Você acaba de ingressar em um novo emprego e se depara com a seguinte situação. A empresa em que foi contratado está comprando computadores novos e, portanto, todos os dados de clientes e contas precisam ser migrados dos computadores antigos para os novos. Porém, os computadores antigos guardam as informações seguindo modelos distintos entre si e incompatíveis diretamente com o modelo novo. A tarefa do seu time é desenvolver uma aplicação que receba os dados das máquinas antigas, identifique o modelo deles e os converta para o modelo novo, utilizando a estratégia de conversão correspondente.

Sua **equipe já contribuiu para o projeto**, cabe a **você finalizar** (vide arquivo compactado contendo código fonte que você deverá utilizar na sua implementação).

A entrada e saída são representadas por arquivos em que a primeira linha possui um número indicando o modelo, conforme tabela a seguir, e o conteúdo restante os dados da máquina que segue aquele modelo. É garantido que os modelos sempre são seguidos corretamente, então não há necessidade de verficar isso.

Código do Modelo	**Modelo**
0 ou qualquer outro valor não presente nesta tabela	UNKNOWN (Desconhecido)
1	**BLUE**
2	**GREEN**
3	**YELLOW**
4	NEW (o novo modelo)

A seguir são apresentados a estrutura de cada modelo e um **rascunho** de código para conversão de cada modelo antigo para o modelo novo.

Modelo NEW
Código: 4
Conteúdo: Caracteres convencionais, todos minúsculos.

Modelo BLUE
Código: 1
Conteúdo: Caracteres convencionais, todos maiúsculos.
Estratégia de conversão: Tornar todos os caracteres em minúsculo.

```
public String convertContent(String contentToConvert){
    String convertedContent = contentToConvert.toLowerCase();
    return convertedContent;
}
```

Modelo GREEN
Código: 2
Conteúdo: Cada caractere é representado por uma sequência de 8 bits. Cada sequência é separada por um espaço.
Estratégia de conversão: Converter cada sequência de 8 bits no caractere correspondente segundo valor na tabela ASCII (https://www.asciitable.com/).

```
public String convertContent(String contentToConvert){
        String convertedContent = "";
        String[] splittedContentToConvert = contentToConvert.split(" ");

        for(String elem : splittedContentToConvert) {
            String asciiRepresentaion = parseBinaryToChar(elem);
            convertedContent += asciiRepresentaion;
        }

        return convertedContent;
    }
```

