

# Projeto 2

LabP em Blocos



Unidade Curricular de  
Laboratório de Programação

2021/2022

# Objetivos

- Manipulação de matrizes;

## Antes de Começar

- Fazer *download* do ficheiro `studentsProject2.zip` acessível na página de LabP;  
Este ficheiro contém:
  - Uma classe `RunProject2.java` com a qual pode experimentar jogar o jogo e os esqueletos das classes que deve implementar;
  - Ficheiros de *input* para a classe `RunProject2.java`;
  - Ficheiros de testes para os métodos que terá que implementar;
  - Uma pasta com ficheiros de *output* que pode utilizar para comparar os resultados obtidos pela sua solução com os resultados esperados.
- No Eclipse,
  - crie um projeto Java e “arraste” para a pasta `src` os ficheiros `RunProject2.java`, `Cell.java`, `Piece.java` e `Game.java`;
  - arraste os ficheiros de *input* para a raiz do projeto;
  - finalmente, importe os testes JUnit das duas pastas `TestesGame` e `TestesPiece` tal como descrito na página 14 do guião “Introdução ao IDE Eclipse”.

## Algumas informações úteis

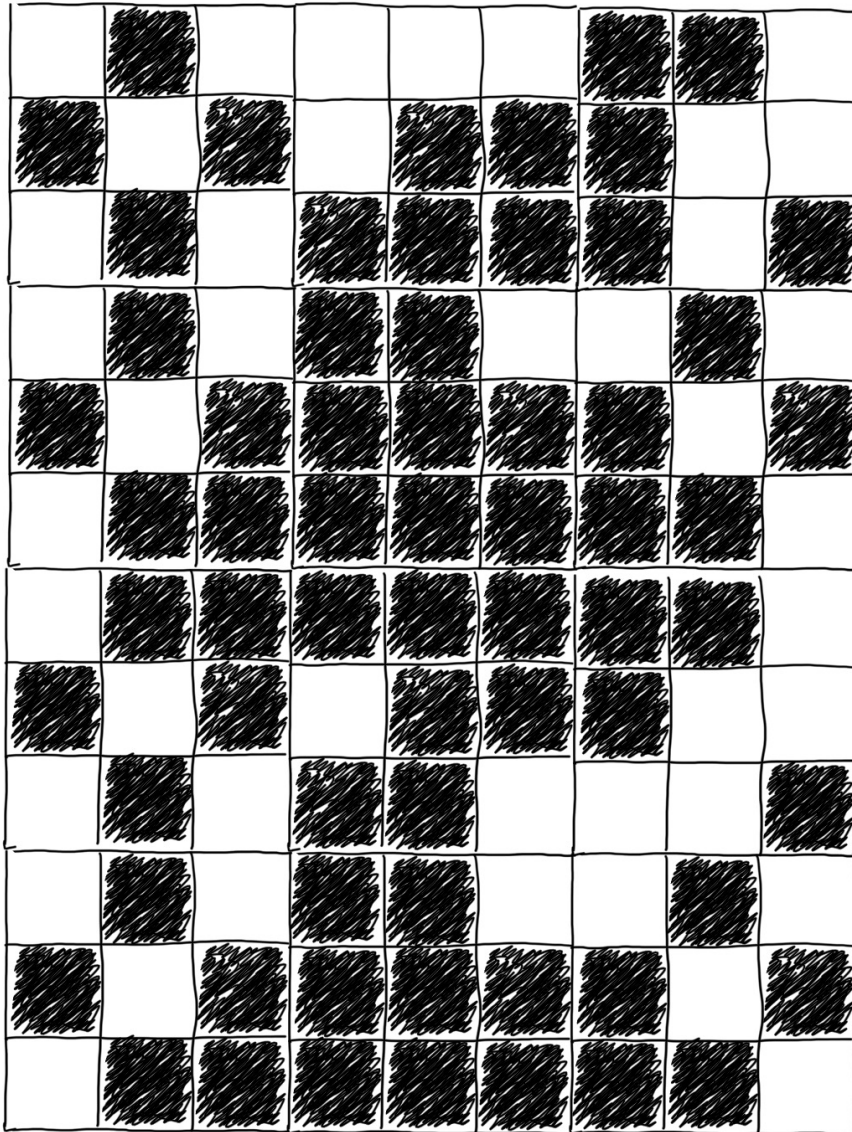
De modo a poder realizar este projeto, poderá consultar informação:

- disponibilizada na disciplina de IP sobre *arrays*,  
(<https://introcs.cs.princeton.edu/java/14array/>);
- Capítulo 9 do livro *online* em  
<http://www.di.fc.ul.pt/~in/PCOlivro/PCOonline/Index.htm>.

