UNIVERSIDADE FEDERAL DE OURO PRETO
Faculdade de Ciência da Computação

Trabalho Prático 01 POO

Daniel Monteiro Valério
Gabriel Mace dos Santos Ferreira
Marcus Vinícius Souza Fernandes

Setembro de 2020

Trabalho Prático 01

Daniel Monteiro Valério
Gabriel Mace dos Santos Ferreira
Marcus Vinícius Souza Fernandes

Trabalho Prático do Curso de Ciência da Computação da Universidade Federal de Ouro Preto.

INTRODUÇÃO

Inicialmente foram idealizadas as classes para o funcionamento do sistema de operações de uma oficina mecânica, foi feita uma análise externa-interna, ou seja, foram identificando os agentes específicos (Administrador, Mecânico, Vendedor, Cliente e Carro) com seus respectivos elementos (Item, mão de obra, serviço e ordem), para assim identificar os elementos em comum e dessa forma criar as classes bases das quais as demais iriam herdar elementos.

DESCRIÇÃO DA ARQUITETURA E IMPLEMENTAÇÃO

O grupo construiu o código de forma a simular o menu de uma oficina, dessa forma foram utilizadas funções para apresentar diferentes opções/menus para o usuário de acordo com sua função dentro da loja (Admin, Mecânico, Vendedor),possibilitando uma melhor modularização do código e tornando os menus mais específicos para o usuário logado no momento.

-Pessoa: A classe base para todos os agentes no código, responsável por armazenar os dados pessoais e passá-los para seus herdeiros.

-Cliente: A necessária para solicitar uma ordem, possui uma lista de carros, visto que um cliente pode possuir mais de um carro.

-Carro: A classe carro contém três variáveis string para armazenar o modelo,placa e cor do veículo, respectivamente, ela também possui uma variável int para armazenar seu id, visto que se encontrar em uma lista, portanto são necessários identificadores únicos. Além disso contém uma lista de ordens para armazenar as diversas ordens realizadas no veículo, funcionando como um histórico.

-Funcionário: A classe base de todos os trabalhadores da oficina, responsável por herdar os dados pessoais da classe Pessoa e criar os itens básicos para o cadastro de um funcionário no sistema da oficina.

-Admin: Classe responsável por gerir os demais trabalhadores, ele deve armazenar todos os funcionários da oficina, sendo separados por funcionalidade (Mecânico e Vendedor), ademais possui métodos para adicionar um funcionário, alterá-lo ou excluí-lo.

-Vendedor: Classe responsável pelo gerenciamento das ordens, clientes e seus respectivos carros, ela possui os dados herdados da classe funcionário, no entanto seu campo user é único sendo formado pela palavra “vende” seguido do Id único do vendedor. A classe vendedor possui métodos para criação, alteração e exclusão de clientes e ordens.

-Mecânico: A classe responsável por adicionar os serviços realizados as ordens que possuam aprovação do cliente e ainda não tenham sido concluídas.O Mecânico deve selecionar uma ordem para um n número de serviços realizados, acrescentando a mão de obra utilizada e custo, bem como os itens usados. De forma análoga ao vendedor seu user é único contendo o prefixo “mecan” seguido do Id do mecânico.

-Ordem: A ordem é criada pelo vendedor, após a consulta com o cliente, ela contém strings que irão definir seu tipo(Orçamento ou não), a motivação para ordem , uma descrição do problema e o cpf do cliente que solicitou a ordem, bem como uma variável double para armazenar o custo dessa, duas variáveis bool definir se a ordem foi concluída ou o cliente à aprovou. Finalmente ela contém uma lista de

serviços, visto que é uma mesma ordem podem ser solicitados diferentes serviços(Troca de pneus e pintura do carro).

-Serviço: O serviço deve conter uma variável mão de obra para armazenar o custo da mão de obra utilizada, além disso contém uma lista de itens, visto que um mesmo serviço pode utilizar diversos itens, a classe contém uma variável int para armazenar seu código, caso seja necessário retirar um serviço da lista contida em Ordens.

-Mão de Obra: Classe utilizada para compor os campos da variável serviço, possui uma variável string para armazenar o seu "nome" e uma variável double para armazenar seu custo.

-Item: Classe utilizada para compor um dos campos da variável serviço, possui uma variável string para nomear o item utilizado, duas variáveis int para armazenar a quantidade de itens utilizados bem como o código do item respectivamente, além disso contém uma variável double para armazenar o valor total dos itens.

