# Teste de Software
# "Testes Automatizados"

**$3^a$ Avaliação (Programa) - Valor final (normalizado): 4 pontos**

Flávio Augusto de Freitas

03 de outubro de 2019

## Especificação do Trabalho Prático

O trabalho prático da disciplina consiste em desenvolver o mesmo sistema computacional para solução do problema descrito abaixo nas duas linguagens de programação apresentadas durante o curso: Java e C++.

## 1 Descrição do problema

A padaria do senhor Esfi está tecnologicamente atrasada. O sistema de controle de conta é à moda antiga, papel e caneta. A padaria possui uma grande clientela local, pois o Sr. Esfi permite que os clientes anotem "na conta" os produtos que estão levando e efetuem o pagamento no final do mês (pagamento fiado). Ele consegue fazer isso pois também negocia com os fornecedores prazo para pagar suas próprias compras no início do mês seguinte. Acontece que a cada fim do mês é sempre o mesmo problema: o senhor Esfi tem dificuldades para calcular quanto tem a receber e quanto tem a pagar somente olhando as anotações em papel. Cansado desta situação, ele deseja um sistema para obter um melhor gerenciamento. O sistema deve fazer o cadastro de clientes e fornecedores, cadastro de produtos, contas a receber e a pagar, além de gerar relatórios. A padaria trabalha também com vendas a vista, que devem ser consideradas durante o fechamento de cada mês.

### 1.1 Controle de clientes e fornecedores

O Sr. Esfi mantém um cadastro de todos os fornecedores e também dos clientes que desejarem abrir uma conta (para pagamento fiado). Dos clientes que pagam a vista não é necessário registrar nada. Dos clientes que possuem conta deseja-se registrar código identificador, nome, endereço, telefone e desde quando o cliente compra na padaria (data do seu cadastro). Clientes podem ser de dois tipos: pessoas físicas ou pessoas jurídicas (empresas). De pessoas físicas deseja-se registrar o CPF, enquanto de pessoas jurídicas deseja-se registrar CNPJ e número de inscrição estadual. Dos fornecedores deseja-se registrar código identificador, nome, endereço, telefone, CNPJ e pessoa de contato. Fornecedores são obrigatoriamente pessoas jurídicas.

### 1.2 Cadastro de produtos

Os produtos são cadastrados no sistema com as seguintes informações: código, descrição, estoque mínimo, quantidade atual em estoque, valor de custo e percentual de lucro. O valor de venda de um produto é calculado com base no valor de custo e percentual de lucro: `<valor de venda> = <valor de custo> + <valor de custo> * <percentual de lucro>`.

### 1.3 Contas a pagar e a receber

Ao longo do mês, os funcionários da Padaria Esfi registram cada compra de produtos dos fornecedores e cada venda de produtos para clientes. Essa informação é fundamental para a geração dos relatórios que o Sr. Esfi precisa para melhor gerenciar sua padaria. Das compras efetuadas pela padaria junto aos fornecedores são registrados o número da nota fiscal, o fornecedor, a data da compra, o tipo de produto comprado e a quantidade. Para simplificar, assuma que para cada tipo de produto comprado é gerada uma entrada separada (mesmo que se refira ao mesmo número de nota fiscal). O valor pago pode ser calculado multiplicando o valor de custo do produto (vide seção 1.2) e a quantidade comprada. Das vendas efetuadas pela padaria aos clientes são registrados o cliente, a data da venda, o produto vendido, a quantidade vendida e o meio de pagamento. Novamente, o valor de cada venda pode ser calculado a partir o valor de venda do produto (vide seção 1.2) e a quantidade vendida. Os meios de pagamento aceito são: dinheiro, cheque, cartão de débito, cartão de crédito, *ticket* alimentação e fiado. Como no caso da compra, para cada produto vendido é criada uma entrada nos registros, mesmo que se repita o cliente, a data e o modo de pagamento. Como mencionado na seção 1.1, somente para pagamento fiado o cliente deve ser registrado. Para pagamentos a vista (dinheiro, cheque, cartão ou *ticket*) não é preciso registrar o cliente.

### 1.4 Relatórios

Após todos os dados registrados, o sistema deve gerar os seguintes relatórios:

- Total a pagar por fornecedor;
- Total a receber por cliente;
- Vendas e lucro por produto;
- Vendas e lucro por forma de pagamento;
- Estado do estoque, incluindo alertas de estoque abaixo do mínimo.

