TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO DA UTFPR: *SISTEMA DE LAVACAR*

Rodrigo Alves Guerra, Gabriel Eugenio Brito, Caio Ormond Araujo
rodrigoa.guerra@hotmail.com, gabriel.eug2@gmail.com, caiooa@hotmail.com

Disciplina: **Técnicas de Programação** / S73 – Prof. Dr. Robson R. Linhares
Departamento Acadêmico de Informática – DAINF - Campus Curitiba
Universidade Tecnológica Federal do Paraná - UTFPR
Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

Resumo – A disciplina de Técnicas de Programação exige o desenvolvimento de um software, no formato de um sistema de informação, para fins de aprendizado de técnicas de programação orientada a objetos em Java/C++, incluindo uma introdução para técnicas de engenharia de software. Para tal, neste trabalho, escolheu-se o desenvolvimento de um sistema de um lavacar, onde o administrador pode gerenciar os seus clientes, os funcionários e os veículos que frequentam a empresa. O programa também faz uma estimativa de tempo de espera para que o cliente saiba quanto tempo falta para seu carro ser lavado. O administrador pode gerenciar pelo sistema os insumos que são gastos nas execuções dos serviços, podendo prever falta de produtos e reabastecer seu estoque.

Palavras-chave: Projeto final de Técnicas de Programação, Orientação a objeto, Lavacar, Diagrama de classes UML.

INTRODUÇÃO

Este relatório tem como fim propor um tema para o projeto a ser executado na disciplina de Técnicas de Programação, ministrada pelo Prof. Robson Ribeiro Linhares.

O tema escolhido consiste na implementação de um sistema projetado para automatizar partes do gerenciamento de um lavacar, feito sob o paradigma de orientação a objeto (OO).

O método a ser utilizado seria o ciclo clássico de Engenharia de Software de forma simplificada (i.e. definição dos requisitos, modelagem-projeto via diagramas em UML, implementação em uma linguagem orientada a objetos e testes pelo uso do software).

Nossa motivação para a escolha do tema vem do fato de um dos membros da equipe, Rodrigo, conhecer o dono de um lavacar e, durante suas conversas, compreender a necessidade de um software que gerencie o lugar. Começaremos explicando algumas funcionalidade do sistema.

EXPLICAÇÃO DO *SOFTWARE*

O software proposto é uma simplificação de um sistema de gerenciamento de um lavacar. Seu objetivo é tornar mais fácil a supervisão e utilização das informações pertinentes ao dono da empresa.

O nome, RG e o telefone de cada cliente que passar pelo lavacar será cadastrado no sistema, junto com o modelo, placa e outras informações de seu veículo. Também serão armazenados o nome, salário atual e outros dados de cada funcionário.

Será possível estimar o tempo de espera para que o próximo veículo seja atendido, baseado no número de veículos à sua frente e o tempo médio de um serviço para veículos de cada modelo (caminhonete, moto, carro, etc).

Também utilizando os dados relativos ao número e tipo dos veículos atendidos será feito uma estimativa dos insumos (esponjas, toalhas, sabão, etc) para auxiliar no controle de estoque. Com as informações relativas à saída de caixa (como salário de funcionários e despesas com materiais utilizados no serviço) e de entrada de caixa (como o valor cobrado pela lavacar para prestar os serviços), o sistema deve ser capaz de gerar relatórios diários, mensais e anuais contendo o total gasto, total de dinheiro bruto e o lucro da empresa.

Ao final de um serviço prestado será gerado um diagnóstico do veículo em questão. Ele deverá conter sugestões de serviços futuros como, por exemplo, polimento do veículo, e verificação do que é preciso fazer , por exemplo, trocar pneus ou trocar óleo.

DESENVOLVIMENTO DO *SOFTWARE* NA VERSÃO ORIENTADA A OBJETOS

Tabela 1. Lista de Requisitos do projeto

N.	Requisitos Funcionais	Peso atribuído	Situação
1	Cadastro de clientes	10	Requisito previsto inicialmente e realizado
2	Cadastro de veículos	10	Requisito previsto inicialmente e realizado
3	Efetuação de serviços	5	Requisito previsto inicialmente e realizado
4	Cadastro de funcionários	10	Requisito previsto inicialmente e realizado
5	Gerenciamento de insumos	10	Requisito previsto inicialmente e realizado
6	Estimativa de tempo de espera	10	Requisito previsto inicialmente e realizado
7	Diagnostico do veículo e serviços futuros.	10	Requisito previsto inicialmente e realizado
8	Geração de relatórios.	10	Requisito previsto inicialmente e realizado
9	Qualidade do relatório final (adequação ao documento de regras)	15	Requisito previsto inicialmente e realizado
10	Salvamento dos dados (cadastros, estoque de insumos)	10	Requisito previsto inicialmente e realizado

