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++.

O trabalho Java é dividido em 4 etapas para que o grupo possa receber feedback do professor nas etapas intermediárias e melhorar o código. O trabalho C++ deve ser entregue em uma única vez.

Os trabalhos podem ser feitos em times de até 3 estudantes. Recomenda-se que o mesmo time realize ambos os trabalhos, porém podemos nos adequar a situações diversas.

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

1. Descrição geral do problema

Devido à pandemia do COVID-19, a UFES implementou o Ensino-Aprendizagem Remoto Temporário Emergencial – EARTE no semestre especial de 2020, que ocorre entre os meses de setembro e dezembro deste ano. O EARTE trouxe uma série de mudanças na interação entre estudantes e docentes, que passaram a planejar atividades de ensino-aprendizagem remotas, síncronas e assíncronas.

Neste contexto, o subcoordenador do curso de Ciência da Computação ficou interessado em saber a opinião dos(as) estudantes sobre as atividades propostas pelos(as) vários(as) docentes que atuam no curso. O seu trabalho é construir um programa que permita o registro do período acadêmico, disciplinas sendo lecionadas, docentes e estudantes envolvidos(as) nestas disciplinas, atividades propostas pelos(as) docentes e a avaliação dos(as) estudantes sobre estas atividades.

O trabalho é dividido em quatro etapas, descritas a seguir. Cada etapa tem um prazo de entrega, que deve ser observado na agenda da disciplina.

2. Etapa 1: cadastros básicos

Ao ser iniciado, seu programa deve exibir um menu de opções e permitir que o(a) usuário(a) escolha uma das funcionalidades disponíveis. Na primeira etapa, são esperadas as seguintes funcionalidades:

- Cadastro de períodos;
- Cadastro de docentes;
- Cadastro de disciplinas;
- Cadastro de estudantes;
- Matrícula de estudantes em disciplinas;
- Cadastro de atividade de disciplina;
- Avaliação de atividade por parte de estudante;
- Sair do programa.

Nesta primeira etapa, ao acessar um cadastro seu programa deve exibir os itens já cadastrados e dar ao(à) usuário(a) as opções de cadastrar um novo item e voltar ao menu principal. Os detalhes de cada cadastro são dados a seguir.

É importante observar que, nesta etapa, não há ainda controle de erros. Portanto, é esperado que o usuário do seu programa informe sempre entradas válidas. Esta preocupação será adicionada apenas na Etapa 3 (vide Seção 4).

2.1. Cadastro de períodos

Dos períodos, deseja-se registrar o ano (valor numérico) e o semestre (um caractere, pois além dos tradicionais 1 e 2, pode-se querer usar a letra E para o semestre especial). Todos os dados são obrigatórios.

Um período deve ser referenciado no formato ano/semestre, ex.: 2019/1, 2020/E.

2.2. Cadastro de docentes

Dos docentes, deseja-se registrar login institucional (o que vem antes do @ufes.br no e-mail institucional, ex.: vitor.souza), nome completo e endereço de sua página Web. Apenas login e nome são obrigatórios.

Um docente deve ser referenciado por seu login institucional.

2.3. Cadastro de disciplinas

Das disciplinas, deseja-se registrar código, nome, período e docente responsável. Todos os dados são obrigatórios.

Portanto, para cadastrar uma disciplina é preciso antes cadastrar o período e o docente responsável, que devem ser referenciados (localizados dentre os já cadastrados) da forma descrita nas seções 2.1 e 2.2.

Uma disciplina deve ser referenciada por código-período, ex.: INF09331-2020/E.

2.4. Cadastro de estudantes

Dos estudantes, deseja-se registrar matrícula (numérico) e nome completo. Ambos são obrigatórios.

Um estudante deve ser referenciado por sua matrícula.

2.5. Matrícula de estudantes em disciplinas

Para matricular um estudante em uma disciplina, deve-se informar o estudante (conforme Seção 2.4, referenciando sua matrícula) e a disciplina (conforme Seção 2.3, referenciando por código-período). O sistema deve, então, registrar o referido estudante na referida disciplina.

2.6. Cadastro de atividade de disciplina

Das atividades, deseja-se registrar o nome da atividade, se a mesma é síncrona e a qual disciplina pertence. Todas as informações são obrigatórias, mas caso o usuário não informe se a atividade é síncrona, ela deve ser considerada assíncrona por padrão.

