

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

Professores do Departamento de Informática da Ufes lecionam disciplinas para os cursos de graduação do Centro Tecnológico e para o Programa de Pós-Graduação em Informática. Em cada disciplina, é necessário efetuar avaliação dos alunos, que pode ser por meio de provas ou trabalhos práticos em grupo. Por exemplo, o prof. Vítor em 2016/1 irá avaliar alunos de Programação III com 1 prova individual e 2 trabalhos práticos em duplas, enquanto os alunos de Desenvolvimento Web farão 3 trabalhos práticos em grupos.

Ao longo do semestre, o professor registra em planilhas informações sobre as disciplinas que leciona naquele semestre, os alunos matriculados em cada disciplina, as avaliações de cada disciplina, os grupos formados para cada trabalho e as notas de cada grupo/aluno em cada avaliação. Ao final do período, o professor gostaria que um sistema automatizado lesse as informações das planilhas e produzisse uma série de relatórios úteis como, por exemplo:

- Relatório de inconsistências: eventuais erros encontrados nos dados das planilhas fornecidas pelo professor como, por exemplo, grupo de trabalho de uma disciplina apontando para aluno não matriculado na mesma, aluno participando de mais de um grupo para a mesma disciplina, etc.;
- Pauta final de cada disciplina: notas, média parcial, prova final e média final de cada aluno, para cada disciplina;
- Estatísticas: para cada disciplina e curso, média das notas finais dos alunos e percentual de aprovados. Para cada avaliação, média das notas dos alunos de cada curso. Inclui estatísticas similares para os alunos da pós-graduação, separados em Mestrado e Doutorado.

Para produzir as planilhas que serão lidas pelo sistema, o professor seguiria os seguintes passos:

1. Existe uma planilha compartilhada por todos os professores do departamento que lista os cursos do CT e seus respectivos códigos (ex.: 05 = Engenharia de Computação; 06 = Engenharia Elétrica; 09 = Engenharia de Produção; 11 = Ciência da Computação, etc.);
2. No início do semestre, o professor preencheria a planilha de disciplinas que irá lecionar com o código e nome de cada uma delas;
3. Em seguida, preencheria em outra planilha as avaliações de cada disciplina: código, nome, peso (para cálculo da média), data, se é prova ou trabalho e, no segundo caso, qual o tamanho máximo dos grupos. As provas são sempre individuais;
4. Assim que recebesse da universidade a lista de alunos de cada disciplina, preencheria a planilha de alunos com número de matrícula na universidade, nome completo, disciplinas que está matriculado neste período e se o aluno é de graduação ou de pós-graduação. No primeiro caso, indica-se também o código do

curso (vide item 1 acima), no segundo caso, indica-se se o aluno é de Mestrado ou de Doutorado;

5. Quando os alunos de cada disciplina formassem os grupos de cada trabalho prático, o professor preenche a última planilha (planilha de notas), que indica, para cada trabalho prático, o número de matrícula dos alunos de cada grupo. No caso de provas, o professor insere a matrícula de cada aluno separadamente para posteriormente registrar a nota da prova;
6. Assim que as avaliações fossem corrigidas, o professor adiciona à planilha de notas as notas de cada aluno/grupo em cada avaliação (prova ou trabalho). Provas finais, quando for o caso, também são adicionadas à planilha de notas.

Com os dados acima, espera-se que o sistema consiga gerar os relatórios citados anteriormente. As próximas seções do documento detalham os dados de entrada e relatórios de saída esperados.

2. Formatos de entrada e saída

Os cadastros dos dados de cursos, alunos, avaliações, etc. são feitos em planilhas eletrônicas. Para o processamento destes dados e geração dos relatórios desejados, as planilhas serão exportadas para um formato 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.: 8,9), os dados serão exportados utilizando ponto-e-vírgula como separadores (ex.: Prog3-P1;2000990274;7,6 – representando que o(a) aluno(a) cuja matrícula é 2000990274 tirou 7,6 na avaliação Prog3-P1).

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, além de ler dados neste formato, também gerar os relatórios em CSV.

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 (ou seja, descartadas durante a leitura das planilhas de entrada e inseridas durante a escrita dos relatórios de saída).