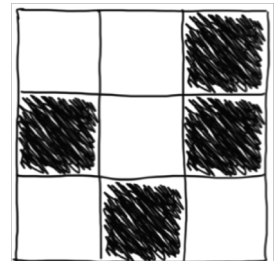
com foco especial em *arrays* multi-dimensionais

# Enunciado

O jogo LabP em Blocos é um jogo de tabuleiro cujo objetivo é encaixar peças que vão sendo geradas aleatoriamente com o intuito de preencher todo o tabuleiro. Cada peça tem tamanho fixo 3x3 em que cada posição está livre ou ocupada. Por cada peça encaixada, o jogador acumula os pontos correspondentes ao número de posições ocupadas dessa peça.



Próxima Peça



PONTUAÇÃO  
66

Se o jogador conseguir encaixar a peça em cima à direita, ganhará mais 4 pontos.

## O que fazer então?

O ficheiro `studentsProject2.zip` contém, entre outros elementos, a estrutura base das classes necessárias ao Jogo – Piece e Game –, que o aluno terá que desenvolver. Contém também um enumerado Cell, que define os valores BUSY e FREE, e ficheiros que permitem experimentar e testar as classes a desenvolver. O

material disponibilizado compreende também a pasta `expectedOutputs` contendo os ficheiros de *output* esperados, correspondentes aos ficheiros de *input* dados.

Descrevem-se de seguida as classes `Piece` e `Game`.

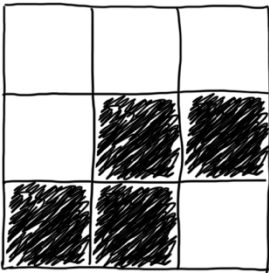
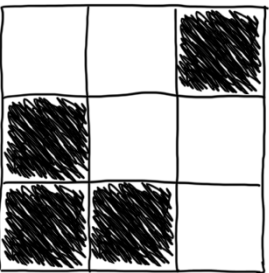
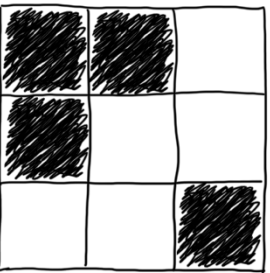
As instâncias da classe `Piece` representam peças a colocar no tabuleiro de jogo. Os métodos públicos que oferece permitem, por um lado, obter informação sobre uma peça e, por outro lado, realizar operações sobre uma peça. São eles:

- `Piece (Cell[][] shape)` – dada uma forma (necessariamente três por três) constrói uma peça com essa forma;
- `Piece (Cell type)` – dado um valor do enumerado, constrói uma peça em que todas as posições têm desse valor;
- `Piece (Random generator)` – dado um gerador aleatório, constrói uma peça em que os valores de todas as posições são definidos aleatoriamente;
- `int weight ()` – indica quantas posições desta peça têm o valor `BUSY`;
- `Cell[][] getCells()` – devolve uma cópia do conjunto de células desta peça;
- `void flipHorizontal()` – reflete esta peça horizontalmente;
- `void flipVertical` – reflete esta peça verticalmente;
- `void rotate90()` – roda esta peça 90 graus no sentido horário;
- `String toString()` – devolve a representação textual desta peça, sendo que as posições `FREE` são representadas por “\_” e as posições `BUSY` por “B”;
- `boolean isEqual(Piece other)` – devolve `true` se esta peça é igual a `other`;
- `boolean fits(Piece other)` – devolve `true` se esta peça, na sua representação atual, encaixa em `other` (no conjunto das duas peças, só pode haver no máximo um `BUSY` por posição);
- `Piece merge(Piece other)` – assumindo que esta peça encaixa em `other`, constrói uma nova peça que resulta de unir esta peça a `other`;
- `Piece copy()` – devolve uma nova peça com a mesma forma desta peça.

Exemplos de operações sobre uma peça:

Original	Reflexo Horizontal	Reflexo Vertical	Rodado 90 graus

No exemplo seguinte pode ver como uma peça encaixa noutra (método `fits`).

Peça 1	Peça 2	Peça 3
		

A peça 2 não encaixa na peça 1, isto é, o resultado das invocações `peca1.fits(peca2)` e `peca2.fits(peca1)` deverá ser *false*;

A peça 3 já encaixa na peça 1, isto é, o resultado das invocações `peca1.fits(peca3)` e `peca3.fits(peca1)` deverá ser *true* (repare que a peça 3 é obtida da peça 2 por rotação).

