TÉCNICAS DE PROGRAMAÇÃO 1

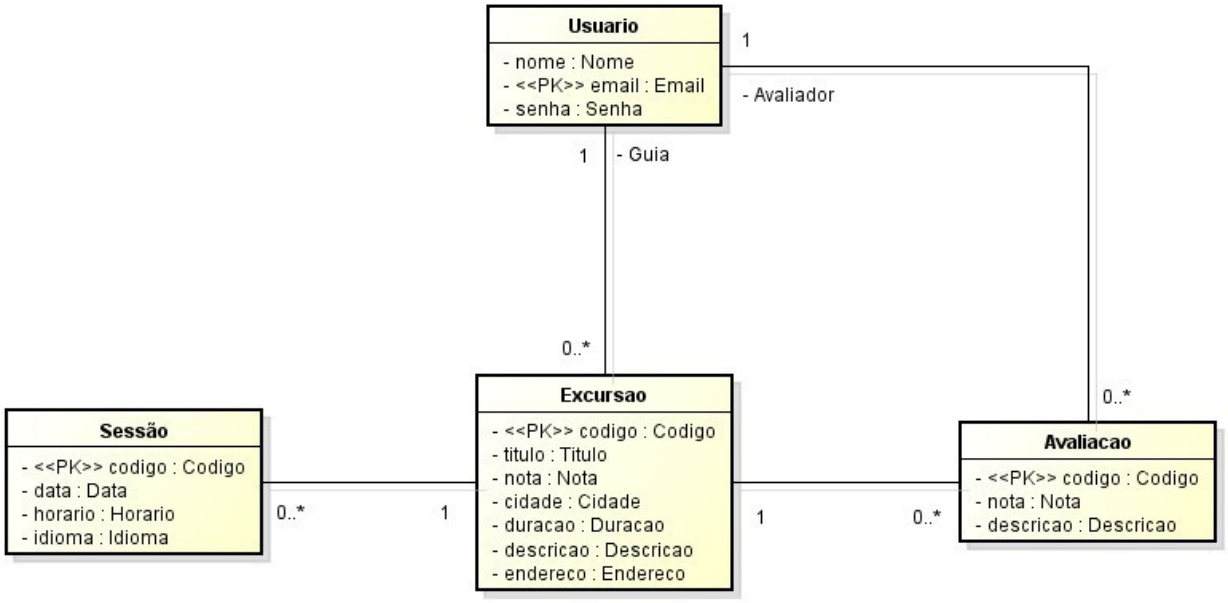
TRABALHO PRÁTICO

1. INTRODUÇÃO

O trabalho prático consiste no desenvolvimento de sistema de software com os requisitos descritos a seguir.

2. REQUISITOS FUNCIONAIS

O sistema de software a ser desenvolvido possibilitará a prestação de serviço de excursão gratuita guiada a pé (vide exemplos dessa classe de sistema em *guruwalk.com*, *freetour.com* e *freewalkertours.com*). Por meio desse sistema, qualquer usuário pode listar excursões disponíveis, acessar dados de excursões, sessões e avaliações. O usuário pode também cadastrar uma conta. Ao cadastrar uma conta, o usuário deve informar nome, endereço de correio eletrônico e senha. Uma vez cadastrada a conta, para ser autenticado, o usuário deve informar endereço de correio eletrônico e senha. Após autenticado, o usuário tem acesso aos seguintes serviços: editar (exceto endereço de correio eletrônico) e descadastrar a sua conta; cadastrar excursão, editar (exceto código) e descadastrar excursão da qual seja guia; cadastrar, editar (exceto código) e descadastrar sessão associada a excursão da qual seja guia; cadastrar avaliação associada a qualquer excursão, editar (exceto código) e descadastrar avaliação da qual seja autor. O sistema deve assegurar, além das regras expressas por meio do seguinte diagrama, as seguintes regras: nota de cada excursão é a média das notas das avaliações associadas à excursão, descadastramento de conta de usuário descadastra excursões nas quais o usuário é guia e avaliações nas quais o usuário é avaliador, descadastramento de excursão resulta no descadastramento de sessões e avaliações associadas à excursão. O sistema deve garantir que os serviços não resultem em inconsistências.



3. REQUISITOS NÃO FUNCIONAIS

1. Adotar o estilo de arquitetura em camadas (*layers*).
2. A arquitetura do software deve ser composta por camada de apresentação e por camada de serviço.
3. A camada de apresentação deve ser responsável pela interface com o usuário e pela validação dos dados de entrada.
4. A camada de serviço deve ser responsável pela lógica de negócio e por armazenar dados.
5. Cada camada deve ser decomposta em módulos de software.
6. Módulos de software devem interagir por meio de serviços especificados em interfaces.
7. Módulos de software devem ser decompostos em classes.
8. Devem ser implementadas classes que representem domínios, entidades e controladoras.
9. Implementar o código na linguagem de programação C++.
10. Prover projeto compatível com o ambiente de desenvolvimento Code::Blocks.

