Especificação do Trabalho Prático

O trabalho prático da disciplina consiste em desenvolver um sistema computacional para solução do problema descrito abaixo na linguagem de programação Java, inicialmente sem interatividade com o usuário e posteriormente adicionando interface gráfica interativa.

1. Descrição do problema

Uma revista de informática, a *EngeSoft*, deseja um novo sistema para gerenciar suas atividades. *EngeSoft* é publicada mensalmente, sendo que uma edição tem diversos artigos, todos versando sobre um mesmo tema. Por exemplo, a edição deste mês é sobre o tema "Qualidade de Software", tendo oito artigos. De uma edição deseja-se saber o volume, número, data prevista de publicação, tema e artigos submetidos.

Autores submetem artigos para uma edição específica. De um artigo deseja-se saber os autores e o título. Os autores devem informar, além de seus nomes, e-mails e as instituições a que pertencem, com endereço. Para artigos com mais de um autor, deve ser indicado um autor como contato.

Para avaliar os artigos submetidos à publicação, a *EngeSoft* possui um conjunto de colaboradores que avaliam artigos (chamados revisores). Dos revisores deseja-se saber o nome, e-mail, instituição e temas para os quais está habilitado a avaliar artigos.

Essas informações são usadas para distribuir os artigos para os colaboradores. Cada artigo é obrigatoriamente avaliado por três revisores, todos habilitados ao tema da edição correspondente, que atribuem notas de 0 a 10 (com até uma casa decimal) a três itens: originalidade, conteúdo e apresentação. Com base nessas avaliações é que se decide se um artigo será publicado ou não. Para simplificar, considere que um revisor é obrigado a avaliar um artigo quando este lhe é atribuído (ou seja, não pode haver cancelamento de atribuição).

Artigos que já foram avaliados pelos seus três revisores estão prontos para a seleção de quais artigos serão publicados na edição em questão, caso contrário encontram-se ainda em avaliação. Apenas quando todos os artigos submetidos para uma edição tiverem sido avaliados é que a seleção pode ser efetuada. Esta seleção é feita pelo editor-chefe da edição, escolhido previamente no conjunto de colaboradores. Finda a seleção, sabem-se quais artigos foram selecionados para publicação e quais foram rejeitados.

A diretoria da revista *EngeSoft* gostaria de um sistema para auxiliar a tarefa do editor-chefe de cada edição. O sistema deverá, dadas as avaliações dos artigos, calcular as médias das avaliações e apresentar ao editor-chefe um relatório dos artigos avaliados, por ordem de média das avaliações. São pedidos, ainda, alguns relatórios complementares, descritos nas próximas seções.

O projeto de construção de um sistema para a revista *EngeSoft* se dará em duas etapas: na primeira etapa, os dados serão mantidos em planilhas eletrônicas e, quando todos os dados estiverem prontos, eles serão passados ao programa, que produzirá relatórios de acordo com os dados fornecidos (vide seção 2). Numa segunda etapa, será construída uma interface gráfica com o usuário para que os dados sejam inseridos diretamente no novo sistema, promovendo uma maior interatividade com o usuário.

2. Primeira etapa

Para a primeira etapa, deverá ser construído um software sem interatividade, que efetua a leitura dos arquivos de entrada e produz automaticamente os relatórios de saída. Para uma transição mais lenta entre o sistema atual (manual) e um sistema completamente informatizado, foi combinado que os cadastros seriam feitos em planilhas eletrônicas ao longo do mês e, assim que estivessem completos (todas as informações da edição), eles seriam passados ao software para geração dos relatórios.

Para o processamento destes dados e geração dos relatórios desejados, um funcionários da *EngeSoft* 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 separador (ex.: 123;456;9,5;7,8;8,0 – representando que o revisor 123 avaliou o artigo 456 e atribuiu-lhe as notas 9,5; 7,8; e 8,0).

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.

