TÉCNICAS DE PROGRAMAÇÃO 1

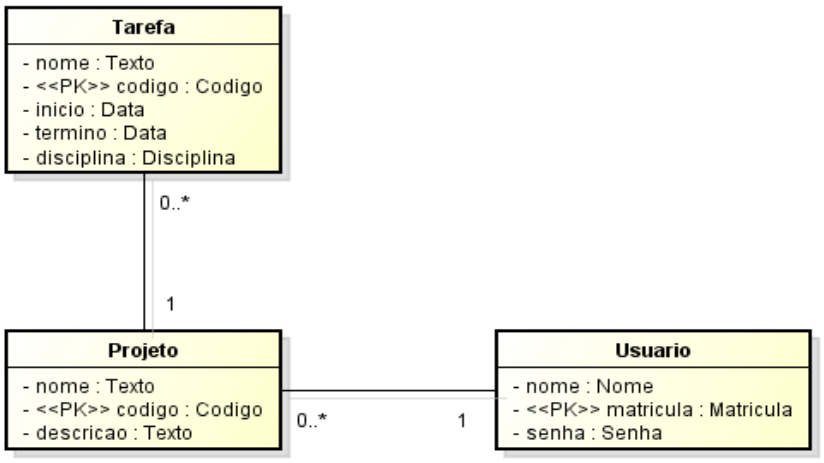
TRABALHO PRÁTICO

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 o suporte ao gerenciamento de projetos. Cada usuário pode cadastrar uma conta informando nome, matrícula e senha. Uma vez cadastrado, para ser autenticado, o usuário deve informar matrícula e senha. Após autenticado, o usuário tem acesso aos seguintes serviços: visualizar, editar (exceto matrícula) e descadastrar a sua conta de usuário; visualizar, cadastrar, editar (exceto código) e descadastrar projeto associado à sua conta de usuário; visualizar, cadastrar, editar (exceto código) e descadastrar tarefa associada a projeto que esteja associado à sua conta de usuário. Para visualizar determinada instância de entidade, o usuário deve informar a chave que a identifica. A visualização de determinada instância de entidade resulta na apresentação dos valores dos seus atributos. O sistema deve assegurar, além das regras expressas no seguinte diagrama, que o descadastramento de conta de usuário resulta no descadastramento dos projetos associados à sua conta de usuário; que o descadastramento de projeto resulta no descadastramento das tarefas associadas ao projeto.



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

DOMÍNIO	FORMATO
CODIGO	Formato DDDDDDDDDDX D é dígito (0-9). X é dígito verificador calculado através de algorítmo módulo 11.
DATA	Formato DD-MES-ANO DD - 01 a 31 MES - 01 a 12 ANO - 00 a 99 Deve ser levado em consideração se ano é ou não é bissexto.
MATRICULA	Formato LLLLDDDD L é letra maiúscula (A-Z). D é dígito (0-9).
NOME	Nome é composto por prenome e até dois sobrenomes. Texto (prenome mais sobrenomes e espaços em branco) é composto por total de até 20 caracteres. Cada caractere é letra (A-Z a-z) ou espaço em branco. Primeira letra de prenome ou de sobrenome é maiúscula (A-Z) e as outras são minúsculas (a-z). Não há espaços em branco em sequência. Acentuação pode ser desconsiderada.
SENHA	Formato XXXXXX Cada caractere X é letra maiúscula (A-Z) ou dígito (0-9). Não pode haver caractere duplicado. Existem pelo menos duas letras maiúsculas e dois dígitos.
TEXTO	10 a 40 caracteres. Cada caractere X é letra (A-Z ou a-z), dígito (0-9) ou sinal de pontuação (. , ; ? ! : -). Não há espaços em branco em sequência. Não há sinal de pontuação (. , ; : ? ! -) em sequência. Acentuação pode ser desconsiderada.
DISCIPLINA	Arquitetura, Desenvolvimento, Gerenciamento, Implantacao, Requisitos, Teste

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.

CRITÉRIOS DE CORREÇÃO

1 – Codificar classe para cada domínio.		PONTOS (% ACERTO)
	Cada domínio contém atributo que pode ser acessado por meio de métodos set e get.	0, 25, 50, 75, 100
	Cada domínio valida formato e método set lança exceção se formato for incorreto.	0, 25, 50, 75, 100
2 – Codificar classe para cada entidade.		
	Cada entidade contém atributos que são instâncias de classes domínio.	0, 25, 50, 75, 100
	Cada entidade contém atributos que podem ser acessados por meio de métodos set e get.	0, 25, 50, 75, 100
3 – Codificar e executar teste de unidade para cada classe domínio.		
	Cada teste é uma classe e exercita domínio com cenário de sucesso e de falha.	0, 25, 50, 75, 100
	Cada domínio funciona corretamente segundo o teste.	0, 25, 50, 75, 100
4 - Codificar e executar teste de unidade para cada classe entidade.		
	Cada teste é uma classe e exercita cada método da entidade em teste.	0, 25, 50, 75, 100
	Cada entidade funciona corretamente segundo o teste.	0, 25, 50, 75, 100
5 - Documentar classes por meio de texto em formato HTML.		
	Provida documentação externa em formato HTML para cada domínio.	0, 25, 50, 75, 100
	Provida documentação externa em formato HTML para cada entidade.	0, 25, 50, 75, 100