UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Faculdade de ciências e tecnologia da Universidade do Algarve

Curso de licenciatura em engenharia informática

Programação Orientada por Objetos

Ano letivo 2015/16

Relatório 4

Problema 4 - 24

Grupo 10

Alexandra Brito nº50075

Cyrill Brito nº52875

Gonçalo Pedras nº51702

UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

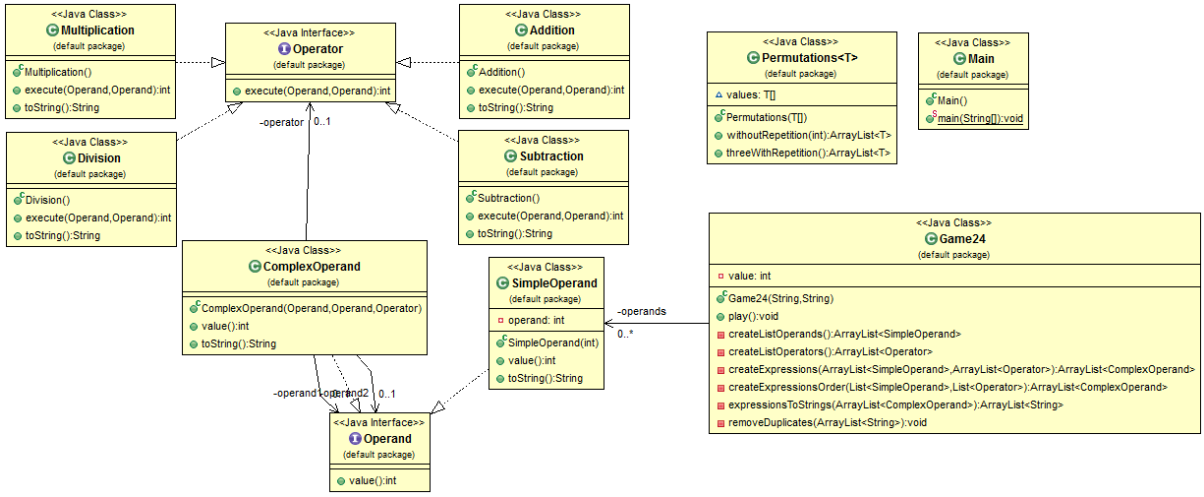
Índice

Introdução

Existe um jogo, o “twenty-four game” onde participam quatro jogadores, onde há um baralho de cartas. Em cada carta há quatro números naturais positivos. O primeiro jogador retira a primeira carta e lê os números. O objectivo é desenvolver uma expressão aritmética envolvendo esses números, cada um desses valores apenas uma vez, e os quatro operadores, e o resultado dessa expressão tem de ser 24. O objectivo deste trabalho é criar um programa que dado um conjunto de quatro números, e um resultado, calcule uma expressão para dar o resultado, que não tem de ser necessariamente 24, ou seja, o programa vai devolver um conjunto de expressões possíveis para tal resultado.

Diagrama UML

Este diagrama UML foi criando através do addon sugerido pelo professor, ObjectAid. Aqui está demostrado todas as classes do programa com respectivos atributos e funções.



Interfaces

- Operand

  Esta interface permite criar um guia para os operados simples e compostos. Esta interface tem a função *'value'*, que devolve o valor do operando.

- Operator

  Esta interface cria um guia para todos os operadores possíveis no problema. A interface apenas tem uma função, '*execute'*, que tem como objectivo receber dois operandos e devolver o valor da operação.

Classes

- Addition, Subtraction, Multiplication Division

  Estas quatro classes representam as quatro operações possíveis no problema, soma, subtracção, multiplicação e divisão. Todas elas implementam a interface '*operator*'. Cada classe apenas tem duas funções, o '*execute*' que faz a operação e o '*toString*' que devolver o caracter representante da operação.