Tais relatórios são utilizados pelos funcionários para fazer e receber pagamentos ao final de cada mês. Na próxima seção serão detalhados os formatos dos arquivos de entrada e saída de dados.

## 2 Formatos de entrada e saída

Para uma transição mais lenta entre o sistema de anotação manual em cadernos e um sistema completo com banco de dados, foi combinado com o Sr. Esfi que os cadastros seriam feitos em planilhas eletrônicas. Os cadastros de clientes, fornecedores e produtos são independentes do

mês e vão sendo atualizados sempre que necessário. Já para compras e vendas são criadas novas planilhas no início de cada mês. Para o processamento destes dados e geração dos relatórios desejados, um funcionários da padaria irá exportar os dados das planilhas para arquivos de texto simples com valores separados por vírgulas, conhecido como CSV (*Comma Separated Values*). No entanto, para evitar conflito com representação de valores decimais (ex.: 3,9), os dados serão exportados utilizando ponto-e-vírgula como separados (ex.: Suco de laranja integral, 1 litro; 9,9 – representando um produto que custa R$ 9,90). Para facilitar a leitura dos relatórios produzidos pelo programa, será feita a importação dos dados dos relatórios do formato CSV para planilha eletrônica. Portanto, seu programa deve ser capaz de ler dados neste formato e gerar os relatórios também no mesmo formato. Esta seção descreve os dados que estarão presentes em cada um dos arquivos de entrada e os dados que devem estar presentes em cada um dos arquivos de saída (relatórios). Para saber como estes dados serão formatados, verifique os arquivos de exemplo disponibilizados juntamente com esta descrição. É muito importante que o programa siga os padrões de formatação prescritos, pois do contrário pode apresentar erro na leitura ou diferenças nos relatórios durante a correção automatizada dos trabalhos (vide seção 4). Note que tanto os arquivos de entrada quanto os de saída possuem linhas de título que devem ser levadas em consideração.

## 2.1 Entrada de dados

São cinco os arquivos de entrada de dados:

- Cadastro de clientes;
- Cadastro de fornecedores;
- Cadastro de produtos;
- Registro de compras (o que a padaria compra dos fornecedores);
- Registro de vendas (o que a padaria vende aos clientes).

Os nomes dos arquivos são especificados durante a execução do programa (vide seção 3). Abaixo encontra-se especificada a ordem que os dados devem aparecer em cada um destes arquivos:

### 2.1.1 Cadastro de clientes

```
<código>; <nome>; <endereço>; <telefone>; <data de cadastro>; <tipo de cliente>;
<cpf ou cnpj>; <número de inscrição estadual>
```

Código e número de inscrição estadual são numéricos (inteiros), nome, endereço, telefone e CPF/CNPJ podem ser lidos como texto. Tipo de cliente pode ser F (pessoa física) ou J (pessoa jurídica).

### 2.1.2 Cadastro de fornecedores

```
<código>; <nome>; <endereço>; <telefone>; <cnpj>; <pessoa de contato>
```

Código é numérico (inteiro), os demais campos podem ser lidos como texto.

### 2.1.3 Cadastro de produtos

`<código>; <descrição>; <estoque mínimo>; <estoque atual>; <valor de custo>; <percentual de lucro>`

Código, estoque mínimo, estoque atual e percentual de lucro são números inteiros. Valor de custo é um número decimal (valor em Reais). A descrição deve ser lida como texto.

### 2.1.4 Registro de compras

`<número da nota fiscal>; <código do fornecedor>; <data da compra>; <código do produto>; <quantidade>`

Número da nota fiscal, código do fornecedor, código do produto e quantidade são números inteiros.

### 2.1.5 Registro de vendas

`<código do cliente>; <data de venda>; <código do produto>; <quantidade>; <modo de pagamento>`

Código do cliente, código do produto e quantidade são números inteiros. Modo de pagamento pode ser $ (dinheiro), X (cheque), D (cartão de débito), C (cartão de crédito), T (*ticket*) ou F (fiado). O código do cliente estará preenchido somente se o modo de pagamento for F, nos demais casos estará vazio.