Ao ser adicionada a uma disciplina, uma atividade deve receber um número sequencial, começando de 1. Dentro do contexto de uma disciplina, atividades podem ser referenciadas por meio deste número sequencial.

2.7. Avaliação de atividade por parte de estudante

Para avaliar uma atividade, deve-se informar o estudante (conforme Seção 2.4, referenciando sua matrícula), a disciplina (conforme Seção 2.3, referenciando por código-período), o número sequencial da atividade (conforme explicado na Seção 2.6) e uma nota de 0 a 10 (número real, até 1 casa decimal). O sistema deve, então, registrar a avaliação do referido estudante em relação à referida atividade.

3. Etapa 2: detalhamento, relatórios e compartilhamento interno

Nesta etapa, seu programa deve registrar as atividades de forma mais detalhada, prover alguns relatórios consolidando as informações cadastradas e permitir o compartilhamento de dados internamente entre os membros do time por meio de serialização. Detalhes a seguir.

3.1. Registro detalhado de atividades

A partir de agora, seu programa não deve mais registrar atividades de forma genérica, mas sim registrar 4 tipos de atividades específicas e as informações associadas a cada tipo:

- **Aula**: além do nome da aula (tema que será discutido), deseja-se registrar a data e a hora da aula, que é uma atividade síncrona por natureza;
- **Estudo**: além do nome do estudo (tema a ser estudado), deseja-se registrar um conjunto de materiais a ser estudado, sendo o estudo assíncrono por natureza. De cada material, deseja-se registrar um nome e um link (URL) onde o material se encontra na Internet;
- **Trabalho**: atividade avaliativa assíncrona, além do nome (ex.: "Trabalho Java", "T1", etc.), deseja-se registrar a data de entrega (prazo), o número máximo de pessoas nos grupos (para trabalhos individuais, basta registrar 1) e a carga horária esperada para conclusão do trabalho (estimativa de quantas horas os alunos deverão dedicar àquela atividade);
- **Prova**: atividade avaliativa síncrona, além do nome (ex.: "Prova Parcial 1"), deseja-se registrar a data e hora em que a prova irá ocorrer, bem como o conteúdo do programa da disciplina que será cobrado na prova.

O item de menu de "Cadastro de atividade de disciplina" deve ser adaptado para atender aos requisitos acima.

3.2. Relatórios

Um novo item de menu chamado "Relatórios" deve ser incluído em seu programa. Ao ser escolhido, deve perguntar ao usuário qual relatório ele gostaria de ver e exibir o relatório na tela assim que um dos possíveis relatórios for escolhido. Nesta etapa, deseja-se os seguintes relatórios:

- **Visão geral do período acadêmico**: pede que o usuário escolha um dos períodos cadastrados e mostra, para o período escolhido, as seguintes informações

das disciplinas do período: código, nome da disciplina, nome do docente, e-mail do docente, número de estudantes matriculados, número de atividades propostas. A lista deve ser apresentada em ordem alfabética de nome da disciplina;

- **Estatísticas dos docentes**: para cada docente registrado, mostrar seu nome, número de disciplinas a ele associadas, número de períodos diferentes em que possui disciplinas associadas, média de atividades por disciplina, percentual de atividades síncronas x assíncronas e média de notas recebidas em avaliações discentes.[1] Mostrar os docentes em ordem alfabética reversa de nome (o prof. Vítor cansou de ser sempre o último da lista);
- **Estatísticas dos estudantes**: para cada estudante registrado, mostrar sua matrícula, seu nome, média de disciplinas matriculadas por período acadêmico, média de avaliações realizadas por disciplina e média de notas, considerando todas as avaliações já feitas. Mostrar os estudantes primeiro por ordem decrescente do número de avaliações feitas, seguido por ordem alfabética do nome;
- **Estatísticas das disciplinas de um docente**: pede que o usuário escolha um docente e mostra, de suas disciplinas, as seguintes informações: período acadêmico, código, nome, número de atividades total, percentual de atividades síncronas x assíncronas, carga horária da disciplina (considera-se que aulas, estudos e provas contam 2 horas na carga horária) e atividades avaliativas. As disciplinas devem ser ordenadas primeiro por período e em seguida por código. Das atividades avaliativas de cada disciplina, mostrar data e nome, ordenando também por data e nome, de forma crescente.

