Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Engenharia de Software
AD2 2° semestre de 2019.

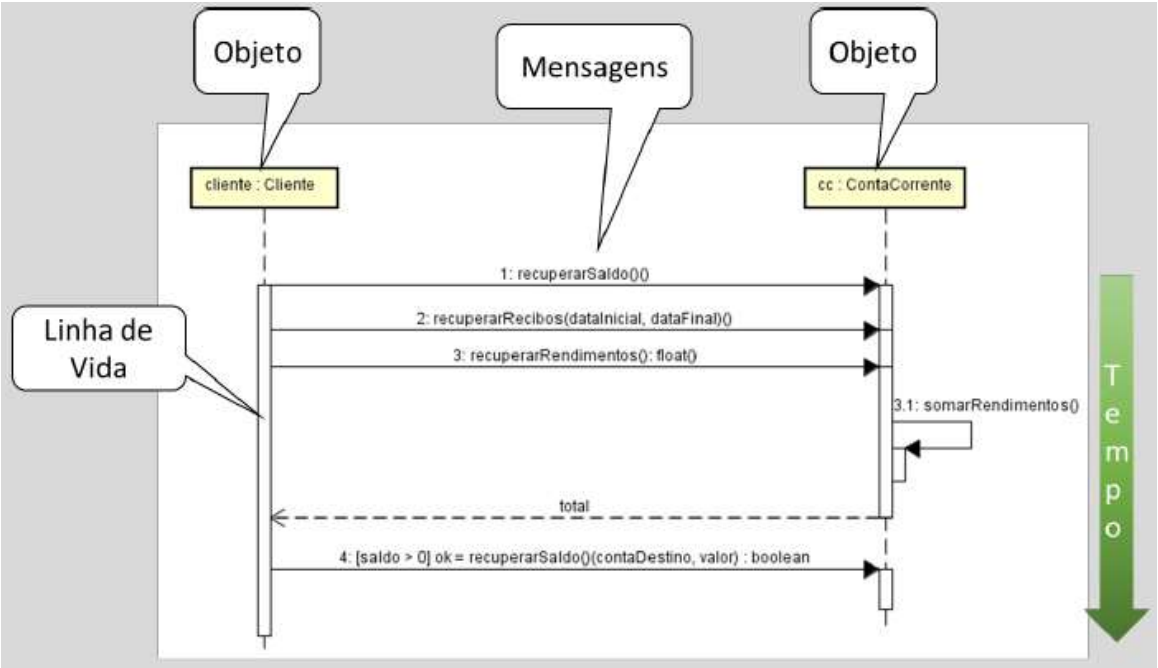
Atenção: Como a AD é individual, caso seja constatado que provas de alunos distintos são cópias uma das outras, independentemente de qualquer motivo, a todas será atribuída a nota ZERO. As soluções para as questões podem sim, ser buscadas por grupos de alunos, mas a redação final das respostas para as questões da prova tem que ser individual!

1. O que diferencia linguagens de programação interpretadas das linguagens de programação compiladas? De dois exemplos de cada tipo e indique para que categoria de software cada uma delas poderia ser mais adequada e por quê. (Valor: 1,5 ponto)

   Nas linguagens de programação compiladas, o código-fonte passa por uma fase inicial de compilação no qual é transformado em código-objeto, ou seja, um código que somente pode ser executado na plataforma para a qual foi compilado. Em seguida, os módulos são reunidos em um programa executável único por um processo chamado de linquedição. Nas linguagens de programação interpretadas, o código-fonte é compilado para uma linguagem intermediária, que é executada em um interpretador, um programa capaz de ler esta linguagem intermediária e executar os seus comandos na plataforma de interesse.