## 2.2 Saída de dados

Os relatórios gerados devem ser escritos em arquivos com os seguintes nomes:

Tabela 1: Relatórios

| **Relatório** | **Nome do Arquivo** |
|---|---|
| Total a pagar por fornecedor | `1-apagar.csv` |
| Total a receber por cliente | `2-areceber.csv` |
| Vendas e lucro por produto | `3-vendasprod.csv` |
| Vendas e lucro por forma de pagamento | `4-vendaspgto.csv` |
| Estado do estoque | `5-estoque.csv` |

Abaixo encontra-se especificada a ordem que os dados devem aparecer em cada um destes arquivos:

### 2.2.1 Total a pagar por fornecedor

`<nome do fornecedor>; <cnpj do fornecedor>; <pessoa de contato do fornecedor>; <telefone do fornecedor>; <valor total a pagar>`

Este relatório deve ser ordenado por nome do fornecedor.

### 2.2.2 Total a receber por cliente

`<nome do cliente>; <tipo do cliente>; <cpf/cnpj do cliente>; <telefone do cliente>; <data de cadastro do cliente>; <valor total a receber>`

Este relatório deve ser ordenado por nome do cliente.

### 2.2.3 Vendas e lucro por produto

`<código do produto>; <descrição do produto>; <receita bruta da venda do produto>; <lucro da venda do produto>`

Este relatório deve ser ordenado por lucro da venda do produto, decrescente. Em caso de empate, deve-se ordenar por código do produto.

### 2.2.4 Vendas e lucro por forma de pagamento

`<modo de pagamento>; <receita bruta deste modo de pagamento>; <lucro deste modo de pagamento>`

Este relatório também deve ser ordenado por lucro, decrescente. Em caso de empate, deve-se ordenar pela letra que representa o modo de pagamento.

### 2.2.5 Estado do estoque

`<código do produto>; <descrição do produto>; <quantidade em estoque após as vendas do mês>; <observações>`

Este relatório deve ser ordenado pela descrição do produto. No campo observações, deve constar o termo "`COMPRAR MAIS`" (sem as aspas) caso o produto, ao final do mês, tenha ficado com uma quantidade em estoque abaixo do limite mínimo para o produto.

## 2.3 Tratamento de exceções

Leitura de dados de arquivos, formatação etc., são fontes comuns de erros e exceções. Seu programa deve tratar **apenas** erros de entrada e saída de dados como, por exemplo, o arquivo especificado não existir ou o programa não ter permissão para ler ou escrever em um arquivo. Nestes casos, o programa deve imprimir apenas a mensagem "`Erro de I/O.`" e ser encerrado.

Isso significa que quaisquer outras situações de erro possíveis (ex.: valor formatado de forma incorreta nos arquivos de entrada, causando erros de *parsing* dos dados) devem ser ignoradas. Pode-se assumir que nos testes feitos durante a avaliação dos trabalhos estes outros erros nunca acontecerão.

## 2.4 Execução

Seu programa deve ser executado especificando os nomes dos arquivos de entrada como opções de linha de comando, especificadas a seguir:

- `-c <arquivo>`: cadastro de clientes;

- `-f <arquivo>`: cadastro de fornecedores;
- `-p <arquivo>`: cadastro de produtos;
- `-a <arquivo>`: registro de compras (o que a padaria compra dos fornecedores);
- `-v <arquivo>`: registro de vendas (o que a padaria vende aos clientes).

Apesar do uso do pacote *default* não ser recomendado, para efeito dos exemplos abaixo vamos supor que a classe do seu programa que possui o método `main()` chama-se `Main` e encontra-se no pacote *default*. Portanto, para executar seu programa lendo os arquivos `cliente.csv`, `fornecedores.csv`, `produtos.csv`, `compras.csv` e `vendas.csv` como arquivos de entrada, o comando seria:

```
java Main -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv
```

Além dos parâmetros acima, a versão Java do seu programa deve suportar dois parâmetros opcionais que estabelecem três modos de execução diferentes. O programa deve poder ser chamado das três formas, como a seguir:

- `java Main -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv`: quando não forem especificadas opções de execução, o programa deve ler os arquivos de entrada, gerar os relatórios e escrevê-los nos arquivos de saída, como descrito anteriormente;
- `java Main --read-only -c cliente.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv`: quando especificada a opção `--read-only`, o programa deve ler os arquivos de entrada, montar as estruturas de objetos em memória e serializar esta estrutura em um arquivo chamado `esfi.dat`. Os relatórios não devem ser gerados neste caso;
- `java Main --write-only`: quando especificada esta opção, o programa deve carregar os objetos serializados no arquivo `esfi.dat`, gerar os relatórios e escrevê-los nos arquivos de saída. Neste caso não há leitura de arquivo CSV.