O restante desta seção especifica em detalhes como o software deve funcionar na primeira etapa. A especificação encontra-se dividida em quatro partes: leitura dos dados, processamento, escrita dos relatórios e tratamento de exceções.

2.1. Leitura dos dados

São cinco os arquivos de entrada de dados:

- Informações da edição;
- Cadastro de temas;
- Cadastro de pessoas;
- Cadastro de artigos;
- Cadastro de revisões.

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 e os tipos de dados esperados. Note, no entanto, que todo arquivo CSV possui uma linha de título (a 1ª linha) que não deve ser lida como dado, mas sim descartada. Consulte os exemplos disponíveis junto ao script de testes.

Informações da edição

```
<Nome do tema>
<Nome do editor-chefe>
<Volume>
<Número>
<Data de publicação>
```

Ao contrário dos cadastros, o arquivo de informações da edição não encontra-se em formato CSV. Ele contém informações de uma única edição (a que está sendo produzida)

e cada informação encontra-se em uma linha diferente. Volume e número são numéricos (inteiros), data de publicação é uma data no formato dia/mês/ano-com-4-dígitos e os demais campos devem ser lidos como texto.

Cadastro de temas

```
<nome>;<códigos dos revisores capacitados para o tema>
```

Os códigos dos revisores são numéricos (inteiros) e separados por vírgula. Nome deve ser lido como texto.

Cadastro de pessoas

```
<código>;<nome>;<email>;<instituição>;<endereço>;<tipo>
```

Código é numérico (inteiro), tipo deve ser A (para autor) ou R (para revisor) e os demais campos devem ser lidos como texto.

Cadastro de artigos

```
<código>;<título>;<códigos dos autores>;<código do autor contato>
```

Todos os códigos são numéricos (inteiros) e os códigos dos autores são separados por vírgula. O código do autor contato só é obrigatório quando o artigo tem mais de um autor. O título deve ser lido como texto.

Cadastro de revisões

```
<código do artigo>;<código do revisor>;<nota originalidade>;<nota
conteúdo>;<nota apresentação>
```

Todos os dados são numéricos, sendo os códigos inteiros e as notas números reais com até 1 casa decimal.

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.

Antes de proceder para os cálculos e escrita dos relatórios, é preciso efetuar uma verificação de consistência dos dados. A tabela abaixo lista os possíveis problemas de consistência que podem ocorrer e o que deve ser incluído no relatório de resumo (vide próxima subseção) no caso dos dados não estarem consistentes.

#	Possível inconsistência	A incluir no resumo
1	O nome do tema em *Informações da edição* não existe no *Cadastro de temas*	O tema "<nome do tema>" não foi encontrado no cadastro.
2	O nome do editor-chefe em *Informações da edição* não existe no *Cadastro de pessoas* como um revisor.	O editor-chefe "<nome do editor-chefe>" não foi encontrado no cadastro.

#	Possível inconsistência	A incluir no resumo
3	Revisores referenciados no *Cadastro de temas* não correspondem a revisores no *Cadastro de pessoas*.	O código <código do revisor> associado ao tema "<nome do tema>" não corresponde a um revisor cadastrado.
4	Não há ao menos 3 revisores no *Cadastro de temas* para o tema da edição em questão.	O tema "<nome do tema>" possui apenas <X> revisores. São necessários no mínimo 3 revisores.
5	Uma das pessoas no *Cadastro de pessoas* possui tipo diferente de A ou R.	O tipo de "<nome da pessoa>" não é um tipo válido: <tipo>.
6	Autores referenciados no *Cadastro de artigos* não correspondem a autores no *Cadastro de pessoas*.	O código <código do autor> associado ao artigo "<título do artigo>" não corresponde a um autor cadastrado.
7	O autor de contato no *Cadastro de artigos* não é um dos autores do artigo em questão.	O autor "<nome do autor de contato>" (<endereço do autor de contato>) informado como autor de contato não corresponde a um dos autores do artigo "<título do artigo>".
8	Revisores referenciados no *Cadastro de revisões* não correspondem a revisores no *Cadastro de pessoas*.	O código <código do revisor> encontrado no cadastro de revisões não corresponde a um revisor cadastrado.
9	Artigos referenciados no *Cadastro de revisões* não se encontram no *Cadastro de artigos*.	O código <código do artigo> encontrado no cadastro de revisões não corresponde a um artigo cadastrado.
10	Revisores referenciados no *Cadastro de revisões* não estão aptos a revisar artigos do tema da edição.	O revisor "<nome do revisor>" avaliou o artigo "<nome do artigo>", porém ele não consta como apto a revisar o tema "<nome do tema>", desta edição.
11	Não há exatamente três revisões para cada artigo no *Cadastro de revisões*.	O artigo "<título do artigo>" possui <X> revisões. Cada artigo deve ter exatamente 3 revisões.

