. Marcos Roberto Sanchez Mena

---

Prof.<sup>a</sup>. Ana Cláudia Antunes Pinto Mantelli

---

Prof.<sup>a</sup>. Maria Elizabete Teixeira

## **AGRADECIMENTOS**

### **Jaystheron:**

Agradeço aos meus amigos e colegas por me apoiarem e estarem comigo.

### **Lucas:**

Agradeço a toda minha família que sempre esteve comigo. E também a todos aqueles que de certa forma colaboraram com este trabalho, meus amigos, e até aquelas pessoas desconhecidas nos fóruns que estavam com a resposta ao meu problema.

### **Pedro:**

Agradeço aos meus pais que me deram apoio e aos meus amigos que me fizeram rir nos meus momentos de decepção.

### **Todos:**

Agradecemos a nossa sanidade mental que não nos abandonou em momento algum. Agradecemos também ao orientador Marcos que durante todos os três anos de curso, sempre lutou e “botou a cara a tapa” por nós, e sempre esteve presente nos ajudando.

## **DEDICATÓRIAS**

### **Jaystheron:**

Dedico este trabalho a minha esposa que me apoiou a não procrastinar.

### **Lucas:**

Dedico este trabalho a minha mãe que sempre esteve comigo, me apoiando, colaborando, em especial por ser a cliente deste protótipo que espero ser útil a ela no futuro.

Dedico também a meu grande “Bro” Lucas Oliveira, já que graças a ele me tornarei um programador e mesmo depois de tantos anos, ainda é um bom e confiável amigo.

### **Pedro:**

Dedico este trabalho a meu amigo Victor que me chama de vadio toda vez que eu digo que cansei, assim me motiva a fazer as coisas...de certo modo.

### **Todos:**

Dedicamos este trabalho a nosso grandioso amigo Tiago Mertens que ouviu nossas reclamações durante todos estes anos e nos apoiou com sua genialidade incompreendida.

## **RESUMO**

Este trabalho tem como objetivo a criação de um software para gerenciamento de cliente e vendas para um pequeno estabelecimento comercial local que depende de atividades manuais para realizar estas tarefas. Fato comum em diversos estabelecimentos do país que ainda não possuem automatização que com ela poderiam ter seus serviços realizados de maneira mais eficaz. Este protótipo foi iniciado com a análise dos requisitos, obtidos por entrevista com o cliente. Após isso, os mesmos foram analisados e serviram de base para criação dos diagramas de caso de uso e classe, parte importante da modelagem de sistemas. Baseado neste resultado, foi decidido então a utilização de plataformas como NetBeans para programação Java, e MySQL para o gerenciamento do banco de dados.

**Palavras-chave:** Engenharia de Requisitos. Modelagem de Sistemas. Programação orientada a objetos. Protótipo Comercial. Gerenciamento de Dados.

## **ABSTRACT**

This final paper aims to create a customer and sales management software for a small local business establishment that depends on manual activities to perform these tasks, a common fact in several establishments in the country that do not have automation yet, which could perform them more effectively with it. This prototype was started by analyzing the requirements obtained by interviewing the customer. After that, they were analyzed and used as basis for the creation of the use case and class diagrams, an important part of the systems modeling. Based on this result, it was then decided to use platforms such as NetBeans for Java programming, and MySQL for database management.

**Keywords: Requirements Engineering. Systems Modeling. Object-Oriented programming. Commercial prototype. Data management.**

## LISTA DE SIGLAS

|      |  |
|------|--|
| DB   | <i>Data Base</i>                       |
| DBMS | <i>Data Base Management System</i>     |
| IDE  | Interface de Desenvolvimento Integrada |
| OMG  | <i>Object Management Group</i>         |
| POO  | Programação Orientada a Objetos        |
| SGBD | Sistema Gerenciador de Banco de Dados  |
| SQL  | <i>Structured Query Language</i>       |
| UML  | <i>Unified Modeling Language</i>       |



## LISTA DE QUADROS

|   |    |
|---|----|
| Quadro 1 - Requisitos do Cliente.....                       | 13 |
| Quadro 2 - Caso de Uso: Registrar Venda.....                | 18 |
| Quadro 3 - Caso de Uso: Gerenciar Clientes.....             | 19 |
| Quadro 4 - Caso de Uso: Alterar Dados Cadastrais.....       | 19 |
| Quadro 5 - Caso de Uso: Cadastrar Clientes.....             | 20 |
| Quadro 6 - Caso de Uso: Registrar Pagamento de Parcela..... | 20 |
| Quadro 7 – Método updateVendas.....                         | 33 |
| Quadro 8 – Método updateVenda.....                          | 34 |
| Quadro 9 – Método updateVencimento.....                     | 35 |

## LISTA DE FIGURAS

|  |    |
|--|----|
| Figura 1 - Exemplo de Diagrama de Caso de Uso..... | 15 |
| Figura 2 - Diagrama de Caso de Uso.....            | 16 |
| Figura 3 - Exemplo de Diagrama de Classe.....      | 22 |
| Figura 4 - Diagrama de Classe.....                 | 22 |
| Figura 5–Tela Principal.....                       | 27 |
| Figura 6–Cadastro de Clientes.....                 | 28 |
| Figura 7–Alterar Cadastro.....                     | 29 |
| Figura 8– Registro de Vendas.....                  | 30 |
| Figura 9–Registro de Pagamentos.....               | 31 |
| Figura 10–Tela de Definição de Juros.....          | 32 |

# SUMÁRIO

|   |           |
|---|-----------|
| <b>1 INTRODUÇÃO.....</b>                                  | <b>11</b> |
| 1.1 A LOJA TRYFATUS.....                                  | 11        |
| <b>2. ENGENHARIA DE REQUISITOS.....</b>                   | <b>12</b> |
| 2.1 OS REQUISITOS DO CLIENTE.....                         | 13        |
| <b>3. MODELAGEM DE SISTEMAS.....</b>                      | <b>13</b> |
| 3.1 LINGUAGEM DE MODELAGEM UNIFICADA.....                 | 14        |
| <b>3.1.1História do UML.....</b>                          | <b>14</b> |
| <b>3.1.2Diagrama de Caso de Uso.....</b>                  | <b>15</b> |
| 3.1.2.1 Componentes do Caso de Uso.....                   | 16        |
| <b>3.1.2.1.1 Ator.....</b>                                | <b>17</b> |
| <b>3.1.2.1.2 Caso de Uso.....</b>                         | <b>17</b> |
| <b>3.1.2.1.3 Relacionamentos.....</b>                     | <b>17</b> |
| 3.1.2.2 Documentação do Caso de Uso.....                  | 18        |
| <b>3.1.3 Diagrama de Classe.....</b>                      | <b>21</b> |
| <b>4. PROGRAMAÇÃO ORIENTADA A OBJETOS.....</b>            | <b>23</b> |
| 4.1 A LINGUAGEM JAVA.....                                 | 23        |
| 4.2 INTERFACE DE DESENVOLVIMENTO INTEGRADO: NETBEANS..... | 24        |
| <b>5. BANCO DE DADOS.....</b>                             | <b>24</b> |
| 5.1 ORIGEM DO BANCO DE DADOS.....                         | 25        |
| 5.2 FUNCIONAMENTO.....                                    | 25        |
| 5.3 STRUCTURED QUERY LANGUAGE.....                        | 26        |
| 5.4 MYSQL.....  | 26        |
| <b>6 INTERFACES DO PROTÓTIPO.....</b>                     | <b>27</b> |
| <b>7 PROGRAMAÇÃO.....</b>                                 | <b>33</b> |
| 7.1 FUNCIONAMENTO DA PROGRAMAÇÃO.....                     | 35        |
| <b>8 CONCLUSÃO.....</b>                                   | <b>36</b> |
| 8.1 TRABALHOS FUTUROS .....                               | 37        |
| <b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>                    | <b>38</b> |

## **1 INTRODUÇÃO**

Diante das barreiras encontradas no dia-a-dia para vendas em um comércio local notou-se que desenvolvendo um pequeno protótipo o mesmo poderia auxiliar na gestão de pequenas tarefas cotidianas. Pensando nisso, e utilizando-se da engenharia de requisitos como uma ferramenta que tende a auxiliar no processo de desenvolvimento de software foi sugerido para este estabelecimento a criação de um protótipo para gerenciar as vendas e clientes, pois estas atividades até então eram feitas manualmente.

