Projeto

André Souto

Unidade Curricular de
Programação Centrada em Objetos

2023/2024

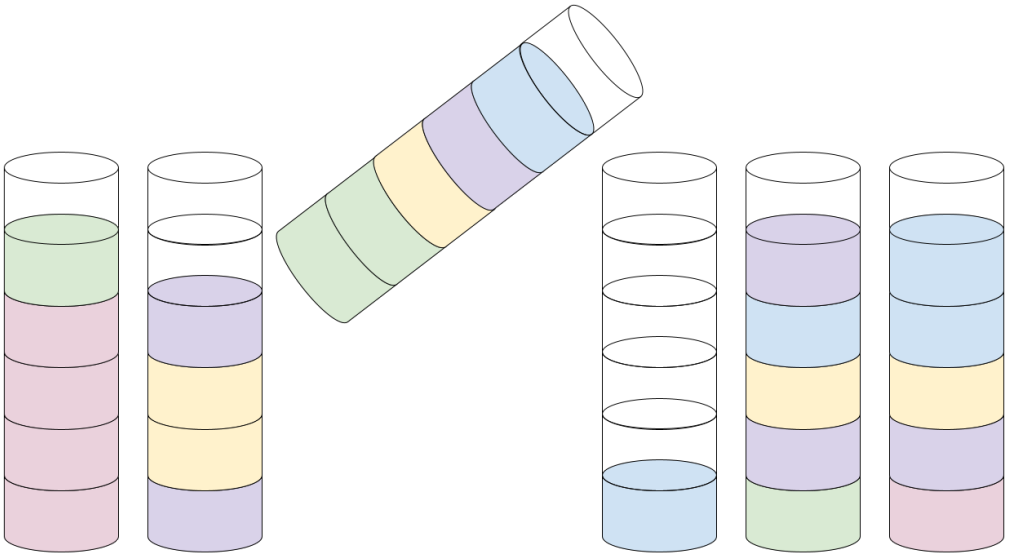
Objetivos

- Algoritmia
- Tipos e subtipos
- Encapsulamento
- Genéricos
- Herança
- Polimorfismo

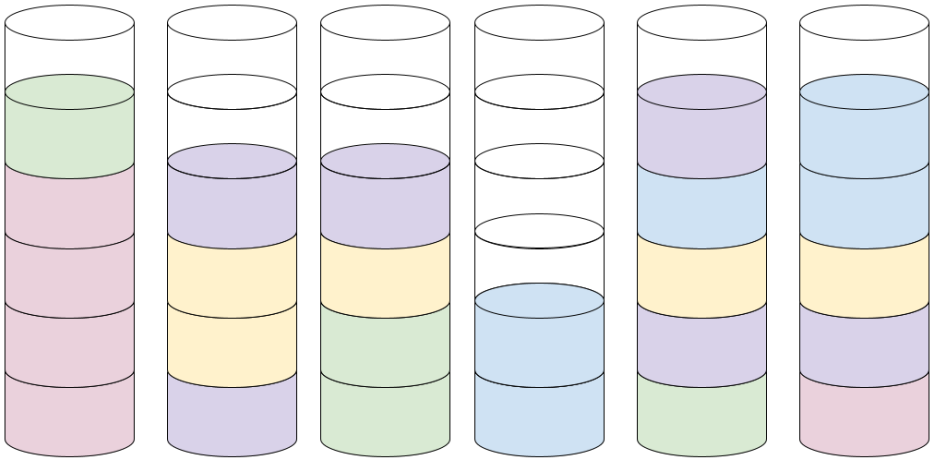
Antes de Começar

De modo a poder realizar este projeto deverá estudar e recordar o que aprendeu sobre os assuntos mencionados anteriormente.

Enunciado



Os "Water Sort Puzzle" são jogos de quebra-cabeça viciante e desafiador que se tornaram muito popular em dispositivos móveis. O objetivo principal do jogo é ordenar corretamente os líquidos coloridos em tubos ou recipientes 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 que cada um dos copos que contem 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 golos. No caso da imagem apresentada cada recipiente tem capacidade para 5 golos e o copo com índice 3 vai receber um golo do recipiente 2. Após o movimento o estado dos recipientes é o seguinte:



Note que no agora já não pode verter do recipiente índice 2 para o recipiente de índice 3 (e nem para o de índice 5 porque está cheio), mas pode por exemplo verter o conteúdo para o recipiente 1.

Regras Básicas:

Recipientes e Líquidos: O jogo geralmente começa com uma mesa contendo vários recipientes, cada um com líquidos de diferentes cores. Estas cores podem variar de um nível para outro ou serem tapados por outros enchimentos, mas com o mesmo significado e efeito.

Objetivo: O objetivo final é organizar os recipientes de forma que cada um contenha apenas uma cor e que não haja líquidos restantes nos outros recipientes. Nalgumas versões o jogo tem um número máximo de jogadas para cada ronda. Portanto, precisas ser estratégico e eficiente na tua abordagem.