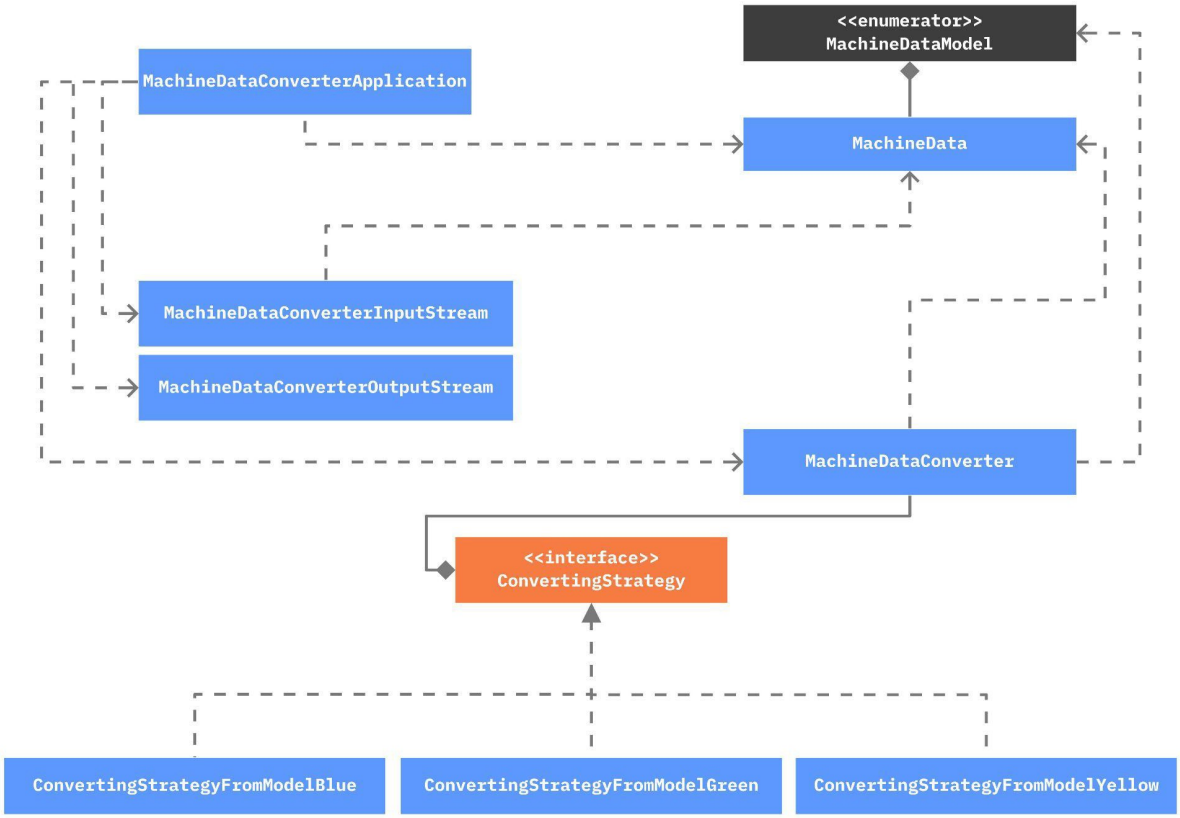
Modelo YELLOW
Código: 3
Conteúdo: Caracteres minúsculos como no modelo NEW, mas os espaços são representados por "+" (sinal de adição) e quebras de linha por "/" (barra inclinada para a direita).
Estratégia de conversão: Percorrer o conteúdo caractere a caractere identificando se corresponde ao sinal de adição ou barra, substituindo por " " (espaço em branco) ou "\n" (valor da nova linha), respectivamente.

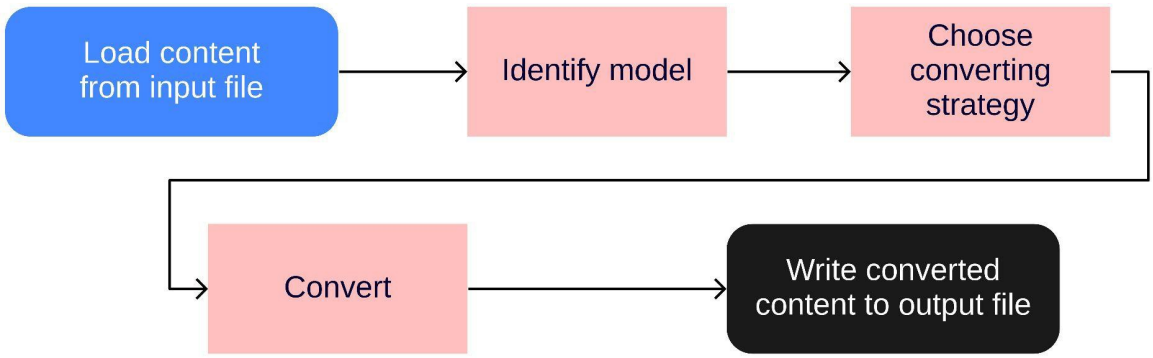
```
public String convertContent(String contentToConvert){
        String convertedContent = "";

        for(char elem : contentToConvert.toCharArray()) {
            if(elem == '+') {
                convertedContent += " ";
            }
            else if(elem == '/') {
                convertedContent += "\n";
            }
            else {
                convertedContent += elem;
            }
        }

        return convertedContent;
    }
```

Quando o modelo de origem for desconhecido ou já estiver no modelo novo, uma mensagem de “There is no strategy for this model” (Não há estratégia para esse modelo) será apresentada ao usuário.