O protótipo tem como objetivo auxiliar estas atividades poupando recursos importantes para uma empresa, sendo eles tempo, dinheiro e excesso de trabalho, já que pesquisas de clientes levarão muito menos tempo, além de acabar com a necessidade de uso de papel e busca manual. Todos estes fatores podem influenciar na movimentação da loja, pois os clientes sairiam mais satisfeitos graças ao atendimento rápido, além de diminuir o trabalho do atendente.

### **1.1 A LOJA TRYFATUS**

A loja TryFatus foi fundada em outubro de 2007, na rua Ernani Cotrim, no centro de Imbituba, Santa Catarina, onde se encontra desde então. Foi criada por Lucia Helena Bittencourt Lopes, familiarmente, com intuito de ajudar a pagar os estudos dos três filhos. Segundo a mesma, “O nome TryFatus foi porque foram os três fatos mais importantes da minha vida, os três filhos, Luana, Mariana e Lucas”.

Seu principal produto são roupas e acessórios de baixo preço, em grande parte femininos, porém conta com produtos dedicados também ao público masculino. A proprietária disse: “Meus produtos são voltados mais ao público feminino com moda evangélica também, além de acessórios como brincos, colares, anéis, bolsas, etc...”

Foi perguntado a cliente o que a mesma esperava sobre o protótipo, Lúcia respondeu: “Eu acredito que vai ajudar na organização dos fichários, controle melhor dos clientes e do fluxo de caixa, geração de recibos e carnês, facilitando assim o funcionamento da loja”.

## 2 ENGENHARIA DE REQUISITOS

A engenharia de requisitos é a responsável por tratar das necessidades do cliente que podem ser avaliadas durante uma entrevista, e transformá-las em informações técnicas sobre as funcionalidades do mesmo. Segundo Espindola, Majdenbaum e Audy (2004, p. 2), Engenharia de Requisitos “É um termo que engloba todas as atividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos para um sistema computacional”.

Nelas serão estudadas as principais necessidades do sistema e estudar quais devem ser implementados no mesmo, além dos possíveis custos para o desenvolvimento.

Sobre a engenharia de requisitos pode-se afirmar que:

Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços a serem oferecidos e as restrições a seu funcionamento. Esses requisitos refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como controlar um dispositivo, colocar um pedido ou encontrar informações. O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado engenharia de requisitos. (SOMMERVILE, 2011, p. 57).

Ou seja, os requisitos são as definições do cliente que devem ser estudadas pelos engenheiros de software para entenderem melhor suas necessidades e vontades. Eles são primariamente divididos em dois: requisitos funcionais e não funcionais.

Requisitos funcionais: São declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais também podem explicitar o que o sistema não deve fazer. 2.

Requisitos não funcionais: São restrições aos serviços ou funções oferecidas pelo sistema. Incluem restrições de timing, restrições no processo de desenvolvimento e restrições impostas pelas normas. Ao contrário das características individuais ou serviços do sistema, os requisitos não funcionais, muitas vezes, aplicam-se ao sistema como um todo. (SOMMERVILE, 2011, p. 59).

Os requisitos funcionais são os principais de um software e também primordiais para o seu funcionamento. Definem o que o sistema deve ou não fornecer. Os requisitos não funcionais não são primordiais ao sistema em si, e podem não estar relacionados com os serviços do software, mas podem afetar de uma forma geral e alguns destes relacionados a confiabilidade do sistema, tempo de resposta ou a linguagem de programação a ser utilizada no desenvolvimento.

Os primeiros passos para a criação de um software são muito importantes em todo o processo, qualquer erro de interpretação, levantamento ou compreensão dos mesmos podem levar todo o projeto ao fracasso (XAVIER, Laís, 2009, p. 8).

## 2.1 OS REQUISITOS DO CLIENTE

Os requisitos do sistema a ser desenvolvido neste trabalho, especificados pelo cliente durante uma entrevista, são os listados a seguir.

Quadro 1 - Requisitos

| Requisitos Funcionais        | Requisitos Não-Funcionais     |
|------------------------------|-------------------------------|
| 1. Cadastro de Clientes      | 1. Cobrar juros opcionalmente |
| 2. Registro de Vendas        | 2. Validar o CPF no cadastro  |
| 3. Gerenciamento de parcelas |                               |
| 4. Pagamentos de débitos     |                               |
| 5. Cálculo de juros          |                               |

## 3 MODELAGEM DE SISTEMAS

Modelagem de sistemas é a parte de desenvolvimento abstrata do sistema, assim planejando como deverá funcionar. Ela irá mostrar as funções do software e as possibilidades de ações do mesmo, assim irá dar uma visão geral sobre todo o programa, suas ações, interações, etc.

Sobre modelagem de sistemas, é possível afirmar que:

Modelagem de sistema é o processo de desenvolvimento de modelos abstratos de um sistema, em que cada modelo apresenta uma visão ou perspectiva, diferente do sistema. A modelagem de sistema geralmente representa o sistema com algum tipo de notação gráfica. (SOMMERVILLE, 2011, p. 82).

A modelagem de um sistema cria uma maneira intuitiva de pensar sobre seu funcionamento, assim irá guiar os programadores e envolvidos no processo de criação e desenvolvimento, o que certamente facilitará a construção de um software de qualidade, havendo menos erros, e assim irá diminuir os custos de manutenção futuros e retrabalho.

### 3.1 LINGUAGEM DE MODELAGEM UNIFICADA

A Linguagem de Modelagem Unificada é basicamente uma família de notações gráficas que apoia um modelo-base único que facilita o desenvolvimento de projetos de sistemas.

Conhecida pelo nome em inglês, *Unified Modeling Language* (UML), e segundo Leandro Ribeiro (2012), “é uma linguagem que define uma série de artefatos os quais nos ajuda na tarefa de modelar e documentar os sistemas orientados a objetos que desenvolvemos”.

Ela define elementos gráficos para modelar sistemas orientados a objetos, esses elementos permitem representar os conceitos de um programa por meio de diagramas que irão representar o funcionamento do sistema (Bezerra, 2007).

#### 3.1.1 HISTÓRIA DO UML

A UML foi criada por um grupo de colaboradores, sendo os principais Grady Booch, James Rumbaugh e Ivar Jacobson, chamados de “Os três amigos”. Eles aproveitaram o melhor das técnicas já existentes com a remoção de certos elementos e adição de novos, com o intuito de deixá-la mais expressiva (Bezerra, 2007).

Ela começou a ser criada em outubro de 1994, quando Rumbaugh e Booch se uniram para unificar seus métodos. Lançaram em 1995 um esboço do resultado, época em que Jacobson se uniu ao projeto e o expandiram para incorporar seu método (O OOSE).

Lançaram em junho de 1996 outra versão, e em 1997 a UML foi aprovada pela OMG (Object Management Group) que é um grupo de empresas que ratifica padrões para a programação orientada a objetos (FELIZARDO, 2013).

### 3.1.2 DIAGRAMA DE CASO DE USO

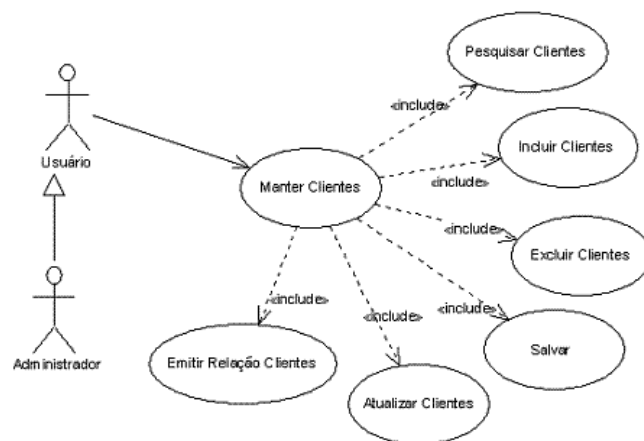
O modelo de caso de uso atualmente é um dos mais utilizados e um dos primeiros a serem criados, já que por ser visualmente simples e de fácil entendimento para leigos na área, assim o cliente pode analisar se o planejamento está decorrendo de acordo com suas necessidades e requisitos.

