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, considerando o que foi entregue na última etapa do trabalho Java.

Os trabalhos podem ser feitos em times de até 2 estudantes (i.e., pode ser feito sozinho ou em dupla). 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

Professores do Departamento de Informática da Ufes lecionam disciplinas para os cursos de graduação do Centro Tecnológico e para o Programa de Pós-Graduação em Informática. Em cada disciplina, é necessário efetuar avaliação dos estudantes, que pode ser por meio de provas ou trabalhos práticos em grupo. Por exemplo, o prof. Vítor em 2023/1 irá avaliar os alunos de Programação Web com 3 trabalhos práticos em grupos, já a profª. Veruska vai avaliar os alunos de Programação I (Engenharia Elétrica) com 2 provas e 1 trabalho prático.

Ao longo do semestre, o(a) professor(a) registra informações sobre as disciplinas que leciona naquele semestre, os alunos matriculados em cada disciplina, as avaliações de cada disciplina, os grupos formados para cada trabalho e as notas de cada grupo/aluno em cada avaliação. Ao final do período, o professor gostaria de produzir uma série de relatórios que indicariam, por exemplo, se há alguma inconsistência no que foi cadastrado, a pauta com notas, médias, etc. de cada disciplina, estatísticas para cada disciplina e curso, etc.

Seu trabalho é construir um programa que permita o registro dos cursos, disciplinas, avaliações, alunos e notas, gerando relatórios úteis para os professores do departamento.

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:

- Cadastrar curso;
- Cadastrar disciplina;
- Cadastrar prova;
- Cadastrar aluno(a);

- Matricular aluno(a) em disciplina;
- Registrar nota de aluno(a) em prova;
- Imprimir dados;
- Sair do programa.

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 em etapa posterior.

2.1. Cadastrar curso

Dos cursos, deseja-se registrar um código numérico (inteiro) e um nome. Todos os dados são obrigatórios.

Exemplo:

- Código: `11`
- Nome: `Ciência da Computação`

Em outros cadastros, um curso deve ser referenciado pelo seu código, não podendo haver dois cursos com o mesmo código.

2.2. Cadastrar disciplina

Das disciplinas, deseja-se registrar um código (alfanumérico) e o nome. Todos os dados são obrigatórios.

Exemplo:

- Código: `INF15933`
- Nome: `Programação Orientada a Objetos`

Em outros cadastros, uma disciplina deve ser referenciada por seu código, não podendo haver duas disciplinas com o mesmo código.

2.3. Cadastrar prova

Das provas, deseja-se registrar em qual disciplina ela será aplicada, código (alfanumérico), nome, peso no cálculo da média (número real) e data que a prova será aplicada (formato dd/mm/aaaa). Com exceção do nome, todos os dados são obrigatórios (no caso de uma prova sem nome, usa-se apenas seu código na hora de imprimir seus dados).

Note que, para cadastrar uma prova é preciso antes cadastrar uma disciplina, que deve ser referenciada (localizada dentre as já cadastradas) da forma descrita na Seção 2.2, ou seja, pelo seu código. Para os próximos cadastros, considere situações análogas.

Exemplo:

- Disciplina: `INF15933`
- Código: `ProgOO-P1`
- Nome: `Prova Parcial 1`
- Peso: `1.0`
- Data: `25/04/2023`

Em outros cadastros, uma prova deve ser referenciada pelo seu código, não podendo haver duas provas com o mesmo código.

2.4. Cadastrar aluno(a)

De um(a) aluno(a), deseja-se registrar sua matrícula (número inteiro), nome e o curso de graduação que ele faz na UFES.

Exemplo:

- Matrícula: `2000000005`
- Nome: `Arya Stark`
- Curso: `11`

Em outros cadastros, um(a) aluno(a) deve ser referenciado(a) pela sua matrícula, não podendo haver dois alunos com a mesma matrícula.

2.5. Matricular aluno(a) em disciplina

Para matricular um(a) aluno(a) em uma disciplina, basta indicar qual o(a) aluno(a) e qual a disciplina.

