# Projeto 3

## Water Sort Puzzle


Unidade Curricular de
Laboratório de Programação

2023/2024

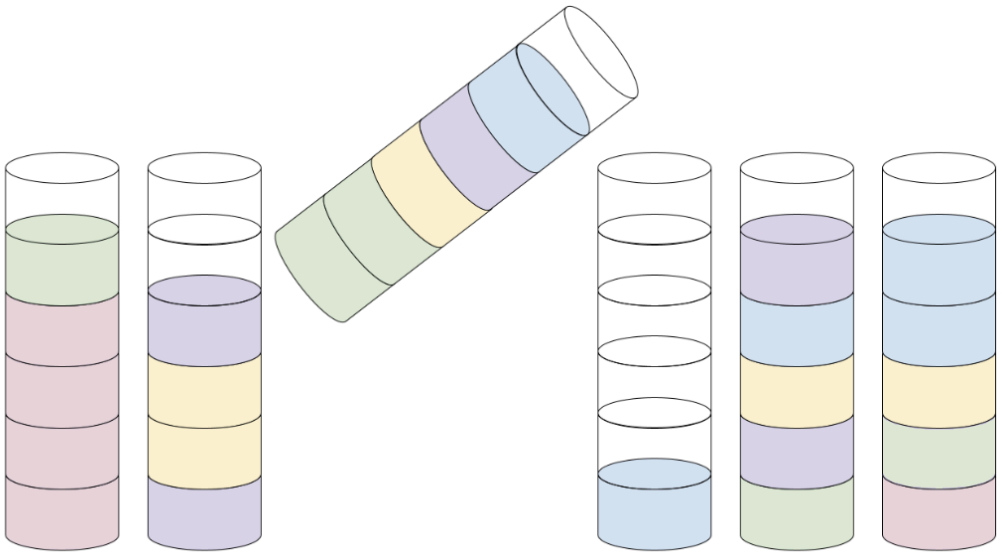
# Objectivos

- Compreensão do conceito de Tipo de Dados Abstrato (TDA)
- Uso de pilhas
- Algoritmia

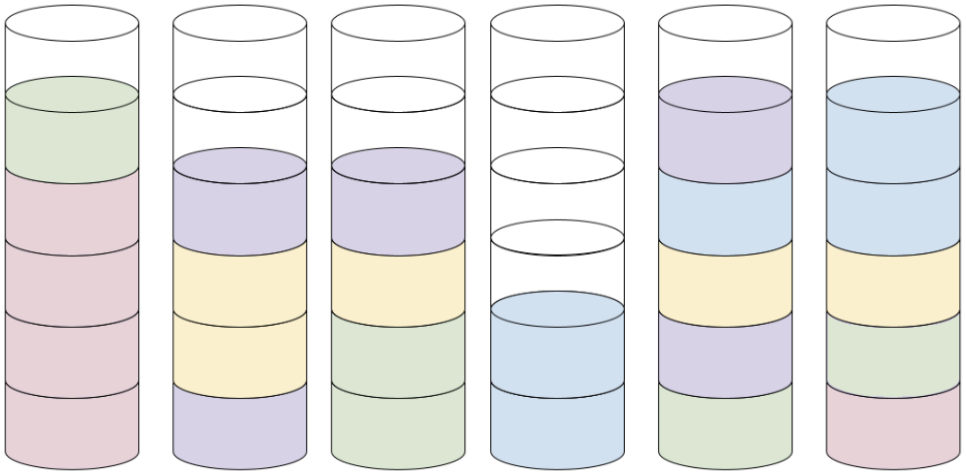
# Antes de Começar

Neste projecto pretende-se que os alunos exercitem o uso de classes e de **Pilhas** para implementar **uma solução do tipo de jogo Water Sort Puzzles**. Para isso devem recordar as operações básicas em pilhas.

# Enunciado



Os "Water Sort Puzzle" são jogos de quebra-cabeças viciantes e desafiadores que se tornaram muito populares em dispositivos móveis. O objetivo principal do jogo é ordenar corretamente líquidos coloridos em tubos de maneira a formar uma sequência de cores uniforme em cada recipiente. Neste jogo são apresentados vários recipientes, cada um contendo uma quantidade variada de líquidos coloridos que não se misturam, e de capacidade limitada. No final de cada ronda, cada recipiente só pode conter um tipo de líquido e cada um dos copos que contêm algum líquido deve ficar completamente cheio. Para atingir o objetivo de completar uma ronda, podem-se verter os líquidos de uns recipientes para outros desde que se o recipiente que recebe esteja vazio ou, tendo espaço, tenha no topo a cor igual ao do recipiente que vai verter. Cada movimento de um recipiente para outro é feito por goles. No caso da imagem apresentada acima, cada recipiente tem capacidade máxima de 5 goles e o copo com índice 3 vai receber um gole do recipiente com índice 2. Após esse movimento o estado dos recipientes é o seguinte:



Note que agora já não pode verter do recipiente de índice 2 para o recipiente de índice 3, uma vez que as cores no topo de um e de outro são diferentes. Repare também que não pode verter do

recipiente de índice 2 para o recipiente de índice 4, uma vez que este último está cheio. No entanto, pode por exemplo verter do recipiente de índice 2 para o recipiente de índice 1.

**Regras Básicas:**

**Recipientes e Conteúdos**: O jogo geralmente começa com uma mesa contendo vários recipientes, cada um contendo líquidos de diferentes cores . Estas cores podem variar de um nível para outro. Os líquidos dos recipientes podem ser trocados por outros enchimentos, com o mesmo significado e efeito.

**Objetivo**: O objetivo final é ter uma distribuição dos líquidos de forma a que cada recipiente (se não estiver vazio, deverá estar cheio) contenha apenas uma cor, e que uma cor pertença apenas a um recipiente. Nalgumas versões o jogo tem um número máximo de jogadas para cada ronda.

**Mistura de Cores e Restrições à Capacidade**: Para reorganizar os líquidos de modo que cada tubo contenha apenas uma cor, pode verter-se o líquido de um tubo para outro com as particularidades de 1) não se pode verter uma cor por cima de outra diferente e 2) não se pode exceder a capacidade máxima de um recipiente.

# Objectivo do trabalho

O objetivo do trabalho é implementar uma solução que permita a um utilizador jogar um jogo do tipo descrito acima com a criação de classes descritas a seguir.

# O que fazer

Os alunos devem implementar:

1. A classe **Bottle**, que deve ser **obrigatoriamente** implementada com uma **pilha**. Para tal podem usar a classe Stack do pacote java.util (https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html) ou, alternativamente, uma das implementações do interface Stack usadas em Algoritmos e Estruturas de Dados, e cujos ficheiros .java lhe fornecemos.
   A classe **Bottle** tem definidas as constantes:

```
public static final int DEFAULT_BOTTLE_CAPACITY = 5;
public static final String EMPTY = "⬜";
public static final String EOL = System.lineSeparator();
```

e tem definidos os métodos:

**public** Bottle() constrói uma garrafa vazia com tamanho DEFAULT_BOTTLE_CAPACITY

**public** Bottle(**int** capacity) constrói uma garrafa vazia com a capacidade dada

**public** Bottle(Filling[] content) constrói uma garrafa com tamanho de content e com o conteúdo dado, sendo que o Filling que fica no topo da garrafa é o primeiro elemento do vetor

**public boolean** isFull() informa se esta garrafa está cheia

**public boolean** isEmpty() informa se esta garrafa está vazia

**public** Filling top() quando a garrafa não está vazia, informa que gole está no topo

**public int** spaceAvailable() diz quantos goles ainda pode receber a garrafa

**public void** pourOut() executa a operação de retirar um único gole da garrafa

**public boolean** receive(Filling s) executa a operação de adicionar um gole s ao topo da garrafa e informa se a operação foi feita com sucesso.

**public int** capacity () diz qual o tamanho da garrafa

**public boolean** isSingleFilling() diz se a garrafa tem um só tipo de conteúdo

**public** Filling[] getContent() devolve uma cópia do conteúdo da garrafa

**public** String toString() devolve uma representação textual, escrita na vertical, do conteúdo da garrafa, do topo para a base; naturalmente, o próximo elemento a sair da garrafa aparece na posição mais acima.

