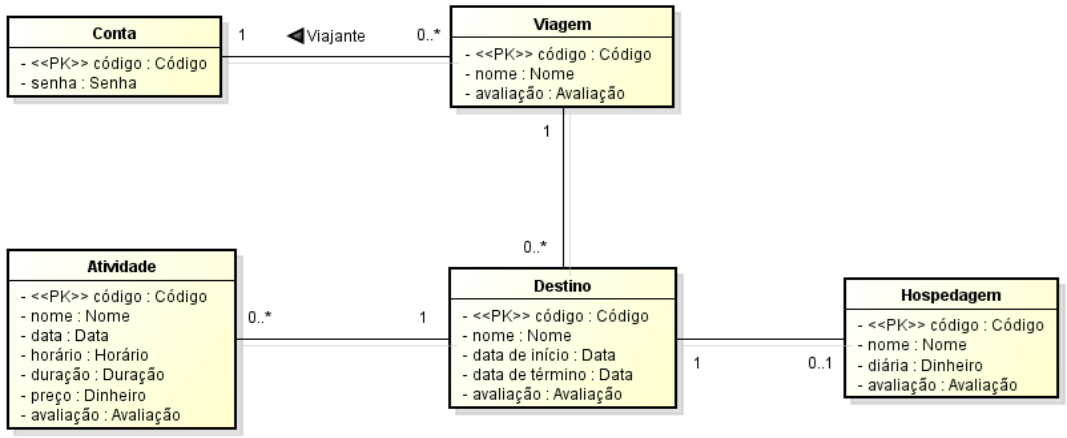
TÉCNICAS DE PROGRAMAÇÃO 1

ESPECIFICAÇÃO DE REQUISITOS DO SISTEMA A SER DESENVOLVIDO

1. REQUISITOS FUNCIONAIS

O sistema a ser desenvolvido possibilitará operações designadas pela sigla CRUD (Create, Read, Update, Delete). O sistema visa facilitar o planejamento de viagens. Para usar o sistema, o viajante deve criar uma conta. Após ser autenticado, o viajante tem acesso aos seguintes serviços: criar, ler, atualizar, e excluir viagem; criar, ler, atualizar, e excluir destino; criar, ler, atualizar, e excluir atividade; criar, ler, atualizar, e excluir hospedagem. A atualização de um registro (viagem, destino, atividade ou hospedagem) não possibilitar atualizar dado que identifica o registro (chave primária). O sistema deve também prover os seguintes serviços: consultar custo de viagem; listar viagens associadas a viajante, listar destinos associados a viagem, listar atividades associadas a destino e listar hospedagens associadas a destino. Ao apresentar uma listagem, o sistema deve ser apresentar código e nome de cada registro integrante da listagem. O sistema deve assegurar as seguintes regras: não pode ser excluído registro se existir registro associado a ele; data de atividade deve estar no intervalo definido pelas datas de início e término do destino associado. O sistema deve também assegurar as regras de negócio representadas no seguinte diagrama:



2. 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.
11. Nas implementações dos códigos de validação não é necessário considerar acentuação e nem a letra ç.

3. DOMÍNIOS

NOME	FORMATO VÁLIDO
Avaliação	Dígito 0, 1, 2, 3, 4 ou 5
Código	Seis caracteres Cada caracter pode ser letra (A - Z ou a - z) Cada caracter pode ser dígito (0 - 9)
Data	Formato DD-MM-AA DD - 00 a 31 MM - 01 a 12 AA - 00 a 99 Levar em consideração anos bissextos
Dinheiro	Valor de 0,00 a 200.000,00
Duração	Valor de 0 a 360
Horário	Formato HH:MM HH pode ser 00 a 23 MM pode ser 00 a 59
Nome	Texto com até 30 caracteres
Senha	Cinco dígitos (0 - 9) Não há dígito duplicado Os seis dígitos não podem estar em ordem crescente (01234, 12345, 23456 etc.) Os seis dígitos não podem estar em ordem descrescente (43210, 54321, 65432 etc.)

TÉCNICAS DE PROGRAMAÇÃO 1

TRABALHO 1

1. ATIVIDADES A SEREM REALIZADAS

1. Projetar, codificar e documentar classe para cada domínio (*domain*).
2. Projetar, codificar e documentar classe para cada entidade (*entity*).
3. Projetar, codificar e executar teste de unidade (*unit test*) para cada classe domínio.
4. Projetar, codificar e executar teste de unidade (*unit test*) para cada classe entidade.

2. REQUISITOS A SEREM CUMPRIDOS

1. Trabalho pode ser realizado individualmente ou por equipe com até cinco participantes.
2. Preencher os documentos com clareza, atentar para a ortografia e adotar um padrão de codificação (*coding standard*).
3. Fornecer os códigos em formato fonte e em formato executável.
4. Em cada classe, identificar por comentários a matrícula do aluno responsável pela implementação da classe.
5. Cada classe domínio deve conter atributo que seja instância de tipo suportado pela linguagem de programação.
6. Cada classe domínio deve permitir acesso ao atributo por meio de métodos públicos *set* e *get*.
7. Método *set* de cada classe domínio deve lançar exceção em caso de formato inválido.
8. Cada classe de entidade deve conter atributos onde cada atributo é instância de classe domínio.
9. Cada classe de entidade deve permitir acesso aos atributos por meio de métodos públicos *set* e *get*.
10. Nesse trabalho, associações entre entidades não são implementadas.
11. Cada teste de unidade deve ser classe com diferentes métodos para diferentes casos de teste.
12. Cada teste de domínio deve exercitar o domínio por meio de um cenário com valor válido e um com valor inválido.
13. Execução de cada teste de domínio deve resultar em sucesso.
14. Cada teste de entidade deve invocar cada método público da entidade pelo menos uma vez.
15. Cada teste de entidade deve comprovar que métodos invocados resultam no comportamento esperado.
16. Execução de cada teste de entidade deve resultar em sucesso.
17. Fornecer projeto Code::Blocks que possibilite compilar e executar códigos sem erros na plataforma de correção.
18. Documentar classes que representam domínios e 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 e atribuir o nome T1-TP1-X-Y-Z.ZIP ao arquivo zip.
21. No nome do arquivo zip, X, Y e Z devem ser 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.

3. CRITÉRIOS DE CORREÇÃO

ITEM	CRITÉRIO	% ACERTO
1	Cada domínio documentado, contém set e get e set lança exceção quando valor é inválido.	0, 25, 50, 75, 100
2	Cada entidade documentada, contém atributos instâncias de domínios acessáveis por set e get.	0, 25, 50, 75, 100
3	Cada teste de domínio é classe que exercita domínio com valor válido e inválido e não ocorre falha.	0, 25, 50, 75, 100
4	Cada teste de entidade é classe que exercita cada método de entidade e não ocorre falha.	0, 25, 50, 75, 100