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++.

Novidades em relação a versões anteriores da especificação estão marcadas em amarelo.

1. Descrição do problema

A UFES precisa de um sistema que facilite o processo de gerenciamento das atividades de seus docentes, lotados nos diferentes departamentos (ex.: Departamento de Informática).

As atividades exercidas devem ser registradas em dois tipos de documentos diferentes: o PAD (Plano Semestral das Atividades Docentes) e o RHA (Relatório de Horas Aulas). Ambos os registros devem ser enviados ao Chefe de Departamento, entretanto o prazo máximo para o envio do PAD é de 60 dias antes do início do período em que as atividades serão desenvolvidas, enquanto o prazo do RHA é de 60 dias antes do fim do ano letivo correspondente. Cabe ao Chefe do Departamento o controle da documentação recebida de todos os seus professores, sendo ele responsável por enviar todos os PADs em até 30 dias antes do início do período letivo, e por enviar todos os RHAs em até 30 dias antes do fim do ano letivo.

O PAD deve indicar, para cada docente, as atividades didático-aulas (na graduação e na pós-graduação), atividades de orientação de discentes da graduação e atividades de orientação de discentes da pós-graduação exercidas em um determinado período. Já o RHA indica o número de horas aulas dadas semestralmente por cada docente por curso.

De posse de todas as informações, o sistema será usado para apoiar a universidade a ser mais transparente para a comunidade ao publicar as atividades exercidas por seu corpo docente. As informações a serem processadas e produzidas pelo sistema a ser desenvolvido são detalhadas na seção seguinte.

2. Formatos de entrada e saída

Os cadastros dos dados dos docentes, dos alunos de graduação e pós-graduação, das disciplinas ofertadas para graduação e pós-graduação, das aulas ministradas, 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.: Jose da Silva;2018101878;G).

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 seis os arquivos de entrada de dados:

- Planilha de docentes;
- Planilha de discentes;
- Planilha de produções científicas;
- Planilha de cursos;
- Planilha de disciplinas (atividades didático-aulas);
- Planilha de orientação de discentes da graduação;
- Planilha de orientação de discentes da pós-graduação.

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 docentes

<código>;<nome>;<departamento>

O código é numérico. O nome e departamento podem ser lidos como texto.

Planilha de discentes

<matricula>;<nome>;<código do curso>

Matricula é um tipo numérico composto por 10 dígitos inteiros; Nome é do tipo texto e código do curso (que pode se referir a um curso de graduação ou pós-graduação) é do tipo inteiro.

Planilha de produções científicas

<código docente>;<titulo da publicacao>;<qualificada?>

O código do docente é do tipo inteiro. O titulo da publicação é do tipo texto. Deve ser marcado um "X" na última coluna para indicar se a publicação é qualificada (segundo o Qualis, da CAPES[1]) ou não.

Planilha de cursos

<código do curso>;<nome do curso>;<graduação?>;<pós-graduação?>

O código do curso é um identificador numérico, o nome do curso é um texto e deve ser marcado um "X" se o curso é de graduação ou pós-graduação, enquanto o outro campo deve ser deixado em branco (;;).

[1] http://www.capes.gov.br/acessoainformacao/perguntas-frequentes/avaliacao-da-pos-graduacao/7422-qualis

Planilha de disciplinas (atividades didático-aulas)

<código>;<nome>;<código docente>;<carga horária semanal>;<carga horaria semestral>;<código do curso>

O código da disciplina é alfanumérico (exemplo: INF01), o nome é do tipo texto, o código do docente, carga horária semanal, carga horaria semestral e código do curso são do tipo inteiro.

Planilha de orientação de discentes da graduação

<código do docente>;<matricula do discente>;<curso>;<carga horaria semanal>

Em código do docente, deve ser informado o código do professor responsável pela atividade de orientação, enquanto matrícula do discente refere-se ao código de matrícula do aluno orientado. Curso refere-se ao código do curso do discente e carga horária semanal é do tipo inteiro.

Planilha de orientação de discentes da pós-graduação

<código do docente>;<matricula do discente>;<data de ingresso do discente no programa>;<programa de pós graduação>;<carga horária-semanal>

Em código do docente, deve ser informado o código do professor responsável pela atividade de orientação, enquanto matricula do discente refere-se ao código de matricula do orientando. Programa de pós-graduação é do tipo texto e carga horária semanal é do tipo inteiro. A data de ingresso está no formato DD/MM/AAAA.

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
Plano Semestral das Atividades Docentes	`1-pad.csv`
Relatório de Horas Aulas	`2-rha.csv`
Tabela de alocação de disciplinas	`3-alocacao.csv`
Lista de discentes de pós-graduação	`4-ppg.csv`

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

Plano Semestral das Atividades Docentes

<nome do docente>;<nome do departamento>;<total horas semanais aulas>;<total horas semestrais aulas>;<total horas semanais orientação>;<número de produções cientificas qualificadas><número de produções científicas não qualificadas>

Este relatório deve ser ordenado por nome do docente, em ordem crescente. Nesse caso, total de horas semanais/mensais refere-se ao total de horas daquela atividade somando todos os cursos em que o docente atua.