O modelo de caso de uso é amplamente utilizado para garantir que os requisitos se encaixem com a necessidade do cliente, por ser um cenário simples que descreve o sistema de forma geral (Sommerville, 2011).

Esse modelo representa as funcionalidades observáveis de um ponto de vista externo, ou seja, não irá exibir como ocorrerá o processamento interno dos dados, no programa. Geralmente representam um refinamento dos requisitos funcionais (Bezerra, 2007).

Abaixo mostra-se a figura de um diagrama de caso de uso para gerência de clientes.

Figura 1 – Exemplo de Diagrama de Caso de Uso

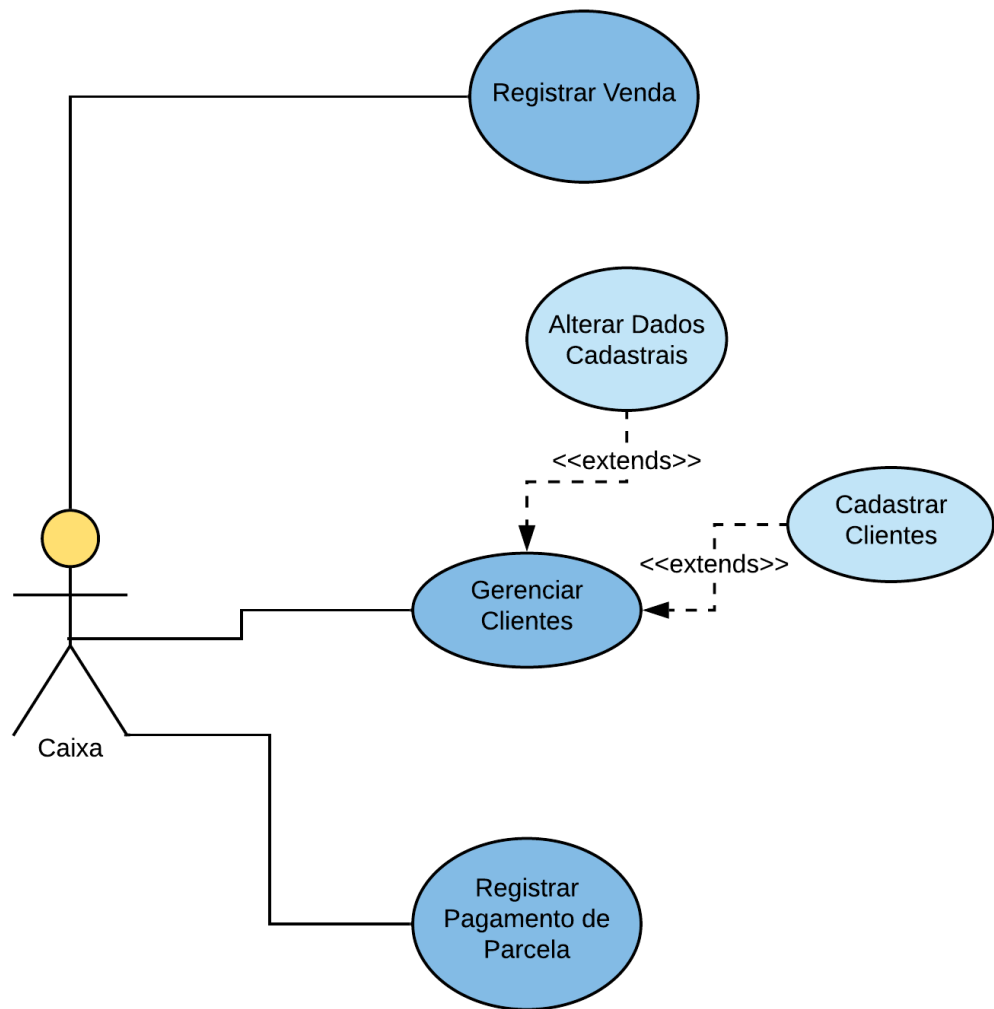


Fonte: Grupo TADS (2014)

Na imagem a seguir, está mostrado o diagrama caso de uso criado para o programa a ser desenvolvido neste projeto.



Figura 2 – Diagrama de Caso de Uso



### 3.1.2.1 COMPONENTES DO CASO DE USO

Um diagrama de caso de uso evolve o uso de três componentes: Atores, Caso de Uso e Relacionamentos entre os casos de uso (Bezerra, 2007).

### **3.1.2.1.1 ATOR**

Segundo Eduardo Bezerra (2007, p. 60), “qualquer elemento externo ao sistema que interage com o mesmo é, por definição, denominado ator”. O termo “externo” nessa definição indica que atores não fazem parte do sistema.

Sendo assim, um ator não precisa ser necessariamente uma pessoa, podendo também ser outros softwares, ferramentas, entre outras coisas.

### **3.1.2.1.2 CASO DE USO**

Um caso de uso é simbolizado por uma elipse e representa as ações e funções do sistema que dependem de fator externo, como um Ator (Sommerville, 2011).

Entretanto, um caso de uso não considera a estrutura e comportamentos internos do sistema, por isso é necessário elaborar a documentação dos mesmos, afim de explicar o seu comportamento. (Bezerra, 2007)

### **3.1.2.1.3 RELACIONAMENTOS**

Os relacionamentos, como o próprio nome diz, servem para exibir a relação entre os atores e os casos de uso. A UML define 4 tipos de relacionamento: inclusão, extensão e generalização (Bezerra, 2007).

Segundo Ventura (2014), o relacionamento de Inclusão significa que os casos de uso sempre serão executados juntos, ou seja, quando o caso e uso X inclui o caso Y, sempre que X for executado, Y também será.

O mesmo ainda explica sobre o caso de uso de Extensão, indicando que sempre onde um dos casos for executado, o outro poderá ser executado, mas talvez não seja.

Ainda segundo Ventura (2014), uma generalização significa que quando um caso de uso X está generalizado a Y, quando X for executado, ele realizará tudo especificado em X e Y.

### 3.1.2.2 DOCUMENTAÇÃO DO CASO DE USO

Segundo Kolb (2014), “A documentação de um Caso de Uso costuma descrever por meio de uma linguagem bastante simples, a função em linhas gerais do Caso de Uso”.

Abaixo, a lista com todos os quadros da documentação, e uma breve explicação sobre cada um.

Quadro 2 – Caso de Uso: Registrar Venda

| Nome do Caso de Uso                  | Registrar Venda   |
|--------------------------------------|---|
| Ator Principal                       | Caixa   |
| Ator secundário                      | Cliente   |
| Resumo                               | Descreve etapas de um caixa ao realizar uma venda. Armazena as informações referente a venda: produto, valor total, quantia de parcelas, meta de pagamento, data da compra, data de vencimento. |
| Pré-condição                         | Cliente Cadastrado  |
| Pós-condição                         | O cliente não poderá estar com as parcelas vencidas   |
| Ações do ator                        | Ações do sistema  |
| 1-Solicitar o registro de venda      |   |
|                                      | 2-Consultar cliente   |
| 3-Inserir dados de registro de venda |   |
| 4-Salvar dados                       |   |
|                                      | 5-Validar dados   |

Se trata do registro das vendas realizadas assim armazenando as informações referentes a mesma, ou seja, método de pagamento, número de parcelas, data da compra, etc.