Alguns problemas de consistência podem ocorrer mais de uma vez (ex.: se no cadastro de revisões houver mais de um artigo com um número de revisões diferente de 3, o problema #11 ocorrerá uma vez para cada artigo). Nesse caso devem ser incluídas no relatório múltiplas mensagens, uma para cada ocorrência do problema.

Caso o programa passe a verificação de consistência sem erros, ele deverá calcular as estatísticas e as médias das revisões dos artigos de modo a possibilitar a escrita dos dados nos relatórios. Os dados exatos a serem calculados podem ser vistos na próxima seção, que descreve em detalhes o que deve ser escrito em cada relatório.

2.3. Escrita dos relatórios

São três os arquivos de saída de dados, listados abaixo com seus respectivos nomes:

- Resumo: `relat-resumo.txt`;
- Relatório de revisões: `relat-revisoes.csv`;
- Relatório de revisores: `relat-revisores.csv`.

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

Resumo

```
EngeSoft, num. <número>, volume <volume> - <data>
Tema: <tema>
Editor-chefe: <editor-chefe>

Consistência dos dados:
<consistência>

Artigos submetidos: <número de artigos submetidos>
Revisores capacitados: <número de revisores capacitados ao tema>
Revisores envolvidos: <número de revisores que fizeram revisão>
Média artigos/revisor: <nº artigos cada revisor revisou em média>
```

Ao contrário dos relatórios, o resumo não deve ser escrito em formato CSV. Ele deve ser escrito exatamente como acima, substituindo os termos entre < e > pelos valores apropriados:

- Número de volume da edição devem ser impressos normalmente como números, sem formatação especial;
- A data deve ser impressa no formato "`<mês-por-extenso>` de `<ano-com-4-dígitos>`". Por exemplo: `Setembro de 2014`;
- Tema e editor-chefe devem ser impressos normalmente como texto, sem formatação especial;
- O campo `<consistência>` deve ser substituído pelas mensagens de erro geradas durante a validação da consistência dos dados. As mensagens devem ser impressas no formato "`- Erro <#>: <mensagem>`", onde `<#>` deve ser substituído pelo número do erro (vide 1ª coluna da tabela de possíveis inconsistências, apresentada anteriormente) e `<mensagem>` deve ser substituída pela mensagem de erro. Por exemplo:

```
Consistência dos dados:
- Erro 8: O código <código do revisor> encontrado no cadastro de revisões não corresponde a um revisor cadastrado.
- Erro 11: O artigo "Exemplo de Artigo" possui 2 revisões. Cada artigo deve ter exatamente 3 revisões.
- Erro 11: O artigo "Outro Exemplo de Artigo" possui 4 revisões. Cada artigo deve ter exatamente 3 revisões.
```

- As mensagens de erro devem ser ordenadas primeiramente pelo número do erro e, em seguida (para erros de mesmo número), em ordem alfabética;
- Caso não haja nenhum problema de consistência, o campo `<consistência>` deve ser substituído por "`- Nenhum problema encontrado.`";
- Por fim, os números inteiros nas estatísticas devem ser impressos normalmente, sem formatação, enquanto a média de artigos por revisor deve ser impressa com 2 casas decimais.

