Desafios Técnicos Java – Supera

1. Desafio Resistores – Código de Cores

Introdução

Os resistores são componentes elétricos marcados com listras/faixas coloridas para indicar tanto o valor de sua resistência em ohms, quanto a tolerância permitida.

Imagine que Diogo, o dono de um "Kit Básico Raspberry Pi", esvaziou diversos sacos zip-lock de resistores, e ao invés de ficar procurando de um por um o resistor para seu projeto, precisa que a partir do valor necessário, tenha em mãos a sequência de cores correta.

Código de Cores dos Resistores

Os códigos básicos dos resistores são:

- 0: preto
- 1: marrom
- 2: vermelho
- 3: laranja
- 4: amarelo
- 5: verde
- 6: azul
- 7: violeta
- 8: cinza
- 9: branco

Todos os resistores possuem pelo menos três bandas, sendo que a primeira e a segunda banda correspondem ao primeiro e segundo dígito do valor de ohms. A terceira indica a potência de 10 que deve ser multiplicada para obter o valor.

Exemplo: Um resistor de 47 ohms é igual a 47 * 10 ^ 0, teria a seguinte sequência de cores: **amarelo violeta preto**

A maioria dos resistores também possuem uma quarta faixa que indica sua tolerância (5% por exemplo), representado por uma faixa dourada. Portanto, no exemplo acima, ficaria **amarelo violeta preto dourado**

Desafio

Sua função deverá receber uma string contendo o valor de ohms a ser convertido, seguido de um espaço e a palavra "ohms" (ex: 47 ohms)

Os valores de entrada seguem as seguintes regras:

- Para resistores menores que 1000 ohms, o valor em ohms é formatado apenas como um número simples. Por exemplo, com o resistor de 47 ohms acima, sua função receberia a string "47 ohms" e retornaria a string "amarelo violeta preto dourado".

- Para resistores maiores ou iguais a 1000 ohms, mas menores que 1.000.000 ohms, o valor de ohms é dividido por 1.000 e tem um "k" minúsculo depois dele. Por exemplo, se sua função recebesse a string "4.7k ohms", ela precisaria retornar a string "amarelo violeta vermelho dourado".

- Para resistores maior ou igual a 1.000.000 ohms, o valor de ohms é dividido por 1.000.000 e tem um “M” maiúsculo depois dele. Por exemplo, se sua função recebesse a string “1M ohms”, ela precisaria retornar a string “marrom preto verde dourado”.

Mais Exemplos

- "10 ohms" => "marrom preto preto dourado"
- "100 ohms" => "marrom preto marrom dourado"
- "220 ohms" => "vermelho vermelho marrom dourado"
- "330 ohms" => "laranja laranja marrom dourado"
- "470 ohms" => "amarelo violeta marrom dourado"
- "680 ohms" => "azul cinza marrom dourado"
- "1k ohms" => "marrom preto vermelho dourado"
- "2M ohms" => "vermelho preto verde dourado"

Observações

Os números decimais de entrada serão sempre separados por **ponto**.

2. Desafio Flores de Samambaia

Introdução

A flor de samambaia (Fern Flower) é um elemento da mitologia eslava, uma flor mágica de grande poder, que é protegida por espíritos malignos, mencionada durante o *Ivana Kupala* (celebração que ocorre no solstício de verão). Diz-se que quem conseguir essa flor, terá sucesso no amor e obterá muitas riquezas.

Para obter uma flor de samambaia, a pessoa deveria procurá-la em uma floresta antes da meia-noite na véspera do *Ivana Kupala*. Exatamente à meia-noite ela floresceria. Para colhê-la seria preciso desenhar um círculo em volta dela. Parece uma tarefa fácil, no entanto, os espíritos malignos que guardam a flor tentariam de tudo para distrair qualquer um tentando colher a flor. Se a pessoa falhasse ao tentar desenhar um círculo em volta da flor, teria sua vida sacrificada.

Desafio

Dados dois círculos, um desenhado por um ambicioso caçador de flores de samambaia e outro representando a área da flor, sua tarefa é determinar se o caçador morre ou fica rico com sua conquista.

Entrada

A entrada consiste em um conjunto de seis inteiros sendo:

- R1 (1 <= R1)
- X1 e Y1
- R2 (R2 <= 1000)
- X2 e Y2

O círculo desenhado pelo caçador possuir raio R1 com centro (X1;Y1). O círculo representado pela área da flor possui raio R2 com centro (X2;Y2).

Saída

Retorne uma única linha contendo **MORTO**, se o caçador morre, ou **RICO** se o caçador consegue colher a flor.

Observações

Deverá ser utilizado a seguinte classe:

```
public class FlorSamambaia {

	public static String tentativaDesenhar(int r1, int x1,int y1,int r2,int
x2,int y2) {
		// Lógica de código aqui
	}
}
```

3. Desafio Snail

Desafio

Dado uma matriz N x N, retorne os valores organizados dos elementos mais externos para os mais internos, em sentido horário.