A classe mais importante do programa é a Company, que representa a empresa a qual utilizará o sistema. Esta classe conhece um objeto da classe UserInterface para fazer interações com o usuário em forma de mensagens e outro objeto da classe DAO para recuperar as informações da execução anterior.

Ela é composta por objetos referentes a serviços, insumos e finanças. Os métodos dos objetos citados possuem acesso somente às informações necessárias para seu funcionamento, mantendo assim um código coeso (os métodos fazem tudo o que devem fazer e somente o que devem fazer). Também é possível adicionar funcionários e clientes em ArrayLists, graças à relação de agregação fraca.

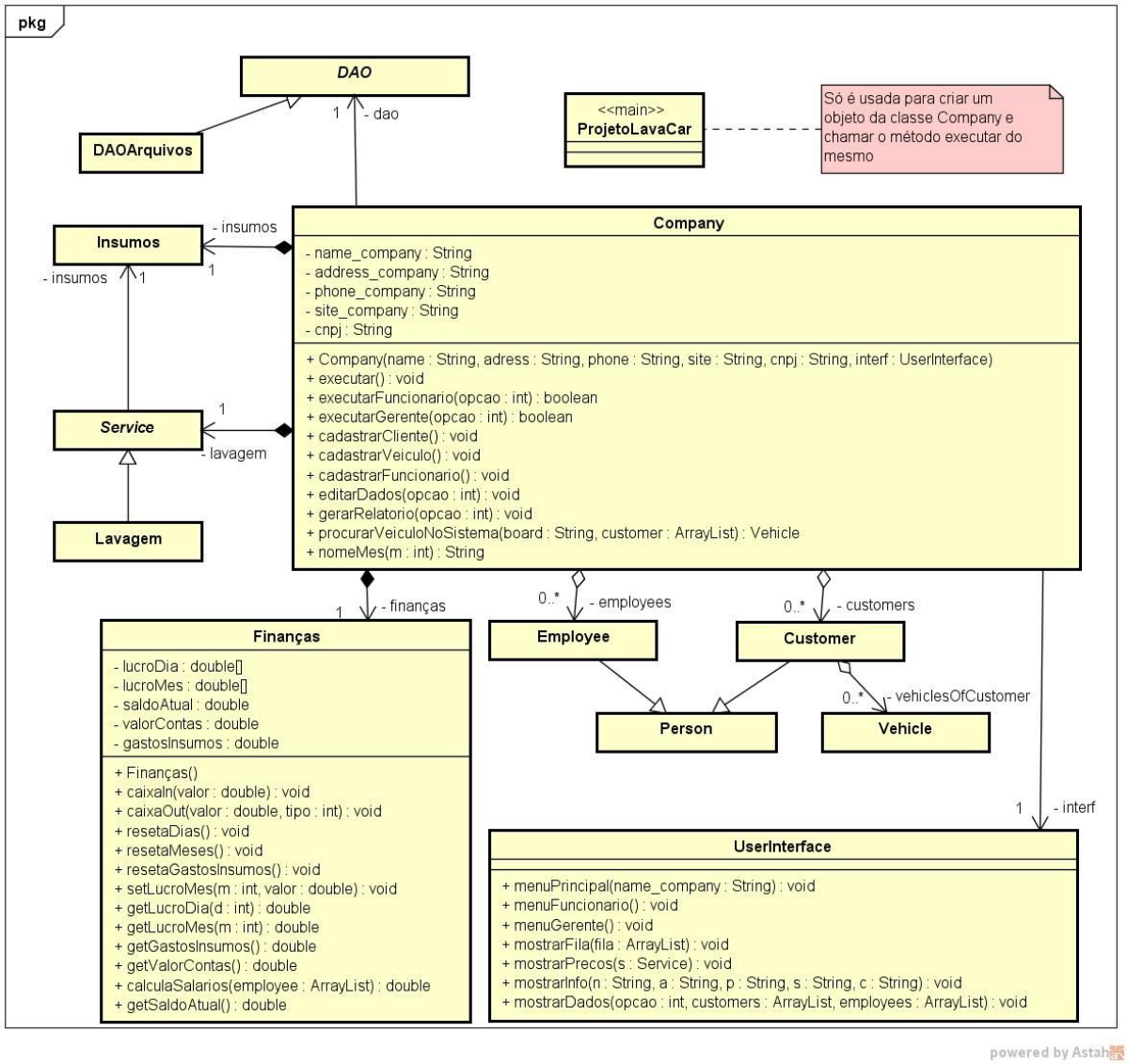

Figura 1. Diagrama de classes UML mostrando todas as relações

Optou-se por criar uma classe Person com atributos como nome, rg, cpf, data de nascimento, etc., e classes derivadas para Customer e Employee. Seria possível ainda criar uma classe chamada Gerente, também derivada de pessoa, afinal clientes, empregados e gerentes tem algumas informações em comum.

Cada cliente possui um ArrayList de veículos. Essa relação é de agregação, pois não se sabe quantos veículos o cliente terá no momento de cadastro (criar o todo não implica em criar as partes). Um funcionário pode adicionar (agregar) outros posteriormente.

