MCTA018 - Programação Orientada a Objetos
Plano de ensino

Prof. Diogo S. Martins
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Q2 2022
v.06/06

1 Informações básicas

- TPI: 2-2-4
- Horários oficiais:
  - DA1:

ter	08-10h	S-301-2	semanal
qui	10-12h	407-2	semanal

  - NA1:

ter	19-21h	S-311-2	semanal
qui	21-23h	407-2	semanal

- Ferramentas de comunicação:
  - Discord

    Para cadastrar-se na sala, utilizar link disponível no Moodle.
  - Comunicação com o professor
    - Prioritariamente via Discord, por mensagem pública ou privada
    - Por email (caso Discord esteja indisponível): `santana.martins@ufabc.edu.br`
- Plantões:

ter	10-11h	S528-2 (presencial) e Google Meet (remoto)
ter	18-19h	S528-2 (presencial) e Google Meet (remoto)

  O objetivo do plantão é esclarecer dúvidas e/ou reforçar temas vistos nas aulas ou outros materiais.

  Aonde: tanto a participação presencial quanto a remota precisam ser agendadas previamente no Google Calendar da disciplina (vide link de agendamento no Moodle).
- Sala no Moodle: `poo-2022.2-da3`

  `https://moodle.ufabc.edu.br/course/view.php?id=1851`

  Todos já foram convidados para a sala, em caso de dificuldade de acesso, entrar em contato com o professor.

2 Descrição da disciplina

A disciplina aborda os fundamentos do paradigma de programação orientada a objetos, e alguns padrões de projeto (*design patterns*) fundamentais.

3 Requisitos recomendados

Para participar dessa disciplina é recomendação oficial ter cursado e sido aprovado em:

- BCM0505 - Processamento da Informação
- MCTA028 - Programação Estruturada

4 Objetivos

- Familiarizar-se com os conceitos básicos do paradigma orientado a objetos, a saber: objetos, classes, passagem de mensagens, encapsulamento, herança e polimorfismo;
- Ser capaz de transferir os conceitos básicos para uma linguagem de programação orientada a objetos, por exemplo, Java;
- Ser capaz de projetar software orientado a objetos;
- Familiarizar-se com os padrões de projeto (*design patterns*) básicos, bem como aplicá-los em problemas concretos.

Ao final do curso, espera-se que o aluno, aprovado com conceito satisfatório, possua habilidades que permitam, a partir de um conjunto de requisitos, projetar software orientado a objeto que seja eficiente, confiável, seguro e de fácil manutenção.

5 Bibliografia

1. Liang, D. Y. Introduction to Java Programming and Data Structures, 11th. edition. Pearson, 2018.
2. Deitel & Deitel. Java, How to Program (Early Objects), 11th. edition. Pearson, 2017.
3. Sedgewick, R. & Wayne, K. Computer Science: An Interdisciplinary Approach. 1st. edition. Addison-Wesley Professional, 2016.

6 Metodologia de ensino-aprendizagem

Vídeo-aulas. Cada aula será disponibilizada com antecedência no Moodle. O conteúdo englobará, além de aulas teóricas expositivas, também tutoriais com práticas de programação, os quais, idealmente, devem ser seguidos de forma síncrona, ou seja, que os alunos tentem implementar os exemplos ao mesmo tempo em que assistem ao vídeo.

Indissociabilidade entre teoria e prática. O conteúdo dos vídeos combina os conteúdos de teoria e prática com dinâmica fluída, ou seja, teremos breves resumos teóricos seguidos de sessões mais longas de programação com introdução de novos conceitos teóricos à medida que mostrem-se necessários nos contextos mais adequados.

Aprendizagem ativa. [1] As atividades das videoaulas enfatizam a análise, a síntese e a avaliação dos conteúdos, em detrimento do consumo passivo de conhecimento pronto. A abordagem segue a linha do "aprender fazendo" ao invés do "aprender olhando". Por esta razão, reforça-se a importância de seguir os tutoriais de programação sincronamente aos vídeos. Além disso, a dinâmica das aulas depende da iniciativa individual do aluno, o qual deve tornar-se protagonista do seu aprendizado, resolvendo os problemas e trazendo suas dúvidas, caso contrário seu aproveitamento nas atividades presenciais será baixo.