3.3. Compartilhamento por serialização

Por fim, seu programa deve incluir duas novas opções no menu para permitir o compartilhamento de dados entre membros do mesmo time:

- **Salvar**: serializar todos os dados cadastrados em um arquivo em disco (perguntar ao usuário o nome do arquivo);
- **Carregar**: desserializar os dados salvos em um arquivo em disco (perguntar ao usuário o nome do arquivo) e carregar em memória os dados cadastrados de uma só vez. Carregar dados de arquivo descarta quaisquer dados que tenham sido carregados (manualmente ou não) anteriormente.

Nota: a versão C++ do programa não precisa implementar serialização.

4. Etapa 3: modularização e controle de erros

Nesta etapa, seu programa deve ser modularizado em pacotes considerando o propósito de cada classe, de modo que os pacotes possuam alta coesão e baixo acoplamento com outros pacotes. Sugere-se a divisão por responsabilidades de alto nível, como entrada/saída (I/O), interface com o usuário, representação de elementos do domínio do problema, etc.

---

[1] Para o percentual de atividades síncronas x assíncronas e a média de notas em avaliações, considere todas as atividades sem agregar com disciplina e todas as avaliações sem agregar por atividade/disciplina. Ou seja, não é para calcular o percentual de síncronas x assíncronas de cada disciplina separadamente e depois tirar a média. Basta considerar o conjunto completo de atividades propostas pelo docente (independente da disciplina) e calcular os percentuais. O mesmo vale para a média de notas em avaliações.

Além disso, seu programa deve agora verificar os casos em que o usuário forneceu entradas inválidas. Espera-se que sejam tratados, no mínimo, os casos descritos na tabela abaixo (substitua valores entre < e > pelo respectivo dado informado pelo usuário).

Situação	Mensagem	Exemplo
O usuário informa um dado inválido. Na descrição dos cadastros, alguns dados possuem formatação específica (ex.: para períodos, ano é valor numérico e semestre é 1 caractere apenas).	`Dado inválido: <valor inválido digitado pelo usuário>.`	`Dado inválido: abcd.`
O usuário informa uma referência inválida. Na descrição dos cadastros, alguns elementos podem ser referencias (ex.: períodos podem ser referenciados no formato ano/semestre, ex.: 2019/1, 2020/E). No entanto, o usuário informa uma referência para um elemento que não existe, não foi ainda cadastrado.	`Referência inválida: <referência digitada pelo usuário>.`	`Referência inválida: 2020/X.`
O usuário cadastra um elemento com a mesma referência de um já cadastrado. Por exemplo, já existe um período com o mesmo ano/semestre, um(a) docente com o mesmo login institucional, uma disciplina com mesmo código naquele período, um(a) estudante com a mesma matrícula.	`Cadastro repetido: <referência que já existe>.`	`Cadastro repetido: 2020/E.`
O usuário matricula um(a) estudante pela segunda vez na mesma disciplina (no mesmo período).	`Matrícula repetida: <referência do estudante> em <referência da disciplina>.`	`Matrícula repetida: 123456789 em INF09331-2020/E.`
O usuário cadastra uma avaliação repetida, ou seja, uma avaliação do(a) mesmo(a) estudante para a mesma atividade (pertencente à mesma disciplina e período).	`Avaliação repetida: estudante <referência do estudante> para atividade <referência da atividade> de <referência da disciplina>.`	`Avaliação repetida: estudante 123456789 para atividade 1 de INF09331-2020/E.`
Erro de entrada/saída. Qualquer problema que ocorrer com leitura/escrita de arquivos (ex.: o arquivo informado para carregar os dados serializados não existe, não há permissão pra escrever em um arquivo, etc.).	`Erro de I/O.`	`Erro de I/O.`

Tratar os casos acima significa que o programa não deve ser interrompido, mas sim exibir a respectiva mensagem e continuar em algum ponto anterior (pode-se retornar ao menu principal ou outro menu intermediário, pedir que se informe o dado novamente, etc.). Para isso, use o mecanismo de exceções de linguagem.

5. Etapa 4: relatórios em arquivo e compartilhamento externo

Nesta etapa, seu programa deve passar a ler os dados e escrever os relatórios em formatos padronizados, utilizando 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, por exemplo:

```
vitor.souza;Vítor Estêvão Silva Souza;http://www.inf.ufes.br/~vitorsouza/
```

Para uma visualização mais amigável dos arquivos CSV, pode-se abrir ou importá-los para um software de planilha eletrônica, como Microsoft Excel, por exemplo.

