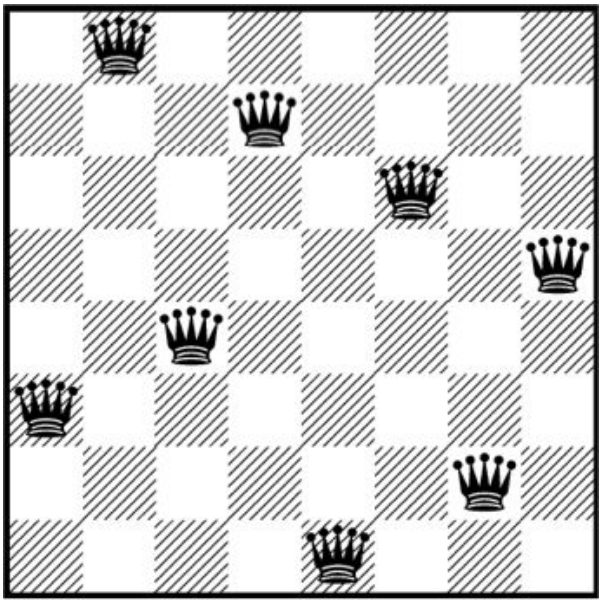
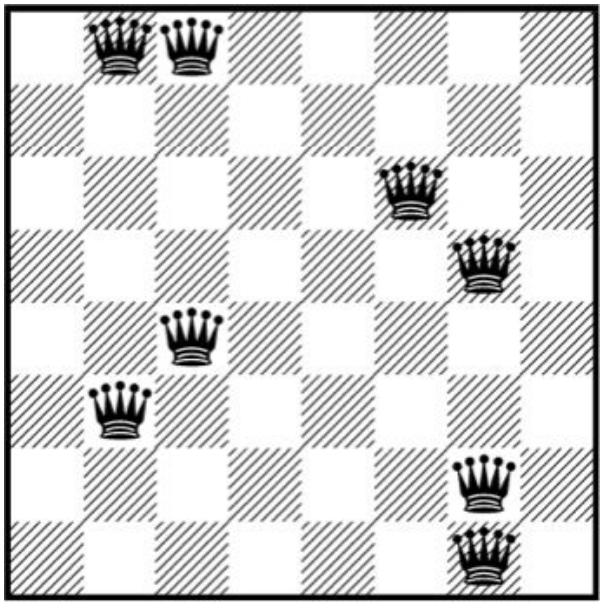
Damas (in)válidas

O puzzle das oito rainhas consiste em colocar oito rainhas num tabuleiro de xadrez sem que nenhuma possa ser capturada. Nesse caso diz-se que as rainhas estão numa configuração válida. Caso contrário, isto é, se alguma rainha poder capturar outra, tem-se uma configuração inválida.

Por exemplo, para um tabuleiro de 8x8 com 8 rainhas, a configuração da esquerda é válida, enquanto a da direita não é:





Programa 1: validador

> Implemente um programa (designado “validador”) que verifique se uma dada configuração é válida. O seu programa deverá ser implementado por uma classe chamada `Validador`.

A configuração é lida do `System.in` e é dada como uma String de comprimento 64 formada apenas pelos carateres `D` ou `-` , que resulta de concatenar as representações das oito linhas. *Deverão ser aceites strings com espaços*, que serão imediatamente ignorados, podendo assim – por exemplo – o input ser dado como uma linha de 64 caracteres, ou ainda como uma linha com 8 blocos de 8 caracteres, separados por espaços.

O seu programa deve:

- Aceitar *argumentos na linha de comando*, i.e. deve olhar para o parâmetro `String[]` passado ao método `public static void main (String[] args)`.
- Ler a configuração do `System.in`. Na variante 1B deverá continuar a ler configurações até ao fim do fluxo.

Variante 1A: validador individual

Se não houver argumentos (i.e. `args.length == 0`), o programa deve escrever (no `System.out`) uma só linha, com o texto `VALIDA` ou `INVALIDA` conforme a configuração dada for, ou não, válida.

Neste caso, espera-se que o programa leia exatamente uma e uma só configuração, por exemplo:

Input	Output
`-D---------D---------D---------D--D-----D------------D-----D---`	`VALIDA`
`-DD-----------------D--------D---D------D------------D-------D-`	`INVALIDA`

Variante 1B: filtro validador

Se o programa for chamado com um argumento igual ao string `"filtro"` (pode-se testar com `args[0].equals ("filtro")`, por exemplo) este deverá repetidamente ler configurações do `System.in`, escrevendo no `System.out` só aquelas que são válidas, no formato mais simples (uma linha de 64 caracteres por configuração de tabuleiros 8x8).

Programa 2: gerador

Implemente um programa que gere configurações, recebendo parâmetros na linha de comando e escrevendo-as no `System.out`. O seu programa deverá ser implementado por uma classe chamada `Gerador`. O output deverá ter o mesmo formato que o input do programa 1 (linhas com uma configuração cada).

O seu programa deverá aceitar as seguintes opções de produção:

- **`random M Q N`**

  Produz N configurações aleatórias de Q rainhas em tabuleiros de dimensão M*M. M, Q e N são inteiros. Verifique que Q <= M.

- **`all M`** (bónus)

  Produz todos os tabuleiros possíveis de tamanho M*M, com M raínhas.

Normas de Resolução

As classes deverão ser definidas **sem package**.