Mistura de Cores: Para reorganizar os líquidos de modo que cada tubo contenha apenas uma cor, pode verter-se o líquido de um tubo para outro com a particularidade de não se poder verter uma cor por cima de outra diferente ou que não caiba no recipiente.

Restrições: Além da restrição da mistura de cores, a quantidade de líquido que se pode verter de um recipiente para outro é limitada ao pelo tamanho do recipiente.

Objetivo 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. Classe Bottle que implementa o serviço de um iterador, tem definidas as constantes:

   ```
   public static int DEFAULT_SIZE = 5;
   public static String empty = " ";
   ```

   e tem definidos os métodos

   public Bottle() que constrói uma garrafa vazia com tamanho DEFAULT_SIZE = 5

   public Bottle(**int** size) que constrói uma garrafa vazia com tamanho dado

   public Bottle(Filling[] content) que constrói uma garrafa com o conteúdo dado

   public boolean isFull() que informa se esta garrafa esta cheia?

   public boolean isEmpty() que informa se esta garrafa esta vazia?

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

   public int spaceAvailable () que diiz quantos golos ainda pode serceber a garrafa.

   public void pourOut() que executa a operação de retirar um golo da garrafa.

   public boolean receive(Filling s) que executa a operação de adicionar um golo da garrafa s ao topo da garrafa.

   public int size () que diz qual o tamanho da garrafa

   public boolean isSingleFilling() a garrafa tem um só tipo de conteúdo?

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

   public String toString() que devolve uma representação textual do conteúdo da garrafa de baixo para cima.

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

2. A classe Cup que é um subtipo de Bootle, tem definidas as constantes:

   ```
   public static final int CUP_SIZE = 1;
   private static final int TIMES_OF_USAGES = 3;
   ```

   e tem, além dos métodos herdados, definida os métodos
   public Cup() que constrói um copo vazio com tamanho DEFAULT_SIZE

   public Cup(Filling[] content) que constrói um copo cujo conteúdo é primeiro elemento de content.

e re-define o método

public boolean receive(Filling s) que quando o número de usos excede 3, deixa de poder receber novos conteúdos.

3. A classe Table que representa um conjunto de garrafas que depois irão permitir jogar. A classe tem as constantes

public static final int DIFICULTY = 3;
public static final int DEFAULT_BOTTLE_SIZE = 5;

e os métodos

public Table(Filling[] contents, **int** numberOfUsedSymbols, **int** seed, **int** bootleSize) Controí uma mesa usando o enumerado contents que é composta em que o número de símbolos usados é o mínimo entre o numberOfUsedSymbols e o tamanho do enumerado utilizado como conteúdo das garrfas, mais a Dificuldade definida por defeito. O tamanho das garrafas é bootleSize e o conteúdo é gerado de forma aleatória escolhendo de entre os símbolos possíveis para formar cada garrafa.

public void regenerateTable() que controí um novo conteúdo para as garrafas da table

public boolean singleFilling(**int** x) que diz se a garrafa com índice x da table é composta por um só conteúdo

public boolean isEmpty(**int** x) que diz se a garrafa com índice x da table está vazia

public boolean isFull(**int** x) que diz se a garrafa com índice x da table está cheia

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

public void pourFromTo(**int** i, **int** j) que concretiza a ação de verter um golo da garrafa i para a garrafa j.

public void addBootle(Bottle bottle) que adiciona uma bootle dada ao conjunto de garrafas da mesa.

public int getSizeBottles() que diz qual o tamanho das garrafas originais;

public Filling top(**int** x) que quando a garrafa não está vazia, bradiz qual o tipo de golo que se encontra no topo da garrafa dada pelo índice x da table.

public String toString() que dá uma descrição textual do conteúdo da table.

4. Uma interface FillingGame:

   void play(**int** x, **int** y) que despoleta uma jogada vertendo o conteúdo da garrafa com o índice x para a garrafa com o índice y;

   boolean isRoundFinished() que diz se a rounda está acabada, isto é se todas as garrafas estão cheias com um único conteúdo ou vazias.

   void startNewRound() que providencia o serviço de gerar uma nova mesa com novas garrafas;

   void provideHelp() que permite ao jogador obter uma ajuda criando uma nova garrafa (ou um seu subtipo) dependendo do jogo ques tá a ser jogado.