2.1. Entrada de dados

São quatro os arquivos de entrada de dados:

- Planilha de cursos;
- Planilha de disciplinas;
- Planilha de avaliações;
- Planilha de alunos;
- Planilha de notas.

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:

Planilha de cursos

<código>;<nome>

O código é numérico (inteiro), nome pode ser lido como texto.

Planilha de disciplinas

<código>;<nome>

Ambos podem ser lidos como texto (string), pois códigos de disciplina usam letras para indicar o prefixo (departamento que oferta o curso), ex.: INF09331 para Programação III.

Planilha de avaliações

<código disciplina>;<código avaliação>;<nome>;<peso>;<tipo>;
<data>;<tamanho máximo grupos>

Peso deve ser lido como número (real). Tipo deve especificar 'T' para trabalhos práticos e 'P' para provas (sem as aspas). Tamanho máximo de grupos é um número inteiro maior que zero e deve ser especificado apenas para trabalhos. Data deve estar no formato dd/mm/yyyy. Os demais campos podem ser lidos como texto.

Planilha de alunos

<matrícula>;<nome>;<disciplinas>;<tipo>;<curso>

Matrícula é numérico (inteiro). Nome pode ser lido como texto. As disciplinas são elencadas por código, separadas por vírgula (eventuais espaços ao redor da vírgula devem ser desconsiderados). Tipo deve especificar 'G' para aluno de graduação e 'P' para aluno de pós-graduação. Curso contém o código (numérico) do curso no caso de alunos da graduação, ou a indicação 'M' (mestrado) ou 'D' (doutorado) para alunos da pós-graduação.

Planilha de notas

<código avaliação>;<matrícula(s) aluno(s)>;<nota>

Conforme citado anteriormente, códigos de avaliação são textos enquanto matrículas de alunos são numéricos. No caso de trabalhos em grupo, as matrículas são separadas por vírgula (eventuais espaços ao redor da vírgula devem ser desconsiderados). A nota é um número real de 0 a 10.

2.2. Processamento

Lidos todos os dados, o programa deve criar objetos em memória representando as informações contidas nos arquivos de entrada. Tais objetos devem estar ligados adequadamente, conforme as associações entre as classes de objetos, observando os princípios da orientação a objetos.

Com os dados em memória (representados pelos objetos), o programa deve proceder para a escrita dos relatórios.

2.3. Escrita dos relatórios

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

Relatório	Nome do Arquivo
Pauta final (por disciplina)	1-pauta-<código>.csv
Estatísticas por disciplina	2-disciplinas.csv
Estatísticas por avaliação	3-avaliacoes.csv

Note que deve ser gerada uma pauta para cada disciplina e o código da disciplina deve ser usado para compor o nome do arquivo (ex.: 1-pauta-INF09331.csv para a disciplina de Programação III). Abaixo encontra-se especificada a ordem que os dados devem aparecer em cada um destes arquivos:

Pauta final (por disciplina)

<matrícula aluno>;<nome aluno>;<N1>;<N2>;...;<Nn>;<média parcial>;<prova final>;<média final>

Este relatório deve ser ordenado por nome da pessoa, em ordem crescente. As colunas <N1>;<N2>;...;<Nn> ilustram que deve haver uma coluna para cada avaliação cadastrada (o código da avaliação deve aparecer no nome da coluna, 1ª linha do arquivo CSV). A média parcial deve ser calculada levando-se em consideração os pesos de cada avaliação. Um traço ('-') deve ser colocado no lugar da prova final quando o aluno tiver média parcial igual ou superior a 7,0. A média final é igual à parcial neste caso. No caso do aluno ter média parcial inferior a 7,0, a média final é calculada por média simples entre média parcial e prova final. Todas as notas devem ser impressas com 2 casas decimais.

Estatísticas por disciplina

<código disciplina>;<nome disciplina>;<curso>;<média notas finais>;<percentual aprovados>

Este relatório deve ser ordenado primeiro pelo código de disciplina, em ordem crescente, seguido pela média das notas finais, em ordem decrescente. Para cada disciplina e cada curso envolvidos na disciplina (considerando Mestrado e Doutorado como cursos, além dos diferentes cursos da Graduação), calcular e exibir a média das notas finais dos alunos daquele curso (com 2 casas decimais) e o percentual de alunos aprovados (com 1 casa decimal).