A sua resolução do programa 1 deverá satisfazer as seguintes condições:

- Implemente as classes públicas **`Linha`**, **`Coluna`**, **`DiagonalAscendente`**, **`DiagonalDescendente`** e **`Tabuleiro`**.
- Implemente uma classe abstrata, **`Peca`**, para descrever o que está numa posição. Essa classe deve ter, pelo menos, duas subclasses concretas, **`Nada`** e **`Rainha`**.
- A numeração da posição das peças obedece ao pressuposto das coordenadas (0, 0) estarem no canto superior esquerdo.

Adicionalmente, será valorizada a conformidade com os seguintes aspetos:

- **`Peca`** deve ter os métodos e variáveis de instância:
  - **`Peca(Tabuleiro tab, int linha, int coluna)`**: construtor para uma nova **`Peca`** dentro do Tabuleiro **`tab`**, que ficará diretamente posicionada na linha e coluna indicadas, sendo também atualizada a peça contida nessa posição, dentro do tabuleiro.

  - **`int linha()`**. Método que devolve a linha em que a peça se encontra. Por exemplo:

    ```
    Peca p = new Rainha(t, 2, 3); assert p.linha() == 2;
    ```

  - **`int coluna()`**. Método que devolve a coluna em que a peça se encontra. Por exemplo:

    ```
    Peca p = new Nada(t, 2, 3); assert p.coluna() == 3;
    ```

- **`boolean podeIrPara (int linha, int coluna`**). Método que indica se a peça se pode deslocar para a posição **`linha`** / **`coluna`**. Por exemplo:

```
Peca r = new Rainha(t, 2, 3); assert r.podeIrPara(3, 4);
```

- **`final boolean ataca (Peca vitima) { return podeIrPara(vitima.linha(), vitima.coluna()); }`**: Método que indica se a peça a que se aplica pode atacar aquela indicada como seu argumento. Este método deve ser definido e implementado na classe abstrata **`Peca`**. Por exemplo:

```
if (estaPeca.ataca (fila[i])) return "Mata Esfola!!";
```

- **`boolean vazia ()`**: Método que indica se a peça que se encontra na casa onde se situa é **`Nada`**. Noutras palavras, deverá retornar **`true`** para a classe **`Nada`** e **`false`** nas outras classes.

- **`Tabuleiro`** deve ter os métodos:
  - **`Tabuleiro(String repr)`**: construtor que tem como argumento um **`String`** que representa uma configuração. Assume-se que **`repr`** tem comprimento **`M*M`**, em que **`M`** é o número de linhas e de colunas. **`M`** poderá ser calculado como a parte inteira da raíz quadrada do comprimento de **`repr`**.

  - **`Peca peca(int linha, int coluna)`** Método que devolve a Peça que está na posição **`linha x coluna`**. Por exemplo:

```
Tabuleiro tab = new Tabuleiro("D-D…");
Peca p = tab.peca(0, 0); // uma Rainha
Peca q = tab.peca(0, 1); // uma Nada
```

  - **`boolean ameacada(int linha, int coluna)`** Método que indica se a posição **`linha x coluna`** está ameaçada por alguma peça no tabuleiro. Por exemplo:

```
Tabuleiro tab = new Tabuleiro("D-D…");
assert tab.ameacada(0, 1); // pela Rainha em 0,0
```

  - **`Linha linha(int linha)`** Método que devolve a linha **`linha`**.
  - **`Coluna coluna(int coluna)`** Método que devolve a coluna **`coluna`**.

    - `DiagonalAscendente diagonalAscendente(int linha, int coluna)` Método que devolve a diagonal ascendente que passa na posição `linha x coluna`.
    - `DiagonalDescendente diagonalDescendente(int linha, int coluna)` Método que devolve a diagonal descendente que passa na posição `linha x coluna`.

- As classes `Linha` , `Coluna` , `DiagonalAscendente` e `DiagonalDescendente` devem todas implementar a interface `Fila` que exige os seguintes métodos:
    - `int comprimento()` para retornar o número de posições nessa fila.
    - `int pecas()` para retornar o número de peças (isto é, posições ocupadas) nessa fila.
    - `Peca peca(int pos) throws IndexOutOfBoundsException` que retorna a ocupação que está na posição pos da fila e, se a posição não for válida, emite uma exceção `IndexOutOfBoundsException`.

Relatório

Deverá entregar um relatório com o **máximo de 3 páginas A4**, em formato PDF, e no qual aborda as seguintes questões:

- Apreciação das características do Java que empregou para realizar o seu código.
- Limites de funcionamento do programa.
- (bónus) Análise de desempenho: até que dimensão é que o seu programa consegue resolver o problema? (`gerador all N` ligado a `validador filtro`).

Pontos Extra (bónus)

As seguintes extensões à funcionalidade do programa poderão resultar num bónus na nota, caso sejam corretamente explicadas, documentadas e implementadas.

O bónus poderá chegar a 5 valores na nota.

1. Documente as classes usando os comentários do Javadoc.
2. Resolva o problema para as outras peças de xadrez. Caso não as conheça, veja na página da wikipedia sobre o xadrez como se deslocam as peças. Use a seguinte designação para as peças:

    a. `rei: R`
    b. `torre: T`
    c. `cavalo: C`
    d. `bispo: B`
    e. `peão: P`
3. Resolva o problema de encontrar todas as soluções do puzzle das oito rainhas.

Modalidades de entrega do trabalho

O trabalho deverá ser submetido no Moodle, em forma a anunciar.