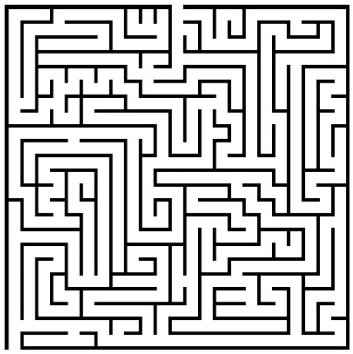
Etapa 1. Arrays bidimensionais, arquivos e recursão

LABIRINTO



Na primeira etapa do Desafio do Laboratório II, você deve criar um programa que simule um labirinto. No nosso sistema, o labirinto será uma determinada estrutura composta por caracteres. No labirinto, posições com 'X' indicam que não é possível passar, enquanto posições com ' ' (espaço) indicam que é possível a passagem. O local de destino está marcado no labirinto com a letra 'D'. Um exemplo de labirinto pode ser visto abaixo:

```
  XXXXXXXXX    XXXXXX
X XXXXXXXXX X    XXXX
X   XXXXXXX XXX XXXX
XXX XXXXXX  XXX XXXX
XXX    XX  XXXX
XXX XX X  XXXXX XXX
XXXXXX X XXXXXX   X
XXXXXX   XXXXXX XXXD
```

O labirinto será um array bidimensional de caracteres. Para a criação do labirinto, seu programa deve ler um arquivo com extensão txt que contém os caracteres correspondentes ao labirinto. Depois de ler o arquivo, você deve passar seu conteúdo para um array bidimensional de caracteres. Desta forma, o local de entrada no labirinto sempre será a posição [0][0] do array.

Um exemplo de array bidimensional de caracteres que representaria um labirinto pode ser visto no código abaixo:

```
char[][] labirinto =
{{' ','X','X','X','X','X','X','X','X','X','X','X','X','X','X','X','X','X'},
{' ',' ',' ',' ',' ',' ',' ',' ','X','X','X','X','X','X','X','X','X','X'},
{'X','X','X',' ','X',' ','X',' ','X','X','X',' ',' ',' ',' ','X','X','X'},
{'X','X','X','X',' ',' ','X',' ','X','X','X',' ','X','X',' ',' ','X','X'},
{'X','X','X','X','X','X','X',' ','X','X','X',' ','X','X',' ',' ','X','X'},
{'X','X','X','X','X','X','X',' ',' ',' ',' ',' ','X',' ','X',' ','X','X'},
{'X','X','X','X','X','X','X','X',' ','X','X',' ','X',' ',' ',' ',' ',' '},
{'X','X','X','X','X','X','X','X',' ',' ',' ',' ','X',' ',' ','X','X','D'}};
```

Neste sentido, nesta etapa você deve:

- criar uma classe `Labirinto`, que possui como atributo a estrutura necessária para o armazenamento do labirinto
- na classe `Labirinto`, crie um método chamado `criaLabirinto`, que recebe uma `String filename`, que corresponde a um arquivo que contém a estrutura do labirinto, conforme supracitado. Este método abre o arquivo `filename` para leitura e preenche o labirinto
- ainda na classe `Labirinto`, você deve criar um método para andar no labirinto. Este método deve retornar verdadeiro, caso haja pelo menos uma saída para este labirinto, ou falso caso contrário. Entretanto, este método deve ser **recursivo** (com os métodos público e privado). Seu método público não recebe parâmetros e deve ser nomeado `percorreLabirinto`.

Etapa 2. Pesquisa e Ordenação

Na segunda etapa, exercitaremos os métodos de Pesquisa e Ordenação, utilizando os métodos estudados para realizar uma ordenação de uma classe criada por você. Para este desafio, é imprescindível **o entendimento da função da estabilidade de algoritmos de ordenação.**

- Crie uma classe `Candidato`, que possui um atributo `nome` (que armazena apenas o primeiro nome), um `partido (String)` e um inteiro chamado `intencoesVotos` que representa a quantidade de intenções de voto obtidas na pesquisa. Crie os métodos de acesso dos atributos e um construtor que recebe informações para inicializar todos os atributos. Crie, também, o método `toString()`.

- Crie uma classe chamada `PrincipalCandidatos`. Nesta classe, crie o método `main`. No `main`, crie um array de `Candidato`, com tamanho aleatório entre 1 e 100. Preencha este array com objetos do tipo `Candidato` com informações aleatórias (utilize sorteios para as informações). Obs.: é importante que exista a possibilidade de ter candidatos com o mesmo nome e também diferentes candidatos do mesmo partido.

- Depois disso, imprima as informações dos candidatos criados. Entretanto, quando você imprimir as informações, **a impressão deve sair ordenada pelo nome dos candidatos**. Caso haja candidatos com o mesmo nome, estes candidatos de nome repetido devem aparecer **ordenados também pela quantidade de intenções de voto**. Caso estes candidatos de mesmo nome possuam a mesma quantidade de intenção de voto, eles devem estar **ordenados por partido**.
    - Para isso, você deve adaptar os métodos de ordenação estudados, de maneira que recebam um array de `Candidato` e ordenem este array de acordo com um determinado atributo da classe `Candidato`. Desta forma, **você deve criar 3 métodos de**

ordenação: um que ordena o array pelo nome dos candidatos, um que ordena por intenção de votos (do maior número de votos para o menor) e outro que ordena o array pelo partido dos candidatos. A ordenação final será dada pela chamada dos métodos de ordenação na ordem correta.