DIAGRAMA UML

O diagrama UML foi criado na plataforma Lucidchart, um website com extensões especializadas. Para uma análise precisa, com uma boa qualidade de leitura, inserimos um arquivo pdf junto ao arquivo do relatório para que assim seja possível realizar este passo com cautela.

DECISÕES DE PROJETO

As decisões para o trabalho prático tangente ao software da oficina mecânica foram tomadas em totalidade de forma conjunta e democrática. Abordamos a estratégia de formalizar o escopo do projeto realizando o mapeamento das camadas externas e bases inicialmente, com elas bem definidas e compreendidas partimos para uma análise mais específica, caminhando internamente nas ramificações, tipos, relações e propriedades.

As classes foram baseadas nos principais agentes identificados nas instruções oferecidas pelo professor, a seguir foram construídos os itens referenciados e por conseguinte seus derivados. A criação da classe ordem provou-se desafiadora, pois poderiam haver diferentes serviços realizados em uma mesma mesma ordem, portanto foi criada uma classe serviço que armazenaria o custo de mão de obra envolvido com tal ação e os itens utilizados, ademais essa classe é inserida na ordem, por meio de listas, o que nos permitiu solucionar o problema previamente exposto.

As tarefas foram bem divididas e executadas seguindo um modelo de metodologia ágil, trabalhávamos de forma paralela em funcionalidades em comum, concluindo uma seção por vez e considerando como concluída após compreendida, integrada e testada.

RECURSOS DE LINGUAGEM

A escolha dos recursos utilizados envolveu uma cautelosa ponderação, visto que sua integração ao código existente deveria torná-lo mais funcional, compreensivo e compacto. Pensando nisso foram utilizados as funcionalidades:

- **Bibliotecas:**

  <string> *: A biblioteca string foi utilizada para armazenar os dados de nomeação das diferentes variáveis, visando suprir a necessidade de um vetor de caracteres, no entanto sem a necessidade da definição prévia do número mínimo de caracteres .*

  <unistd> *: A biblioteca unistd foi utilizada devido a necessidade da implementação de um temporizador para adicionar um delay meio a execução de comandos, com a utilização do comando system("clear // cls") para manter o terminal limpo tornou-se necessário a presença da mesma para evidenciar as mensagens de êxito previamente a limpeza das informações em tela.*

  <iostream> *: A biblioteca iostream foi utilizada com o objetivo de expor e receber diferentes tipos de dados para a criação de distintas classes e/ou apresentar os campos dessas.*

  <list> *: A biblioteca list foi utilizada com o intuito de armazenar grandes quantidades de estruturas de dados, visto que algumas classes são quase exclusivamente compostas por diferentes classes, por vezes sendo necessário armazenar um número dinâmico dessas, por exemplo a previamente citada classe Ordem.*

- **Recursos externos:**

  Makefile : O Makefile foi utilizado de forma a facilitar a compilação, visto que a uma grande quantidade de arquivos seria extremamente demorado a compilação manual de cada um, ademais será utilizado um arquivo de texto para preencher os dados iniciais e tornar a execução do programa mais ágil.

- **Boas Práticas:**

  Indentação: Uma boa indentação é essencial para garantir uma boa legibilidade no código.

  Comentários: Comentários são de extrema importância para assegurar a compressão das funções e passos desenvolvidos.

  Nomenclatura: Seguir padrões de nomes de variáveis e funções também garantem uma excelente compressão e legibilidade do código.

INSTRUÇÕES DE COMPILAÇÃO

Como relatado anteriormente, o programa é constituído também por um recurso externo para agilizar a compilação, o Makefile. Na imagem abaixo se encontra a configuração do arquivo, ele basicamente lista todos os arquivos e comandos de execução seguindo sua própria estrutura.