Quadro 3 – Caso de Uso: Gerenciar Clientes

| Nome do Caso de Uso                                   | Gerenciar Clientes   |
|---|--|
| Ator Principal  | Caixa  |
| Ator Secundário                                       | Cliente  |
| Resumo  | Responsável pelo gerenciamento de dados do cliente, seja cadastrar ou alterar. |
| Pré-condição  | O cliente deve ser registrado  |
| Pós-condição  | Os dados do cliente devem estar atualizados                                    |
| Ações do ator   | Ações do sistema   |
| 1-Solicitar dados ao cliente                          |  |
| 2-Inserir dados do cliente                            |  |
|   | 3-Verificar cadastro   |
| 4-Solicitar cadastro do cliente ou alteração de dados |  |
|   | 5-Cadastrar ou alterar dados do cliente  |

Gerencia os dados do cliente, alterando ou criando um novo cadastro.

Quadro 4 – Caso de Uso: Alterar Dados Cadastrais

| Nome do Caso de Uso                 | Alterar Dados Cadastrais  |
|-------------------------------------|---|
| Ator Principal                      | Caixa   |
| Ator Secundário                     | Cliente   |
| Resumo                              | Responsável pela alteração dos dados de um cliente já cadastrado. |
| Pré-condição                        | Cliente cadastrado  |
| Pós-condição                        | Novos dados   |
| Ações do ator                       | Ações do sistema  |
| 1-Solicitar novos dados ao cliente  |   |
| 2-Inserir os novos dados do cliente |   |
|                                     | 3-Salvar a alteração dos dados                                    |

Se trata de uma parte do gerenciamento dos clientes, responsável pela alteração dos dados do cliente, como por exemplo, o endereço de onde ele mora caso haja uma mudança.

Quadro 5 – Caso de Uso: Cadastrar Clientes

| Nome do Caso de Uso             | Cadastrar Clientes                                 |
|---------------------------------|--|
| Ator Principal                  | Caixa  |
| Ator Secundário                 | Cliente  |
| Resumo                          | Responsável pela atividade de cadastrar o cliente. |
| Pré-condição                    | CPF válido   |
| Pós-condição                    | Cadastro concluído                                 |
| Ações do ator                   | Ações do sistema                                   |
| 1-Solicitar os dados do cliente |  |
| 2-Inserir dados do cliente      |  |
|                                 | 3-Validar CPF                                      |
|                                 | 4-Concluir cadastro                                |

Se trata do cadastro de um novo cliente para que ele possa fazer compras no estabelecimento.

Quadro 6 – Caso de Uso: Registrar pagamento de parcela

| Nome do Caso de Uso                       | Registrar Pagamento de Parcela  |
|---|---|
| Ator Principal                            | Caixa   |
| Ator Secundário                           | Cliente   |
| Resumo                                    | Responsável pela atividade de registrar o pagamento das parcelas da compra. |
| Pré-condição                              | Compra registrada   |
| Pós-condição                              | Deduzir pagamento   |
| Ações do ator                             | Ações do sistema  |
| 1-Solicitar pagamento por parcela         |   |
|   | 2-Solicitar valor e quantidade das parcelas                                 |
| 3-Inserir valor e quantidade das parcelas |   |
|   | 4-Registrar pagamento de parcela  |

Atividade de realizar os registros dos pagamentos das parcelas.

### 3.1.3 DIAGRAMA DE CLASSE

Esse tipo de diagrama tem o objetivo de organizar as classes e seus relacionamentos, assim também, apresentando seus métodos e atributos. Classes são abstrações de entidades reais, nos quais nelas são atribuídas as características como por exemplo: a classe Cachorro pode ter características como Raça, Tamanho, Peso, Cor, entre outros. Além disso, podem possuir métodos, como andar, latir, correr, entre outros.

Sobre diagramas de classe, pode-se afirmar:

Os diagramas de classe são usados no desenvolvimento de um modelo de sistema orientado a objetos para mostrar as classes de um sistema e as associações entre essas classes. Em poucas palavras, uma classe de objeto pode ser pensada como uma definição geral de um tipo de objeto do sistema. Uma associação é um link entre classes que indica algum relacionamento entre essas classes. Consequentemente, cada classe pode precisar de algum conhecimento sobre sua classe associada (SOMMERVILE, 2011, p. 90).

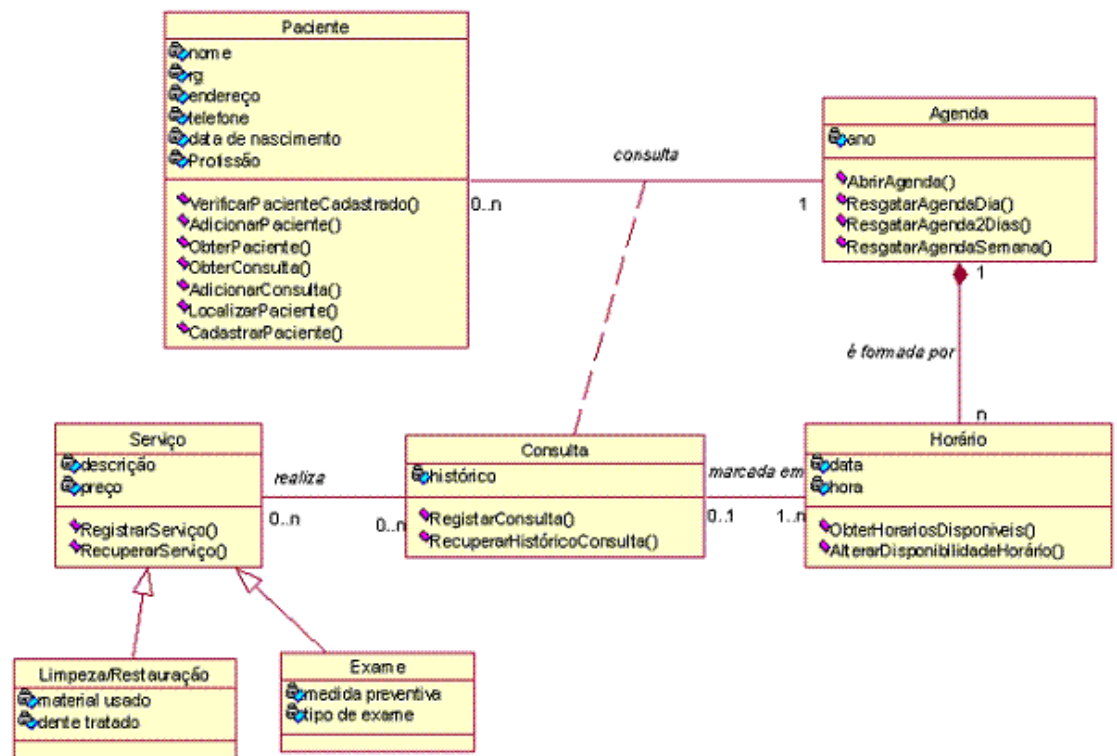
As classes são formadas por retângulos contendo até 3 compartimentos, sendo eles nome, atributos e operações. Pôr convenção, o nome deve ser iniciado com letra maiúscula e no singular (Bezerra, 2007).

Segundo Eduardo Bezerra (2007, p. 112), “No segundo compartimento, são declarados os atributos, que correspondem às informações que um objeto armazena”. Podemos ter como por exemplo: tipo, nome, sexo, entre outras coisas.

No terceiro compartimento, ficam as operações que segundo Eduardo Bezerra (2007, p. 113) “Correspondem às ações que um objeto sabe realizar”. Como por exemplo, consultar, validar, etc...

Abaixo, mostre um exemplo de um diagrama de classes de uma clínica, mostrando os atributos das classes e suas relações.