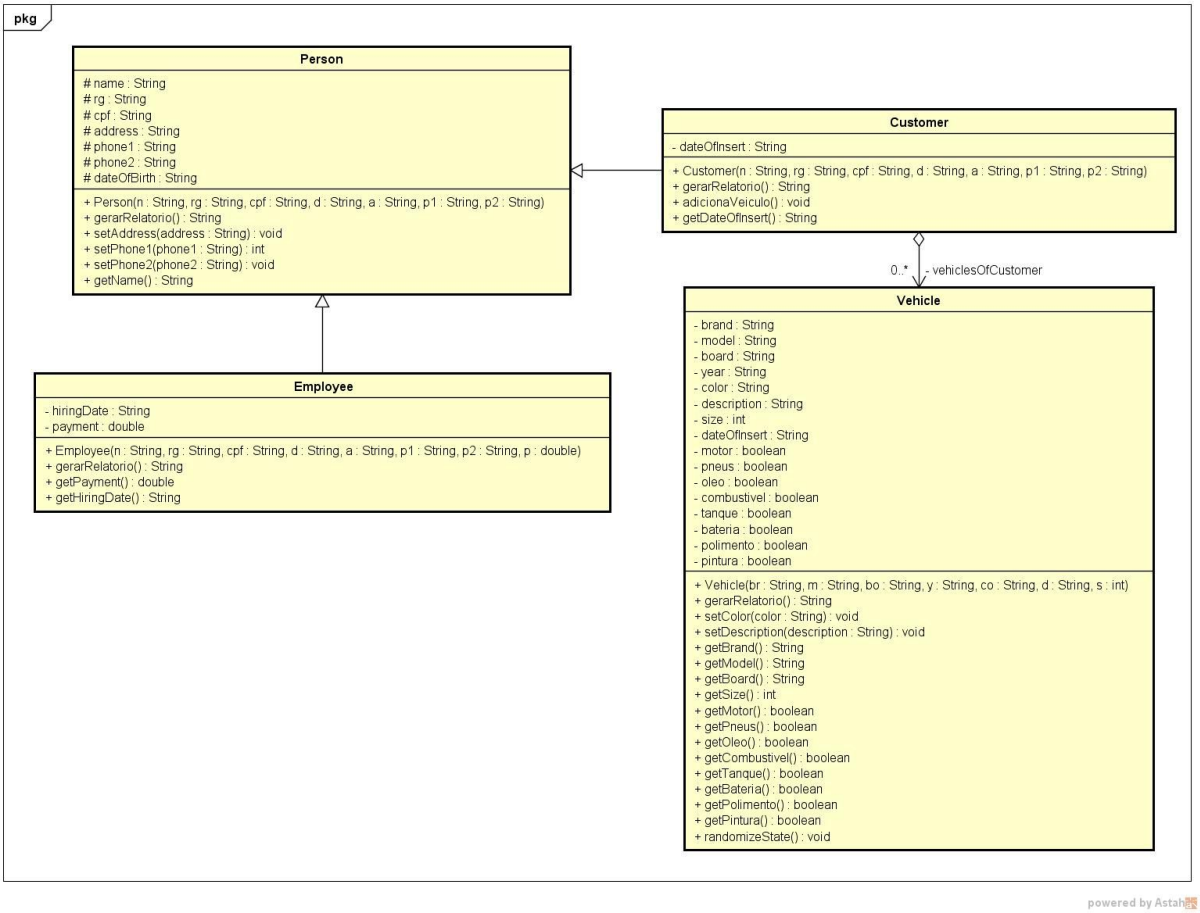

Figura 2. Diagrama detalhando as relações entre pessoas

Temos também a classe Service, cujos objetos possuem um preço para diferentes tamanhos de veículos e atributos para guardar o número de serviços realizados em cada dia e mês. Um objeto de Service também conhece um objeto da classe Insumos para poder descontar os produtos do estoque. Devido à essa relação de associação, seria possível lidar com vários estoques, um para cada serviço.

Esta classe é abstrata pois não podemos executar um serviço genérico. Foi implementado apenas uma classe derivada desta, Lavagem, com sua própria implementação de executar(), mas como o projeto foi desenvolvido pensando sempre no desacoplamento, para adicionar um novo serviço bastaria criar outra classe derivada de Service com sua própria definição do método executar(), evitando grandes mudanças no código.

