Universidade Estadual da Paraíba (UEPB)
Curso: Computação
Disciplinas: Linguagem de Programação II e Laboratório de Programação II
Professor: Kláudio Medeiros
Semestre: 2024.1

Projeto de Linguagem e Laboratório de Programação 2

Observações gerais

- Os projetos podem ser feitos em equipes de, no máximo, 4 (quatro) pessoas.
- Os alunos de um mesmo grupo podem pertencer a turmas diferentes.
- O projeto compõe as notas das Unidades I e II de Laboratório de Programação II, além de metade da nota da Unidade II de Linguagem de Programação II.

Preparação para projeto

- Previamente, é pedido aos alunos que indiquem os grupos formados e os temas de projeto.
- A alocação de grupos para temas deve ser informada por algum aluno como resposta da postagem da atividade no Classroom.
- Os alunos sem grupo ou tema precisam entrar em contato com o professor ou entre si para definição de grupo e tema.

Realização de projeto

- As implementações pedidas constam abaixo. Além disso, o grupo deve mandar comentários sobre o projeto, realizando os comentários Javadoc para a classe e o que a compõe, assim como justificando as decisões de orientação a objeto tomadas (e.g. por que um atributo x é static, para que foi utilizada sobrecarga em um método).
- Cada projeto deve fazer uso de, pelo menos, uma API externa para alguma finalidade (e.g. Swing para interface gráfica, JDBC para conexão a banco de dados). Quanto mais, melhor.
- Os grupos irão listar alguns requisitos que o sistema tem. O próprio grupo pode definir esses requisitos, recebendo apenas um feedback e um ok do professor.

Definição de classes, atributos e métodos

- Criem classes para representar as entidades relevantes para o contexto do projeto (e.g. um cliente, um item a ser vendido).

- Pensem em quais classes seriam relevantes e aplicáveis no contexto do projeto.
- Façam o mesmo para definir os atributos (e.g. nome) e os métodos (e.g. calcularAlgo()) dessas classes.

- Aplique boas práticas de programação na criação das classes, dos atributos e dos métodos (e.g. encapsulamento, ocultação de informação, divisão de funcionalidades em métodos).

Controle de acesso aos atributos e métodos

- Utilize os modificadores de acesso de atributos e métodos de maneira adequada (e.g. atributos privados, métodos de acesso públicos, proteger métodos que servem apenas como métodos auxiliares para uma funcionalidade maior).

Sobrecarga (overload) de método

- Faça uso de sobrecarga de método em, pelo menos, um método do projeto (e.g. uma funcionalidade, um construtor). Justifique a importância desse método ser sobrecarregado no contexto do projeto.

Atributos e métodos static

- Faça com que, pelo menos, um atributo ou um método seja static. Justifique por que esse atributo ou método deveria ser static.

Utilização de coleções (e.g. ArrayList)

- Utilize listas de tamanho dinâmico (e.g. ArrayList) para representar coleções de coisas dentro do contexto do projeto (e.g. lista de clientes). É interessante também considerar o uso de mapas (Maps) ou conjuntos (Sets).

Herança

- Verifique a hierarquia entre as classes definidas, considerando alguns pontos como os abaixo:

    - Quem herda de quem?
    - Que classes não deveriam ser instanciadas? Por que?
    - Como estabelecer as relações de herança de modo que mais código seja reutilizado?

- Que métodos são abstratos e que métodos possuem implementações padrão que podem ser substituídos em subclasses?

- A herança deve ser utilizada, pelo menos, uma vez no projeto (veja onde e por que).

Interfaces

- Verifique possíveis “contratos” entre as classes definidas, considerando alguns pontos abaixo:
    - Que interfaces poderiam ser criadas?
    - Quais os métodos definidos nessas interfaces que devem ser implementados?
    - Por que utilizar interfaces nos casos em que foram utilizadas?
    - Em que situações é melhor utilizar interface e em que situações seria melhor utilizar herança?

- O projeto deve conter, pelo menos, uma interface (veja onde e por que)

Composição

- Verifique em que situações poderiam ser utilizadas a composição (por exemplo, no lugar de herança). Justifique no seu código a utilização de composição, enfatizando os benefícios dessa utilização.

Polimorfismo

- Verifique em que situações no projeto o polimorfismo ocorre. Comente sobre o uso de polimorfismo e a relevância dele na funcionalidade onde o polimorfismo ocorre.

Tratamento de exceções

- O projeto poderá fazer uso de tratamento de exceções para alguma situação excepcional. Caso o grupo opte por outras maneiras de evitar ou resolver essas exceções, poderá fazer uso dessas outras maneiras, dispensando o tratamento com try/catch/finally.

- É pedido que o grupo comente no código qual foi a maneira escolhida de tratar situações específicas (e.g. se utilizaram if para verificar um valor, se utilizaram Exception para tratar de uma regra de negócio).

- O tratamento de exceções não é obrigatório, mas deve ser utilizado em situações que pedem a sua utilização. Analisem as situações do seu projeto, pois é extremamente provável que ela precise ser utilizada em algum lugar.

Testes de unidade

- Serão criados testes de unidade para as classes criadas e funcionalidades feitas.

- Serão levados em consideração na avaliação a abrangência dos testes. Quanto mais cenários possíveis forem testados, melhor.

UML básico

- Os grupos irão criar diagramas (e.g. de classe) para representar o projeto feito.

- Os diagramas pedidos são:
    - diagramas de classe (obrigatório)
    - diagramas de caso de uso (pelo menos um exemplo)
    - diagramas de sequência (pelo menos um exemplo)
    - diagramas de atividade (pelo menos um exemplo)
    - diagramas de estado (pelo menos um exemplo)

Apresentação

- Os detalhes de apresentação serão dados ao longo da unidade.

Critérios de avaliação

- Para o projeto, serão considerados:

    - Funcionamento (se o sistema está funcionando corretamente)
    - Utilização adequada dos conceitos vistos ao longo da disciplina
    - Utilização adequada de boas práticas de programação no geral
    - Complexidade do sistema (recursos externos, dificuldade, etc)

- Para a apresentação, serão considerados:

    - Domínio do Conteúdo
    - Clareza na Comunicação
    - Utilização dos Recursos
    - Distribuição do Tempo

Cronograma de entregas e apresentações

Unidade I

	Unidade I	
25/03	Descrição do projeto e lista de requisitos	até 1,0 ponto
05/04	Diagrama de classes (UML)	até 2,0 pontos
15/04	Protótipo do projeto; a apresentação será focada nos conceitos de orientação a objeto aplicados (tudo até interfaces, exceto este)	até 2,0 pontos
22/04	Protótipo do projeto com conceitos de orientação a objeto aplicados (tudo até interfaces, exceto este) e funcionalidades feitas (s/ API externa); o grupo indicará	até 5,0 pontos

	quais funcionalidades fez e quais não fez, baseado na lista de requisitos	
	Unidade II	
29/04	Diagramas UML (de classe atualizado e os demais);	até 1,5 ponto
06/05	Tratamento de exceções e outros tipos de erros que possam acontecer no sistema	até 0,5 ponto
13/05	Testes de unidade	até 1,0 ponto
23/05	Entrega final do projeto (com pelo menos uma API externa)	até 7,0 pontos