- Utilize **EXCLUSIVAMENTE** os métodos de ordenação `Inserção Direta` e `Seleção Direta` (com as alterações necessárias) para realizar a ordenação do array de `Candidato`. Assim sendo, os métodos devem receber um array de `Candidato` e ordenar este array por algum dos critérios. Os métodos de ordenação devem ser estáticos (para poderem ser chamados no `main`) e devem estar na classe `PrincipalCandidatos`. O nome dos métodos devem seguir o seguinte:
  - `ordenaCandidatosPorNome`
  - `ordenaCandidatosPorVotos`
  - `ordenaCandidatosPorPartido`

- Na classe `PrincipalCandidatos`, crie um método estático chamado `pesquisaBinariaCandidatos`. Este método é uma versão do método de Pesquisa Binária visto em aula, que deve ser capaz de pesquisar um determinado `Candidato` pelo seu primeiro nome. O método deve receber um array unidimensional de `Candidato` e um nome de candidato a ser procurado. O método deve retornar a posição na qual aquele `Candidato` se encontra ou -1 caso não exista candidato com aquele nome no array (caso haja mais de um candidato com o mesmo nome, o método deve retornar o primeiro que for encontrado).

- Depois disso, teste seu método com o array ordenado pelo seu método de ordenação do Exercício 2 e imprima as informações do `Candidato` que possui o nome que o usuário informar pelo teclado, utilizando o seu método `pesquisaBinariaCandidatos`.

Etapa 3. Listas Estáticas

Na terceira etapa, o desafio do módulo solicita que você implemente alguns métodos utilizando Estruturas de Dados Estáticas.

- Na classe que representa uma lista estática (classe criada em aula), implemente um método **RECURSIVO** que recebe um elemento por parâmetro e retorna a quantidade de vezes que este elemento aparece na lista. Você deve fazer os métodos público e privado. Você deve utilizar a seguinte assinatura para o método público (utilizando exceções na declaração, caso seja necessário):

```
public int contaElementos(E el)
```

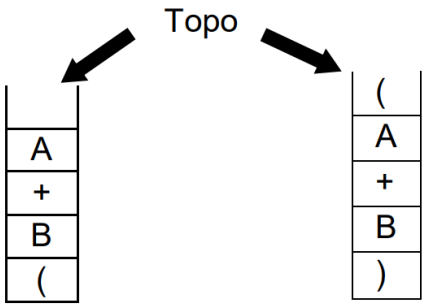
- Implemente, em uma classe chamada `Etapa3`, o seguinte método:

```
public boolean checkBrackets(Stack<Character> s1)
```

Esse método verifica se uma expressão matemática tem os parênteses agrupados de forma correta, isto é: **(1)** *se o número de parênteses à esquerda e à direita é igual*; *e* **(2)** *se todo parêntese aberto é seguido, posteriormente, por um fechamento de parêntese.* Por exemplo, as expressões "`((A+B)` e "`A+B(`" violam a regra (1) e as expressões "`)A+B(-C`" e "`(A+B))-(C+D`" violam a regra (2). Um exemplo de expressão correta seria: "`((A+B)-(C+D))`".

O método recebe como parâmetro uma pilha que contém os caracteres de uma expressão matemática. O método retorna verdadeiro se os parênteses da expressão estão agrupados de forma correta, ou falso, caso contrário. Uma pilha armazena apenas uma única expressão. As expressões são armazenadas na pilha da direta para a esquerda, ou seja, os caracteres da direita (ou seja, do final da expressão) são empilhados primeiro.

A imagem abaixo ilustra duas possíveis pilhas de entrada. A pilha da esquerda armazena a expressão "`A+B(`", enquanto a pilha da direita armazena a expressão "`(A+B)`".



Etapa 4. Listas Dinâmicas

De acordo com o que estudamos, você deve implementar uma pilha e uma fila dinâmica. Para isto, você deve criar um projeto no Eclipse e colocar suas implementações neste projeto. Utilizando as interfaces `Stack` e `Queue` e a classe `Node` (ambas vistas em aula), faça o que se pede:

- A partir da classe `Node` e da interface `Stack`, crie uma classe chamada `LinkedStack`, que implementa `Stack`, ou seja, é uma pilha, porém, encadeada.
- A partir da classe `Node` e da interface `Queue`, crie uma classe chamada `LinkedQueue`, que implementa `Queue`, ou seja, é uma fila, porém, encadeada.

Atente para o fato de que você deve criar as classes solicitadas utilizando a classe `Node` que vimos em aula, não a classe `DNode`. Além disto, a classe `Node` não pode ser alterada para ser utilizada neste desafio.