Codigo > Makefile

```
all: ./O/Pessoa.o ./O/Administrador.o ./O/Vendedor.o ./O/Carro.o ./O/Cliente.o ./O/Funcionario.o ./O/Item.o ./O/MaoDeObra.o ./O/Mecanico.o ./O/Ordem.o ./O/Servico.o ./O/MenuAdmin.o ./O/Menu
    g++ -o ./O/main main.cpp ./O/Pessoa.o ./O/Administrador.o ./O/Vendedor.o ./O/Carro.o ./O/Cliente.o ./O/Funcionario.o ./O/Item.o ./O/MaoDeObra.o ./O/Mecanico.o ./O/Ordem.o ./O/Servico.o

./O/Pessoa.o: ./Cpp/Pessoa.cpp
    g++ -o ./O/Pessoa.o -c ./Cpp/Pessoa.cpp

./O/Ordem.o: ./Cpp/Ordem.cpp
    g++ -o ./O/Ordem.o -c ./Cpp/Ordem.cpp

./O/Administrador.o: ./Cpp/Administrador.cpp
    g++ -o ./O/Administrador.o -c ./Cpp/Administrador.cpp

./O/Vendedor.o: ./Cpp/Vendedor.cpp
    g++ -o ./O/Vendedor.o -c ./Cpp/Vendedor.cpp

./O/Carro.o: ./Cpp/Carro.cpp
    g++ -o ./O/Carro.o -c ./Cpp/Carro.cpp

./O/Cliente.o: ./Cpp/Cliente.cpp
    g++ -o ./O/Cliente.o -c ./Cpp/Cliente.cpp

./O/Funcionario.o: ./Cpp/Funcionario.cpp
    g++ -o ./O/Funcionario.o -c ./Cpp/Funcionario.cpp

./O/Item.o: ./Cpp/Item.cpp
    g++ -o ./O/Item.o -c ./Cpp/Item.cpp

./O/MaoDeObra.o: ./Cpp/MaoDeObra.cpp
    g++ -o ./O/MaoDeObra.o -c ./Cpp/MaoDeObra.cpp

./O/Mecanico.o: ./Cpp/Mecanico.cpp
    g++ -o ./O/Mecanico.o -c ./Cpp/Mecanico.cpp

./O/Servico.o: ./Cpp/Servico.cpp
    g++ -o ./O/Servico.o -c ./Cpp/Servico.cpp

./O/MenuAdmin.o: ./Cpp/MenuAdmin.cpp
    g++ -o ./O/MenuAdmin.o -c ./Cpp/MenuAdmin.cpp

./O/MenuMeca.o: ./Cpp/MenuMeca.cpp
    g++ -o ./O/MenuMeca.o -c ./Cpp/MenuMeca.cpp

./O/MenuVend.o: ./Cpp/MenuVend.cpp
    g++ -o ./O/MenuVend.o -c ./Cpp/MenuVend.cpp

clean:
    rm -f ./O/*.o ./O/main
```

Faz-se essencial realizar a compilação do código executando inicialmente o comando *"make clean"* para limpar os resíduos dos arquivos *".o"* gerados e em seguida executar o comando *"make all"* para compilar o código passando por toda a estrutura listada no próprio Makefile e por fim inserir o comando *"make run"* para executar o programa. A imagem a seguir é um exemplo do terminal após a execução destes passos.

```
marcus@Fernandes-AcerVX77:/mnt/d/Área de Trabalho/Marcus/UFOP/University/Matérias 3 período PLE/POO/Trabalhos Praticos/TP01-POO/Codigo$ make clean
rm -f ./O/*.o ./O/main
marcus@Fernandes-AcerVX77:/mnt/d/Área de Trabalho/Marcus/UFOP/University/Matérias 3 período PLE/POO/Trabalhos Praticos/TP01-POO/Codigo$ make all
g++ -o ./O/Pessoa.o -c ./Cpp/Pessoa.cpp
g++ -o ./O/Administrador.o -c ./Cpp/Administrador.cpp
g++ -o ./O/Vendedor.o -c ./Cpp/Vendedor.cpp
g++ -o ./O/Carro.o -c ./Cpp/Carro.cpp
g++ -o ./O/Cliente.o -c ./Cpp/Cliente.cpp
g++ -o ./O/Funcionario.o -c ./Cpp/Funcionario.cpp
g++ -o ./O/Item.o -c ./Cpp/Item.cpp
g++ -o ./O/MaoDeObra.o -c ./Cpp/MaoDeObra.cpp
g++ -o ./O/Mecanico.o -c ./Cpp/Mecanico.cpp
g++ -o ./O/Ordem.o -c ./Cpp/Ordem.cpp
g++ -o ./O/Servico.o -c ./Cpp/Servico.cpp
g++ -o ./O/MenuAdmin.o -c ./Cpp/MenuAdmin.cpp
g++ -o ./O/MenuMeca.o -c ./Cpp/MenuMeca.cpp
g++ -o ./O/MenuVend.o -c ./Cpp/MenuVend.cpp
g++ -o ./O/main main.cpp ./O/Pessoa.o ./O/Administrador.o ./O/Vendedor.o ./O/Carro.o ./O/Cliente.o ./O/Funcionario.o ./O/Item.o ./O/MaoDeObra.o ./O/Mecanico.o ./O/
Ordem.o ./O/Servico.o ./O/MenuAdmin.o ./O/MenuMeca.o ./O/MenuVend.o -Wall
marcus@Fernandes-AcerVX77:/mnt/d/Área de Trabalho/Marcus/UFOP/University/Matérias 3 período PLE/POO/Trabalhos Praticos/TP01-POO/Codigo$ make run
```