Caso **haja qualquer problema de consistência**, as estatísticas ao final do resumo e os demais relatórios **não devem ser gerados**. O programa gera só o resumo e só até a lista de inconsistências.

Relatório de revisões

```
<título do artigo>;<autor de contato>;<média>;<nome do 1º
revisor>;<nome do 2º revisor>;<nome do 3º revisor>
```

O relatório de revisões deve ser escrito em CSV e conter as informações de um artigo por linha. O título do artigo, o autor de contato e os nomes dos revisores devem ser escritos como texto, sem formatação especial. A média deve ser formatada para ter apenas 2 casas decimais e usar separação com vírgula (brasileira). Os revisores de uma mesma linha do arquivo devem aparecer em ordem alfabética. O arquivo como um todo deve ser ordenado em ordem decrescente de média. Em caso de empate, ordene alfabeticamente pelo título do artigo.

Relatório de revisores

```
<nome do revisor>;<número de artigos revisados>;<média das notas
atribuídas>
```

O relatório de revisores deve conter apenas os revisores que participaram da edição, ou seja, que revisaram artigos. Eles devem ser incluídos no relatório em ordem alfabética dos nomes, que devem ser escritos normalmente, sem formatação. O número de artigos revisados é um número inteiro e também não necessita de formatação. A média das notas atribuídas deve ser formatada com 2 casas decimais e separação com vírgula.

2.4. Tratamento de exceções

Leitura de dados de arquivos, formatação, etc. são fontes comuns de erros e exceções. 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);
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).

No programa sem interatividade, as mensagens devem ser impressas na tela e o programa deve ser terminado em seguida. No caso do programa com interface gráfica, as

mensagens devem ser exibidas de alguma forma na interface com o usuário e o programa deve continuar rodando, permitindo que o usuário modifique os parâmetros.

Quaisquer outras situações de erro possíveis 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 (primeira etapa)

Esta seção descreve como o programa deve ser executado após a primeira etapa (ou seja, sem interface gráfica com o usuário). Seu programa deve ser executado especificando os nomes dos arquivos de entrada como opções de linha de comando, especificadas a seguir:

- -e <arquivo>: informações da edição;
- -t <arquivo>: cadastro de temas;
- -p <arquivo>: cadastro de pessoas;
- -a <arquivo>: cadastro de artigos;
- -r <arquivo>: cadastro de revisões.

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 edicao.txt, temas.csv, pessoas.csv, artigos.csv, revisoes.csv como arquivos de entrada, o comando seria:

```
java Main -e edicao.txt -t temas.csv -p pessoas.csv -a
artigos.csv -r revisoes.csv
```

Recuperação de pontos[1]: além dos parâmetros acima, é possível recuperar pontos perdidos: basta que seu programa aceite dois parâmetros opcionais que estabelecem três modos de execução diferentes. Neste caso, o programa deve poder ser chamado das três formas, como a seguir:

a) `java Main -e edicao.txt -t temas.csv -p pessoas.csv -a artigos.csv -r revisoes.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;
b) `java Main --read-only -e edicao.txt -t temas.csv -p pessoas.csv -a artigos.csv -r revisoes.csv`: quando especificada a opção `--read-only`, o programa deve ler os arquivos de entrada, montar as estruturas de objetos em memória e serializar essas estruturas em um arquivo chamado `engesoft.dat`. Os relatórios não devem ser gerados neste caso;
c) `java Main --write-only`: quando especificada esta opção, o programa deve carregar os objetos serializados no arquivo `engesoft.dat`, gerar os relatórios e escrevê-los nos arquivos de saída. Neste caso não há leitura de arquivo CSV.

---

[1] Recupera pontos perdidos, o que significa que a nota do trabalho não poderá ultrapassar o valor máximo de 10 pontos.