Aula invertida. [2] O tempo da aula será utilizado essencialmente para a resolução de problemas e discussão de dúvidas. A aquisição do conhecimento teórico-conceitual e prático é abordado via estudo prévio com os materiais indicados no roteiro da respectiva aula, os quais serão compostos, basicamente, de vídeo-aulas e textos. Cada aula é estruturada em ciclos, cada ciclo composto dos seguintes momentos:

[1] `https://en.wikipedia.org/wiki/Active_learning`
[2] `https://en.wikipedia.org/wiki/Flipped_classroom`

1. **Estudo prévio**. Alunos estudam os conteúdos da aula, com base nos recursos indicados pelo professor (textos, vídeos, tutoriais interativos, etc.), antes da aula;
2. **Problemas e dúvidas**. Durante as aulas práticas, teremos solução de problemas ou mini-projetos sobre o conteúdo estudado. Durante as aulas teóricas, teremos discussão de dúvidas relacionadas ao material de estudo e exercícios correlatos ou às atividades avaliativas.

Dinâmica das atividades presenciais. Dispomos de um dia de teoria (em sala de aula) e um dia de prática (em laboratório). Dada essa configuração, utilizaremos a infra-estrutura dos espaços físicos do seguinte modo:

- **Aulas teóricas**. Como não dispomos de computadores individuais, teremos eventualmente exercícios focados na análise de códigos prontos, ou questões mais conceituais, mas, principalmente, a discussão de dúvidas levadas pelos alunos. Para evitar riscos de contaminação pelo compartilhamento de materiais impressos, todos os conteúdos gerados pelo professor serão digitais e disponibilizados no Moodle. Similarmente, os materiais trazidos pelos alunos também devem estar em formato digital. Sugere-se que, dentro das possibilidades, os alunos tragam equipamentos individuais (e.g. notebook, smartphone, tablet, etc.) ou que mantenham, alternativamente, os códigos em repositórios públicos (e.g. GitHub), para que o professor possa clonar esses repositórios e discutí-los com os interessados durante as aulas.
- **Aulas práticas**. Como neste espaço há disponibilidade de computadores individuais, teremos exercícios regulares de programação, com enunciados disponíveis no Moodle. As atividades serão sempre formativas, i.e., não valerão nota. O tempo em aula pode ser usado também para a discussão de dúvidas.

7 Avaliação

A avaliação consiste nos componentes dados pela Equação 1, onde:

$$N_F = 0.5 \cdot N_{atv} + 0.25 \cdot N_{P1} + 0.25 \cdot N_{P2} \tag{1}$$

- N_{atv} é a média de 75% das atividades com as melhores notas. Como total para calcular a porcentagem, consideramos todas as atividades enunciadas durante o quadrimestre;
- N_{P1} e N_{P2} são as notas da Prova 1 e Prova 2, respectivamente;

O conceito final será obtido de acordo com a Equação 2.

$$C_F = \begin{cases} \texttt{A}, \text{ se } N_F \in [8.5, 10.0] \\ \texttt{B}, \text{ se } N_F \in [7.0, 8.5) \\ \texttt{C}, \text{ se } N_F \in [5.5, 7.0) \\ \texttt{D}, \text{ se } N_F \in [5.0, 5.5) \\ \texttt{F}, \text{ se } N_F \in [0.0, 5.0) \\ \texttt{O}, \text{ se ausência exceder } 25\% \end{cases} \tag{2}$$

Sobre as Atividades

Teremos um total de 4 atividades somativas, ou seja, valendo nota. Caberá ao aluno demonstrar disciplina e organização de tempo para executá-las. Para computar a N_{atv}, serão consideradas as três maiores notas obtidas[3].