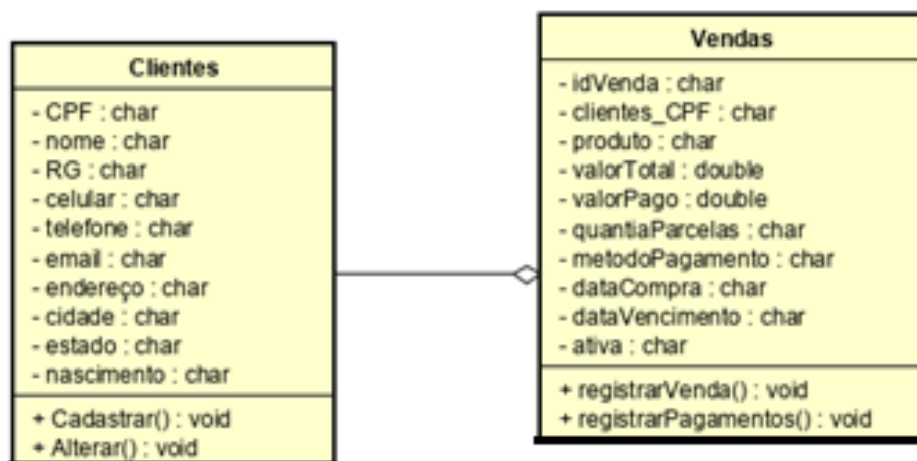
Figura 3 – Exemplo de Diagrama de Classes



Fonte: Macoratti (2013)

A imagem a seguir mostra o diagrama de classes criado para este projeto.

Figura 4 – Diagrama de Classe



## 4 PROGRAMAÇÃO ORIENTADA A OBJETOS

A Programação Orientada a Objetos (POO) é o paradigma de programação mais utilizado atualmente e está evoluindo cada vez mais. Isso se deve ao fato de que os códigos nesse paradigma não são complexos de criar, bastando adaptar aos requisitos (Gasparotto, 2014). O programador quando utiliza POO deve pensar nas coisas de maneira distinta, para transformá-las em classes e objetos. Graças a esta distinção, um dos pontos positivos desta linguagem é que seu código tende a ser facilmente reutilizável.

As classes e objetos são geralmente abstrações da realidade e são nelas em que estão contidos os códigos. Podem se implementar diversos atributos e funcionalidades nestas classes utilizando as variáveis e métodos. Por exemplo: a classe Pessoa pode ter os atributos Nome, Sexo, Idade, e os métodos Andar, Correr, Falar (Santana, 2015).

As principais linguagens com suporte a POO atualmente são o Java, C++, C#, Python, Ruby, Object Pascal, entre diversas outras.

### 4.1 A LINGUAGEM JAVA

Java é uma linguagem de POO que começou a ser criada pela Sun Microsystems em 1991. Teve início com o Green Project os quais os mentores foram Patrick Naughton, Mike Sheridan, e James Gosling. Este projeto não tinha o objetivo de criar uma nova linguagem de programação, e sim antecipar a “nova era” que estava chegando na área da informática.

A ideia era possibilitar a criação de programas portáteis que pudessem ser executados em diversos dispositivos. Mais a equipe teria que desenvolver programas específicos para cada tipo de dispositivo, daí surgiu a ideia de desenvolver um sistema operacional que permitiria a utilização de seus programas pelos mais diversos tipos de equipamento. A nova linguagem foi batizada de Oak (carvalho), uma referência ao carvalho que James Gosling visualizava a partir de seu escritório. O sistema operacional que foi desenvolvido foi chamado de GreenOS, e junto com ele foi construída uma interface gráfica padronizada (Filgueiras, 2015).

Com a internet ficando cada vez mais popular, começaram a pensar em aplicações para o Oak na mesma, onde a palavra-chave é interação. Após adaptar a linguagem Oak para a web, em 1995, foi criado o Java que era uma versão atualizada do mesmo para a internet.



O nome da linguagem desenvolvida pelo Green Project teve seu nome modificado em homenagem à uma ilha da Indonésia, de onde os Norte-Americanos importava o café que era consumido pela equipe de James Gosling (Pacievitch, 2013).

## **4.2 INTERFACE DE DESENVOLVIMENTO INTEGRADO: NETBEANS**

NetBeans é um IDE (ambiente de desenvolvimento integrado), Java desenvolvido pela Sun Microsystems que atualmente está sendo usado por diversas plataformas como Windows e Linux.

O NetBeans foi iniciado em 1996, por dois estudantes tchecos na universidade de Charles, em Praga, isso quando a linguagem de programação Java não era tão popular quanto atualmente. O nome inicial era Xelfi, em alusão ao Delphi (Ambiente de desenvolvimento na linguagem Object Pascal), que tinha IDE's populares e atrativas (M3 Mídia, 2008).

## **5 BANCO DE DADOS**

Atualmente, um sistema em Banco de Dados é muito comum, pois faz-se necessário ter um acesso rápido as informações de determinados negócios, sendo utilizado para trabalhar com desenvolvimento de softwares, armazenando diversos tipos de arquivos, como por exemplo, cadastrar clientes, funcionários, produtos, entre outras diversas coisas (Alves, 2013).

## 5.1 ORIGEM DO BANCO DE DADOS

No passado, as empresas utilizavam fichas de papel e materiais físicos para armazenar os dados, porém, além de ocupar muito espaço, foram analisados e desenvolvidos por Edgar Frank Codd e Dr. Peter Chenque que seria menos custoso e mais rápido armazenar as informações de maneira digital, via banco de dados. No início, os softwares eram simples e apenas cadastravam, alteravam, excluía e consultavam arquivos digitais, porém as entidades precisavam se relacionar, por exemplo um produto é fornecido por um fornecedor, e com os arquivos digitais relacioná-las não era: uma tarefa muito trivial, os softwares simples para manipular os arquivos digitais começaram a ficar complexos para permitir os relacionamentos entre entidades (Alves, 2013).

## 5.2 FUNCIONAMENTO

Um banco de dados é um conjunto de dados existentes num sistema informatizado e está disponível a todos os utilizadores ou processamentos de organizações em que o acesso e atualização são realizados através de um software específico.

O SGBD (Sistema de Gestão de Bases de Dados) ou Data Base Management System (DBMS), o software utilizado para gerir Bases de Dados (Data Bases - DB), no que permite criar, modificar ou excluir DBs, inserindo e/ou eliminando os dados nela contidos. Eles têm um conjunto de requisitos funcionais: a Segurança, a Integridade (que só inclui dados válidos relativamente à realidade), o Controle de Concorrência (Locking, Etiquetagem ou Optimista), e a recuperação e tolerância a falhas (Backup e *Transactionlogging*) (Schimiguel, 2014).

### 5.3 STRUCTURED QUERY LANGUAGE

O SQL (Linguagem de Consulta Estruturada) é uma linguagem que manipula banco de dados relacionais através dos SGBDs. Alguns dos principais sistemas que utilizam SQL são: MySQL, Oracle, Firebird, Microsoft Access, PostgreSQL, HSQLDB. Alguns dos principais comandos SQL para manipulação de dados são: INSERT (inserir), SELECT (consulta), UPDATE (atualizar), DELETE (excluir). SQL possibilita ainda a criação de relações entre tabelas e o controle do acesso aos dados, ou seja, é necessário ter conhecimento de SQL onde irá ser desenvolvido um software, pois os sistemas de informações geralmente necessitam de um banco de dados para tais atividades empresariais (Alves, 2013).

### 5.4 MYSQL

MySQL é uma interface de criação de banco de dados de código aberto e o mais utilizado atualmente pelo seu bom desempenho, funcionalidades bem desenvolvidas, e sua facilidade de uso, sendo confiável suficiente para ser utilizado pelos maiores sites da rede Web, como Facebook, Twitter, YouTube. foi desenvolvido pela empresa Oracle Corporation, sua versão inicial foi lançada em, 23 de maio de 1995, tendo o nome MySQL. O MySQL recebeu o nome da filha do coo-fundador Monty Widenius, o nome do golfinho que está na logo é Sakila e foi escolhido por um fórum, onde o nome foi enviado por Ambrose Twebaze, desenvolvedor de software de código aberto da Suazilândia, África (Oracle, 2007).