A peça 3, apesar de ser obtida da peça 2 por rotação, não é igual à peça 2 (isto é, o resultado das invocações `peca2.isEqual(peca3)` e `peca3.isEqual(peca2)` deverá ser *false*).

O método `weight` invocado sobre qualquer uma destas peças deverá devolver 4.

As instâncias da classe `Game` representam jogos cujo objetivo é encaixar o maior número possível de peças num tabuleiro. Os métodos públicos que oferece permitem saber qual é a peça correntemente em jogo, aplicar movimentos sobre essa peça e (tentar) encaixá-la no tabuleiro. O jogo termina quando não é possível encaixar a peça corrente ou quando todo o tabuleiro está ocupado.

São os seguintes os métodos públicos da classe `Game`:

- `Game(int x, Random generator)` – cria uma instância do jogo em que `x` posições, escolhidas aleatoriamente usando `generator`, estão ocupadas; por omissão, o tabuleiro terá 20 linhas por 12 colunas; considere que se `x` for maior que o número de posições do tabuleiro, este fica todo ocupado; é gerada aleatoriamente (usando generator) a primeira peça a jogar;
- `Game(int l, int c, int x, Random generator)` – que cria uma instância do jogo cujo tabuleiro tem `l` linhas e `c` colunas e em que `x` posições, escolhidas aleatoriamente usando `generator`, estão ocupadas; considere que se `x` for maior que o número de posições do tabuleiro, este fica todo ocupado; é gerada aleatoriamente (usando generator) a primeira peça a jogar;
- `Piece getCurrentPiece()` – devolve uma cópia da peça correntemente em jogo;
- **boolean** `isFinished()` – devolve *true* se o jogo já terminou (se o tabuleiro está todo ocupado ou a última peça jogada não encaixou);
- **void** `rotatePiece(int n)` – faz uma rotação à peça em jogo que é `n` vezes múltipla da rotação de 90°;
- **void** `flipHorizontal()` – reflete horizontalmente a peça em jogo;
- **void** `flipVertical()` – reflete verticalmente a peça em jogo;

- **void** play(**int** linha, **int** coluna) – efetua uma jogada com a peça correntemente em jogo, de tal modo que o centro dessa peça deverá encaixar no tabuleiro exatamente na linha e coluna dadas; os valores linha e coluna devem ser válidos para a colocação de uma peça 3x3 (não podem, por exemplo, ser 0 e 0); se a peça em jogo não encaixar, deve ficar registado que o jogo terminou; se o jogo não terminou, é gerada a nova peça corrente;
- **int** getScore() – devolve o valor atual da pontuação deste jogo, que corresponde ao número total de posições ocupadas no tabuleiro;
- **String** toString() – devolve a representação textual do jogo, composta pelo tabuleiro, pela peça correntemente em jogo e pela pontuação atual (ver formato nos ficheiros de *output* disponibilizados).

**ATENÇÃO:** Os ficheiros de *output* na pasta `expectedOutputs` fornecida, e os resultados esperados nos testes JUnit, estão de acordo com uma maneira bem definida de fazer a geração aleatória do tabuleiro e das peças a jogo.

Se os alunos querem usar os testes JUnit e a classe `RunProject2` fornecidos para verificarem a sua soluções, devem respeitar o seguinte:

- Na classe `Game`:
  - Para todas as gerações de aleatórios, usar sempre o mesmo gerador que é dado no construtor;
  - gerar primeiro os valores para preencher o tabuleiro e só depois a primeira peça a jogo;
- Na classe `Piece`:
  - Por ordem crescente de linha e, para cada linha, por ordem crescente de coluna, gerar um inteiro aleatório no intervalo [0,1] e interpretar o 1 como `Cell.FREE` e o 0 como `Cell.BUSY`.

## Antes de entregar

Antes de entregar, certifique-se da correção da formatação e inclua comentários *javadoc* para todos os métodos.

Inclua nas classes `Game` e `Piece` uma linha contendo a *tag* `@author` onde deve constar o seu nome e número de aluno.

## O que entregar

Deve criar o ficheiro `P2fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros `Game.java` e `Piece.java`

## Importante

O facto das vossas classes passarem nos testes fornecidos não significa que a classe esteja 100% correta. Há testes e verificações que não são feitas de propósito, de modo a incentivar os alunos a irem à procura de pontos de eventuais falhas no código. Além disso, os pesos para a avaliação dados a cada um dos testes pode ser diferente.