Exemplo:

- Aluno(a): `2000000005`
- Disciplina: `INF15933`

2.6. Registrar nota de aluno(a) em prova

Para registrar a nota de um(a) aluno(a) em uma prova, basta indicar qual o(a) aluno(a), qual a prova e qual a nota recebida (número real).

Exemplo:

- Aluno(a): `2000000005`
- Prova: `ProgOO-P1`
- Nota: `9.9`

2.7. Imprimir dados

Ao ser solicitada a impressão dos dados, seu programa deve exibir o seguinte:

- Uma listagem de disciplinas e os alunos matriculados;
- Uma listagem de provas e as notas recebidas por cada aluno(a);

Apesar dos códigos e matrículas terem sido usados na entrada de dados, nas listagens devem ser usados os nomes dos cursos, disciplinas, alunos, provas (exceto se a prova não tem um nome, então usa-se o código), etc. Não há um formato padrão para as listagens, mas segue uma sugestão abaixo com base nos exemplos dados anteriormente:

```
Disciplinas e alunos matriculados:
- Programação Orientada a Objetos (INF15933)
     - Arya Stark (Ciência da Computação)
```

```
Provas e notas recebidas:
- Programação Orientada a Objetos - Prova Parcial 1 (25/04/2023)
    - Arya Stark: 9.9
```

Note que nesta etapa estamos considerando apenas alunos de graduação e provas. Nas próximas etapas, passaremos a considerar alunos de pós-graduação, trabalhos e demais requisitos do sistema, como verificação de erros nos dados, produção de relatórios, etc.

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 de alunos de pós, trabalhos práticos e provas finais

A partir de agora, seu programa deve considerar também estudantes de pós-graduação. Ao invés de associá-los a um dos cursos de graduação cadastrados, estudantes de pós-graduação devem ser associados ao nível de pós que estão cursando: Mestrado ou Doutorado.

Além disso, seu programa deve considerar trabalhos práticos além de provas como avaliações que podem ser utilizadas nas diversas disciplinas cadastradas. Assim como provas, trabalhos práticos também possuem um código, nome, peso e data (no caso do trabalho, é a data de entrega). Além disso, devem indicar o tamanho máximo dos grupos que podem ser formados para fazê-los.

Por fim, outro tipo de avaliação que deve ser considerada é a Prova Final. Uma prova final é cadastrada da mesma forma que uma prova, porém ao ser indicada como prova final ela não entra no cálculo da média parcial dos estudantes.

Os itens de menu "Cadastrar prova" e "Cadastrar aluno" devem ser adaptados para dar suporte aos registros mais detalhados descritos acima. Além disso, o menu "Registrar nota de aluno(a) em prova" também deve ser modificado para permitir indicar mais de um(a) aluno(a) no caso de trabalhos que permitam grupos de 2 ou mais alunos.

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:

- **Pauta final de disciplina:** dada uma disciplina, listar, para cada aluno(a) matriculado(a) nesta disciplina, sua matrícula, nome, notas obtidas nas avaliações, média parcial, nota na prova final e média final. A média parcial deve ser calculada levando-se em consideração os pesos de cada avaliação. Quando o aluno tiver média parcial igual ou superior a 7,0, não é necessário exibir sua nota de prova final e sua média final é igual à parcial. No caso do aluno ter média parcial inferior a 7,0, a média final é calculada por média simples entre média parcial e prova final;

- **Estatísticas por disciplina:** para cada disciplina, exibir seu código, nome e, para cada curso envolvido na disciplina (considerando Mestrado e Doutorado como cursos, além dos diferentes cursos da Graduação), calcular e exibir a média das notas finais dos alunos daquele curso e o percentual de alunos aprovados daquele curso;
- **Estatísticas por avaliação:** para cada avaliação cadastrada, exibir o código da disciplina, o código da avaliação, seu nome, data e a média das notas obtidas.

Neste ponto, a formatação de datas, números, etc. e a ordenação dos relatórios é livre. Formatações e ordenações específicas serão indicadas em outra etapa do trabalho.