Auto-avaliação formativa. As atividades contarão, sempre que possível, com recursos de verificação automática de corretude e estilo de programação, com o objetivo de auxiliar o estudante a detectar, de modo automatizado, problemas recorrentes de programação. Na ausência de recursos de verificação automática, o estudante contará com uma rubrica para analisar a aderência do seu resultado ao que é esperado. Cabe ao aluno garantir que o programa passe por *todas* as verificações automáticas. Além disso, nas atividades em que haja rubrica, deve garantir que todos os critérios sejam atendidos. Submissões com falhas terão descontos substanciais na nota.

[3] A justificativa para essa política consta no mecanismo de avaliação substitutiva.

Avaliação somativa. Além dos testes automáticos, o professor analisará as submissões em atenção aos aspectos qualitativos e éticos detalhados neste plano de ensino. Ou seja, passar em todos os testes automáticos não implica necessariamente em nota máxima, visto que análise posterior pode implicar em desconto parcial ou total na nota da atividade.

Prazos. Cada atividade tem prazo de 15 dias para entrega. É possível entregar atividades atrasadas, sendo o limite máximo para entrega a data da prova 2. Atividades atrasadas terão desconto de nota proporcional ao atraso.

Sobre as Provas

As provas terão as seguintes características:

- Presenciais;
- Com duração de 100 minutos;
- 100% práticas, envolvendo problemas de programação;
- Efetuadas no computador do laboratório (não é permitido o uso de computador pessoal nem de dispositivos móveis);
- Com consulta, mas somente a materiais offline (ou seja, sem acesso à Internet);
- Entregue no sistema de submissão da disciplina.

O prazo será controlado automaticamente pelo Moodle. Não serão aceitas, em hipótese alguma, provas entregues fora do prazo regulado pelo Moodle ou enviadas por meios de submissão alternativos.

Sobre o sistema de submissão

Tanto as atividades quanto as provas serão submetidas via Moodle e via GitHub Classroom. O conteúdo da primeira semana de aula envolve treinamento sobre o uso combinado destas duas plataformas de submissão.

Critérios de avaliação

Todas as avaliações somativas, isto é, as atividades e as provas, qualificam-se como atividades de programação.

A nota máxima de cada atividade será obtida apenas se a mesma for entregue no prazo e executada correta e completamente.

Os programas solicitados em atividades de avaliação serão submetidos aos seguintes tipos de verificações:

- **Verificações automáticas**. Testes de corretude, de estilo de programação, de qualidade de software, de erros comuns e de detecção de plagiarismo.
- **Verificações manuais**. O professor inspecionará os programas para verificar os seguintes critérios gerais:
  - **Eficiência**: os programas desenvolvidos deverão ter bom desempenho, o que pode englobar o tratamento adequado dos seguintes fatores:
    - Ler e escrever dados nas quantidades mínimas necessárias para resolver o problema;
    - Não desperdiçar memória primária (RAM);
    - Acessar memória secundária (disco) somente quando necessário e sem redundância;
    - entre outros.
  - **Acurácia**: o programa deverá atender adequadamente a todos os requisitos enunciados para a atividade;
  - **Corretude**: a solução deverá estar funcionalmente correta;
  - **Estrutura e organização do código**: atentar principalmente aos seguintes aspectos:
    - **Auto-documentação**: nomes intuitivos para variáveis, métodos, classes, interfaces e pacotes;

    - **Modularização**: métodos e classes com alta concisão e baixo acoplamento, isto é, que sejam em sua maioria curtos, e que realizem preferencialmente uma única tarefa;
    - **Comentários**: documentação completa porém ao mesmo tempo concisa (sem poluição visual, apenas nos lugares adequados e necessários).
  - **Autenticidade**: o código é original e não foi copiado de outras fontes (i.e. web, livros, terceiros, etc.), sob pena das sanções previstas no código de honra.