Note que as opções de execução podem ser passadas em qualquer ordem. Portanto, o comando

```
java Main --read-only -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv
```

É equivalente a:

```
java Main -p produtos.csv -f fornecedores.csv -v vendas.csv -a compras.csv --read-only -c clientes.csv
```

Por fim, a versão C++ do programa não precisa implementar as opções `--read-only` e `--write-only`.

## 3 Condições de entrega

O trabalho deve ser feito obrigatoriamente em dupla e em duas versões: uma utilizando a linguagem Java, outra utilizando a linguagem C++. O primeiro deve ser entregue até o dia **31/10/2019** e o segundo até o dia **21/11/2019**, impreterivelmente. Alunos que não fizerem o trabalho em dupla sofrerão penalidade de 2 pontos na nota do trabalho. No caso de haver um número ímpar de alunos matriculados, o professor indicará um aluno que poderá, a seu

critério, fazer o trabalho sozinho ou juntar-se a uma dupla para formar um trio. As duplas para os trabalhos Java e C++ não precisam necessariamente ser as mesmas.

Dado que existem várias versões dos compiladores Java e C++, fica determinado o uso das versões instaladas nas máquinas do **Lab1** como versões de referência para o trabalho prático. Seu trabalho deve compilar e executar corretamente nas máquinas do **Lab1**. Além disso, os arquivos de código-fonte devem estar em arquivos codificados com Unicode (`UTF-8`) para evitar erros de compilação. O código-fonte e o arquivo de *build* (vide instruções do *script* de teste automático) de sua solução deverá ser compactado e enviado por *e-mail* (anexo ao *e-mail*) para o professor (`flavio.freitas@ifsudestemg.edu.br`). Serão aceitos trabalhos entregues até as 23:59 h da data limite. O assunto do *e-mail* deverá ser o seguinte:

```
TesteSoftware - TP1 - Trab1 - <Nomes dos alunos>
```

substituindo `<Nomes dos alunos>` pelos nomes dos alunos do grupo, separado por vírgula. Para a entrega do trabalho de C++, substitua `Trab1` por `Trab2`. Assim que possível, o professor responderá o *e-mail* com um *hash* MD5[1] do arquivo recebido. Para garantir que o arquivo foi recebido sem ser corrompido, gere o *hash* MD5 do arquivo que você enviou[2] e compare com o *hash* recebido na confirmação. Caso você não receba o *e-mail* de confirmação ou caso o valor do *hash* seja diferente, envie o trabalho novamente. Se o problema persistir, contate o professor o mais rápido possível. Dada a quantidade de trabalhos que devem ser avaliados, a correção dos trabalhos passará primeiro por um processo de testes automáticos e, em seguida, por uma avaliação subjetiva (mais detalhes na próxima seção). Para que os testes automáticos funcionem, o arquivo compactado enviado por *e-mail* deve estar no formato zip com o nome `trabalho.zip` e conter os arquivos fonte e demais arquivos necessários para a execução dos testes (arquivos de automação da compilação: `build.xml` (página 11) do Ant para projeto Java, `Makefile` para projeto C++). No SIGAA encontra-se disponível o *script* de teste automático. Espera-se que cada grupo execute-o antes do trabalho ser enviado, garantindo, assim, que o programa passe sem erros nos testes automáticos.

## 4 Critérios de avaliação

Os trabalhos serão avaliados em duas etapas:

- Avaliação objetiva (*veja próximo tópico*, com testes automáticos), valendo 10 pontos;
- Avaliação subjetiva, valendo 10 pontos.

A nota final do trabalho é a média aritmética simples entre as notas acima. Para a avaliação objetiva, todo trabalho possui inicialmente nota 10 e sofre penalidades nas situações descritas na tabela abaixo:

[1] Para saber mais sobre *Hash* MD5, visite sua página na *Wikipedia*: https://pt.wikipedia.org/wiki/MD5#Hashes_MD5

[2] Para gerar *hash* MD5 de arquivos no Linux, veja as instruções em http://roneymedice.com.br/2009/07/30/gerando-hash-md5-dos-arquivos-no-linux/. Já na página http://www.mundodoshackers.com.br/como-gerar-e-checar-hashs-md5 você encontra instruções também para Windows (porém os exemplos mostram geração de *hash* para *strings*, não para arquivos).