Com o programa em execução, de início é evidenciado uma tela para que o usuário possa fazer login no sistema. Como ainda não há vendedores e mecânicos cadastrados, faz-se necessário o administrador realizar seu acesso com os dados padrões (admin/admin) e cadastrar estes funcionários para que assim as funcionalidades do sistema possam ser executadas.

```
Sistema ativado.

Insira seus dados de acesso:
User  - admin
Senha - admin
```

O sistema é composto por vários menus de ações, o exemplo a seguir é tangente ao menu do administrador da plataforma. Todos os menus são bem intuitivos e possuem comandos bem descritivos para que não gere dúvidas ao usuário que esteja interagindo com eles.

```
As ações possíveis para o Administrador são:


1.  Cadastrar um Vendedor
2.  Alterar dados de um Vendedor
3.  Selecionar um Vendedor
4.  Deletar um Vendedor
5.  Cadastrar um Mecanico
6.  Alterar dados de um Mecanico
7.  Selecionar um Mecanico
8.  Deletar um Mecanico
9.  Listar Vendedores
10. Listar Mecanicos
11. Logout

Ação desejada: 1
```

Sempre que algum usuário realiza o logout ele é redirecionado para a tela ilustrada abaixo, deixando como escolha desligar o sistema ou realizar login novamente, podendo ser com os dados de qualquer funcionário para prosseguir para seus respectivos menus onde possuem autonomia para realizar suas funções.

```
0. Para realizar login.
1. Para sair do programa.

Ação desejada:
```

Caso a ação desejada seja realmente sair do sistema, o mesmo é desligado e uma mensagem é mostrada em tela para que o usuário tenha clareza que o sistema está se encerrando.

```
Sistema desligado.

marcus@Fernandes-AcerVX77:/mnt/d/Área de Trabalho/Marcus/UFOP/University/Matérias 3 período PLE/POO/Trabalhos Praticos/TP01-POO/Codigo$
```

- Dados auxiliares para teste do programa:

1. Passo 1:

. **Efetuar login no sistema:** User - admin / Senha - admin
. **Cadastrar um vendedor:** Vendedores necessitam ter o prefixo vende seguido de 4 dígitos no user (ex: *vende1000*).
. **Cadastrar um mecânico:** Mecânicos necessitam ter o prefixo mecan seguido de 4 dígitos no user (ex: *mecan1000*).
. Qualquer outra ação se torna opcional, caso queira excluir algum funcionário, outro deverá ser criado para garantir que as demais funcionalidades possam ser executadas.
. **Realizar o logout:** Pressionar o comando 11 de acordo com o menu.

2. Passo 2:

. **Efetuar login no sistema:** 0
. **Introdução do user:** (ex: *vende1000*).
. **Introdução da senha: (**ex: *vende1000*).
. **Opção de cadastrar cliente:** 1 (É necessário criar um cliente para utilizar a opção 3).
. **Opção de gerar ordem de serviço:** 2 (É necessário criar uma ordem para utilizar as opções 4 e 5).
. **Opção de visualizar clientes**: 3 (É possível alterar os campos do cliente, mas você tem que digitar um cpf válido inicialmente).
. **Opção de visualizar ordens de serviço pendentes:** 4 (É possível alterar os campos da ordem, mas você tem que digitar um id de ordem válido).
. **Opção de visualizar ordens de serviço executada:** 5
. **Opção de sair do menu:** 6

3. Passo 3:

. **Efetuar login no sistema:** 0
. **Introdução do user:** (ex: *mecan1000*)
. **Introdução da senha: (**ex: *mecan1000*)
. **Opção no menu mecânico:** 1 (Necessário utilizar a opção 1 primeiro, visto que o mecânico é iniciado com uma ordem vazia)
. **Escolha da Ordem por meio de seu Id:** (ex: *1000*)
. **Retorno ao menu mecânico**
. **Opção no menu mecânico:** 2
. **Número de serviços efetuados no veículo:** (ex: *1*)
. **Mão de obra utilizada:** (ex: *Troca de pneu*)
. **Valor da mão de obra:** (ex: *14.50*)
. **Tipos de itens utilizados:** (ex: *1*)
. **Item utilizado**:(ex: *Rodas*)
. **Quantidade utilizada:** (ex: *4*)
. **Preço dos itens:** (ex: *14.50*)
. **Código do item:** (ex: *100*)
. **Saindo do menu mecânico:** 3