Estatísticas por avaliação

<código disciplina>;<código avaliação>;<nome avaliação>;<data avaliação>;<média notas>

Este relatório deve ser ordenado primeiro pelo código da disciplina (crescente), seguido pela data da avaliação (crescente). A data deve ser impressa no formato dd/mm/yyyy e a média das notas em cada avaliação com 2 casas decimais.

2.4. Tratamento de exceções

Leitura de dados de arquivos, formatação, etc. são fontes comuns de erros e exceções. Além disso, é possível que os dados das planilhas não estejam consistentes. Seu programa deve tratar **apenas** os seguintes tipos de erro:

1. 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 exibir a mensagem "Erro de I/O" (sem as aspas) e terminar sem produzir arquivos de saída;
2. Erro de formatação dos dados nos arquivos, ou seja, um valor formatado de forma incorreta nos arquivos de entrada (ex.: encontrado caractere onde esperava-se um número), causando erros de *parsing* dos dados. Nestes casos, o programa deve exibir a mensagem "Erro de formatação" (sem as aspas) e terminar sem produzir arquivos de saída;
3. Inconsistência nos dados, ou seja, não conformidade com a especificação dos arquivos de entrada da seção 2.1 ou com a descrição do problema na seção 1. Seu programa deve tratar **apenas** as inconsistências elencadas abaixo, exibindo a mensagem associada e terminar sem produzir arquivos de saída:

Inconsistência	Mensagem
O mesmo código foi usado para dois cursos, disciplinas ou avaliações diferentes.	Código repetido para <objeto>: <código>. (Substituir <objeto> por curso, disciplina ou avaliação; substituir <código> pelo código em questão. Aplicar este mesmo conceito nas mensagens abaixo).
A mesma matrícula foi usada para dois alunos diferentes.	Matrícula repetida para aluno: <matrícula>.
Código de disciplina usado na planilha de avaliações ou na planilha de alunos não foi definido na planilha de disciplinas.	Código de disciplina não definido usado no <objeto> <código/matricula objeto>: <código>.
Peso de uma avaliação é um número negativo ou zero.	Peso de avaliação inválido para <código avaliação>: <peso>.
Tipo de uma avaliação não é nem 'T' nem 'P'.	Tipo de avaliação desconhecido para <código>: <tipo>.
Tamanho máximo de grupo foi especificado para prova.	Tamanho máximo de grupo especificado para a prova <código>: <tamanho>.
Tamanho máximo de grupo é um número negativo ou zero.	Tamanho máximo de grupo inválido para trabalho <código>: <tamanho>.
Tipo de um aluno não é nem 'G' nem 'P'.	Tipo de aluno desconhecido para <matrícula>: <tipo>.
Código de curso especificado para um aluno de graduação não foi definido na planilha de cursos.	Código de curso não definido usado no aluno <matrícula>: <código>.

Inconsistência	Mensagem
Código de curso especificado para um aluno de pós-graduação não é nem 'M' nem 'D'.	Tipo de aluno de pós-graduação desconhecido para <matrícula>: <tipo>.
Código de avaliação usado na planilha de notas não foi definido na planilha de avaliações.	Código de avaliação não definido usado na planilha de notas, associado ao(s) aluno(s) <matrícula(s)>: <código>.
Matrícula de aluno usada na planilha de notas não foi definida na planilha de alunos.	Matrícula de aluno não definida usada na planilha de notas, associada à avaliação <código>: <matrícula>.
Nota na planilha de notas é um número negativo ou maior do que 10.	Nota inválida para avaliação <código> do(s) aluno(s) <matrícula(s)>: <nota>.
Disciplina não possui nenhuma avaliação cadastrada.	A disciplina <código> não possui nenhuma avaliação cadastrada.
A planilha de notas contém nota de um aluno em uma avaliação de uma disciplina que ele não está matriculado.	O aluno <matrícula> possui nota na avaliação <código> da disciplina <código>, porém não encontra-se matriculado nesta disciplina.