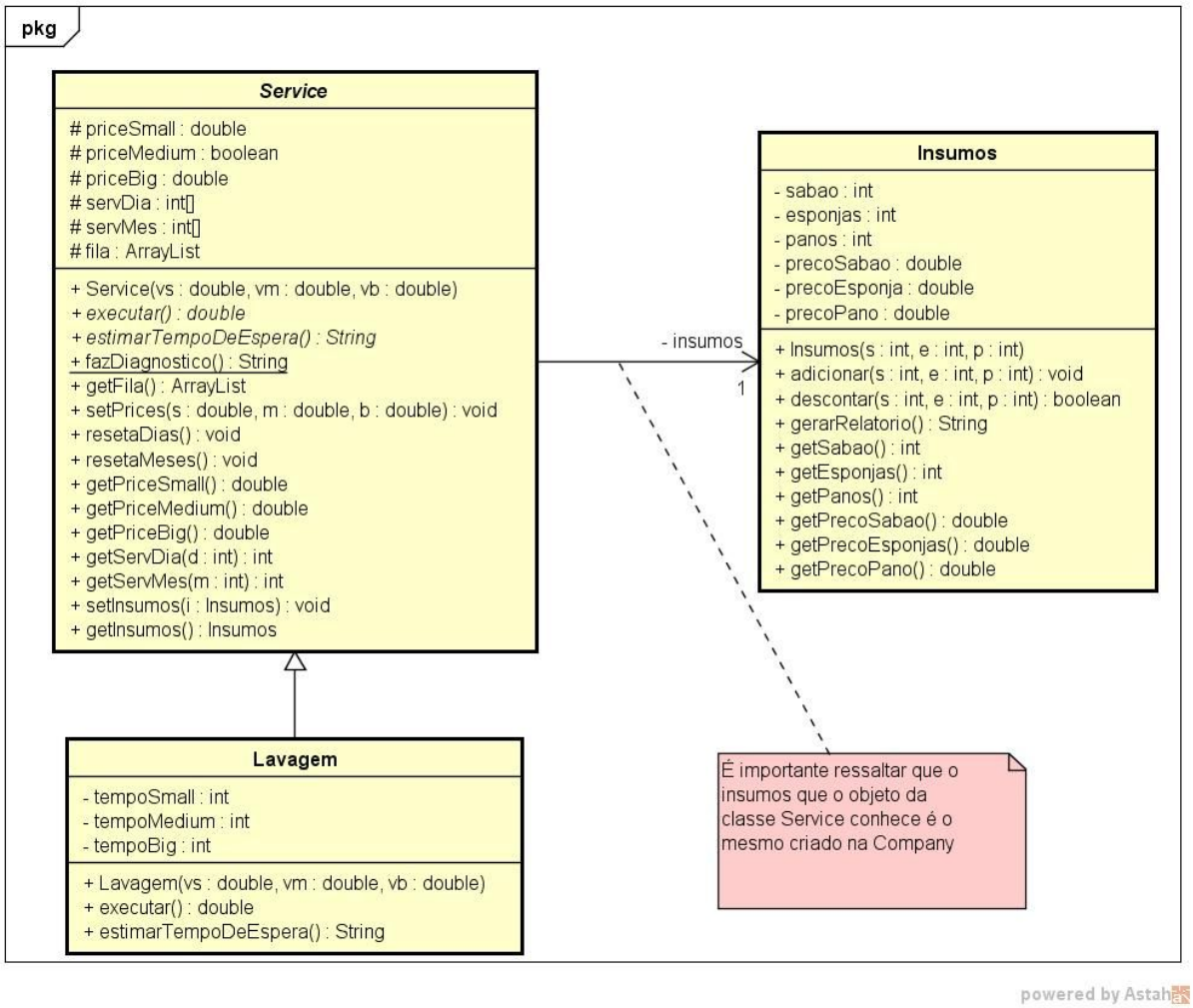

Figura 3. Diagrama de classes relacionadas aos serviços

O padrão de projeto DAO (*Data Access Object*) foi utilizado para a persistência de objetos no sistema. Como a fonte de leitura dos arquivos pode variar de um lavacar para outro, os métodos desta classe são abstratos. Foi implementado apenas uma classe derivada, DAOArquivos, mas se um lavacar decidisse usar banco de dados para salvar seus dados, só seria preciso criar outra classe, DAOBancoDeDados, com sua própria forma de salvar e recuperar informações.

No caso de arquivos, foi utilizado o recurso *Serializable*, que permite salvar objetos inteiros. Um método recebe