**public** Iterator<Filling> iterator() que cria um iterador sobre o conteúdo da garrafa

2. A classe **Table**, que representa um conjunto de garrafas que depois irão ser usadas no jogo. A classe tem as constantes:

```
public static final String EMPTY = "⬜";
public static final String EOL = System.lineSeparator();
public static final int DIFFICULTY = 3;
public static final int DEFAULT_BOTTLE_CAPACITY = 5;
```

e os métodos:

**public** Table(Filling[] symbols, **int** numberOfUsedSymbols, **int** seed, **int** capacity) constrói uma mesa cujas garrafas estão preenchidas com elementos de symbols, que corresponde a uma representação vetorial do enumerado (i.e. symbols contém todos os símbolos possíveis de usar no jogo). O número de símbolos usados é o mínimo entre o numberOfUsedSymbols e o tamanho de symbols. A mesa deve ter espaço para albergar uma garrafa para cada símbolo usado mais a dificuldade definida por defeito. A capacidade máxima das garrafas é capacity. O conteúdo das garrafas é gerado de forma aleatória, escolhendo entre os símbolos possíveis para formar cada garrafa e usando a semente seed. Para tal, deve começar por construir vetores com tamanho igual a capacity, que deve preencher de trás para a frente. Só depois deve construir a garrafa e adicioná-la à sua estrutura que guarda as garrafas desta mesa.

**public void** regenerateTable() constroí um novo conteúdo para as garrafas desta mesa com o mesmo esquema do construtor anterior (i.e. há de usar os mesmos símbolos usados anteriormente, garrafas com a mesma capacity e a mesma seed)

**public boolean** singleFilling(**int** i) diz se a garrafa com índice i desta mesa é composta por um só tipo de conteúdo

**public boolean** isEmpty(**int** i) diz se a garrafa com índice i desta mesa está vazia

**public boolean** isFull(**int** i) diz se a garrafa com índice i desta mesa está cheia

**public boolean** areAllFilled() diz se todas as garrafas não vazias estão totalmente cheias com um só tipo de conteúdo

**public void** pourFromTo(**int** i, **int** j) concretiza a ação de verter um único gole da garrafa no índice i para a garrafa no índice j

**public void** addBottle(Bottle bottle) que adiciona uma dada garrafa ao conjunto de garrafas da mesa

**public int** getSizeBottles() diz qual a capacidade das garrafas

**public** Filling top(**int** i) quando a garrafa não está vazia, diz qual o tipo de gole que se encontra no topo da garrafa no índice i

**public** String toString() dá uma descrição textual do conteúdo da mesa. Note que precisa de pensar numa forma de devolver o conteúdo das garrafas representando cada uma das garrafas na vertical. Veja os exemplos dos testes para concretizar a ideia

3. A classe **Game** que tem os métodos:

   **public** Game(Filling[] symbols, **int** numberOfUsedSymbols, **int** seed, **int** capacity) constrói um jogo em que os conteúdos para usar nas garrafas na mesa são dados por contents, o número de símbolos a usar por numberOfUsedSymbols, a semente do gerador de aleatórios usado para gerar o enchimento de cada garrafa e o tamanho a usar das garrafas por capacity

   **public** Game(Filling[] symbols, **int** numberOfUsedSymbols, **int** seed, **int** capacity, **int** score) constrói um jogo igual ao anterior com um dado score

   **public** Bottle getNewBottle() gera uma garrafa extra para ser usada na mesa como ajuda

   **void** play(**int** i, **int** j) efetua uma jogada vertendo o conteúdo da garrafa com o índice i para a garrafa com o índice j. De notar que o conteúdo a verter são **todos** os goles possíveis de verter da garrafa no índice i para a garrafa no índice j

   **boolean** isRoundFinished() diz se a ronda está acabada, isto é, se todas as garrafas estão totalmente cheias com um único conteúdo ou vazias

   **void** startNewRound() gera uma nova mesa com novas garrafas

   **int** score() indica qual a pontuação atual do jogo

   **int** jogadas() retorna o número de jogadas efetuadas até ao momento

   **void** provideHelp() permite ao jogador obter uma ajuda criando uma nova garrafa vazia, resultando numa penalização de 100 pontos

   **int** updateScore() se o jogador resolver o enigma da ronda, dá 1000 pontos no caso do número de jogadas ser inferior a 10 (inclusive), 500 se o número de jogadas for entre 10 (exclusive) e 15 (inclusive), 200 entre 15 (exclusive) e 25 (inclusive) e 0 nos restantes casos

4. A classe **Main** com um método main cuja criação é da total responsabilidade de cada aluno e que exercite todas as funcionalidades do jogo que acabaram de criar.

# Material Fornecido

São fornecidos:

- o enumerado **Filling** cujos valores serão usados como conteúdos das garrafas;
- o interface Stack e as implementações LinkedStack e ArrayStack das aulas de Algoritmos e Estruturas de Dados (de uso opcional; podem optar pela classe Stack de java.util);
- os esqueletos das classe **Bottle**, **Table** e **Game**.

# O que entregar

Deve criar o ficheiro `P3fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros: `Bottle.java, Table.java, Game.java` e `Main.java`