Apesar de terminar sem produzir arquivos de saída, seu programa não deve ser interrompido abruptamente com uma chamada a `System.exit()`, mas sim seguir até o final do método `main()` e terminar normalmente.

Quaisquer outras situações de erro devem ser ignoradas. Pode-se assumir que nos testes feitos durante a avaliação dos trabalhos outros tipos de erros diferentes dos listados acima nunca acontecerão.

3. Execução

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

- -c <arquivo>: planilha de cursos;
- -d <arquivo>: planilha de disciplinas;
- -p <arquivo>: planilha de avaliações (provas e trabalhos);
- -a <arquivo>: planilha de alunos;
- -n <arquivo>: planilha de notas.

Supondo que a classe do seu programa que possui o método `main()` chama-se `Main` e encontra-se no pacote `trabalho`, para executar seu programa lendo os arquivos `curso.csv`, `disciplinas.csv`, `avaliacoes.csv`, `alunos.csv` e `notas.csv` como arquivos de entrada, o comando seria:

```
java trabalho.Main -c curso.csv -d disciplinas.csv -p
avaliacoes.csv -a alunos.csv -n notas.csv
```


Recuperação de pontos¹: além dos parâmetros acima, a versão Java do seu programa pode, se quiser recuperar pontos perdidos (vide Seção 5), suportar dois parâmetros opcionais que estabelecem três modos de execução diferentes. Neste caso, o programa deve poder ser chamado das três formas, como a seguir:

- `java trabalho.Main -c cursos.csv -d disciplinas.csv -p avaliacoes.csv -a alunos.csv -n notas.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 trabalho.Main --read-only -c cursos.csv -d disciplinas.csv -p avaliacoes.csv -a alunos.csv -n notas.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 `semestre.dat`. Os relatórios não devem ser gerados neste caso;
- `java trabalho.Main --write-only`: quando especificada esta opção, o programa deve carregar os objetos serializados no arquivo `semestre.dat`, gerar os relatórios e escrevê-los nos arquivos de saída. Neste caso não há leitura de arquivo CSV.

Importante: as opções de execução podem ser passadas em qualquer ordem. Portanto, o comando

```
java trabalho.Main --read-only -c cursos.csv -d
disciplinas.csv -p avaliacoes.csv -a alunos.csv -n notas.csv
```

É equivalente a:

```
java trabalho.Main -a alunos.csv -p avaliacoes.csv -d
disciplinas.csv -c cursos.csv -n notas.csv --read-only
```

Por fim, a versão C++ do programa não deve implementar as opções `--read-only` e `--write-only`, ou seja, a recuperação de pontos não é oferecida para o trabalho 2.

4. Condições de entrega

O trabalho deve ser feito obrigatoriamente em dupla² e em duas versões: uma utilizando a linguagem Java, outra utilizando a linguagem C++. O primeiro deve ser entregue até o dia **02/06/2016** e o segundo até o dia **05/07/2016**, impreterivelmente. As duplas para os trabalhos Java e C++ não precisam, necessariamente, ser as mesmas. No entanto, é preciso avisar com antecedência sobre eventuais trocas.

Dado que existem várias versões dos compiladores Java e C++, fica determinado o uso das versões instaladas nas máquinas dos LabGrads (Laboratórios de Graduação do Departamento de Informática) como versões de referência para o trabalho prático. Seu

¹ Diferentemente dos pontos extras (vide Seção 6), a recuperação de pontos apenas recupera pontos perdidos, o que significa que a nota do trabalho não poderá ultrapassar o valor máximo de 10 pontos.

² Exceto quando permitido extraordinariamente pelo professor, acordado com antecedência. Caso contrário, o aluno/grupo sofrerá uma penalidade de 2 pontos.

trabalho deve compilar e executar corretamente nas máquinas destes laboratórios. Além disso, os arquivos de código-fonte devem estar codificados com Unicode (UTF-8) para evitar erros de compilação.