o objeto (Finanças, Insumos, Service) ou ArrayList contendo os objetos (Customer, Employee) e salva em um arquivo de texto e outro, usado para recuperar as informações, recebe um objeto ou ArrayList vazio e o preenche com os armazenados no arquivo.

Seria possível criar uma classe genérica que diminuiria a quantidade de métodos da classe Arquivos, tornando-a mais enxuta, passar também o tipo do objeto a ser recuperado dos arquivos ou outra fonte qualquer.

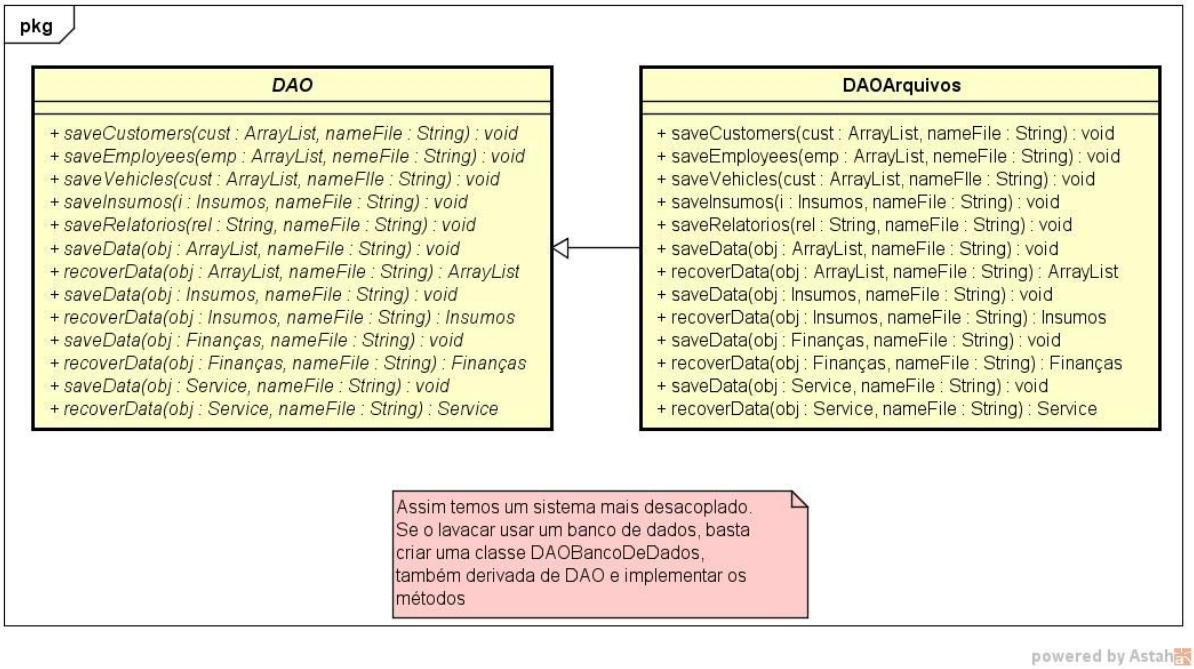

Figura 4. Diagrama de classes UML - DAO

TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Tabela 2. Lista de conceitos utilizados e não utilizados no trabalho