## 6 INTERFACES DO PROTÓTIPO

As figuras a seguir representam a interface do protótipo:

Figura 5 – Tela Principal

The screenshot displays the main interface of the TryFatus system. The window title is "TryFatus". The interface is divided into several sections:

- Top Bar:** Contains the "Admin" label and a search bar for "Cliente: Pedro Souza Pittigliani de Carvalho - 08342055812". There are buttons for "Pesquisar" (Search) and "Cadastrar Novo" (Register New).
- Gerenciamento (Management):** This section includes buttons for "Adicionar Venda" (Add Sale) and "Registrar Pagamento" (Register Payment).
- Registro de Vendas (Sales Record):** A table displaying sales data:
 

| ID | Produto      | Valor Total | Valor Pago | Parcelas | Método de ... | Data da Co... | Vencimento |
|----|--------------|-------------|------------|----------|---------------|---------------|------------|
| 1  | Calça Jeans  | 120         | 120        | 1        | A Vista       | 2019-11-08    |            |
| 2  | Camisa So... | 149.9       | 50         | 2        | A Prazo       | 2019-11-08    | 2019-12-08 |
- Dados do Cliente (Client Data):** A sidebar on the right containing client information:
  - Nome: Pedro Souza Pittigliani de Carvalho
  - CPF: 08342055812
  - RG: 6758216
  - Celular: 85468465468
  - Telefone: 489865468465
  - Email: pedro-s-carv@hotmail.com
  - Endereço: Casa
  - Bairro: Paes Leme
  - Cidade: Imbituba
  - Estado: SC
  - Nascimento: 02/11/2001
 There is an "Alterar Dados" (Change Data) button at the bottom of this section.
- Resumo Financeiro (Financial Summary):** A bottom bar showing financial metrics:
  - Total em Compras: 149.9
  - Em Aberto: 99.9
  - Compras Abertas: 1
  - Juros Total: 0.0

Esta é a tela principal do programa que será a primeira a ser aberta ao executar o mesmo. Inicialmente, ela começará com todos os dados vazios, pois não terá um cliente selecionado, e as opções Adicionar Venda, Registrar Pagamento e Alterar Dados estarão bloqueadas.

Na aba Gerenciamento, há uma barra de pesquisa, onde permite-se pesquisar o cliente pelo nome e sempre mostrando seu CPF ao lado. Escolhendo o cliente, clica-se em pesquisar, assim irá preencher todos os campos das demais abas, liberar as funções de venda e alteração.

A interface irá mostrar todos os dados a direita, as vendas na tabela ao meio e um resumo do financeiro do cliente na aba inferior. Para realizar qualquer ação, basta clicar no botão correspondente. Para escolher um novo cliente, basta buscar na barra de pesquisa, e clicar em pesquisar que os campos serão automaticamente preenchidos novamente.

Na barra superior, há a opção Admin, abre a opção Juros, onde é permitido alterar o juro base do sistema.

Figura 6 – Cadastro de Clientes

**TryFatus**

**Dados do Cliente**

Nome:

CPF:  RG:

Nascimento:

**Dados de Contato**

Telefone:

Celular:

Email:

**Endereço**

Endereço:

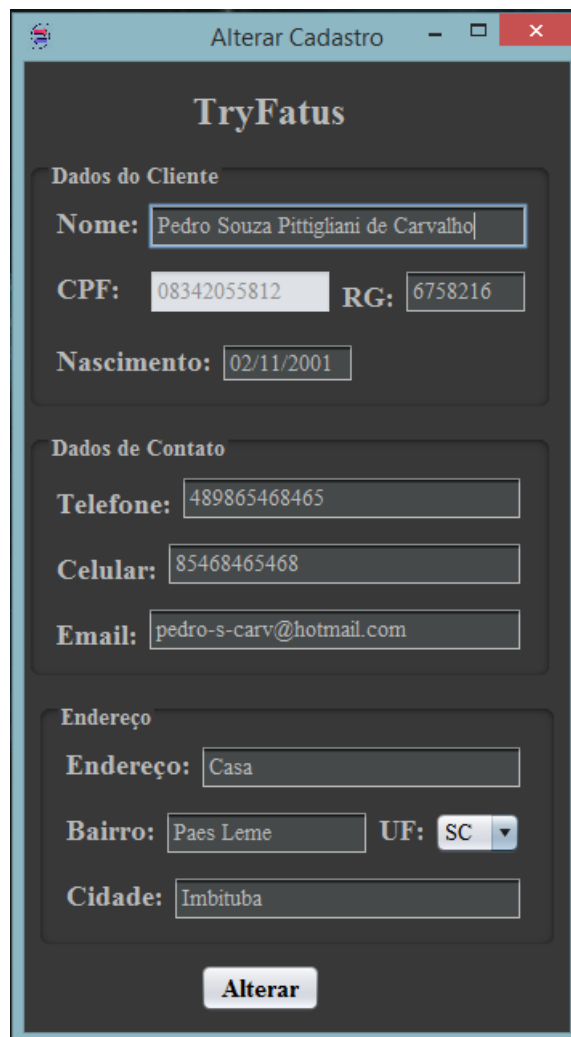
Bairro:  UF: SC

Cidade:

**Cadastrar**

É nessa tela, onde ocorre o cadastro de clientes no sistema. Os campos Nome, CPF, RG e Nascimento são obrigatórios, enquanto os campos Telefone, Celular, Email, Endereço, Bairro e Cidade não são de preenchimento obrigatório.

Figura 7 – Alterar cadastro



**TryFatus**

**Dados do Cliente**

Nome:

CPF:  RG:

Nascimento:

**Dados de Contato**

Telefone:

Celular:

Email:

**Endereço**

Endereço:

Bairro:  UF:

Cidade:

**Alterar**

É nessa tela onde ocorre a alteração dos dados do cliente, já cadastrado no sistema da loja. Por ser o identificador primário, o CPF não pode ser alterado.

Figura 8 – Registro de vendas

**Registro de Vendas**

**TryFatus**

**Dados do Cliente**

**Nome:** Pedro Souza Pittigliani de Carvalho

**CPF:** 08342055812 **RG:** 6758216

**Dados da Venda**

**Produto:**

**Valor:** **Data:** 08/11/2019

**Pagamento:** A Vista **Parcelas:** 1

**Vencimento:** / / **Valor Pago:**

**Cadastrar Venda**

Essa tela é a responsável pelo registro das vendas que ocorrem na loja. Os campos Nome, CPF e RG, já são preenchidos automaticamente, enquanto Produto, Valor e Parcelas são de preenchimento obrigatório. E os campos Data, Vencimento e Valor Pago não são de preenchimento obrigatório.

Figura 9 – Registro de Pagamentos

**Registro de Pagamentos**

**TryFatus**

**Dados do Cliente**

Nome: Pedro Souza Pittigliani de Carvalho

CPF: 08342055812 RG: 6758216

**Dados de Compras**

Em Aberto: 1 Vencidas: 0

**Dados Financeiros**

Total: 149.9 Em Aberto: 99.9

Vencido: 0.0 Juros: 0.0

**Pagamento**

Valor: 0.00 Parcelas: 1

Novo Vencimento: / /

☒ Cobrar Juros ☒ Contar para todas

**Registrar Pagamento**

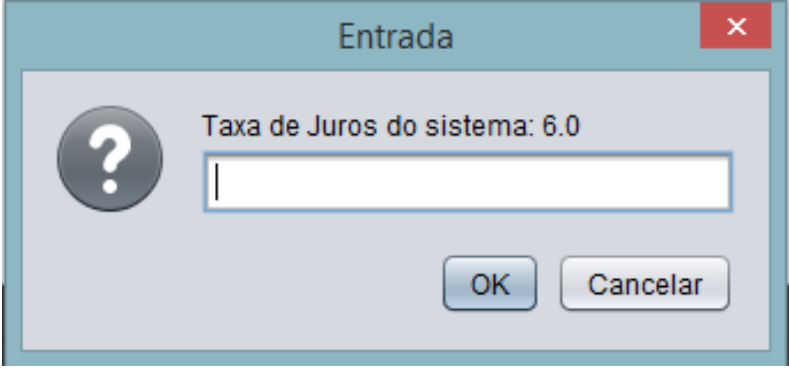
Essa tela é a responsável pelo registro de pagamentos. Os campos Nome, CPF, RG, Em Aberto, Vencidas, Total, Em Aberto, Vencido, Juros são preenchidos automaticamente, como meio do caixa ter um controle sobre a informação. Já os campos Valor, Parcelas e Novo Vencimento são de preenchimento obrigatório, pois indicam de fato o pagamento em si. As caixas Cobrar Juros e Contar para todas, já vem automaticamente marcadas, podendo ser desmarcadas.