Relatório de Horas Aulas

<nome do departamento>;<nome do docente>;<código do curso>;<nome do curso>;<total horas semestrais aulas>

Neste relatório, o total de horas semestrais é agrupado por curso. Por exemplo, se o docente "Vítor Souza" leciona duas disciplinas de 50h para o curso de código 11 e outras duas de 40h para o curso de código 05, o RHA deve conter:

```
Departamento de Informática;Vítor Souza;11;Ciência da Computação;100
Departamento de Informática;Vítor Souza;05;Engenharia de Computação;80
```

Esse relatório deve ser ordenado primeiramente por nome do departamento, seguido pelo nome do docente e então por nome do curso, todos em ordem crescente.

Tabela de alocação de disciplinas

<nome do docente>;<código da disciplina>;<nome da disciplina>;<carga horária semestral>

Este relatório deve ser ordenado por nome do docente, seguido pelo código da disciplina, ambos em ordem crescente.

Lista de discentes de pós-graduação

<nome do programa de pós-graduação>;<data de ingresso do discente no programa>;<matrícula do discente>;<nome do discente>

Este relatório deve ser ordenado por nome do programa de pós-graduação, seguido pela data de ingresso no programa, seguido pelo nome do discente, todos em ordem crescente.

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 docentes diferentes.	Código repetido para <objeto>: <código>. *(Substituir <objeto> por docente; substituir <código> pelo código em questão. Aplicar este mesmo conceito nas mensagens abaixo).*
A mesma matrícula foi usada para dois discentes diferentes.	Código repetido para <objeto>: <código>.
O mesmo código de curso foi usado para dois cursos diferentes.	Código repetido para <objeto>: <código>.
O mesmo código de disciplina foi usado para duas disciplinas diferentes.	Código repetido para <objeto>: <código>.
O código de docente em disciplina refere-se a um código inválido.[2]	Código de docente inválido na disciplina "<nome>": <código>.
O código de docente em orientação refere-se a um código inválido.	Código de docente inválido na orientação do aluno "<nome>": <código>.
O código do docente em publicação científica refere-se a um código inválido.	Código de docente inválido na publicação "<titulo>": <código>.
O código do curso em orientação refere-se a um código inválido.	Código de curso inválido na orientação do aluno "<nome>": <código>.
O código do curso em disciplina refere-se a um código inválido.	Código de curso inválido na disciplina "<nome>": <código>.
Um curso não foi marcado nem como graduação nem como pós-graduação, ou foi marcado como ambos.	Inconsistência ao definir o nível do curso: <código> - <nome>.
A data de ingresso de um discente em um programa de pós-graduação está no futuro.	Data de ingresso do aluno "<nome>" está no futuro: <data (DD/MM/AAAA)>.

[2] Neste contexto, inválido significa que não há um docente com este código na planilha de docentes. O mesmo se aplica às demais utilizações da palavra "inválido" nesta tabela.

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 e o ano a ser considerado no recredenciamento como opções de linha de comando, especificadas a seguir:

- -d <arquivo>: planilha de docentes;
- -a <arquivo>: planilha de discentes;
- -p <arquivo>: planilha de produções científicas;
- -c <arquivo>: planilha de cursos;
- -r <arquivo>: planilha de disciplinas;
- -og <arquivo>: planilha de orientação de discentes da graduação;
- -op <arquivo>: planilha de orientação de discentes da pós-graduação.

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 docentes.csv, discentes.csv, producoes.csv, cursos.csv, aulas.csv, orientagrad.csv e orientapos.csv, o comando seria:

```
java trabalho.Main -d docentes.csv -a discentes.csv -p producoes.csv -c cursos.csv -r aulas.csv -og orientagrad.csv -op orientapos.csv
```

Recuperação de pontos[3]: 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 -d docentes.csv -a discentes.csv -p producoes.csv -c cursos.csv -r aulas.csv -og orientagrad.csv -op orientapos.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 -d docentes.csv -a discentes.csv -p producoes.csv -c cursos.csv -r aulas.csv -og orientagrad.csv -op orientapos.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 **`dados.dat`**. Os relatórios não devem ser gerados neste caso;

[3] 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.

- **`java trabalho.Main --write-only`**: quando especificada esta opção, o programa deve carregar os objetos serializados no arquivo **`dados.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 -d docentes.csv -a discentes.csv -p producoes.csv -c cursos.csv -r aulas.csv -og orientagrad.csv -op orientapos.csv
```

É equivalente a:

```
java trabalho.Main -op orientapos.csv -d docentes.csv -c cursos.csv -a discentes.csv -p producoes.csv --read-only -r aulas.csv -og orientagrad.csv
```

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 dupla[4] e em duas versões: uma utilizando a linguagem Java, outra utilizando a linguagem C++. O primeiro deve ser entregue até o dia **04/06/2017** e o segundo até o dia **06/07/2018**, 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 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 monitor da disciplina (chbernabe@inf.ufes.br) e com cópia para o professor (vitorsouza@inf.ufes.br). Serão aceitos trabalhos entregues até as 23h59 da data limite.[5] O assunto do e-mail deverá ser o seguinte:

`Prog3 - Trab1 - `*`Nomes dos alunos`*

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

[5] 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 (ainda que incompleto ou incorreto) no momento da entrevista.

substituindo `Nomes dos alunos` pelos nomes dos alunos do grupo, separado por vírgula. Para a entrega do trabalho de C++, substitua **`Trab1`** por **`Trab2`**. Além disso, ao invés do build.xml do Ant, o arquivo deverá conter (além do código-fonte) um arquivo chamado Makefile com instruções para que o programa make possa compilar e executar o programa (em resumo, deve funcionar com o script de testes):

- O comando make (ou make all) deve compilar o programa;
- O comando make run deve executar o programa no modo normal;
- O comando make clean deve excluir os arquivos gerados (classes compiladas, etc.).

Assim que possível, o monitor responderá o e-mail com um *hash* MD5[6] do arquivo recebido. Para garantir que o arquivo foi recebido sem ser corrompido, gere o *hash* MD5 do arquivo que você enviou[7] 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 monitor e 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, 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-prog3-20181-script-java.zip`**;
2. Descompacte o arquivo em alguma pasta do seu sistema;

---

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

[7] 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).

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 2018/1 - 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 May 29 17:09 ./
drwxr-xr-x@ 14 vitor  staff   476B May 29 17:09 ../
drwxr-xr-x@  5 vitor  staff   170B May 29 17:26 professor/
-rwxr-xr-x@  1 vitor  staff   2.0K May 29 17:12 test.sh
drwxr-xr-x@  6 vitor  staff   204B May 29 17:11 testes/

$ ./test.sh
Script de teste Prog3 2018/1 - Trabalho 1

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em 1-pad.csv
[I] Testando professor: teste 01, tudo OK em 2-rha.csv
[I] Testando professor: teste 01, tudo OK em 3-alocacao.csv
[I] Testando professor: teste 01, tudo OK em 4-ppg.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 03, tudo OK em 1-pad.csv
[I] Testando professor: teste 03, tudo OK em 2-rha.csv
[I] Testando professor: teste 03, tudo OK em 3-alocacao.csv
[I] Testando professor: teste 03, tudo OK em 4-ppg.csv
[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: alguns testes podem indicar 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
`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 **`-d docentes.csv -a discentes.csv -p producoes.csv -c cursos.csv -r aulas.csv -og orientagrad.csv -op orientapos.csv`** como parâmetro.
`ant run-read-only`	O programa deve ser executado no modo **`--read-only`** (vide seção 3), especificando além disso as mesmas opções do comando **`ant run`** acima.
`ant run-write-only`	O programa deve ser executado no modo **`--write-only`** (vide seção 3).
`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*).

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_2018_1" default="compile" basedir=".">
    <description>Arquivo de build do trabalho de Prog3, 2018/1.</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="${mainClass}">
            <arg value="-d" />
            <arg value="docentes.csv" />
            <arg value="-a" />
            <arg value="discentes.csv" />
            <arg value="-p" />
            <arg value="producoes.csv" />
            <arg value="-c" />
            <arg value="cursos.csv" />
            <arg value="-r" />
            <arg value="aulas.csv" />
            <arg value="-og" />
            <arg value="orientagrad.csv" />
            <arg value="-op" />
            <arg value="orientapos.csv" />
            <classpath>
                <pathelement path="${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="-d" />
            <arg value="docentes.csv" />
            <arg value="-a" />
            <arg value="discentes.csv" />
            <arg value="-p" />
            <arg value="producoes.csv" />
            <arg value="-c" />
            <arg value="cursos.csv" />
```

```
                <arg value="-r" />
                <arg value="aulas.csv" />
                <arg value="-og" />
                <arg value="orientagrad.csv" />
                <arg value="-op" />
                <arg value="orientapos.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 pontos[8]: 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 (docentes.csv, discentes.csv, producoes.csv, cursos.csv, aulas.csv, orientagrad.csv e orientapos.csv) que atendam aos seguintes critérios:

- Não conter trechos iguais a outros arquivos de teste disponíveis no site (ou seja, não copiar de outros grupos ou do professor);
- Conter o cadastro de pelo menos 5 docentes, 30 discentes (espalhados entre graduação e pós-graduação), 30 publicações científicas, 15 cursos (entre graduação e pós-graduação), 30 aulas espalhadas pelos 15 cursos e 20 atividades de orientação (entre graduação e pós-graduação);
- 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.

[8] Idem seção 3.

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.

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 (YCBM) 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 3".

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 14) 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
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.).	-0,5 se forem poucas, -1 se forem muitas
Grandes diferenças das saídas em relação ao teste automático (ex.: valores sensivelmente diferentes).	-1 se forem poucas, -2 se forem muitas
Entrega fora do prazo.	-1 por dia
Opções `--read-only` e `--write-only` 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) 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 *for-each*, *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 extra[9]

Para incentivar alunos que desejarem aprender conteúdos avançados da linguagem Java por conta própria,[10] são oferecidos pontos extra para os alunos que demonstrarem na

[9] 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.

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, se desejado.	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.

[10] 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.