Tabela 2: Penalidades (em pontos)

| **Situação** | **Penalidade** |
|---|---|
| Não foi feito em dupla. | -2 |
| Não compilou nos testes automáticos. | -7 |
| Não compilou nos testes automáticos, mas foi possível corrigir manualmente (ex.: arquivos não codificados em `UTF-8`). | -2 |
| Não compilou nem manualmente. | -10 |
| Não gerou saídas nos testes automáticos. | -5 |
| Não gerou saídas nem manualmente. | -7 |
| Não implementou serialização. | -1 |
| Pequenas diferenças das saídas em relação ao teste automático (ex.: formatação, arredondamentos etc.). | -1 |
| Grandes diferenças das saídas em relação ao teste automático (ex.: valores sensivelmente diferentes). | -2 |
| Entrega fora do prazo. | -1 por dia |

Para avaliação subjetiva, novamente os trabalhos começam com nota 10 e perdem pontos (que variam de acordo com a avaliação feita pelo professor) caso não estejam bem escritos ou organizados. Critérios utilizados na avaliação subjetiva incluem (mas não estão limitados a):

- Uso dos princípios básicos da orientação a objetos, como encapsulamento, abstração e modularização;
- Legibilidade (nomes de variáveis bem escolhidos, código bem formatado, uso de comentários quando necessário etc.);
- Consistência (utilização de um mesmo padrão de código, sugere-se a convenção de código do Java: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf);
- Eficiência (sem exageros, tentar evitar grandes desperdícios de recursos);
- Uso eficaz da API Java (leitura com `Scanner`, API de coleções etc.) e das funcionalidades das novas versões da plataforma (ex.: tipos genéricos, laço `for each`, `try` com recursos fecháveis etc.).

Alguns alunos poderão ser aleatoriamente escolhidos para explicar o trabalho ao professor em entrevista. Tal entrevista consiste em explicar trechos do trabalho escolhidos arbitrariamente pelo professor. Neste caso, a nota subjetiva do trabalho poderá ser afetada pelo resultado desta entrevista.

# 5 Avaliação objetiva

## 5.1 Script de Teste Automático

O trabalho prático da disciplina será avaliado em duas etapas, sendo a primeira uma avaliação objetiva, com testes automáticos. Foi disponibilizado aos alunos um *script* para execução de alguns testes automáticos, sendo portanto possível garantir que o trabalho passa nesses testes antes de submetê-lo ao professor. Este documento explica como preparar seu trabalho para execução do *script*.

### 5.1.1 Obtendo e executando o *script*

O *script* de teste funciona somente em ambientes Linux/MacOS e foi testado no **Lab1**. Recomenda-se fortemente que os testes finais do seu trabalho sejam feitos no **Lab1**, pois as versões das ferramentas instaladas nas máquinas do laboratório serão consideradas como versões de referência para a correção do trabalho. Para obter e executar o *script* Java, siga os passos abaixo:

1. No SIGAA, obtenha o arquivo `teaching-br-testesoftware-20192-script-java.zip`;
2. Descompacte o arquivo em alguma pasta do seu sistema;
3. Abra um terminal, acesse esta pasta;
4. Execute o *script*: `./test.sh` e o resultado deve ser parecido com a saída abaixo:

Listing 1: bash

```
$ ./test.sh
Script de teste TesteSoftware 2019/2 - Trabalho Java
$
```

O *script* C++ funciona de maneira idêntica, substituindo **java** por **cpp** no nome do arquivo disponibilizado no *site*. Ambos os *scripts* reconhecem trabalhos se forem colocados em pastas no mesmo diretório em que se encontra o *script* `test.sh` e se os trabalhos seguirem as instruções contidas nas seções 2 e 3, abaixo. Note que há já uma pasta testes, que contém os arquivos de teste executados pelo *script*. O *script* já é configurado a não considerar esta pasta como uma pasta de trabalho. No exemplo abaixo, o comando `ls` mostra que há um diretório `professor` dentro do qual encontra-se a implementação do *trabalho do professor*. Em seguida, mostra a execução do *script* de testes sem erro algum:

Listing 2: bash

```
$ ls -Flpah
total 8
drwxr-xr-x 5 flavio freitas 170B May 30 18:00 ./
drwxr-xr-x@ 55 flavio freitas 1.8K May 30 17:54 ../
drwxr-xr-x 4 flavio freitas 136B May 30 12:30 professor/
-rwxr-xr-x@ 1 flavio freitas 2.0K May 30 17:57 test.sh
drwxr-xr-x 6 flavio freitas 204B May 30 17:24 testes/
```

```
$ ./test.sh

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em 1-apagar.csv
[I] Testando professor: teste 01, tudo OK em 2-areceber.csv
[I] Testando professor: teste 01, tudo OK em 3-vendasprod.csv
[I] Testando professor: teste 01, tudo OK em 4-vendaspgto.csv
[I] Testando professor: teste 01, tudo OK em 5-estoque.csv
[I] Testando professor: teste 02
[I] Testando professor: teste 02, serializacao OK!
[I] Testando professor: teste 03
[I] Testando professor: teste 03, serializacao OK!
[I] Testando professor: teste 03, tudo OK em 1-apagar.csv
[I] Testando professor: teste 03, tudo OK em 2-areceber.csv
[I] Testando professor: teste 03, tudo OK em 3-vendasprod.csv
[I] Testando professor: teste 03, tudo OK em 4-vendaspgto.csv
[I] Testando professor: teste 03, tudo OK em 5-estoque.csv
[I] Testando professor: pronto!
$
```

### 5.1.2 Testando o trabalho Java

O arquivo compactado do trabalho Java, a ser enviado por *e-mail* conforme descrito na especificação do trabalho deve conter os arquivos fonte e um arquivo de *build* do Ant. Ele não deve conter classes compiladas (`.class`). Estas devem ser geradas pelo software de *build* (construção de programas) Apache Ant (`http://ant.apache.org`).

Os arquivos fonte podem estar organizados da forma que você achar melhor, desde que o Ant consiga compilá-los, executar as *classes* geradas e limpar o projeto. Para que isso seja feito de forma automatizada, o arquivo de *build* do Ant deve, obrigatoriamente, encontrarse na raiz do arquivo compactado e chamar-se `build.xml`. Além disso, ele deve ser feito de forma a responder aos seguintes comandos:

Tabela 3: Formas de compilar Java

| Comando | Resultado esperado |
|---|---|
| `ant compile` | O código-fonte deve ser compilado, gerando os arquivos `.class` para todas as *classes* do trabalho. |
| `ant run` | O programa deve ser executado especificando as opções `-c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv` como parâmetro. |
| `ant run-read-only` | O programa deve ser executado no modo `--read-only` (ver especificação do trabalho), especificando além disso as mesmas opções do comando `ant run` acima. |
| `ant run-write-only` | O programa deve ser executado no modo `--write-only` (ver especificação do trabalho). |
| `ant clean` | Todos os arquivos gerados (*classes* compiladas, relatórios de saída, arquivo de serialização) e eventuais arquivos de entrada de dados devem ser excluídos, sobrando somente o conteúdo original do arquivo compactado (ou seja, o código-fonte e o arquivo de *build*). |

Segue abaixo um exemplo de arquivo `build.xml` que atende às especificações acima. Em negrito encontram-se marcados os dados que devem ser adaptados dependendo do projeto:

- Subpasta onde encontra-se todo o código-fonte;
- Subpasta onde serão colocadas as classes compiladas;
- Nome da classe principal do programa, ou seja, aquela que possui o método `main()`.