   Filling top (**int** x) que indica qual o golo que está no topo da garrafa x;

   boolean singleFilling(**int** x) que indica se a garrafa x tem apenas um conteúdo;

   boolean areAllFilled(**int** x) que retorna true se todas as garrafas estão cheias com o mesmo conteúdo ou vazias.

   int score() que indica qual a pontuação do jogo;

5. Classe abstrata `AbstractFillingGame` que implementa a interface Game. O construtor e os métodos abstratos são:

   public `AbstractFillingGame(Filling[] contents, int numberOfUsedSymbols, int seed, int bootleSize)` que constrói um jogo abstracto em que os conteúdos 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 é dado por seed e o tamanho a usar das garrafas por bootleSize.

   protected abstract `Bottle getNewBootle();` que há-de gerar uma garrafa extra para ser usada na mesa;

   public abstract void `updateScore();` que há-de atualizar o score do utilizador no jogo

   public abstract int `score();` que há-de dizer o score atual do jogo

6. Classe concreta `FinalScoringGame` que é um subtipo da classe `AbstractFillingGame` e implementa a interface Game e tem dois construtores naturais:

   public FinalScoringFillingGame(Filling[] symbols, **int** numberOfUsedSymbols, **int** seed, **int** bootleSize)

   public FinalScoringFillingGame(Filling[] symbols, **int** numberOfUsedSymbols, **int** seed, **int** bootleSize, **int** score)

   @Override
   public int updateScore() que se o jogador resolver o enigma da ronda dá 1000 pontos se o número de jogadas for inferior a 10, 500 se o número de jogadas for entre 10 e 15, 200 entre 15 e 25 e 0 caso contrário.

e redefine os métodos score(), toString() bem como outros que ache necessários para que o score do jogo seja calculado quando é gerada uma nova mesa e que depende do número de jogadas. Se a ronda for completada com sucesso, com no máximo 10 jogadas, o score é 1000 pontos, se for completado entre 10 e 20, o score é 500 pontos e 200 caso contrário. Se não for completada com sucesso deve-se descontar 100 pontos.

7. Classe concreta `BettingScoringGame` que é um subtipo da classe `AbstractFillingGame`, implementa a interface `Game` e tem o construtor:

   public BettingFillingGame(Filling[] symbols, **int** numberOfUsedSymbols, **int** seed, **int** bootleSize, **int** score, **int** bet, **int** maxPlays) que constrói um jogo em que os conteúdos 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 é dado por seed e o tamanho a usar das garrafas por bootleSize. Além disso é dado também dado um score inicial uma aposta com a qual o jogador vai jogar bet e maxPlays é o número abaixo do qual o utilizador se propõe a resolver cada ronda.

   Além disso redefine os métodos score(), updateScore(), isRoundFinished toString() bem como outros que ache necessários para que o score do jogo seja atualizado quando uma ronda terminada (o que pode acontecer no fim de uma jogada) e que depende se já foi atingido o número máximo de jogadas e se as garrafas estão finalizadas.

   @Override
   public void provideHelp() - retirando uma jogada ao número máximo de jogadas permitidas e criando um Cup em vez de uma bottle.

   @Override
   public int score() que devolve o score atual do jogo.

   @Override
   public int updateScore() que se o jogador resolver o enigma da ronda em menos de maxPlays jogadas recebe 2 vezes o valor que apostou e fica com o valor da bet caso contário.

   @Override
   public boolean isRoundFinished() indicando se a ronda com aquelas garrafas já terminou ou o número de jogadas máximo já foi ultrapassado.

   @Override
   public String toString() que dá uma descrição textual do jogo.

   @Override
   protected Bottle getNewBootle() - que devolve como ajuda a criação de um Cup.

8. Classe Main com um main de criação da total responsabilidade de cada grupo que exercite todas as funcionalidades do jogo que acabaram de criar.

9. Um documento com a descrição em UML da estrutura da sua solução.

Material Fornecido

São fornecidos: A interface Filling e os enumerados Balls, Animals, Squares que implementam Filling para serem usados como conteúdos das garrafas e os esqueletos das classe Bottle, Cup, Table, AbstractFillingGame, FinalScoringGame e BettingScoreGame.

Observações durante a realização do projeto

- Algumas das especificações são deliberadamente deixadas laxas para dar oportunidade ao alunos de apresentarem soluções originais;
- Durante a realização do projeto é natural que surjam alguns problemas com algumas implementações que eu não antecipei. Aí devem falar comigo para esclarecer,
- Devem inferir a formação dos métodos toString dos inputs e outputs to testes;
- Os testes fornecidos não são exaustivos havendo situações limite que poderão não estar testadas com o intuito de serem avaliadas durante a prova oral;
- A classe Main não está implementada. os alunos deverão usar a classe Scanner na ajuda da implementação para permitir ao utilizador jogar o jogo.

Atenção

Antes de submeter o trabalho, certificando-se que **TODOS** os passos descritos no documento são cumpridos, que entrega todas as classes incluindo a classe Main bem como o documento com a descrição do UML.

O que entregar

Deve criar o ficheiro **projectoGrupoX.zip**, contendo todos os ficheiros pedidos. Cada grupo deve fazer apenas uma submissão (ou seja, só um dos elementos do grupo é que deve submeter o trabalho, não precisam de submeter todos). Devem fazer as submissões até dia 30 de Novembro às 23h59.

Avaliação

Na avaliação do projeto serão tidos em conta, entre outras coisas, a qualidade e complexidade do código, a documentação, a correção da solução, desempenho e contributo de cada membro do grupo. Não se esqueçam que a nota é individual.