



1ª Avaliação

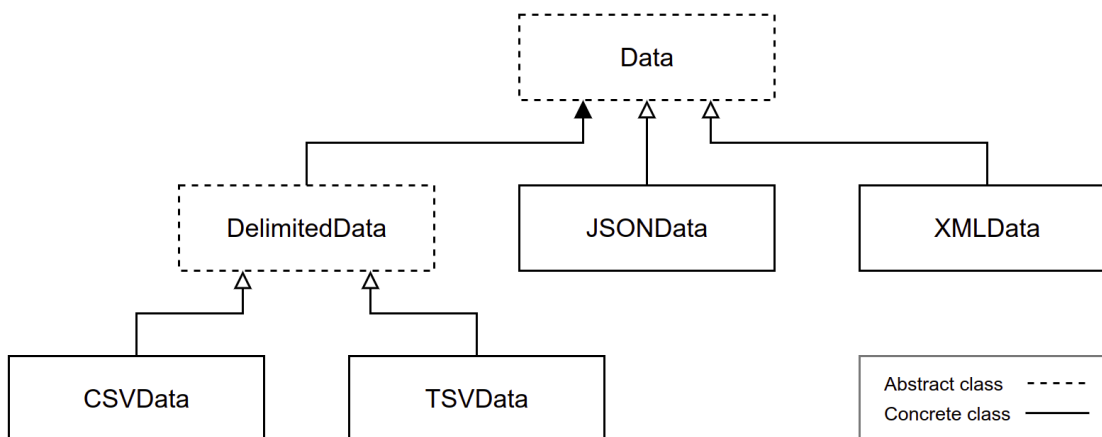
FORMATOS DE INTERCÂMBIO DE DADOS

A transferência de dados entre sistemas de computação é uma tarefa comum entre sistemas que não compartilham uma mesma base de dados. Essa transferência pode ocorrer por diversos meios, como arquivos em mídias removíveis, rede de computadores, etc. Independente do meio por onde os dados irão transitar entre sistemas, o formato com o qual os dados são transmitidos é extremamente relevante, especialmente entre sistemas com arquitetura de hardware diferentes.

Ao longo do tempo diversos formatos para intercâmbio de dados foram sendo desenvolvidos. Alguns dos mais comuns são apresentados abaixo:

- **CSV** (*Comma Separated Values*): formato de arquivo texto onde cada registro é armazenado em uma única linha e os valores correspondentes a cada campo do registro são separados por vírgula. A primeira linha é composta pelos nomes dos campos separados por vírgula.
- **TSV** (*Tab Separated Values*): formato similar ao CSV, onde o separador de campos é o caractere TAB (código ASCII 9).
- **JSON** (*Javascript Object Notation*): padrão aberto independente, de troca de dados simples e rápida entre sistemas.
- **XML** (*Extensible Markup Language*): Formato para a criação de documentos com dados organizados de forma hierárquica por meio de marcações.

Com base nas informações acima foi elaborada a seguinte hierarquia de classes para implementar suporte para uma interface de intercâmbio de dados via arquivo com suporte para os formatos descritos acima.



a) Sabe-se que a classe *Data* é abstrata e possui os seguintes métodos abstratos:

void load(String fileName)

Carrega o arquivo texto “fileName” para a memória.

void save(String fileName)

Grava os dados armazenados em memória no arquivo texto “fileName”.

void clear()

Limpa os dados armazenados na memória interna do objeto.

bool hasData

Getter que indica se existem dados carregados na memória do objeto.

String data

Getter/setter que retorna/armazena dados na memória do objeto.

List<String> fields

Getter que retorna os nomes dos campos de um registro.

b) A classe *DelimitedData* também é abstrata e possui os seguintes métodos abstratos:

String delimiter

Getter que retorna o caractere separador utilizado para separar os campos de um registro.

TAREFA:

Desenvolva uma aplicação em Dart que implemente a hierarquia de classes descrita acima e que observe os requisitos especificados.