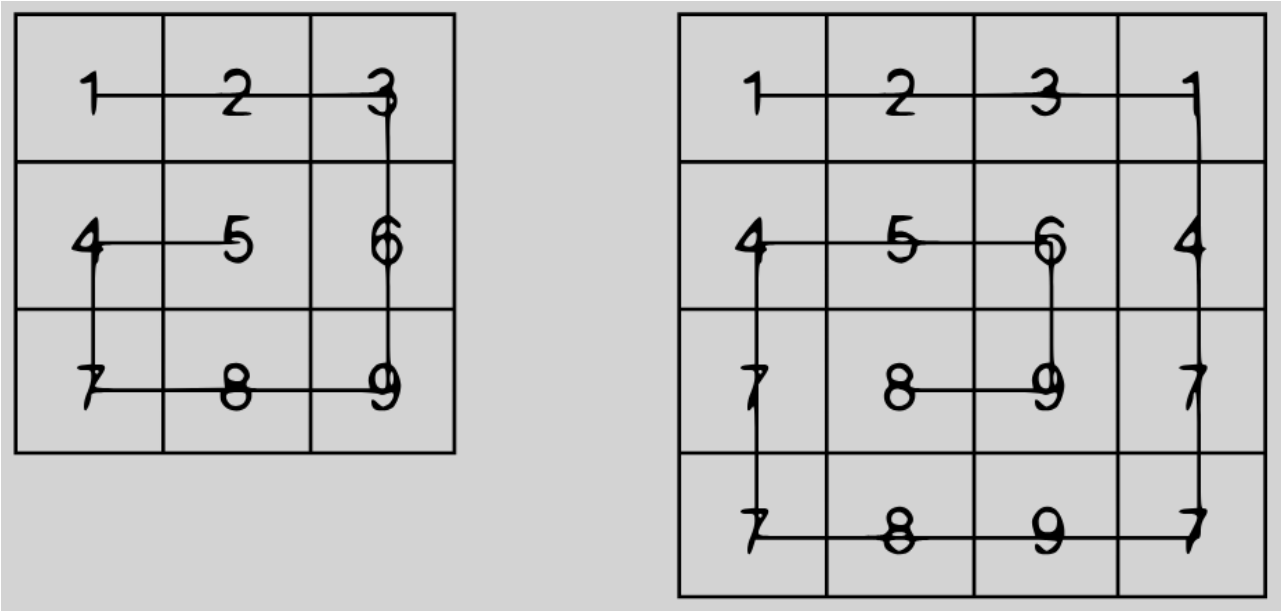
Exemplo.:

```
-----------
1  |  2  |  3
-----------
4  |  5  |  6        => [1, 2, 3, 6, 9 , 8, 7, 4, 5]
-----------
7  |  8  |  9
-----------

-----------
1  |  2  |  3
-----------
4 | 10 | 12              => [1, 2, 3, 12, 33 , 7, 6, 4, 10]
-----------
6 | 7 | 33
-----------
```

Observações

- A ideia é percorrer a matriz de duas dimensões em um padrão de caracol no sentido horário
- Uma entrada com uma matriz vazia também deve ser considerada



Desafio Técnico - Sistema de Gerenciamento de Tarefas Abstrato

Objetivo: Desenvolver uma aplicação web que permita aos usuários gerenciar listas e itens de forma eficiente.

Requisitos Funcionais:

1. **Criação de Listas:** O usuário deve poder criar e gerenciar listas, cada uma contendo itens associados.
2. **Gerenciamento de Itens:** Dentro de cada lista, o usuário deve poder adicionar, editar, remover e alterar o estado de itens.
3. **Visualização e Filtragem:** O usuário deve poder visualizar e organizar as listas e itens de forma intuitiva, com opções de filtragem disponíveis.
4. **Prioridade de Itens:** O usuário pode destacar itens dentro das listas para indicar prioridade.

Regras de Negócio:

1. **Validação de Dados:** Os itens dentro de uma lista devem seguir critérios básicos de validação, como comprimento mínimo do título.
2. **Estado dos Itens:** Cada item deve possuir um estado que pode ser alterado pelo usuário.
3. **Ordenação e Destaque:** Itens destacados devem ser priorizados na visualização.

Requisitos Não Funcionais:

1. **Persistência de Dados:** As listas e itens devem ser armazenados de forma persistente.
2. **Exposição de API:** A aplicação deve fornecer uma API para as operações principais.
3. **Testes Automatizados:** Deve haver testes automatizados para as funcionalidades principais.

Tecnologias Obrigatórias:

- **Framework de Desenvolvimento:** Utilize um framework adequado para o desenvolvimento da aplicação.
- **Banco de Dados:** A persistência dos dados deve ser gerenciada por um banco de dados relacional.
- **Ferramentas de Teste:** Inclua testes automatizados utilizando ferramentas de sua escolha.

Critérios de Avaliação:

- **Funcionalidade:** A aplicação deve atender aos requisitos funcionais especificados.
- **Qualidade e Clareza do Código:** Avaliação da estrutura e clareza do código.
- **Documentação:** A aplicação deve ser acompanhada de uma documentação mínima explicando como configurá-la e usá-la.

- **Testes:** A qualidade dos testes implementados será considerada na avaliação.