Cobrar Juros irá cobrar juros adicionando o valor do juro, dependendo da quantia de parcelas, ao total do pagamento da compra.

Contar para todas é como se todas as compras abertas recebessem o pagamento, assim atualizando seu valor total e vencimento, mas não necessariamente o valor pago. Quando desmarcada, atualiza apenas as vendas em que o valor foi suficiente para pagar qualquer quantia.



Figura 10 – Tela de Definição de Juros



The image shows a software dialog box titled "Entrada" (Input) with a red close button in the top right corner. Inside the dialog, there is a circular icon with a question mark on the left. To the right of the icon, the text "Taxa de Juros do sistema: 6.0" is displayed. Below this text is a white rectangular input field with a blue border. At the bottom right of the dialog, there are two buttons: "OK" and "Cancelar" (Cancel).

É nessa tela que será definido o juro do sistema, não afeta compras antigas, apenas o cálculo do mesmo em um pagamento.

## 7 PROGRAMAÇÃO

Nos quadros 7, 8 e 9 será mostrado três métodos da parte de programação do software julgada mais trabalhosa e complexa, pertencentes a mesma classe.

Quadro 7 – Método updateVendas

```
public static void updateVendas(double valorPago, Cliente cliente, int parcelasPagas, java.sql.Date
novoVencimento, boolean cobrarJuros, boolean attTodas) {
    String sql = "SELECT idVenda,valorTotal,valorPago,quantiaParcelas,dataVencimento FROM
tryfatus.vendas WHERE Clientes_CPF='"+cliente.getCPF()+"' AND ativa=1 ORDER BY dataVencimento";
    try {
        PreparedStatement stmt = connection.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery();
        double a = valorPago;
        double valJuros=0;
        while(rs.next()) {
            if (a>0) {
                if (rs.getDate("dataVencimento").before(DateConvert.LocalToUtil(LocalDate.now())) {
                    valJuros = JurosCalc.Juros((rs.getDouble("valorTotal")-
rs.getDouble("valorPago"))/rs.getInt("quantiaParcelas"), JurosDAO.pegarJuros(),
rs.getDate("dataVencimento"));
                }
                double aberto=rs.getDouble("valorTotal")+valJuros-rs.getDouble("valorPago");
                if (aberto>=a) {
                    updateVenda(a, rs.getInt("idVenda"), rs.getDouble("valorTotal"), rs.getDouble("valorPago"),
rs.getInt("quantiaParcelas"), parcelasPagas, rs.getDate("dataVencimento"), novoVencimento, cobrarJuros,
attTodas);
                    a=0;
                }
                else if (aberto<a) {
                    updateVenda(aberto, rs.getInt("idVenda"), rs.getDouble("valorTotal"),
rs.getDouble("valorPago"), rs.getInt("quantiaParcelas"), parcelasPagas, rs.getDate("dataVencimento"),
novoVencimento, cobrarJuros, attTodas);
                    a=a-aberto;
                }
            }
            else if (attTodas==true){
                if (rs.getDate("dataVencimento").before(novoVencimento)) {
                    int par = rs.getInt("quantiaParcelas")-parcelasPagas;
                    if (par<1) {
                        par=1;
                    }
                    updateVencimento(rs.getInt("idVenda"), novoVencimento, par,
decimal(rs.getDouble("valorTotal")+valJuros));
                }
            }
            else {
                break;
            }
        }
        if (a>0) {
            JOptionPane.showMessageDialog(null, "Sobraram "+a+"R$ de troco!");
        }
        stmt.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

## Quadro 8 – Método updateVenda

```

public static void updateVenda(double pagoAgora, int idVenda, double valorTotal, double valorPago, int
quantiaParcelas, int parcelasPagas, Date dataVencimento, java.sql.Date dataNVencimento, boolean
cobrarJuros, boolean attTodas) {
    String sql = "UPDATE tryfatus.vendas SET valorTotal = ?, valorPago = ?, quantiaParcelas = ?,
dataVencimento = ?, ativa=? WHERE idVenda = ?";
    double pagAgora = pagoAgora;
    try {
        PreparedStatement stmt = connection.prepareStatement(sql);          double valorTot = valorTotal;
        double valJuros = 0;
        if (dataVencimento.before(DateConvert.LocalToUtil(LocalDate.now())) {
            valJuros = JurosCalc.Juros((valorTotal-valorPago)/quantiaParcelas, JurosDAO.pegarJuros(),
dataVencimento);
        }
        if (cobrarJuros==true) {
            valorTot=valorTot+valJuros;
        }
        stmt.setDouble(1, decimal(valorTot));
        double valPag = 0;
        if (pagoAgora>0) {
            if (pagAgora>valorTot) {
                valPag=valorTot;
                pagAgora=pagAgora-valorTot;
            }
            else if (pagAgora==valorTot) {
                valPag=valorTot;
                pagAgora=0;
            }
            else if (pagAgora<valorTot) {
                valPag=valorPago+pagAgora;
                pagAgora=0;
            }
        }
        } else {
            valPag=valorPago;
        }
        stmt.setDouble(2, decimal(valPag));
        if (pagoAgora>0 || attTodas==true) {
            if (quantiaParcelas-parcelasPagas<=0) {
                stmt.setInt(3, 1);
            } else {
                stmt.setInt(3, quantiaParcelas-parcelasPagas);
            }
        } else {
            stmt.setInt(3, quantiaParcelas);
        }
        if (pagAgora>0) {
            stmt.setDate(4, dataNVencimento);
        } else if (attTodas==true) {
            stmt.setDate(4, dataNVencimento);
        } else {
            stmt.setDate(4, dataVencimento);
        }

        //
        stmt.setInt(5, valPag>=valorTot ? 0 : 1);

        //
        stmt.setInt(6, idVenda);
        stmt.executeUpdate();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

```

### Quadro 9 – Método updateVencimento

```
public static void updateVencimento(int idVenda, java.sql.Date dataVencimento, int parcelas, double juros) {
    String sql = "UPDATE tryfatus.vendas SET dataVencimento = ?, quantiaParcelas = ?, valorTotal = ?
    WHERE idVenda = ?";
    try {
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setDate(1, dataVencimento);
        stmt.setInt(2, parcelas);
        stmt.setDouble(3, decimal(juros));
        stmt.setInt(4, idVenda);
        stmt.executeUpdate();
        stmt.close();
    } catch (SQLException ex) {
        Logger.getLogger(VendaDAO.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

## 7.1 FUNCIONAMENTO DA PROGRAMAÇÃO

Esses três métodos fazem parte de uma única ação: a de realizar pagamentos. Inicialmente, o método *updateVendas* irá se conectar ao banco, realizar as verificações necessárias e, por fim, buscar todas as comprar do cliente que estão ativas. Feito isso, dependendo da situação, irá utilizar ou o método *updateVenda* ou *updateVencimento*, sendo o segundo aplicável apenas quando a opção “Contar para todas” estiver habilitada.

O protótipo irá então realizar um laço entre as vendas, fazendo o pagamento de cada uma buscando o método *updateVenda* (Destacado em amarelo) e inserindo os valores. Feito isso, irá descontar da variável que contém o valor que está sendo pago o que foi deduzido, agora, e iniciará o mesmo processo em outra venda.

Quando o valor desta variável chegar em zero, irá verificar se a opção Contar para Todas está ativa, caso sim, irá utilizar o método *updateVencimento* (Também destacado em amarelo) para atualizar a data de vencimento, quantia de parcelas e juros das vendas (Quando aplicável) restantes.

Caso a opção não esteja ativa, encerrará o processo de pagamentos e retornará uma mensagem de sucesso. Se sobrar algum valor no pagamento (Quando não há comprar para serem pagas) irá retornar um aviso, com o valor restante de troco.