REQUISITOS E FUNCIONALIDADES:

1. As classes *Data* e *DelimitedData* são abstratas, portanto, não podem ser instanciadas.
2. Os nomes dos campos de dados são obrigatórios em todos os dados manipulados. A não presença dos nomes dos campos nas strings de dados é condição de erro.
3. As demais classes são concretas e devem implementar os métodos abstratos das classes *Data* e *DelimitedData* (*getters* e *setters* também são métodos).
4. As implementações dos métodos *load()* e *save()* em cada classe concreta devem ser capazes de ler e escrever em arquivos texto no formato que sua respectiva classe representa.
5. O *getter hasData* deve indicar se existem dados armazenados internamente no objeto. O armazenamento interno não precisa obedecer o formato representado pela classe, podendo ser utilizado qualquer estrutura de dados do Dart que seja conveniente.
6. O membro *data* representa dois métodos, um *getter* e um *setter* (em Dart é permitido ter um *getter* e um *setter* na mesma classe com o mesmo nome).
7. O *getter data* deve retornar uma única string contendo os dados armazenados internamente no objeto no formato que a classe representa. O método deve retornar uma string vazia (ou, opcionalmente, *null*) caso não existam dados armazenados no objeto.
8. O *setter data* deve receber uma única string no formato representado pela classe. Caso o formato seja inválido, uma exceção deve ser levantada.
9. O método *clear()* limpa os dados armazenados no objeto. Após um *clear()* o *getter hasData* deverá retornar *false*.
10. A implementação do *getter "fields"* deve retornar uma lista de strings representando os nomes dos campos de dados. Esta lista deve ser atualizada a cada atualização da memória interna do objeto. Caso não existam dados armazenados, a lista deve estar vazia.
11. O *getter delimiter* da classe *SeparatedData* representa um método que retorna o caractere utilizado como separador de campo pelo formato representado pela classe concreta que o implementa.
12. Em todos os métodos devem ser levantadas exceções em caso de erro.
13. Para cada tipo de erro deve ser levantada uma tipo de exceção customizada (criada por você) diferente.
14. Todos os identificadores utilizados na aplicação devem usar a língua inglesa e a notação Camel Case.
15. É facultada a utilização de bibliotecas externas para manipulação dos formatos solicitados.
16. A aplicação deverá ser organizada em arquivos e pastas conforme a necessidade.
17. Deve ser implementado um programa para demonstrar todas as funcionalidades de cada classe, incluindo casos de levantam exceções. Os exemplos de exceção devem ser manipulados por instruções *try-catch*.
18. A última linha exibida pelo programa deverá conter os nomes dos membros da equipe.
19. Seu código será testado com a função *main()* existente no arquivo “main.dart”, que poderá ser baixado do repositório GitHub disponibilizado na seção “Recursos do Projeto” apresentada adiante.
 - O código do arquivo “main.dart” não deverá ser alterado. Seu código deve se adaptar às chamadas lá existentes.
20. A codificação do projeto deverá observar os princípios de estilo do Effective Dart. <https://dart.dev/guides/language/effective-dart/style>
21. 15. O projeto deverá ser disponibilizado em um repositório com acesso público do GitHub, com “main” como seu branch principal (default do GitHub). Qualquer outro branch não será considerado.

22. O código fonte do projeto deverá ser armazenado no GitHub (github.com) em um repositório com acesso público.
23. O repositório do projeto deverá conter um arquivo “README.md” contendo os nomes dos membros da equipe além de, obviamente, os arquivos do projeto (código fonte e os arquivos de dados).

RECURSOS DO PROJETO:

1. Todos os arquivos necessários para o projeto estão disponíveis no repositório <https://github.com/uespi-phb/prog2-20222/tree/main/aval1>
2. Os arquivos disponibilizados não deverão ser alterados.

ARQUIVO	DESCRIÇÃO
main.dart	Arquivo principal para teste do código submetido.
data.*	Arquivos em formato CSV, TSV, JSON e XML que contém os dados para execução dos testes.
output.txt	Saída esperada ao executar o programa a partir do arquivo “main.dart”

COMPOSIÇÃO DA EQUIPE:

1. A implementação poderá ser realizada em equipes de ATÉ 3 (três) membros;
2. Os autores de cada implementação SERÃO questionados sobre o código implementado, com o objetivo de comprovar a participação de cada membro na execução do projeto;
3. A comprovação de não participação do projeto de algum membro da equipe implicará em:
 - Sua exclusão da equipe, e atribuição de nota zero;
 - Penalidade de 30% na nota da equipe.

INTRUÇÕES PARA REMESSA DO PROJETO:

- Ao finalizar o projeto, remeter o *link* do repositório para o e-mail: cyder@phb.uespi.br com o seguinte assunto: “PROG2 – AVAL1”, juntamente com os nomes dos membros da equipe no corpo do e-mail;
- Prazo de entrega: 04/06/2022;
- O projeto deverá ser compatível com a versão 2.17.0 (ou superior) do Dart.

CRITÉRIOS DE AVALIAÇÃO:

70%	Correção	A solução atende a todos os requisitos e funcionalidades solicitadas?
30%	Organização	O código fonte encontra-se organizado e seus componentes devidamente segmentados em arquivos e/ou pastas?