4. DOMÍNIOS

NOME	FORMATO
CIDADE	Hong Kong, Bangkok, Macau, Singapura, Londres, Paris, Dubai, Delhi, Istambul, Kuala Lumpur, Nova Iorque, Antalya, Mumbai, Shenzhen, Phuket
CÓDIGO	Formato DDDDDDX D é dígito (0-9). Não existe o código 000000. X é dígito verificador (informar algoritmo usado).
DATA	Formato DD-MES-AAAA DD - 01 a 31 MES - Jan, Fev, Mar, Abr, Mai, Jun, Jul, Ago, Set, Out, Nov, Dez AA - 2000 a 9999 Data considera a ocorrência de anos bissextos.
DESCRICAO	0 a 30 caracteres. Não há espaços em branco em sequência. Não há pontos (.) em sequência.
DURACAO	30, 60, 90, 120 ou 180.
EMAIL	Formato *parte-local@domínio* *parte-local* é composta por até 64 caracteres. Caractere de parte local pode ser letra maiúscula (A-Z) ou minúscula (a-z). Caractere de parte local pode ser dígito (0-9). Caractere de parte local pode ser ! # $ % & ' * + - / = ? ^ _ ` { | } ~ Caractere de parte local pode ser ponto (.) desde que não seja o primeiro ou o último caractere e que não ocorra em sequência. *domínio* é composto por até 253 caracteres. Caractere de domínio pode ser letra maiúscula (A-Z) ou minúscula (a-z). Caractere de domínio pode ser dígito (0-9). Caractere de domínio pode hífen (-). Caractere de domínio pode ser ponto (.) desde que não seja o primeiro caractere e não ocorra em sequência.
ENDERECO	0 a 20 caracteres. Não há espaços em branco em sequência. Não há pontos (.) em sequência.
HORÁRIO	HH:MM HH - 00 a 23 MM - 00 a 59
IDIOMA	Inglês, Chinês Mandarim, Hindi, Espanhol, Francês, Árabe, Bengali, Russo, Português, Indonésio Desconsiderar a acentuação.
NOME	5 a 20 caracteres. Cada caractere é letra (A-Z ou a-z), ponto (.) ou espaço em branco. Ponto (.) é precedido por letra. Ponto (.) é último caractere ou é seguido por um espaço em branco. Não há espaços em branco em sequência. Primeira letra de cada termo é letra maiúscula (A-Z).
NOTA	0, 1, 2, 3, 4, 5
SENHA	Formato XXXXXX Cada caractere X é letra (A-Z ou a-z) ou dígito (0-9). Não existe caracter repetido. Existe pelo menos uma letra maiúscula, uma letra minúscula e um dígito.
TITULO	5 a 20 letras (A-Z) (a-z) Não há espaços em branco em sequência. Não há pontos (.) em sequência.

TÉCNICAS DE PROGRAMAÇÃO 1

TRABALHO 1

1. ATIVIDADES A SEREM REALIZADAS

1. Codificar classe para cada domínio (*domain*).
2. Codificar classe para cada entidade (*entity*).
3. Codificar e executar teste de unidade (*unit test*) para cada classe domínio.
4. Codificar e executar teste de unidade (*unit test*) para cada classe entidade.
5. Documentar classes que representam domínios e entidades por meio de texto em formato HTML.

2. REQUISITOS A SEREM CUMPRIDOS

1. Trabalho pode ser realizado individualmente ou por equipe com até três participantes.
2. Desenvolver o sistema de software seguindo os requisitos especificados (funcionais e não funcionais).
3. Preencher os documentos com clareza e atentar para ortografia.
4. Adotar um padrão de codificação (*coding standard*).
5. Fornecer os códigos em formato fonte e em formato executável.
6. Em cada classe, identificar por comentários, a matrícula do aluno responsável pela implementação da classe.
7. Cada classe domínio deve conter atributo que seja instância de tipo suportado pela linguagem de programação.
8. Cada classe domínio deve permitir acesso ao atributo por meio de métodos públicos *set* e *get*.
9. Método *set* de cada classe domínio deve lançar exceção em caso de formato incorreto.
10. Cada classe de entidade deve conter atributos onde cada atributo é instância de classe domínio.
11. Cada classe de entidade deve permitir acesso aos atributos por meio de métodos públicos *set* e *get*.
12. Nesse trabalho, associações entre entidades não são implementadas.
13. Cada teste de unidade deve ser classe com diferentes métodos para diferentes casos de teste.
14. Cada teste de domínio deve exercitar o domínio por meio de um cenário de sucesso e de um de falha.
15. Cada teste de entidade deve invocar cada método público da entidade em teste pelo menos uma vez.
16. Classes devem funcionar corretamente segundo os testes de unidade fornecidos.
17. Fornecer projeto Code::Blocks que possibilite compilar e executar códigos sem erros na plataforma de correção.
18. Gerar documentação dos domínios e das entidades em formato HTML por meio da ferramenta Doxygen.
19. Escrever documentação das classes em formato HTML segundo perspectiva dos usuários das classes.
20. Incluir todos os artefatos construídos em um arquivo zip com nome T1-TP1-X-Y-Z.ZIP.
21. No nome do arquivo, os valores de X, Y e Z são os números de matrícula dos autores do trabalho.
22. Testar se o arquivo pode ser descompactado com sucesso e se não há vírus no mesmo.
23. Enviar o arquivo dentro do prazo.
24. Não cumprimento de requisitos resulta em redução de nota do trabalho.