As próximas seções detalham os formatos esperados dos arquivos de entrada (dados) e saída (relatórios), bem como a forma que os arquivos devem ser especificados pela linha de comando e como proceder com o tratamento de erros. Exemplos de arquivos de entrada e saída esperados serão disponibilizados juntamente com esta especificação. Além disso, são oferecidos pontos extras aos grupos que produzirem novos arquivos de entrada para testes (vide Seção 7.2).

5.1. Formato dos arquivos de entrada

Todos os arquivos de entrada possuem na primeira linha um cabeçalho que deve ser descartado pelo seu programa. As demais linhas contêm os dados dos objetos a serem criados, como se estivessem sendo cadastrados manualmente pelo usuário (como na Parte 1 do trabalho). O formato dos arquivos de entrada é detalhado a seguir:

Cadastro de Períodos

```
<ano>;<semestre>
```

Os dados devem obedecer aos mesmos formatos especificados na Seção 2.1.

Cadastro de Docentes

```
<login institucional>;<nome completo>;<endereço web>
```

Os dados devem obedecer aos mesmos formatos especificados na Seção 2.2.

Cadastro de Disciplinas

```
<ref. período>;<código>;<nome>;<ref. docente responsável>
```

Os dados devem obedecer aos mesmos formatos especificados na Seção 2.3.

Cadastro de Estudantes

```
<matrícula>;<nome completo>
```

Os dados devem obedecer aos mesmos formatos especificados na Seção 2.4.

Matrículas de Estudantes em Disciplinas

```
<ref. disciplina>;<ref. estudante>
```

Os dados devem obedecer aos mesmos formatos especificados na Seção 2.5.

Cadastro de Atividades

```
<ref. disciplina>;<nome>;<tipo>;<data/prazo>;<hora>;<materiais/conteúdo>;
<tamanho máximo grupos>;<carga horária esperada>
```

A referência à disciplina e o nome seguem os mesmos formatos especificados na Seção 2.6. Os tipos possíveis de atividade são representados por letras: 'A' para aulas, 'E' para estudos, 'T' para trabalhos práticos e 'P' para provas. Os demais dados são especificados da seguinte maneira:

- Data/prazo: em formato `dd/mm/yyyy`, especifica a data de uma aula ou prova, ou o prazo de um trabalho;
- Hora: em formato `hh:mm`, especifica a hora de uma aula ou prova;
- Materiais/conteúdo: especifica os materiais associados a um estudo ou o conteúdo de uma prova. Conteúdo é especificado como texto simples. Materiais são especificados em formato de link MarkDown,[2] um seguido do outro (ex.: `[material1](url1)[material2](url2)[material3](url3)`);
- Tamanho máximo grupos: número inteiro, associado apenas a trabalhos práticos;
- Carga horária esperada: número inteiro, associado apenas a trabalhos práticos.

Avaliações de Atividades por parte dos Estudantes

```
<ref. disciplina>;<ref. estudante>;<ref. atividade>;<nota>
```

Os dados devem obedecer aos mesmos formatos especificados na Seção 2.7.

5.2. Formato dos arquivos de saída

Para viabilizar a execução automática (sem interação com o usuário) dos programas, alguns dos relatórios pedidos na Seção 3.2 foram adaptados, conforme especificação abaixo. Seu programa deve, então, produzir sempre todos os relatórios, sempre utilizando o nome de arquivo, cabeçalho e o formato especificado a seguir:

Visão geral dos períodos acadêmicos

Nome do arquivo: 1-visao-geral.csv.

Cabeçalho:
```
Período;Código Disciplina;Disciplina;Docente Responsável;E-mail Docente;Qtd. Estudantes;Qtd. Atividades
```

Formato:
```
<período>;<código disciplina>;<nome disciplina>;<nome docente responsável>;<e-mail docente responsável>;<qtd. Estudantes matriculados>;<qtd. Atividades propostas>
```

Diferente da Seção 3.2, este relatório deve mostrar todas as disciplinas de todos os períodos, não apenas de um período escolhido. Os dados devem ser ordenados primeiro por período, depois em ordem alfabética de nome da disciplina. Períodos devem ser ordenados primeiro pelo ano (ordem numérica crescente) depois pelo semestre (ordem alfabética crescente).