Listing 3: build.xml

```
<project name="TrabalhoTesteSoftware_2019_2" default="compile" basedir=".">
  <description>Arquivo de build do trabalho de Teste de Software, 2019/2.</
     description>

  <!-- Propriedades do build. -->
  <property name="src" location="src" />
  <property name="bin" location="bin" />
  <property name="mainClass" value="meupacote.MinhaClassePrincipal" />

  <!-- Inicializacao. -->
  <target name="init" description="Inicializa as estruturas necessarias.">
```

```
  <tstamp/>
  <mkdir dir="${bin}" />
</target>

<!-- Compilacao. -->
<target name="compile" depends="init" description="Compila o codigo-fonte.">
  <javac includeantruntime="false" srcdir="${src}" destdir="${bin}" />
</target>

<!-- Execucao normal. -->
<target name="run" depends="compile" description="Executa o programa
    principal, em modo normal.">
  <java classname="${mainClass}">
    <arg value="-c" />
    <arg value="clientes.csv" />
    <arg value="-f" />
    <arg value="fornecedores.csv" />
    <arg value="-p" />
    <arg value="produtos.csv" />
    <arg value="-a" />
    <arg value="compras.csv" />
    <arg value="-v" />
    <arg value="vendas.csv" />
    <classpath>
      <pathelement path="${bin}" />
    </classpath>
  </java>
</target>

<!-- Execucao somente leitura. -->
<target name="run-read-only" depends="compile" description="Executa o
    programa principal, em modo somente leitura.">
  <java classname="${mainClass}">
    <arg value="-c" />
    <arg value="clientes.csv" />
    <arg value="-f" />
    <arg value="fornecedores.csv" />
    <arg value="-p" />
    <arg value="produtos.csv" />
```

```
      <arg value="-a" />
      <arg value="compras.csv" />
      <arg value="-v" />
      <arg value="vendas.csv" />
      <arg value="--read-only" />
      <classpath>
        <pathelement path="${bin}" />
      </classpath>
    </java>
  </target>

  <!-- Execucao somente escrita. -->
  <target name="run-write-only" depends="compile" description="Executa o
      programa principal, em modo somente escrita.">
    <java classname="${mainClass}">
      <arg value="--write-only" />
      <classpath>
        <pathelement path="${bin}" />
      </classpath>
    </java>
  </target>

  <!-- Limpeza. -->
  <target name="clean" description="Limpa o projeto, deixando apenas o codigo-
      fonte." >
    <delete dir="${bin}"/>
    <delete><fileset dir="." includes="*.csv"/></delete>
    <delete><fileset dir="." includes="*.dat"/></delete>
  </target>
</project>
```

### 5.1.3 Testando o trabalho C++

O arquivo compactado do trabalho C++, a ser enviado por *e-mail* conforme descrito na especificação do trabalho deve conter os arquivos fonte e um arquivo de *build* do `Make` (ou seja, um `Makefile`). Ele não deve conter arquivos objeto (`.o`). Estes devem ser gerados pelo `make` (http://www.gnu.org/software/make/).

Os arquivos fonte podem estar organizados da forma que você achar melhor, desde que o `make` consiga compilá-los, executar as *classes* geradas e limpar o projeto. Para que isso seja feito de forma automatizada. O *script* espera que o seu `Makefile` seja preparado para responder

aos seguintes comandos:

Tabela 4: Formas de compilar C++

| Comando | Resultado esperado |
|---|---|
| `make` | O código-fonte deve ser compilado, gerando o programa executável. |
| `make run` | O programa deve ser executado especificando as opções `-c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv` como parâmetro. |
| `make clean` | Todos os arquivos gerados (*classes* compiladas, relatórios de saída) e eventuais arquivos de entrada de dados devem ser excluídos, sobrando somente o conteúdo original do arquivo compactado (ou seja, o código-fonte e o `Makefile`). |

Para o trabalho de C++ será considerado para a correção subjetiva se o aluno utilizou compilação separada. Por exemplo, suponha que o trabalho seja composto pelo arquivo `main.cpp` e uma classe chamada `MinhaClasse` (dividida em `MinhaClasse.h` e `MinhaClasse.cpp`). Este seria um exemplo de `Makefile` sem compilação separada (que acarretará perda de pontos no critério subjetivo):

```
all:
  g++ -o main main.cpp MinhaClasse.cpp

run:
  ./main -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv

clean:
  rm -rf main *.csv *.o
```

Enquanto este seria um exemplo de `Makefile` com compilação separada (OK):

```
all: main

main: main.cpp MinhaClasse.h MinhaClasse.o
  g++ -o main main.cpp MinhaClasse.o

MinhaClasse.o: MinhaClasse.h MinhaClasse.cpp
  g++ -c MinhaClasse.cpp

run:
  ./main -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv

clean:
  rm -rf main *.csv MinhaClasse.o
```

# 6 Pontos extra

Será dado 1 ponto extra[3] ao grupo que preparar e enviar ao professor até o prazo do trabalho 1 (Java) um conjunto de arquivos de entrada (`clientes.csv`, `fornecedores.csv`, `produtos.csv`, `compras.csv` e `vendas.csv`) que atenda aos seguintes critérios:

- Não conter trechos iguais a outros arquivos de teste disponíveis no site;
- Conter o cadastro de pelo menos 10 clientes, 5 fornecedores e 15 produtos;
- Conter o registro de ao menos 30 compras e 30 vendas, sendo que nestes registros devem estar associados ao menos 7 clientes, 4 fornecedores, 10 produtos e 3 modos de pagamento.