N.	Conceitos	Uso	Onde
1	**Elementares:**		
	- Classes, objetos,	Sim	No projeto
	- Atributos (privados), variáveis e constantes	Sim	Maioria das classes
	- Métodos (com e sem retorno).	Sim	Todas as classes
	- Métodos (com retorno const e parâmetro const).	Não	---
	- Construtores (sem/com parâmetros) e destrutores	Sim	Todas as classes
	- Classe Principal	Sim	SistemaLavaCar.java
2	**Relações de:**		
	- Associação	Sim	Company-DAO, Company-UserInterface e Service-Lavagem
	- Agregação (fraca)	Sim	Company-Employee, Company-Customer e Customer-Vehicle
	- Composição (agregação forte)	Sim	Company-Insumos, Company-Service e Company-Finanças
	- Herança elementar.	Sim	DAO-DAOArquivos, Service-Lavagem, Person-Customer, Person-Employee
	- Herança em diversos níveis	Não	---
	- Herança múltipla.	Não	---
3	**Ponteiros, generalizações e exceções**		
	- Operador this	Sim	Company.java, Person.java e Vehicle.java
	- Alocação de memória (new & delete)	Sim	Company.java
	- Gabaritos/Templates criada/adaptados pelos autores (e.g. Listas Encadeadas via Templates)	Não	---
	- Uso de Tratamento de Exceções	Sim	DAOArquivos
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Customer, Employee, Lavagem,

			DAOArquivos
	- Operadores	Não	---
	Persistência de Objetos		
	- Texto via Arquivos de Fluxo	Não	---
	- Binário	Sim	DAOArquivos
5	**Virtualidade:**		
	- Métodos Virtuais.	Sim	Todos os métodos do DAO, Service.executar() e Service.estimarTempoDeEspera()
	- Polimorfismo	Sim	Todos os métodos do DAOArquivos
	- Métodos Virtuais Puros / Classes Abstratas	Sim	DAO
	- Coesão e Desacoplamento	Sim	No projeto
6	**Engenharia de Software**		
	- Levantamento de Requisitos Textualmente e Tabelado	Sim	No projeto
	- Diagrama de Classes em UML	Sim	No projeto
	- Outros diagramas em UML	Não	---
7	**Biblioteca Gráfica**		
	- Funcionalidades Elementares	Não	---
	- Funcionalidades Avançadas como: - tratamento de colisões - duplo buffer	Não	---
	Interdisciplinaridades por meio da utilização de Conceitos de Matemática, Física, etc		
	- Ensino Médio (especificar quais Conceitos aqui)	Não	---
	- Ensino Superior (especificar quais Conceitos aqui)	Não	---
8	**Organizadores:**		
	Pacotes criados pelos autores.	Sim	No projeto
	Classes aninhadas.	Não	---
	Estáticos e String		
	Atributos estáticos e chamadas estáticas de métodos	Sim	Service.java
	A classe Pré-definida String ou equivalente.	Sim	Maioria das classes
9	**Standart Template Library (*STL*)**		
	Vector da STL (p/ objetos ou ponteiros de objetos de classes definidos pelos autores).	Não	---
	List da STL (p/ objetos ou ponteiros de objetos de classes definidos pelos autores).	Sim	Company.java
	Pilhas, Filas, Bifilas, Filas de Prioridade, Conjuntos, Multi-Conjuntos, Mapas ou Multi-Mapas.* **Obs.: Listar apenas os utilizados*	Não	---
10	**Uso de Conceito Avançado no tocante a Orientação a Objetos.**		
	Padrões de Projeto: DAO	Sim	No projeto

Tabela 3. Lista de Justificativas para Conceitos Utilizados e Não Utilizados no Trabalho

No.	**Conceitos**	**Situação**
1	**Elementares**	Classes, objetos, atributos, métodos e constantes e construtores foram, obviamente, usados, pois são a base da Orientação a Objeto.
2	**Relações**	Associação foi usada para a classe Company acessar os métodos do DAO. Composição e Agregação foram utilizadas para associar boa parte das classes à Company. Herança, juntamente com classes abstratas, foi usada principalmente para permitir um código mais desacoplado.
3	**Ponteiros, generalizações e exceções**	Alocação de memória foi usada para alocar objetos e alguns vetores. Exceções foram adicionadas somente na manipulação de arquivos.