4.1. Entrega do trabalho

O código-fonte e o arquivo de *build* (vide seção 4.2) de sua solução deverão ser compactados e enviados por e-mail (anexo ao e-mail) para o professor (vitor.souza@ufes.br). Serão aceitos trabalhos entregues até as 23h59 da data limite³. O assunto do e-mail deverá ser o seguinte:

Prog3 – 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* MD5⁴ do arquivo recebido. Para garantir que o arquivo foi recebido sem ser corrompido, gere o *hash* MD5 do arquivo que você enviou⁵ 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 o arquivo de *build* (explicações a seguir) e o código-fonte. O arquivo enviado não deve conter nenhuma classe compilada. Os testes automáticos serão executados no diretório onde encontra-se o arquivo de *build*. O código-fonte pode ser organizado da forma que a dupla achar melhor, desde que o arquivo de *build* esteja adequado a esta estrutura.

4.2. Preparação e execução do script de testes

Os trabalhos práticos da disciplina serão avaliados em duas etapas (vide seção 5), sendo a primeira uma avaliação objetiva, com testes automáticos. Foi disponibilizado aos alunos um script para execução de alguns testes automáticos, sendo possível, portanto, garantir que o trabalho passa nesses testes antes de submetê-lo ao professor.

O script de teste funciona somente em ambientes Linux/MacOS e foi testado no LabGrad. Recomenda-se fortemente que os testes finais do seu trabalho sejam feitos no LabGrad,

³ Além da entrega do trabalho, será feita uma entrevista com cada grupo, separadamente. Tais entrevistas devem acontecer até a data de entrega do trabalho em questão, em horário de atendimento do professor. Os alunos podem enviar o trabalho depois da entrevista (até as 23h59 da data limite), porém devem trazer uma cópia do trabalho para apresentação no momento da entrevista.

⁴ Para saber mais sobre *hash* MD5, visite sua página na Wikipedia: https://pt.wikipedia.org/wiki/MD5#Hashes_MD5

⁵ 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-hashes-md5> você encontra instruções também para Windows (porém os exemplos mostram geração de *hash* para strings, não para arquivos).

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, siga os passos abaixo:

1. Na página da disciplina, obtenha o arquivo `teaching-br-prog3-20161-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:

```
$ ./test.sh
Script de teste Prog3 2016/1 - Trabalho 1
$
```

O script reconhece 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 a seguir. 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:

```
$ ls -Flpah
drwxr-xr-x   6 vitor  staff   204B Oct 13 15:31 ./
drwxr-xr-x@ 16 vitor  staff   544B Oct 13 15:01 ../
drwxr-xr-x   4 vitor  staff   136B Oct 13 15:26 professor/
-rwxr-xr-x@  1 vitor  staff   2.0K Oct  9 08:21 test.sh
drwxr-xr-x   7 vitor  staff   238B Oct 13 15:31 testes/

$ ./test.sh
Script de teste Prog3 2016/1 - Trabalho 1

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em 1-pauta-INF09331.csv
[I] Testando professor: teste 01, tudo OK em 2-disciplinas.csv
[I] Testando professor: teste 01, tudo OK em 3-avaliacoes.csv
[I] Testando professor: teste 02
[I] Testando professor: teste 02, serialização OK!
[I] Testando professor: teste 03
[I] Testando professor: teste 03, serialização OK!
[I] Testando professor: teste 01, tudo OK em 1-pauta-INF09331.csv
[I] Testando professor: teste 01, tudo OK em 2-disciplinas.csv
[I] Testando professor: teste 01, tudo OK em 3-avaliacoes.csv
[I] Testando professor: teste 04
[I] Testando professor: teste 04, tudo OK em output.txt
[I] Testando professor: pronto!

$
```

O script compara cada arquivo de saída gerado pelo trabalho do aluno com o arquivo de saída gerado pela implementação do professor. No exemplo acima, nenhum erro foi encontrado e tudo está OK. Quando diferenças são encontradas, as mesmas são mostradas na tela e implicam perda de pontos na correção automática. O aluno deve, portanto, tentar reduzir o número de diferenças ao máximo possível antes de entregar o trabalho.

Nota: o teste 04 indica tudo OK no arquivo output.txt, porém este arquivo não foi citado na especificação. Na verdade, este é um arquivo temporário criado pelo script para os casos em que há inconsistência no trabalho e o programa deve imprimir uma mensagem na tela. O script direciona as impressões de tela para este arquivo temporário e compara com a resposta oficial do professor.