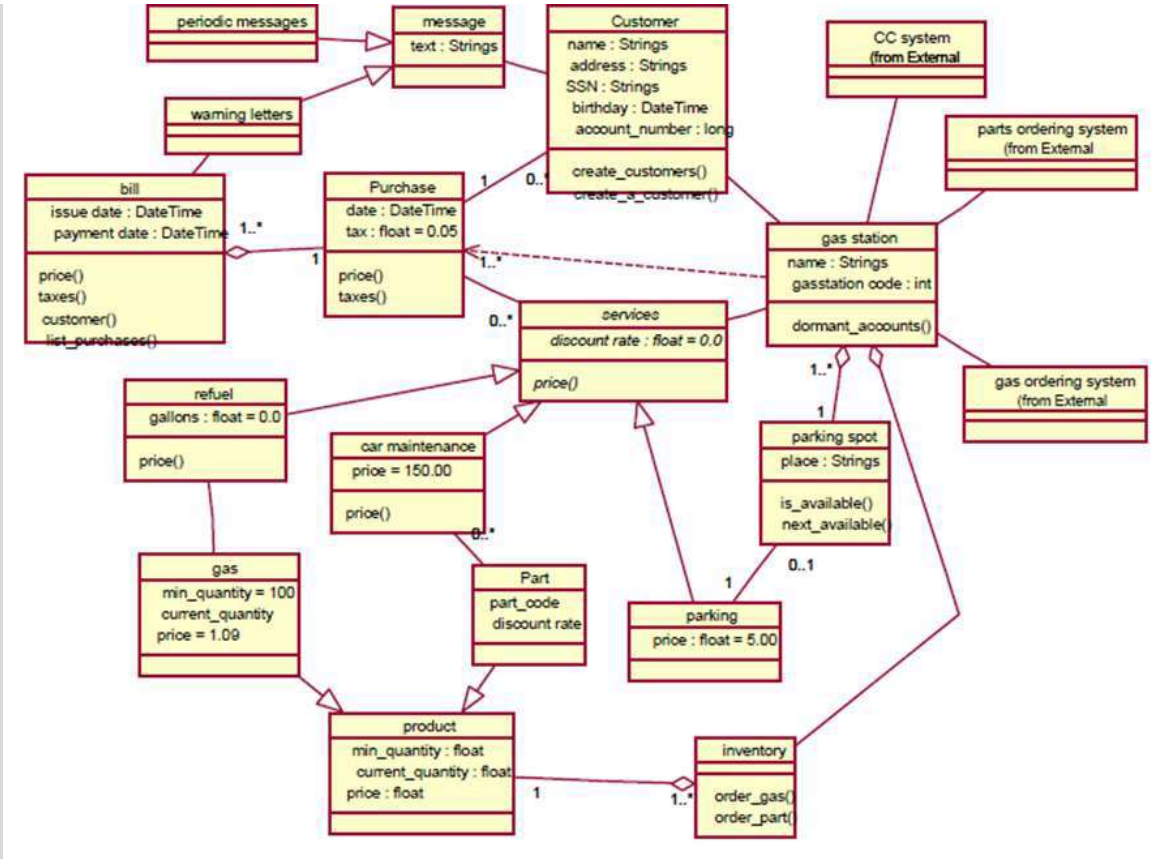
2. Quais são os componentes de um diagrama de sequência da UML? Mostre um diagrama de sequência e explique como estes componentes se relacionam e o quê cada um representa no diagrama. (Valor 1,5 ponto)

   O Diagrama de Sequência permite representar como os objetos de um sistema OO trocam mensagens entre si para cumprir com algum objeto (por exemplo, realizar um caso de uso).
   Os principais componentes de um diagrama de sequência são Objetos, Linha de Vida e Mensagens.
   A Figura a seguir apresenta um exemplo de um diagrama de sequência. Perceba que a linha do tempo é representada na vertical.



3. O que representa o conceito de acoplamento em software. Como podemos observar acoplamento no software? Como medir acoplamento? Apresente um exemplo de excesso de acoplamento em software. (Valor: 1,5 ponto)

Acoplamento indica o grau de dependência entre os componentes (ou módulos) do software. Um software com baixo acoplamento é sempre desejável. Um software com alto grau de acoplamento indica que a dependência entre módulos é grande, aumentando o risco de efeito colateral e, consequentemente, de falhas. Em relação a manutenção, o esforço aumenta (e os riscos de falha também) tendo em vista que alterações realizadas em um determinado módulo tem o potencial de afetar os módulos dependentes.
Uma métrica que pode ser usada para medir acoplamento é a CBO (*coupling between objects*). Basicamente ela mede o número de relacionamentos existente entre as classes que irão instanciar os objetos.
No diagrama de classes abaixo, a classe Gas Station possui acoplamento alto (8) em comparação com a classe Periodic Messages (1). Portanto, objetos da classe Gas Station são altamente acoplados, dependendo de 8 outros objetos para desempenhar suas responsabilidades.