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.

Além disso, seu programa deve agora verificar os casos em que o usuário forneceu entradas inválidas. Espera-se que sejam tratados exatamente 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 alunos, matrícula é valor numérico e para provas a data deve obedecer ao formato dd/mm/aaaa).	`Erro de formatação.`	`Erro de formatação.`
O mesmo código foi usado para dois cursos diferentes.	`Código repetido para curso: <código>.`	`Código repetido para curso: 5.`
A mesma matrícula foi usada para dois alunos diferentes.	`Matrícula repetida para aluno: <matrícula>.`	`Matrícula repetida para aluno: 2000990274.`

Situação	Mensagem	Exemplo
Código de disciplina usado na planilha de avaliações ou na planilha de alunos não foi definido na planilha de disciplinas.	Código de disciplina não definido usado <no/na> <objeto> <código/matrícula objeto>: <código>.	Código de disciplina não definido usado na avaliação ProgOO-P1: INF09375.
Peso de uma avaliação é um número negativo ou zero.	Peso de avaliação inválido para <código avaliação>: <peso>.	Peso de avaliação inválido para ProgOO-P1: -1.
Tipo de uma avaliação não é nem 'T' nem 'P'.	Tipo de avaliação desconhecido para <código>: <tipo>.	Tipo de avaliação desconhecido para ProgOO-P1: H.
Tamanho máximo de grupo foi especificado para prova.	Tamanho máximo de grupo especificado para a prova <código>: <tamanho>.	Tamanho máximo de grupo especificado para a prova PAC-PF: 2.
Tamanho máximo de grupo é um número negativo ou zero.	Tamanho máximo de grupo inválido para trabalho <código>: <tamanho>.	Tamanho máximo de grupo inválido para trabalho ProgOO-T1: -2.
Tipo de um aluno não é nem 'G' nem 'P'.	Tipo de aluno desconhecido para <matrícula>: <tipo>.	Tipo de aluno desconhecido para 2000990274: H.
Código de curso especificado para um aluno de graduação não foi definido na planilha de cursos.	Código de curso não definido usado no aluno <matrícula>: <código>.	Código de curso não definido usado no aluno 2000990274: 20.
Código de curso especificado para um aluno de pós-graduação não é nem 'M' nem 'D'.	Tipo de aluno de pós-graduação desconhecido para <matrícula>: <tipo>.	Tipo de aluno de pós-graduação desconhecido para 2000000007: H.
Código de avaliação usado na planilha de notas não foi definido na planilha de avaliações.	Código de avaliação não definido usado na planilha de notas, associado ao(s) aluno(s) <matrícula(s)>: <código>.	Código de avaliação não definido usado na planilha de notas, associado ao(s) aluno(s) 2000990274: ProgOO-F0.
Matrícula de aluno usada na planilha de notas não foi definida na planilha de alunos.	Matrícula de aluno não definida usada na planilha de notas, associada à avaliação <código>: <matrícula>.	Matrícula de aluno não definida usada na planilha de notas, associada à avaliação ProgOO-P1: 3000990274.

Situação	Mensagem	Exemplo
Nota na planilha de notas é um número negativo ou maior do que 10.	`Nota inválida para avaliação <código> do(s) aluno(s) <matrícula(s)>: <nota>.`	`Nota inválida para avaliação ProgOO-P1 do(s) aluno(s) 2000990274: 11,6.`
Disciplina não possui nenhuma avaliação cadastrada.	`A disciplina <código> não possui nenhuma avaliação cadastrada.`	`A disciplina INF09373 não possui nenhuma avaliação cadastrada.`
A planilha de notas contém nota de um aluno em uma avaliação de uma disciplina que ele não está matriculado.	`O aluno <matrícula> possui nota na avaliação <código> da disciplina <código>, porém não encontra-se matriculado nesta disciplina.`	`O aluno 2009837130 possui nota na avaliação PFE-P1 da disciplina FIS06326, porém não encontra-se matriculado nesta disciplina.`
A planilha de notas contém mais de uma nota para o mesmo aluno e a mesma avaliação, seja numa prova, seja num grupo de trabalho.	`O aluno <matrícula> foi registrado em mais de um grupo para a avaliação <código>.`	`O aluno 2000990274 foi registrado em mais de um grupo para a avaliação ProgOO-T1.`
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 – que agora devem ser representados no formato usado no Brasil, com vírgulas (ex.: 8,9) – os dados serão exportados utilizando ponto-e-vírgula como separadores. Por exemplo (mesmos dados das seções 2.4 e 2.6):

```
2000000005;Arya Stark;11

2000000005;ProgOO-P1;9,9
```