Para testar o seu trabalho, crie uma pasta com um nome qualquer dentro do mesmo diretório em que se encontra o script `test.sh` e copie seu código-fonte para esta pasta. Além do código-fonte, crie um arquivo de *build* do Apache Ant (<http://ant.apache.org>) que indique como compilar e executar seu programa.

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, encontrar-se na raiz da pasta criada e chamar-se `build.xml`. Além disso, ele deve ser feito de forma a responder aos seguintes comandos:

Comando	Resultado esperado
<code>ant compile</code>	O código-fonte deve ser compilado, gerando os arquivos <code>.class</code> para todas as classes do trabalho.
<code>ant run</code>	O programa deve ser executado especificando as opções <code>-c cursos.csv -d disciplinas.csv -p avaliacaoes.csv -a alunos.csv -n notas.csv</code> como parâmetro.
<code>ant run-read-only</code>	O programa deve ser executado no modo <code>--read-only</code> (vide seção 3), especificando além disso as mesmas opções do comando <code>ant run</code> acima.
<code>ant run-write-only</code>	O programa deve ser executado no modo <code>--write-only</code> (vide seção 3).
<code>ant clean</code>	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 <i>build</i>).

Caso você não implemente as opções *read-only* e *write-only*, faça com que os respectivos comandos funcionem da mesma forma que o comando *run*, ou seja, efetuem a execução normal.

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:

- `src`: subpasta onde encontra-se todo o código-fonte;
- `bin`: subpasta onde serão colocadas as classes compiladas;
- `meupacote.MinhaClassePrincipal`: nome da classe principal do programa, ou seja, aquela que possui o método `main()`.

```
<project name="TrabalhoProg3_2016_1" default="compile" basedir=".">
  <description>Arquivo de build do trabalho de Prog3, 2016/1.</description>

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

  <!-- Inicialização. -->
  <target name="init" description="Inicializa as estruturas necessárias.">
    <tstamp/>
    <mkdir dir="${bin}" />
  </target>

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

  <!-- Execução normal. -->
  <target name="run" depends="compile" description="Executa o programa
principal, em modo normal.">
    <java classname="${mainClass}">
      <arg value="-c" />
      <arg value="cursos.csv" />
      <arg value="-d" />
      <arg value="disciplinas.csv" />
      <arg value="-p" />
      <arg value="avaliacoes.csv" />
      <arg value="-a" />
      <arg value="alunos.csv" />
      <arg value="-n" />
      <arg value="notas.csv" />
      <classpath>
        <pathelement path="${bin}" />
      </classpath>
    </java>
  </target>

  <!-- Execução 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="cursos.csv" />
      <arg value="-d" />
      <arg value="disciplinas.csv" />
      <arg value="-p" />
      <arg value="avaliacoes.csv" />
      <arg value="-a" />
      <arg value="alunos.csv" />
    </java>
  </target>
```

```
<arg value="-n" />
<arg value="notas.csv" />
<arg value="--read-only" />
<classpath>
  <pathelement path="${bin}" />
</classpath>
</java>
</target>

<!-- Execução 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
código-fonte." >
  <delete dir="${bin}" />
  <delete><fileset dir="." includes="*.txt" /></delete>
  <delete><fileset dir="." includes="*.csv" /></delete>
  <delete><fileset dir="." includes="*.dat" /></delete>
</target>
</project>
```

Recuperação de pontos⁶: será dado 1 ponto extra ao grupo que preparar e enviar ao professor, até o prazo do trabalho 1, dois conjuntos de arquivos de entrada (cursos.csv, disciplinas.csv, avaliacoes.csv, alunos.csv e notas.csv) que atendam aos seguintes critérios:

- Não conter trechos iguais a outros arquivos de teste disponíveis no site (ou seja, não copiar);
- Conter o cadastro de pelo menos 2 cursos, 2 disciplinas, 5 avaliações, 20 alunos (incluindo todos os cursos, inclusive mestrado e doutorado) além, claro das respectivas notas;
- Assim como o código-fonte do trabalho, os arquivos devem estar em formato Unicode (UTF-8);
- Os dois conjuntos de arquivos devem ser quase iguais: um deles não deve ter inconsistência nenhuma enquanto o outro deve apresentar 1 inconsistência de dados (a escolha do grupo), como descrito na seção 2.4.