## 8 CONCLUSÃO

Dessa forma, atualmente há um alto consumo do papel, várias empresas utilizam folhas para registrar seus dados, contratos, registros, entre outros. Além da desvantagem de o mesmo ocupar espaço no estabelecimento, seu preço e custeio não é assim tão barato. Sendo isso, de grande risco, pois pode ocasionar a perda de todas essas coisas de imensa importância em catástrofes naturais, como uma enchente ou muitas vezes simplesmente por conta da perda das folhas.

Baseando-se nessas informações foi realizada a produção do protótipo. Seu principal objetivo é agilizar os processos de venda na loja TryFatus. De acordo com as necessidades citadas pela proprietária em entrevista, após foi realizada a parte de engenharia de requisitos.

Sabemos que a engenharia de requisitos é essencial para o desenvolvimento de um software com qualidade, e ela que vai especificar o que o software fará e de que maneira cada requisito acontecerá. A engenharia de requisitos engloba toda a documentação de software que irá auxiliar o programador, quando o mesmo estiver desenvolvendo. Ainda podemos ressaltar que a mesma tem como meta descobrir, analisar e documentar todas as fases de desenvolvimento de software.

Para o desenvolvimento do software foi utilizado a linguagem de programação Java e gerenciamento de banco de dados SQL nas plataformas NetBeans e MySQL, ambas selecionadas por serem as de maior domínio dos membros de nossa equipe, por já terem sido utilizadas nas aulas práticas e teóricas, assim já tendo um maior conhecimento das mesmas.

É importante ressaltar que as maiores dificuldades encontradas foram em relação a programação. O cadastro de pagamento do protótipo demandou muito tempo e pesquisa, ao final da codificação pudemos notar que a classe tinha mais de cento e trinta linhas.

Após desenvolvimento do protótipo em conversa com a proprietária mostramos o mesmo e pedimos para a mesma avaliar seu layout e suas funções.

De acordo com Lúcia, o protótipo é de fácil entendimento e manuseio, as funções são úteis e práticas. A mesma achou que as telas são discretas e intuitivas, além de serem bonitas. De maneira geral, superou as expectativas.

Ela concluiu que o protótipo chegou ao ideal e poderá ser de maior aproveitamento automatizado quando houver funções para impressão de carnês, sendo esta necessidade um trabalho futuro a ser realizado.

## 8.1 TRABALHOS FUTUROS

Buscando aprimorar melhor os conhecimentos obtidos, fica como sugestão para trabalhos futuros os seguintes:

- Formatação de valores em campos de texto melhorada;
- Adicionar opções para impressão de registro, carnês e recibo;
- Otimizar a estrutura do código;
- Criar opção para realizar backup do banco de dados facilmente;
- Adicionar sistema de login para utilização do sistema.
- Adicionar opção de pagamento e gerenciamento de uma única venda.

## REFERÊNCIAS

ALVES, Gustavo Furtado. **A história dos bancos de dados** Disponível em: <<https://dicasdeprogramacao.com.br/a-historia-dos-bancos-de-dados/>> Acesso em: 02 set. 2019.

ALVES, Gustavo Furtado. **Você precisa saber o que é SQL!** Disponível em: <<https://dicasdeprogramacao.com.br/o-que-e-sql/>> Acesso em: 02 set. 2019.

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**. 2. ed. Rio de Janeiro: Elsevier, 2007.

ESPINDOLA, Rodrigo; MAJDENBAUM, Azriel; AUDY, Jorge. **Uma análise crítica dos desafios para engenharia de requisitos em manutenção de software**. Disponível em: <[http://www.metodoiron.com.br/iron/wp-content/uploads/2014/03/Artigo\\_EngReq\\_ManutencaoSoftware.pdf](http://www.metodoiron.com.br/iron/wp-content/uploads/2014/03/Artigo_EngReq_ManutencaoSoftware.pdf)> Acesso em: 09 out. 2019.

FILGUEIRAS, Felipe. **Java – A Origem** Disponível em: <<https://tableless.com.br/java-origem/>> Acesso em: 10 out. 2019.

FELIZARDO, José. **História da UML** Disponível em: <<https://www.projetodiario.net.br/historia-da-uml>> Acesso em: 09 out. 2019.

GASPAROTTO, Henrique Machado. **Os 4 Pilares da Programação Orientada a Objetos** Disponível em: <<https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>> Acesso em: 02 set. 2019.

GRUPO TADS. **Diagramas de Caso de Uso** Disponível em: <<http://grupotads2014.blogspot.com/2014/04/diagramas-de-casos-de-uso-o-modelo-de.html>> Acesso em: 10 out. 2019.

KOLB, Juliana Jenny. **Documentação de Casos de Uso** Disponível em: <<https://jkolb.com.br/documentacao-de-casos-de-uso/>> Acesso em: 09 set. 2019.

MACORATTI, José Carlos. **Modelando Sistemas em UML – Casos de Uso** Disponível em: <[http://www.macoratti.net/net\\_uml2.htm](http://www.macoratti.net/net_uml2.htm)> Acesso em: 02 set. 2019.

MACORATTI, José Carlos. **UML - Diagrama de Classes e objetos** Disponível em: <[http://www.macoratti.net/net\\_uml1.htm](http://www.macoratti.net/net_uml1.htm)> Acesso em: 03 dez. 2019.

MARTINEZ, Mariana. **UML** Disponível em: <<https://www.infoescola.com/engenharia-de-software/uml/>> Acesso em: 23 ago. 2019.

MIRANDA, Rosenildo Pinheiro. **Programação Orientada a Objetos** Disponível em: <<https://meuartigo.brasilecola.uol.com.br/informatica/programacao-orientada-objetos.htm>> Acesso em: 02 set. 2019.

M3 MÍDIA, O que é NetBeans? Disponível em: <[https://www.oficinadanet.com.br/artigo/1061/o\\_que\\_e\\_o\\_netbeans](https://www.oficinadanet.com.br/artigo/1061/o_que_e_o_netbeans)> Acesso em: 02 set. 2019.

ORACLE, History of MySQL Disponível em: <<https://dev.mysql.com/doc/refman/8.0/en/history.html>> Acesso em: 02 set. 2019.

ORACLE, MySQL Disponível em: <<https://www.oracle.com/technetwork/database/mysql/index.html>> Acesso em: 02 set. 2019.

PACIEVITCH, Yuri. **História do Java** Disponível em: <<https://www.infoescola.com/informatica/historia-do-java/>> Acesso em: 02 set. 2019.

RIBEIRO, Leandro. **O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML** Disponível em: <<https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>> Acesso em: 23 ago. 2019.

SANTANA, Eduardo Felipe. **Principais Conceitos da Programação Orientada a Objetos** Disponível em: <<https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objetos/32285>> Acesso em: 02 set. 2019.



SCHIMIGUEL, Juliano. **Gerenciamento de Banco de Dados: Análise Comparativa de SGBD'S** Disponível em: <<https://www.devmedia.com.br/gerenciamento-de-banco-de-dados-analise-comparativa-de-sgbd-s/30788>> Acesso em: 02 set. 2019.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

VARGAS, Thânia Clair. **A história da UML e seus diagramas** Disponível em: <[https://projetos.inf.ufsc.br/arquivos\\_projetos/projeto\\_721/artigo.tcc.pdf](https://projetos.inf.ufsc.br/arquivos_projetos/projeto_721/artigo.tcc.pdf)> Acesso em: 23 ago. 2019.

VENTURA, Plínio. **Caso de Uso – Include, Extend e Generalização** Disponível em: <<https://www.ateomomento.com.br/caso-de-uso-include-extend-e-generalizacao/>> Acesso em: 08 set. 2019.

XAVIER, Laís. **Integração de Requisitos Não-Funcionais a processos de negócio: Integrando BPMN e RFN**. Recife, 2009. Disponível em: <<https://repositorio.ufpe.br/bitstream/123456789/13961/1/MASTER%20THESIS%20VERSAO%20BIBLIOTECA.pdf>> Acesso em: 9 out. 2019.