Estatísticas dos docentes

Nome do arquivo: 2-docentes.csv.

Cabeçalho:
```
Docente;Qtd. Disciplinas;Qtd. Períodos;Média Atividades/Disciplina;% Síncronas;% Assíncronas;Média de Notas
```

[2] Vide https://daringfireball.net/projects/markdown/basics, seção "Links".

Formato: `<nome>;<qtd. disciplinas associadas>;<qtd. períodos que possui disciplina associada>;<média de qtd. de atividades por disciplina>;<percentual atividades síncronas>;<percentual atividades assíncronas>;<média das notas recebidas em avaliações>`

Este relatório segue o especificado na Seção 3.2, devendo os docentes serem listados em ordem alfabética reversa de nome. Os percentuais devem ser arredondados (zero casas decimais) e a média das notas deve ser exibida com 1 casa decimal. Caso o docente não tenha nenhuma atividade proposta por ele, ambos os percentuais devem mostrar o valor zero (`0,0`).

Estatísticas dos estudantes

Nome do arquivo: 3-estudantes.csv.

Cabeçalho: `Matrícula;Nome;Média Disciplinas/Período;Média Avaliações/Disciplina;Média Notas Avaliações`

Formato: `<matrícula>;<nome>;<média qtd. disciplinas matriculadas por período>;<média qtd. avaliações realizadas por disciplina>;<média das notas das avaliações>`

Este relatório segue o especificado na Seção 3.2, devendo os estudantes serem listados primeiro por ordem decrescente do número de avaliações feitas, seguido por ordem alfabética do nome. As médias devem ser exibidas com 1 casa decimal.

Estatísticas das disciplinas dos docentes

Nome do arquivo: 4-disciplinas.csv.

Cabeçalho: `Docente;Período;Código;Nome;Qtd. Atividades;% Síncronas;% Assíncronas;CH;Datas Avaliações`

Formato: `<login docente>;<período>;<código disciplina>;<nome disciplina>;<qtd. atividades disciplina>;<percentual atividades síncronas>;<percentual atividades assíncronas>;<carga horária>;<datas das atividades avaliativas>`

Diferente da Seção 3.2, este relatório deve mostrar todas as disciplinas de todos os docentes e não só de um docente escolhido. Além disso, inclui apenas a data das atividades avaliativas. As disciplinas devem ser ordenadas primeiro por período e em seguida por código. Os percentuais devem ser arredondados (zero casas decimais) e as datas devem ser exibidas em formato `dd/mm/yyyy`, separadas por espaços e em ordem temporal crescente. Caso a disciplina não tenha atividades, os percentuais devem mostrar o valor zero (`0,0`).

5.3. Especificação por linha de comando

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

- `-p <arquivo>`: planilha de períodos;
- `-d <arquivo>`: planilha de docentes;
- `-o <arquivo>`: planilha de disciplinas (oferta);
- `-e <arquivo>`: planilha de estudantes;
- `-m <arquivo>`: planilha de matrículas de estudantes em disciplinas;
- `-a <arquivo>`: planilha de atividades de disciplinas;

- `-n <arquivo>`: planilha de avaliações de atividades pelos estudantes;
- `--read-only`: dados devem ser lidos por serialização;
- `--write-only`: dados devem ser escritos por serialização.

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 periodos.csv, docentes.csv, disciplinas.csv, estudantes.csv, matriculas.csv, atividades.csv e avaliacoes.csv, o comando seria:

```
java trabalho.Main -p periodos.csv -d docentes.csv -o disciplinas.csv
-e estudantes.csv -m matriculas.csv -a atividades.csv -n avaliacoes.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.

Quando especificada a opção `--write-only`, 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 -p periodos.csv -d docentes.csv -o
disciplinas.csv -e estudantes.csv -m matriculas.csv -a atividades.csv -n
avaliacoes.csv
```

É equivalente a:

```
java trabalho.Main -m matriculas.csv --read-only -e estudantes.csv -a
atividades.csv -o disciplinas.csv -d docentes.csv -p periodos.csv -n
avaliacoes.csv
```

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

5.4. Tratamento de erros

Nesta versão, seu programa deve continuar detectando os erros especificados na Seção 4, porém ao invés de dar continuidade ao programa, o mesmo deve ser interrompido logo após a impressão da mensagem de erro na tela.

