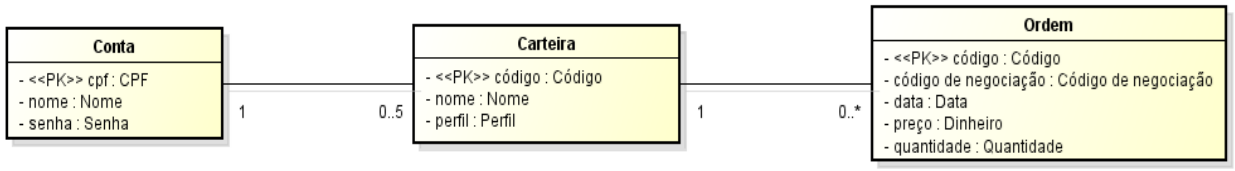
TÉCNICAS DE PROGRAMAÇÃO 1

1. REQUISITOS FUNCIONAIS DO SISTEMA DE SOFTWARE

O sistema tem o propósito de promover o aprendizado sobre investimentos financeiros. Para usar o sistema, o usuário deve criar uma conta. Após criar uma conta, deve ser autenticado. Para isso, deve informar CPF e senha. Após ser autenticado, pode acessar os serviços disponibilizados pelo sistema. A seguir, relação desses serviços: ler, editar e excluir conta; criar, ler, editar e excluir carteira; criar, ler e excluir ordem; listar carteiras associadas à conta; listar ordens associadas a carteira associada à conta.

Ao criar uma ordem, o código de negociação do papel deve corresponder a um código de papel no arquivo com dados históricos. O preço da ordem é o preço médio do papel na data informada vezes a quantidade de papéis. O preço médio do papel é obtido do arquivo com dados históricos. Uma ordem pode ser excluída, mas não editada. A leitura de conta deve apresentar dados (CPF, nome, senha) e saldo da conta. O saldo da conta é a soma dos saldos das carteiras associadas à conta. A leitura de carteira deve apresentar dados (código, nome, perfil) e saldo da carteira. O saldo da carteira é a soma dos preços das ordens associadas à carteira. A leitura de ordem deve apresentar dados (código, código de negociação, data, preço e quantidade) da ordem. A listagem de carteiras deve apresentar dados (código, nome, perfil) e saldo de cada carteira associada à conta. A listagem de ordens deve apresentar dados (código, código de negociação, data, preço e quantidade) de cada ordem. Ao solicitar listagem de ordens, o usuário deve informar o código da carteira à qual as ordens a serem listadas estão associadas. Não é possível editar dado que identifica registro (chave primária). Finalmente, o sistema deve assegurar as multiplicidades informadas nos relacionamentos presentes no seguinte diagrama e não deve possibilitar excluir registro se houver registro associado.



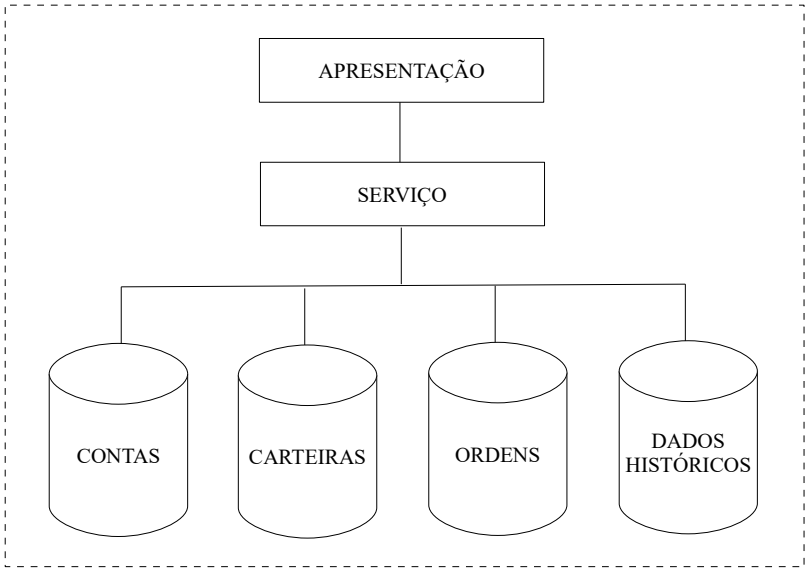
2. DOMÍNIOS

NOME	FORMATO VÁLIDO
Código	Cinco dígitos (0 – 9).
Código de negociação	Texto com até 12 caracteres. Cada caracter pode ser dígito (0 – 9), letra (A – Z ou a – z) ou espaço em branco.
CPF	Representação de número de CPF segundo norma.
Data	AAAAMMDD Ano (AAAA), mês (MM) e dia (DD). Data deve ser válida considerando os anos bissextos.
Nome	Texto com até 20 caracteres. Cada caracter pode ser dígito (0 – 9), letra (A – Z ou a – z) ou espaço em branco. Não podem existir espaços em branco em sequência.
Perfil	Conservador, Moderado, Agressivo
Dinheiro	0,01 a 1.000.000,00
Quantidade	1 a 1.000.000
Senha	Seis caracteres. Cada caracter pode ser dígito (0 – 9), letra (A – Z ou a – z), #, $, % ou &. Não há caracter duplicado. Existe pelo menos um dígito (0 – 9). Existe pelo menos uma letra maiúscula (A – Z). Existe pelo menos uma letra minúscula (a – z). Existe pelo menos um caracter especial (#, $, % ou &).

3. REQUISITOS NÃO FUNCIONAIS DO SISTEMA DE SOFTWARE

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. A arquitetura do software deve ser composta por ao menos quatro módulos.
9. Devem ser implementadas classes que representem domínios, entidades e controladoras.
10. Implementar o código na linguagem de programação C++.
11. Prover projeto compatível com o ambiente de desenvolvimento Code::Blocks.
12. Nas implementações dos códigos de validação não é necessário considerar acentuação e nem a letra ç.

4. ELEMENTOS DA ARQUITETURA DO SISTEMA DE SOFTWARE



5. ELEMENTOS DE CÓDIGO PARA ACESSAR ARQUIVO COM DADOS HISTÓRICOS

```
string linha, codigo = "IVVB11      ";                    // Código do papel procurado composto por 12 caracteres.

cout << "CODIGO      = " << codigo << "\n";

ifstream Arquivo("DADOS_HISTORICOS.TXT");                 // Abrir arquivo.

while (getline (Arquivo, linha)) {                        // Ler cada linha do arquivo.

    if(codigo.compare(linha.substr(12,12)) == 0){
        cout << "DATA        = " << linha.substr(2,8) << "\n";
        cout << "PRECO_MEDIO = " << std::stod(linha.substr(113,13))/100 << "\n";
    }
}

Arquivo.close();                                          // Fechar arquivo
```

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 seis 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

PONTUAÇÃO POR TRABALHO EM EQUIPE	
PONTOS SÃO SOMADOS APENAS SE A NOTA ORIGINAL DO TRABALHO FOR SUPERIOR A 0.00 PONTUAÇÃO APÓS A SOMA NÃO PODE SER SUPERIOR A 10.00	
NÚMERO DE PARTICIPANTES	PONTUAÇÃO
1	+ 0.00
2	+ 0.25
3	+ 0.50
4	+ 0.75
5 a 6	+ 1.00