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.

⁶ Idem seção 3.

4.3. Apresentação do trabalho em entrevista

Os trabalhos devem ser também apresentados ao professor por meio de entrevista. Para tal, os alunos devem acessar o sistema de marcação de horários YouCanBook.Me no seguinte endereço:

<https://vitorsouza.youcanbook.me>

Na tabela de horários que se apresenta, cada dupla deve agendar um horário, dentre os horários disponíveis, até as respectivas datas limite, com duração de 30 minutos e fornecendo os dados solicitados pelo formulário (nome e e-mail). Para o propósito da reunião, selecionar a opção "Aluno(a) de Programação Aplicada de Computadores".

Atenção aos seguintes detalhes sobre o agendamento:

- O sistema só permite agendamentos com antecedência mínima de 8 horas (e máxima de 2 semanas);
- O sistema bloqueia automaticamente horários já reservados ou em que o professor tenha outros compromissos;
- É de responsabilidade do aluno achar um horário disponível. Planeje-se com antecedência para evitar problemas de última hora (ex.: falta de horários adequados).

Uma vez agendada a reunião, os alunos devem comparecer à sala do professor (Ufes, CT-7, térreo, sala 17) para a entrevista pontualmente no dia e hora marcados. A apresentação do trabalho pode ser feita em computador portátil trazido pelos alunos ou no computador do professor. Em qualquer caso, os alunos devem também trazer o código-fonte do trabalho em disco removível (*pen-drive*) para o professor (ou tê-lo enviado por e-mail anteriormente).

A entrevista consiste em uma apresentação do código do trabalho feita pelos alunos. Durante esta apresentação, os alunos serão questionados **individualmente** sobre detalhes do trabalho e serão avaliados com relação às respostas fornecidas. Os critérios de avaliação são descritos na seção a seguir.

5. Critérios de avaliação

Os trabalhos serão avaliados em duas etapas:

- Avaliação objetiva (com testes automáticos), valendo 10 pontos;
- Avaliação subjetiva (entrevistas), 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 modificações nas situações descritas na tabela abaixo:

Situação	Modificação
Não foi feito em dupla (exceto casos previamente autorizados)	-2

Situação	Modificação
Não observou as regras para envio do trabalho.	-1
Não compilou nos testes automáticos, mas foi possível compilar manualmente (ex.: arquivos não codificados em UTF-8).	-3
Não compilou nem manualmente.	-10
Não gerou saídas nos testes automáticos.	-5
Não gerou saídas nem manualmente.	-8
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
Opções <code>--read-only</code> e <code>--write-only</code> funcionaram no teste automático.	+2

Para avaliação subjetiva, novamente os trabalhos começam com nota 10 e perdem pontos (que variam de acordo com a avaliação feita pelo professor e pelo monitor) 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: <http://www.oracle.com/technetwork/java/codeconv-138413.html>);
- 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 *foreach*, *try* com recursos fecháveis, etc.);
- Se o aluno sabe responder questões formuladas pelo professor durante a entrevista sobre o código-fonte escrito pela dupla.

6. Pontos extra⁷

Para incentivar alunos que desejarem aprender conteúdos avançados da linguagem Java por conta própria⁸, são oferecidos pontos extra para os alunos que demonstrarem na

⁷ Ao contrário da recuperação de nota, os pontos extras permitem que a nota do trabalho Java ultrapasse o valor máximo de 10 pontos. No entanto, no cálculo da média parcial do aluno, a nota máxima continua sendo 10, não podendo ser ultrapassada.

⁸ Os alunos devem buscar recursos próprios para aprender tais tecnologias, visto que não há tempo hábil durante o semestre para aulas destes assuntos. O professor, no entanto, possui alguns materiais que pode indicar para os alunos interessados.

entrevista do trabalho 2 que adicionaram uma ou mais das seguintes funcionalidades ao programa do trabalho 1:

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 3 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 avaliada a qualidade do código, que deve ser explicado pelos alunos, e do resultado (ex.: quão bem feita é a interface gráfica/web?).

Note que o trabalho Java para correção automática deve ser enviado no prazo 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 da correção objetiva.

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

7. 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.