Os arquivos de teste enviados poderão, a critério do professor, ser disponibilizados aos demais alunos como parte do *script* de testes. Atualizações do *script* serão divulgadas em sala de aula, pelo SIGAA e/ou pelo nosso grupo do WhatsApp.

Para incentivar alunos que desejarem aprender conteúdos avançados da linguagem Java por conta própria[4], são oferecidos pontos extra[5] para os alunos que agendarem uma reunião com o professor até o final do período letivo e demonstrarem que adicionaram uma ou mais das seguintes funcionalidades ao programa básico:

Tabela 5: Funcionalidades opcionais

| **Funcionalidade opcional** | **Pontos extra**⋆ |
|---|---|
| Implementação de uma interface gráfica (janelas) que permita indicar onde encontram-se os arquivos de entrada e onde salvar os arquivos de saída e gerar os relatórios a partir de um clique. | Até 3 pontos |
| Implementação de uma interface *Web* que permita fazer o *upload* dos arquivos de entrada, gere os relatórios e os disponibilize para download. **Nota:** *Applets* não serão consideradas interfaces *Web*. | Até 3 pontos |
| Substituir a serialização descrita na seção 3 por armazenamento em banco de dados relacional. Podem ser utilizadas soluções de mapeamento objeto/relacional. | Até 2 pontos |

⋆A coluna "Pontos extra" indica o máximo de pontos extra que podem ser obtidos pela implementação da funcionalidade extra correspondente. A pontuação exata será estabelecida pelo professor após avaliado o código, que deve ser explicado pelos alunos.

[3]Apesar dos pontos extras permitirem que a nota do trabalho Java ultrapasse o valor máximo de 10 pontos, no cálculo da média parcial do aluno, a nota máxima continua sendo 10, não podendo ser ultrapassada.

[4]Se houver tempo disponível ao final do período, podem ser ministradas aulas de conteúdos avançados Java nestes dias.

[5]Apesar dos pontos extras permitirem que a nota do trabalho Java ultrapasse o valor máximo de 10 pontos, no cálculo da média parcial do aluno, a nota máxima continua sendo 10, não podendo ser ultrapassada.

Note que o trabalho Java para correção automática deve ser enviado no prazo do mesmo para avaliação da nota do trabalho 1. Posteriormente ele pode ser utilizado como base para criação do programa com interface gráfica/*Web* ou banco de dados para obtenção dos pontos extra, porém ao enviá-lo para a correção no prazo inicial ele deve responder aos *scripts* dos testes automáticos, do contrário sofrerá as penalidades descritas na especificação dos Critérios de avaliação (vide Tabela 2).

Esta opção não é oferecida para os trabalhos C++.

# 7 Pontuação final

Após a avaliação objetiva e subjetiva que totalizará no máximo 20 pontos, a nota final do trabalho será normalizada para 4 pontos.

# 8 Observações finais

Caso haja algum erro neste documento, serão publicadas novas versões e divulgadas erratas em sala de aula. É responsabilidade do aluno manter-se informado, frequentando as aulas ou acompanhando as novidades na página da disciplina na Internet.

O seu programa deve começar com um cabeçalho (uma sequência de linhas de comentários) como o seguinte:

```
/*************************************************************/
/* Aluno: Fulano de Tal                                      */
/* Matrícula: 9999-17                                        */
/* Curso: Ciência da Computação                              */
/* 3º Trabalho Prático -- Testes Automatizados               */
/* DCC05234 -- 2019/2 -- IFSEMG                              */
/* Prof. Flávio Augusto de Freitas                           */
/* Compilador: ... (gcc ou Code::Blocks) versão ...          */
/* Sistema Operacional: ...                                  */
/*************************************************************/
```

**Guarde uma cópia do seu programa pelo menos até o final do semestre.**