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

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

```
java Main --read-only -e edicao.txt -t temas.csv -p
pessoas.csv -a artigos.csv -r revisoes.csv
```

É equivalente a:

```
java Main -r revisoes.csv -p pessoas.csv -e edicao.txt --read-
only -t temas.csv -a artigos.csv
```

4. Segunda etapa

A segunda etapa consiste na criação de uma interface gráfica com o usuário (GUI) que substitua a execução do programa conforme vista na seção 3 (chamada via terminal/linha de comando) por uma forma mais interativa.

Ao ser executado sem parâmetros (ex.: `java Main`) seu programa deve abrir uma janela que permita ao usuário indicar os arquivos de entrada, executar o programa, ver os arquivos de saída e qualquer outra interação que seja necessária para executar as funcionalidades desenvolvidas na primeira etapa.

A segunda etapa do trabalho será apresentada ao professor por meio de entrevista, conforme descrito na seção 5.3.

5. Condições de entrega

O trabalho deve ser feito obrigatoriamente em dupla e em duas versões, ambas em Java: uma sem interatividade (primeira etapa) e outra com interface gráfica (segunda etapa). O primeiro deve ser entregue até o dia **07/12/2014**, enquanto o segundo deve ser apresentado ao professor (por meio de entrevista) até o dia **15/12/2014**, 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 1 e 2 devem ser as mesmas. No caso de desistência de aluno, o aluno que se encontrar sozinho para o trabalho 2 deve informar ao professor para que a situação possa ser solucionada.

Dado que existem várias versões dos compiladores Java, fica determinado o uso das versões instaladas nas máquinas do LCEE como versões de referência para o trabalho prático. Seu trabalho deve compilar e executar corretamente nas máquinas do LCEE. Além disso, os arquivos de código-fonte devem estar em arquivos codificados com Unicode (UTF-8) para evitar erros de compilação.

5.1. Entrega do trabalho sem interatividade (correção automática)

Para o trabalho da primeira etapa (vide seções 2 e 3), sua solução deverá ser compactada e enviados por e-mail (anexo ao e-mail) para o monitor da disciplina (caducbraga@gmail.com) e com cópia para o professor (vitorsouza@inf.ufes.br). Serão aceitos (sem penalidade) trabalhos entregues até as 23h59 da data limite. O assunto do e-mail deverá ser o seguinte:

`PAC - Trab1 - Nomes dos alunos`

substituindo *`Nomes dos alunos`* pelos nomes completos dos alunos do grupo, separado por vírgula.

Assim que possível, o monitor responderá o e-mail com um *hash* MD5[2] do arquivo recebido. Para garantir que o arquivo foi recebido sem ser corrompido, gere o *hash* MD5 do arquivo que você enviou[3] 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 da primeira etapa passará por um processo de testes automáticos. 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.

5.2. Preparação e execução do script de testes

O trabalho prático da disciplina será avaliado em duas etapas (vide seção 6), 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.

O script de teste funciona somente em ambientes Linux/MacOS e foi testado no LCEE. Recomenda-se fortemente que os testes finais do seu trabalho sejam feitos no LCEE, 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-pac-20142-script-java.zip;
2. Descompacte o arquivo em alguma pasta do seu sistema;

---

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

[3] 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 PAC 2014/2 - 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
total 8
drwxr-xr-x   5 vitor  wheel   170B May 30 18:00 ./
drwxr-xr-x@ 55 vitor  wheel   1.8K May 30 17:54 ../
drwxr-xr-x   4 vitor  wheel   136B May 30 12:30 professor/
-rwxr-xr-x@  1 vitor  wheel   2.0K May 30 17:57 test.sh
drwxr-xr-x   6 vitor  wheel   204B May 30 17:24 testes/

$ ./test.sh
Script de teste PAC 2014/2 - Trabalho 1

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em relat-resumo.txt
[I] Testando professor: teste 01, tudo OK em relat-revisoes.csv
[I] Testando professor: teste 01, tudo OK em relat-revisores.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 relat-resumo.txt
[I] Testando professor: teste 03, tudo OK em relat-revisoes.csv
[I] Testando professor: teste 03, tudo OK em relat-revisores.csv
[I] Testando professor: teste 04
[I] Testando professor: teste 04, tudo OK em relat-resumo.txt
[I] Testando professor: teste 05
[I] Testando professor: teste 05, tudo OK em relat-resumo.txt
[I] Testando professor: teste 06
[I] Testando professor: teste 06, tudo OK em relat-resumo.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.