Mecanismos de avaliação substitutivos

Teremos dois mecanismos de avaliação substitutiva:

- Para as atividades, como consideraremos 75% das maiores notas, para substituir basta entregar com atraso até atingir no mínimo 75% de entregas;
- Para provas, é preciso apresentar justificativa prevista na Resolução ConsEPE 227. O documento de justificativa deve ser encaminhado por email ao professor, que providenciará o agendamento da prova substitutiva.

Mecanismo de recuperação

A recuperação será aplicada apenas aos alunos que tiverem conceito final D ou F. Ocorrerá no início do quadrimestre subsequente, em formato similar ao estabelecido para as provas regulares.

A nota obtida na prova de recuperação (N_R) será usada para obter a nota final com recuperação (N_{FR}), que consiste na média estabelecida pela Equação 3.

$$N_{FR} = 0.5 \cdot N_F + 0.5 \cdot N_R \tag{3}$$

O conceito final obtido na recuperação (C_{FR}) é o conceito que entrará no histórico, obtido de acordo com os limiares para a nota final de recuperação (N_{FR}) dados pela Equação 4.

$$C_{FR} = \begin{cases} \texttt{C}, \text{ se } N_{FR} \geq 5.5 \\ \texttt{D}, \text{ se } N_{FR} \in (5.0, 5.5) \\ \texttt{F}, \text{ se } N_{FR} \leq 5.0 \end{cases} \tag{4}$$

No caso dos alunos que não participarem da recuperação, $C_{FR} = C_F$.

8 Código de honra

A aprovação na disciplina é baseada exclusivamente no esforço e trabalho pessoal do discente, ao qual cabe garantir que não ajudará ou receberá ajuda não-permitida em qualquer atividade avaliativa (e.g. provas, trabalhos, listas, etc.).

Exemplos de violação do código de honra incluem:

- Copiar atividades avaliativas (e.g. listas, trabalhos, provas, etc.) ou permitir que outros discentes copiem suas atividades avaliativas;
- Colaboração não-permitida entre indivíduos ou grupos (e.g. oferecer vantagens em troca de soluções prontas, doar trechos para o trabalho de outro grupo, etc.);
- Permitir que outros assumam sua identidade em atividades avaliativas (e.g. entregar trabalho que não fez ou permitir que outros façam provas por você);
- Plagiarismo (i.e. aplicável a textos, programas de computador, etc.), o que envolve copiar porções significativas de textos ou programas de terceiros, sem atribuição de autoria ou, mesmo que haja atribuição de autoria, demonstre-se que não houve trabalho original significativo;
- Receber ou conceder ajuda em atividades avaliativas quando o contexto mostra que não é sensato receber tal ajuda.

Como consequências de violação do código de honra tem-se:

- Reprovação automática na disciplina, com conceito F, sem direito ao mecanismo de recuperação;
- Denúncia na Comissão de Transgressões Disciplinares Discentes da Graduação, a qual decidirá sobre a punição adequada à violação, o que pode levar a advertência, suspensão ou desligamento, de acordo com os arts. 78-82 do Regimento Geral da UFABC.

9 Cronograma de aulas

O cronograma a seguir pode variar de acordo com o aproveitamento aferido nas turmas durante o quadrimestre.

Aula #	Data	T/P	Tema
1	07/06	T	Configuração de ambientes
2	09/06	P	Revisão Java (Programação estruturada)
3	14/06	T	Objetos
	16/06		Feriado
4	21/06	T	Objetos
5	23/06	P	Classes
6	28/06	T	Classes
7	30/06	P	Herança
8	05/07	T	Herança
9	07/07	P	Polimorfismo
10	12/07	T	Polimorfismo
11	14/07	P	Prova 1
12	19/07	T	Exceções
13	21/07	P	Exceções
14	26/07	T	GUI
15	28/07	P	GUI
16	02/08	T	Arquivos
17	04/08	P	Arquivos
18	09/08	T	Programação genérica
19	11/08	P	Programação genérica
20	16/08	T	Aplicações
21	18/08	P	Aplicações
22	23/08	T	Aplicações
23	25/08	P	Prova 2
24	29/08	P	Prazo-limite para atividades atrasadas