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.

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:

Planilha de cursos

```
<código>;<nome>
```

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

Planilha de disciplinas

```
<código>;<nome>
```

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

Planilha de avaliações

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

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

Planilha de alunos

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

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

Planilha de notas

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

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

5.2. Formato dos arquivos de saída

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

Pauta final (por disciplina)

Nome do arquivo: 1-pauta-<código>.csv. Um para cada disciplina.

Cabeçalho: `Matrícula;Aluno;<código 1>;<código 2>;...;<código N>;Média Parcial;Prova Final;Média Final`

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

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

Estatísticas por disciplina

Nome do arquivo: 2-disciplinas.csv.

Cabeçalho: `Código;Disciplina;Curso;Média;% Aprovados`

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

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

Estatísticas por avaliação

Nome do arquivo: 3-avaliacoes.csv.

Cabeçalho: `Disciplina;Código;Avaliação;Data;Média`

Formato: `<código disciplina>;<código avaliação>;<nome avaliação>;<data avaliação>;<média notas>`

Este relatório deve ser ordenado primeiro pelo código da disciplina (crescente), seguido pela data da avaliação (crescente). A data deve ser impressa no formato dd/mm/aaaa e a média das notas em cada avaliação com 2 casas decimais. A média dos trabalhos deve ser feita dividindo pelo número de grupos, não pelo número de alunos. As provas finais das disciplinas não devem constar neste relatório.

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:

- `-c <arquivo>`: planilha de cursos;
- `-d <arquivo>`: planilha de disciplinas;
- `-p <arquivo>`: planilha de avaliações (provas e trabalhos);
- `-a <arquivo>`: planilha de alunos;
- `-n <arquivo>`: planilha de notas.
- `--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 cursos.csv, disciplinas.csv, avaliacoes.csv, alunos.csv e notas.csv, o comando seria:

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

Quando especificada a opção `--read-only`, o programa deve ler os arquivos de entrada, montar as estruturas de objetos em memória e serializar esta estrutura em um arquivo chamado **`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 -c cursos.csv -d disciplinas.csv -p avaliacoes.csv -a alunos.csv -n notas.csv
```

É equivalente a:

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

Por fim, a versão C++ do programa não 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.

Nestes 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 devem criar um repositório privativo no GitHub ou em serviço similar e convidar o professor para que tenha acesso ao repositório.[1] O repositório deve ter instruções de como compilar e executar o programa no arquivo README.md. Usaremos o Ant para automatizar estas tarefas (vide Seção 7.2), porém quem se interessar por uma ferramenta mais robusta pode instalar e experimentar o Maven.[2]

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 também agendar um horário para entrevista com o professor sobre o código-fonte e o funcionamento do trabalho. Instruções para agendamento de horário com o professor encontram-se no site http://www.inf.ufes.br/~vitorsouza/atendimento/. Atenção aos seguintes detalhes sobre o agendamento:

- O sistema só permite agendamentos com antecedência mínima de 1 dia e máxima de 2 semanas (i.e., nenhum horário disponível antes ou depois destes prazos);
- 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 (preferencialmente) ou ao link do Google Meet (também possível) para a entrevista pontualmente no dia e hora marcados. A apresentação do trabalho pode ser feita em computador dos alunos ou no computador do professor. No segundo caso, o código-fonte do trabalho deve estar atualizado no repositório de código compartilhado com o professor.

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.

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.

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.

---

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

[2] 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

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[3]);
- 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.). Critérios análogos para o trabalho em C++;
- Respostas dadas pelos membros do time durante a entrevista.

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 entrevista) 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 ou superior.[4]

Script de testes Java

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

1. Obtenha o arquivo `progoo-20231-script-java-v1.zip` (ou versão mais recente);
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 ProgOO 2023/1 - 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.

---

[3] Há um formatador disponível para a IDE Eclipse que também funciona no VSCode, vide instruções para Eclipse no antigo Manual do Marvin e para VSCode no Catálogo do LabES.

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

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 ProgOO 2023/1 - Trabalho Java

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em 1-pauta-INF09324.csv
[I] Testando professor: teste 01, tudo OK em 1-pauta-INF09331.csv
[I] Testando professor: teste 01, tudo OK em 1-pauta-INF09373.csv
[I] Testando professor: teste 01, tudo OK em 2-disciplinas.csv
[I] Testando professor: teste 01, tudo OK em 3-avaliacoes.csv
[I] Testando professor: teste 02
[I] Testando professor: teste 02, serialização OK!
[I] Testando professor: teste 03
[I] Testando professor: teste 03, serialização OK!
[I] Testando professor: teste 03, tudo OK em 1-pauta-INF09324.csv
[I] Testando professor: teste 03, tudo OK em 1-pauta-INF09331.csv
[I] Testando professor: teste 03, tudo OK em 1-pauta-INF09373.csv
[I] Testando professor: teste 03, tudo OK em 2-disciplinas.csv
[I] Testando professor: teste 03, tudo OK em 3-avaliacoes.csv
[I] Testando professor: teste 04
[I] Testando professor: teste 04, tudo OK em output.txt

... (muitas outras linhas de resultado)

[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.

Nota 2: caso a sua saída esteja diferente da saída do professor e você considere que a sua saída esteja correta, entre em contato com o professor imediatamente explicando porque você acredita que a saída esperada do script de testes está errada. Tais saídas esperadas são geradas pela implementação do trabalho feita pelo professor que, como qualquer código, pode conter erros.

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 `-c cursos.csv -d disciplinas.csv -p avaliacoes.csv -a alunos.csv -n notas.csv` como parâmetros.
`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`.
`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="TrabalhoProgOO_2023_1" default="compile" basedir=".">
      <description>Arquivo de build do trabalho de ProgOO, 2023/1.</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 em modo normal.">
            <java classname="${mainClass}">
                  <arg value="-c" />
```

```
                <arg value="cursos.csv" />
                <arg value="-d" />
                <arg value="disciplinas.csv" />
                <arg value="-p" />
                <arg value="avaliacoes.csv" />
                <arg value="-a" />
                <arg value="alunos.csv" />
                <arg value="-n" />
                <arg value="notas.csv" />
                <classpath>
                        <pathelement path="${bin}" />
                </classpath>
        </java>
    </target>

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

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

Caso o time esteja usando alguma biblioteca externa, deve inclui-la como parte do *classpath* do Java tanto para compilação quanto para execução. Seguem abaixo trechos do exemplo anterior, modificados para incluir uma biblioteca chamada bib.jar colocada dentro de uma subpasta chamada lib.

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

```
    <!-- Execução normal. -->
    <target name="run" depends="compile" description="Executa o programa em modo normal.">
        <java classname="${mainClass}">
                <arg value="-c" />
                <arg value="cursos.csv" />
                <arg value="-d" />
                <arg value="disciplinas.csv" />
                <arg value="-p" />
```

```
            <arg value="avaliacoes.csv" />
            <arg value="-a" />
            <arg value="alunos.csv" />
            <arg value="-n" />
            <arg value="notas.csv" />
            <classpath>
                  <pathelement path="${bin}" />
                  <pathelement location="lib/bib.jar"/>
            </classpath>
      </java>
</target>
```

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 `-c cursos.csv -d disciplinas.csv -p avaliacoes.csv -a alunos.csv -n notas.csv` como parâmetros.
`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).

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.