4. Explique o que é teste funcional. Apresente três exemplos de critérios de teste que podem ser utilizados para projetar casos de teste para este tipo de teste. (valor: 1,5 ponto)

   Os testes funcionais, também chamados testes de caixa fechada, enxergam o software de forma macro, na perspectiva do usuário, sem perceber detalhes de sua estrutura interna. Os testes funcionais baseiam-se na especificação do software para derivar os casos de testes. Exemplos de critérios que podem ser aplicados para testes funcionais são particionamento em classes de equivalência e Análise de Valor Limite.

5. Atividade de pesquisa I (não vale copiar e colar! Você deve pesquisar e explicar com suas palavras. Indique fontes alternativas que usar!): (valor 2,0 pontos)

   Leia o Código de ética do Profissional de Informática da Sociedade Brasileira de Computação (http://www.sbc.org.br/institucional-3/codigo-de-etica). Como ele se compara ao Código de ética e de Prática Profissional da Engenharia de Software, segundo as recomendações da ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices (https://www.computer.org/cms/Computer.org/professional-education/pdf/doc.pdf)? Quais a consequências em não seguirmos um código de ética em nossos projetos de engenharia de software?

Tanto o código de ética da SBC quanto o da ACM/IEEE-CS fornecem diretrizes para a conduta dos profissionais de informática.
O código da SBC consiste em um conjunto de 12 artigos que descrevem os deveres dos profissionais de informática. Já o código da ACM/IEEE-CS consiste em um conjunto de 8 artigos.
Uma diferença entre os códigos é que o ACM/IEEE-CS, em sua versão completa, possui um detalhamento dos 8 artigos propostos.

6. Atividade de pesquisa II (não vale copiar e colar! Você deve pesquisar e respondem com suas palavras. Indique fontes alternativas que usar!): (valor 2,0 pontos)

   A percepção da qualidade do produto de software tem evoluído ao longo do tempo. Abordagens clássicas e ainda muito importantes, observam a qualidade externa do produto através, por exemplo, da presença de defeitos. Entretanto, a evolução dos processos de desenvolvimento tem trazido preocupações com a evolução dos produtos. Ciclos interativos e incrementais dependem não apenas da qualidade externa, mas, principalmente, da qualidade interna do produto. Neste sentido o conceito de Dívida Técnica (Technical Debt) tem sido atualmente discutido e aplicado. Portanto, o que é Dívida Técnica? Como um engenheiro de software pode identificar ou medir a dívida técnica no produto? Quais são seus efeitos na evolução do produto? Toda dívida técnica deve ser paga? Exemplifique.

   O termo **dívida técnica** é uma metáfora utilizada no contexto de desenvolvimento de software referindo-se a tarefas atrasadas e artefatos imaturos que constituem uma "dívida" porque incorrem em custos adicionais no futuro, sob a forma de aumento de custo de mudança durante a evolução e a manutenção.
   Por exemplo, imagine que uma funcionalidade do sistema esteja finalizada, mas o código não está aderente aos padrões de qualidade estabelecidos pela organização (nome de variáveis muito longos, número grande de comandos em cada método, etc.). Nesse cenário, pode haver uma decisão em disponibilizar o software para os usuários, mesmo com os problemas no código fonte. É importante perceber que esses problemas não são percebidos de imediato pelos usuários. Entretanto, cria-se uma dívida (dívida técnica), pois esse código precisa ser refatorado futuramente para atender aos padrões de qualidade da organização. A dívida técnica pode ser observada através de métricas que representam a qualidade interna do produto. Por exemplo, a complexidade ciclomática. O acúmulo de muitas dívidas pode prejudicar a evolução do produto, pois apesar de não serem diretamente percebidas pelos usuários finais, as dívidas aumentam o esforço de manutenção. É importante explicitar que defeito não é dívida técnica, pois defeito é percebido diretamente pelos usuários do software.

   Avgeriou P, Kruchten P, Ozkaya I, et al. (2016) Managing Technical Debt in Software Engineering Edited by. Dagstuhl Reports 6:110–138. doi: 10.4230/DagRep.6.4.110. Disponível em: http://drops.dagstuhl.de/opus/volltexte/2016/6693/pdf/dagrep_v006_i004_p110_s16162.pdf.