Além disso, seu programa deve detectar agora 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).

Em ambos os casos, é esperado que o programa termine sem produzir arquivos de saída, porém execute seu código até o final do método `main()`. Ou seja, utilize o mecanismo de lançamento de exceções para interromper o fluxo do programa, evitando outros mecanismos como, por exemplo, chamar o método `System.exit()`.

6. Condições de entrega

Os times de até 3 estudantes devem criar um repositório privativo no GitHub[3] ou em serviço similar e convidar o professor para que tenha acesso ao repositório.[4] O repositório deve ter instruções de como compilar e executar o programa no arquivo README.md. Aos(às) estudantes que quiserem utilizar uma ferramenta de gerenciamento da construção (*build*) do software, recomendo instalar e experimentar o Maven.[5]

A cada etapa concluída, um dos membros do time deve realizar a entrega da atividade no Google Sala de Aula, indicando o nome dos membros do time e a URL do repositório em que se encontra o trabalho.

Ao final da última etapa do Trabalho Java e na entrega do Trabalho de C++, o time deve preparar, também, um vídeo de 10-15 minutos que demonstre ao professor e aos demais times:

- O programa em funcionamento;
- Como o código-fonte foi organizado para que o programa funcione;
- Como os princípios da Orientação a Objetos foram aplicados.

Todos os membros do time devem participar do vídeo e contribuir com seu conteúdo (dividindo a fala entre todos). Caso não seja possível fazer a gravação com o Google Meet com conta de estudante, sugere-se o uso do Jitsi Meet, que é gratuito.

7. Critérios de avaliação

Os trabalhos serão avaliados de forma objetiva e subjetiva, ambos os aspectos influenciando em sua nota final. Será disponibilizado um script que pode ser usado para testar seu programa e garantir uma melhor avaliação objetiva. Há também a possibilidade de pontos extras, detalhados a seguir.

7.1. Avaliação do código-fonte

Na avaliação objetiva, o programa entregue será compilado e executado para verificação se o mesmo está completo e funcional, de acordo com as especificações do trabalho.

Para avaliação subjetiva, os critérios 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 da Oracle ou do Google[6]);
- Eficiência (sem exageros, tentar evitar grandes desperdícios de recursos);

---

[3] https://github.com

[4] No GitHub, meu usuário é vitorsouza. Em outros serviços, os(as) alunos(as) devem me consultar.

[5] https://medium.com/@giu.drawer/criando-um-projeto-java-maven-no-eclipse-cf7326d4db37, https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

[6] Há um formatador disponível para a IDE Eclipse, vide instruções no Manual do Marvin.

- 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.). Critérios análogos para o trabalho em C++;
- Conteúdo do vídeo que apresenta o trabalho.

Todos os trabalhos entregues passarão por um procedimento de detecção de plágio. Caso seja detectado que houve plágio, a nota de todos os alunos envolvidos será 0 (zero).

As etapas intermediárias do trabalho de Java serão avaliadas de forma a prover retorno (*feedback*) aos times, porém apenas a nota final do trabalho (após a última etapa e entrega do vídeo) será considerada para a nota da disciplina.

7.2. Script de testes

Será disponibilizado um script de testes para execução automática dos programas a partir de arquivos de teste e seus resultados esperados. O script de teste funciona somente em ambientes que possam executar scripts Bash, como Linux, MacOS e Windows 10.[7]

Script de testes Java

Para obter e executar o script do trabalho Java, siga os passos abaixo:

1. Obtenha o arquivo `prog3-2020E-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 2020/E - Trabalho Java

$
```

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 -p
professor/    test.sh  testes/

$ ./test.sh
Script de teste Prog3 2020/E - Trabalho Java

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em 1-visao-geral.csv
[I] Testando professor: teste 01, tudo OK em 2-docentes.csv
[I] Testando professor: teste 01, tudo OK em 3-estudantes.csv
[I] Testando professor: teste 01, tudo OK em 4-disciplinas.csv
```