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 `-e edicao.csv -t temas.csv -p pessoas.csv -a artigos.csv -r revisoes.csv` como parâmetro.
`ant run-read-only`	O programa deve ser executado no modo `--read-only` (ver 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` (ver 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="TrabalhoPAC_2014_2" default="compile" basedir=".">
    <description>Arquivo de build do trabalho de PAC, 2014/2.</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="${mainClass}">
            <arg value="-r" />
            <arg value="revisoes.csv" />
            <arg value="-p" />
            <arg value="pessoas.csv" />
            <arg value="-e" />
            <arg value="edicao.txt" />
            <arg value="-t" />
            <arg value="temas.csv" />
            <arg value="-a" />
            <arg value="artigos.csv" />
            <classpath>
                <pathelement path="${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="${mainClass}">
            <arg value="-e" />
            <arg value="edicao.txt" />
            <arg value="-t" />
            <arg value="temas.csv" />
            <arg value="-p" />
            <arg value="pessoas.csv" />
            <arg value="--read-only" />
            <arg value="-a" />
            <arg value="artigos.csv" />
            <arg value="-r" />
            <arg value="revisoes.csv" />
            <classpath>
                <pathelement path="${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[4]: 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 (edicao.txt, temas.csv, pessoas.csv, artigos.csv e revisoes.csv) que atendam aos seguintes critérios:

- Não conter trechos iguais a outros arquivos de teste disponíveis no site;
- Conter o cadastro de pelo menos 5 temas, 5 revisores, 15 autores e 10 artigos;
- Os dois conjuntos de arquivos devem ser quase iguais: um deles não deve ter inconsistência nenhuma enquanto o outro deve apresentar de 2 a 4 inconsistências (a escolha do grupo), de acordo com a tabela da seção 2.2.

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.

5.3. Apresentação do trabalho em entrevista

O trabalho completo (primeira e segunda etapas concluídas) deve ser apresentado 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

[4] Idem seção 3.

Na tabela de horários que se apresenta, cada dupla deve agendar um horário, dentre os horários disponíveis, até a data limite (15/12/2014), 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, 1º andar, sala 28) 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.

A entrevista consiste em uma apresentação do trabalho em funcionamento feita pelos alunos, seguida de uma entrevista feita pelo professor. Na entrevista, 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.

6. Critérios de avaliação

O trabalho será avaliado em duas etapas, conforme descrito na seção anterior:

- Avaliação objetiva com testes automáticos, valendo 10 pontos;
- Avaliação subjetiva em entrevista, 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 observou as regras para envio do trabalho 1.	-1
Não foi feito em dupla.	-2
Não compilou nos testes automáticos, mas foi possível corrigir 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

Situação	Modificação
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 `--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.

7. Pontos extra[5]

Para incentivar alunos que desejarem aprender conteúdos avançados da linguagem Java por conta própria, são oferecidos pontos extra para os alunos que demonstrarem na entrevista final do trabalho que adicionaram uma ou mais das seguintes funcionalidades ao programa:

Funcionalidade opcional	Pontos extra
Substituição da serialização descrita na seção 3 por armazenamento em banco de dados utilizando a tecnologia JDBC.	Até 2 pontos

[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.

Substituição da serialização descrita na seção 3 por armazenamento em banco de dados utilizando alguma tecnologia de mapeamento objeto/relacional (ex.: Hibernate).	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.

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.