Agora são apresentados os detalhes de implementação e o fluxo da aplicação, bem como exemplos de entradas e saídas esperadas.

Este é o diagrama de classe proposto pelo time para desenvolvimento da aplicação.
Neste link (https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial) há um guia rápido para entendimento da UML, mas basicamente este diagrama representa as entidades presentes na aplicação e como se relacionam.





O fluxo esperado da aplicação é receber um nome de arquivo que será buscado no diretório **inputFiles**, identificar o modelo dos dados, converter o conteúdo para o padrão novo e escrever em um arquivo novo no diretório **outputFiles**.

Exemplos de conteúdo antes e depois da conversão:

Entrada	Saída
#exemplo 1 - conteúdo do arquivo original# 1 TESTE DA ESTRATEGIA DE CONVERSAO 1	#exemplo 1- conteúdo do arquivo novo# 4 teste da estrategia de conversao 1
#exemplo 2 - conteúdo do arquivo original# 2 01110100 01100101 01110011 01110100 01100101 00100000 01100100 01100101 00100000 01100101 01110011 01100011 01110010 01101001 01110100 01100001 00100000 01110000 01100001 01110010 01100001 00100000 01100011 01101111 01101110 01110110 01100101 01110010 01110011 01100001 01101111 00001010 01100101 01101101 00100000 01100100 01110101 01100001 01110011 00100000 01101100 01101001 01101110 01101000 01100001 01110011 00100000 01110111 01101111 01110111 01110111 01110111	#exemplo 2- conteúdo do arquivo novo# 4 teste de escrita para conversao em duas linhas wowww
#exemplo 3 - conteúdo do arquivo original# 3 teste+de+conversao+de+texto/so+que+agora+com+duas+linhas	#exemplo 3- conteúdo do arquivo novo# 4 teste de conversao de texto so que agora com duas linhas

Submissão

Antes de submeter, coloque seu nome completo e *número usp* no código-fonte Java, na forma de comentário JavaDoc no início de cada arquivo, utilizando a *tag* **`@author`**:

```
@author <nome> <número usp>
```

Ou seja, substitua **`<nome>`** pelo seu nome completo e **`<número usp>`** pelo seu número usp. Por exemplo, o aluno de nome Pedro Marcondes cujo número usp é 11111 deve substituir a linha acima por:

```
@author Pedro Marcondes 11111
```

Para documentação de código novo, siga o mesmo padrão existente no código já existente em JavaDoc.

Sinta-se livre para adicionar arquivos, código e documentar o código, mas **NÃO altere código ou documentação existentes bem como a estrutura de pastas e arquivos.**

Submeta sua implementação no sistema Tidia-AE em um único arquivo compactado .ZIP.

Avaliação

Na nota do trabalho também serão considerados os seguintes critérios (além dos já mencionados nas Disposições Gerais entregues no início do semestre):

- **Correção:** O programa faz o que foi solicitado? Faz tudo o que foi solicitado? Utiliza encapsulamento de informação? (i.e., acessa adequadamente os objetos definidos?)
- **Eficiência:** As operações são executadas da maneira mais eficiente para cada estrutura de dados? Evita código duplicado/redundante/não atingível?
- **Interface:** É simples de usar, genérico, prático, tolera os erros mais óbvios? O trabalho foi entregue dentro das especificações?
- **Código fonte:** é claro (*layout*, espaçamento, organização em geral), nomes de variáveis são sugestivos, e há documentação/comentários apropriados no código? Quando aplicável, faz uso de subalgoritmos (funções, procedimentos ou métodos) adicionais que melhoram a legibilidade do(s) método(s) solicitado(s) sem comprometer sua eficiência (por exemplo, na notação assintótica **O(*n*),** onde ***n*** representa o tamanho da entrada). Segue e atende os fundamentos (Abstração, Encapsulamento, Reúso, Polimorfismo e Herança) do Paradigma Orientado a Objetos.