[7] https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2016/04/como-instalar-e-usar-o-shell-bash-do-linux-no-windows-10.html

```
[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-visao-geral.csv
[I] Testando professor: teste 03, tudo OK em 2-docentes.csv
[I] Testando professor: teste 03, tudo OK em 3-estudantes.csv
[I] Testando professor: teste 03, tudo OK em 4-disciplinas.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. Neste caso, tente corrigir o trabalho e reduzir o número de diferenças ao máximo possível antes de entregá-lo.

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 erro nos dados 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 `-p periodos.csv -d docentes.csv -o disciplinas.csv -e estudantes.csv -m matriculas.csv -a atividades.csv -n avaliacoes.csv` como parâmetro.
`ant run-read-only`	O programa deve ser executado no modo `--read-only`, especificando além disso as mesmas opções do comando `ant run` acima.
`ant run-write-only`	O programa deve ser executado no modo `--write-only`.

Comando	Resultado esperado
`ant clean`	Todos os arquivos gerados (classes compiladas, relatórios de saída, arquivo de serialização) e eventuais arquivos de entrada de dados devem ser excluídos, sobrando somente o conteúdo original do arquivo compactado (ou seja, o código-fonte e o arquivo de *build*).

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

- **`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_2020_E" default="compile" basedir=".">
      <description>Arquivo de build do trabalho de Prog3, 2020/E.</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, normal.">
            <java classname="${mainClass}">
                  <arg value="-p" />
                  <arg value="periodos.csv" />
                  <arg value="-d" />
                  <arg value="docentes.csv" />
                  <arg value="-o" />
                  <arg value="disciplinas.csv" />
                  <arg value="-e" />
                  <arg value="estudantes.csv" />
                  <arg value="-m" />
                  <arg value="matriculas.csv" />
                  <arg value="-a" />
                  <arg value="atividades.csv" />
                  <arg value="-n" />
                  <arg value="avaliacoes.csv" />
                  <classpath>
                        <pathelement path="${bin}" />
                  </classpath>
            </java>
      </target>

      <!-- Execução somente leitura. -->
      <target name="run-read-only" depends="compile" description="Executa em modo leitura.">
            <java classname="${mainClass}">
                  <arg value="-p" />
                  <arg value="periodos.csv" />
                  <arg value="-d" />
                  <arg value="docentes.csv" />
                  <arg value="-o" />
                  <arg value="disciplinas.csv" />
                  <arg value="-e" />
                  <arg value="estudantes.csv" />
                  <arg value="-m" />
                  <arg value="matriculas.csv" />
                  <arg value="-a" />
                  <arg value="atividades.csv" />
```

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

    <!-- Execução somente escrita. -->
    <target name="run-write-only" depends="compile" description="Executa em modo 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>
```

Script de testes C++

O script de testes do trabalho C++ funciona exatamente como o do trabalho Java, porém usando Make para compilação e execução do trabalho ao invés de Ant, conforme a tabela abaixo:

Comando	Resultado esperado
`make`	O código-fonte deve ser compilado, gerando um executável.
`make run`	O programa deve ser executado especificando as opções `-p periodos.csv -d docentes.csv -o disciplinas.csv -e estudantes.csv -m matriculas.csv -a atividades.csv -n avaliacoes.csv` como parâmetro.
`make clean`	Todos os arquivos gerados (classes compiladas, relatórios de saída) e eventuais arquivos de entrada de dados devem ser excluídos, sobrando somente o conteúdo original do arquivo compactado (ou seja, o código-fonte e o Makefile).

7.3. Ponto extra: produção de arquivos de teste

Serão dados 2 pontos extras aos grupos que prepararem e enviarem ao professor, **até o dia 06/11/2020**, dois conjuntos de arquivos de entrada (periodos.csv, docentes.csv, disciplinas.csv, estudantes.csv, matriculas.csv, atividades.csv e avaliacoes.csv) que atendam aos seguintes critérios:

- Não conter trechos iguais a outros arquivos de teste disponíveis (ou seja, não copiar de outros grupos ou do professor);
- Conter o cadastro de pelo menos 2 períodos, 4 docentes, 6 disciplinas (cada docente tendo no máximo 2 disciplinas por período), 30 estudantes, 60 matrículas (cada estudante tendo no máximo 3 matrículas por período), 30 atividades (divididas em pelo menos 3 disciplinas) e 100 avaliações (cada estudante tendo, no máximo, 15 avaliações);

- 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 erro nenhum enquanto o outro deve apresentar 1 erro (a escolha do grupo), dentre os descritos na Seção 4.

Os arquivos de teste enviados serão disponibilizados aos demais alunos para que possam também experimentá-los e avaliar se seus programas estão funcionando corretamente antes da entrega do Trabalho Java.

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.