**Atividade prática AT1/N1**

Você foi contratado por um banco digital para desenvolver um sistema de detecção de fraudes baseado em **dados históricos de cadastro de clientes**. No último mês de abril, por uma falha no cadastro de clientes, milhares de contas foram abertas — e agora, o banco quer fazer uma **auditoria retroativa** para identificar inconsistências nos CPFs utilizados, que estão armazenados em 30 arquivos (1 por dia).

O desafio: analisar milhões de registros antigos e verificar se os CPFs presentes são válidos segundo o **algoritmo oficial da Receita Federal**.

Nesse contexto, desenvolva 8 versões de um código em Java, cada uma com um número de threads distinto:

- 1 thread (30 arquivos para essa thread)
- 2 threads (15 arquivos por thread)
- 3 threads (10 arquivos por thread)
- 5 threads (6 arquivos por thread)
- 6 threads (5 arquivos por thread)
- 10 threads (3 arquivos por thread)
- 15 threads (2 arquivos por thread)
- 30 threads (1 arquivo por thread)

Cada versão deve ser capaz de realizar o seguinte:

- Ler 30 arquivos de texto, cada um contendo uma lista com os CPFs cadastrados em um dia;
- Realizar a checagem de cada um dos CPFs (o código Java da checagem está na segunda página);
- Exibir a contagem de CPFs válidos e inválidos encontrados;
- Salvar o tempo total de execução em um arquivo separado (**confira arquivo de exemplo "versao_1_thread.txt"**)

**Avaliação (2.5 pontos)**

- **Criação das 8 versões do experimento:** 1.5
    - Implementação das 8 versões conforme requisitos apresentados acima;
        - **Um arquivo .java com cada versão!**
    - Correto processamento dos dados dos arquivos;
    - Correta exibição dos resultados (núm. de CPFs válidos e inválidos) e criação dos arquivos com os tempos coletados.
- **Estruturação e organização do código:** 0.5**;**
    - Divisão do projeto em classes;
    - Modularização do código, utilizando métodos sempre que possível;
    - Organização/clareza do código (nomes significativos de variáveis, indentação, etc.);
- **Armazenamento dos dados obtidos e gerados em arquivos distintos:** 0.5
    - Armazenar tempo de execução para cada versão em arquivo com nomes descritivos:
        - versao_1_thread.txt
        - versao_2_threads.txt
        - versao_5_threads.txt
        - …

**Envio**

- O código produzido juntamente com os arquivos gerados devem ser armazenados em um repositório do GitHub. Somente o link para o repositório deve ser enviado pelo AVA!
    - Lembre-se de não deixar o repositório como privado
- **O trabalho deve ser feito em grupo, mas qualquer tipo de plágio/cola será penalizado (o projeto receberá nota 0 (SEM EXCEÇÕES!!!!!!!!!!!!!!!!!)).**
- **Projetos com erro de sintaxe ou que não possam ser executados irão receber nota 0 também**

**Código de exemplo de validação do CPF**

```
public class CPFValidator {

    public static boolean validaCPF(String cpf) {
        // Remove caracteres não numéricos
        cpf = cpf.replaceAll("\\D", "");

        if (cpf.length() != 11 || cpf.chars().distinct().count() == 1) {
            return false;
        }

        for (int i = 9; i <= 10; i++) {
            int soma = 0;
            for (int j = 0; j < i; j++) {
                soma += (cpf.charAt(j) - '0') * ((i + 1) - j);
            }

            int digito = (soma * 10) % 11;
            if (digito == 10) digito = 0;

            if (digito != (cpf.charAt(i) - '0')) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        String cpf = "12312312342";
        System.out.println("CPF válido? " + validaCPF(cpf));
    }
}
```

**Referências**

- Como ler arquivos de texto em Java: https://www.datacamp.com/pt/doc/java/read-files