4	**Sobrecarga e Persistência de Objetos**	Sobrecarga foi utilizada na manipulação de arquivos e devido à necessidade de gerar relatórios diferentes, caso fosse um cliente ou um empregado. Arquivos de texto foram usados para auxiliar na persistência de objetos.
5	**Virtualidade**	Métodos virtuais e polimorfismo foram utilizados para garantir a modularidade do programa, e como consequência melhorar a legibilidade e facilitar uma possível expansão/melhoria futura.
6	**Engenharia de Software**	Foi utilizado diagramas de classe UML para o desenvolvimento do sistema e uma melhor compreensão das interações entre os objetos.
7	**Biblioteca Gráfica**	Bibliotecas gráficas não foram utilizadas por não ter sido consideradas prioritárias.
8	**Organizadores, Estáticos e String**	Um método static foi usado pois fazia sentido que ele fosse chamado mesmo sem a criação de um objeto daquele tipo. A classe String foi utilizada pois permite fácil manipulação de informações como nome, data de cadastro e outras.
9	**Standart Template Library (*STL*)**	Foram utilizados ArrayLists pois não há como saber quantos clientes, empregados e veículos serão armazenados.
10	**Uso de Conceito Avançado**	O padrão de projetos DAO foi utilizado para aumentar o desacoplamento do sistema quanto à forma como os dados são lidos e gravados.

DISCUSSÃO E CONCLUSÕES

Durante o desenvolvimento do sistema, chegamos a conclusão de que embora a ideia inicial seja simples, a conceitualização e a implementação são bem complexas, pelo fato de haver muitas relações entre classes e objetos.

A escolha da linguagem Java foi unânime entre nós por ser um dos objetos de estudo centrais da disciplina. Sua sintaxe simples facilitou em muito o desenvolvimento de nosso trabalho. É interessante notar que desenvolvemos o trabalho enquanto o conteúdo ainda estava sendo exposto nas aulas. Frequentemente reescrevemos trechos de códigos implementados dias antes por um conteúdo novo dar uma solução melhor para nosso problema. Nesses momentos a linguagem Java mostrou a diferença que sua modularidade pode dar. Também nos familiarizamos com o diagrama de classes UML, que melhorou a visualização da sistemática do software e facilitou a elaboração do código fonte.

Optamos por não utilizar nenhuma interface gráfica, e sim nos concentrar nas funcionalidades do sistema e nas relações entre as classes e os objetos. Afinal, não adianta ter uma interface amigável se o programa não funcionar como deveria.

Estamos satisfeitos com o resultado que conseguimos. Tivemos a oportunidade de trabalhar em um grupo onde cada membro se esforçou para entender e colaborar no desenvolvimento do projeto. Não só isso, todos estavam presentes para tirar dúvidas uns dos outros, ajudar a resolver eventuais problemas no funcionamento ou discutir possíveis melhorias.

Os conteúdos foram fixados numa aplicação mais concreta e nos familiarizamos com métodos de gerenciar versões (GitHub) e/ou edições partindo de diversos membros (Google Drive). Dessa forma, o projeto acabou nos conferindo certa experiência não só na linguagem Java, nos mostrando as vantagens e limitações dela, mas em como pesquisar e desenvolver trabalhos em grupo.

DIVISÃO DO TRABALHO

Atividades.	Responsáveis
Levantamento de Requisitos	Todos
Diagramas de Classes	Caio
Programação em Java	Todos
Implementação de Cadastros	Todos
Implementação dos Serviços (lavar, tempo de espera, diagnósticos)	Todos
Implementação das Finanças	Gabriel
Implementação da Geração de Relatórios	Rodrigo e Gabriel
Implementação da Persistência dos Objetos	Rodrigo
Escrita do Trabalho	Todos
Revisão do Trabalho	Todos

REFERÊNCIAS

Uso de arquivos
http://www.botecodigital.info/java/gravando-e-lendo-dados-de-um-arquivo-em-java/
http://www.guj.com.br/t/gravar-arraylist-em-arquivo-txt/71441/8
http://www.javaprogressivo.net/2014/01/Arquivos-Files-em-Java.html
http://www.javaprogressivo.net/2013/02/Arquivos-Files-Tutorial-Java.html
http://www.devmedia.com.br/criando-e-gravando-dados-em-txt-com-java/23060