- SimpleOperand

  Permite criar um operando simples, ou seja um operando que apenas tem um número. A classe implementa a interface '*Operand*', logo tem a função value que devolve o valor do número e tem a função '*toString*' que devolve a representação do operando.

- ComplexOperand

  Permite criar um operando composto, ou seja quando o operando já é um resultado de uma operação. A class implementa a interface '*Operand*', logo tem a função '*value*' que devolve o valor do operando e também tem a função '*toString*' que devolve a representação do operando composto.

- Permutations

  Esta é uma classe genérica com o objectivo de criar permutações de vários tipos, a classe aceita um array de qualquer tipo e permite criar todas as permutações sem repetição dos objectos do array ou fazer todas as permutações para três posições com repetição.

- Game24

  Esta é a classe em que é resolvido o problema.

Testes Unitários

No primeiro teste são inseridos quatro numeros todos diferentes. Este teste é o mais simples, em que existe menos posibilidade de ocurer erros.

No segundo teste são inseridos dois numeros iguais, para teste se o programa não esta a devolver resultaos repetidos.

```
@Test
public void test1() {
    String numbers = "12 5 10 8";
    String value = "34";

    Game24 game = new Game24(numbers, value);

    ArrayList<String> gameList = game.play();
    ArrayList<String> testList = new ArrayList<>();

    testList.add("((10-8)*(12+5))");
    testList.add("((10-8)*(5+12))");
    testList.add("((12+5)*(10-8))");
    testList.add("((5+12)*(10-8))");

    for (int i = 0; i < testList.size(); i++)
        assertEquals(gameList.get(i), testList.get(i));
}
```

```
@Test
public void test2() {
    String numbers = "12 16 22 12";
    String value = "150";

    Game24 game = new Game24(numbers, value);

    ArrayList<String> gameList = game.play();
    ArrayList<String> testList = new ArrayList<>();

    testList.add("(((12*12)+22)-16)");
    testList.add("(((12*12)-16)+22)");
    testList.add("((12*12)+(22-16))");
    testList.add("((22+(12*12))-16)");
    testList.add("((22-16)+(12*12))");
    testList.add("(22+((12*12)-16))");

    for (int i = 0; i < testList.size(); i++)
        assertEquals(gameList.get(i), testList.get(i));
}
```

No terceiro são iseridos quatro numeros iguais.

No ultimo test foram isereidos numeros que não têm resultados para verificar se o programa não devolve zero valores.

```
@Test
public void test3() {
    String numbers = "1 1 1 1";
    String value = "4";

    Game24 game = new Game24(numbers, value);

    ArrayList<String> gameList = game.play();
    ArrayList<String> testList = new ArrayList<>();

    testList.add("(((1+1)+1)+1)");
    testList.add("((1+(1+1))+1)");
    testList.add("((1+1)*(1+1))");
    testList.add("((1+1)+(1+1))");
    testList.add("(1+((1+1)+1))");
    testList.add("(1+(1+(1+1)))");

    for (int i = 0; i < testList.size(); i++)
        assertEquals(gameList.get(i), testList.get(i));
}
```

```
@Test
public void test4() {
    String numbers = "12 6 5 7";
    String value = "34";

    Game24 game = new Game24(numbers, value);

    ArrayList<String> gameList = game.play();

    assertEquals(gameList.size(), 0);
}
```

Discussão

Inicialmente criámos as interfaces '*Operand*' e '*Operator*', que vão ser implementadas nos operandos e nos operadores respectivamente, como foi explicado anteriormente. O programa lê da consola 4 números, e de seguida lê o resultado para calcular, ambas como String e cria um objecto da classe '*Game24*' onde vai receber essas duas Strings. Esse construtor vai separar a String dos quatro números num array, converte essas String para Integer e passa-as para o array de '*SimpleOperand*', e guarda finalmente o valor do resultado a calcular na variável '*value*' como um Integer.

De seguida é executado pela '*Main*' o método '*play*()' do objecto game24. Este método vai criar um ArrayList de operandos chamando o método "createListOperands()", este que vai criar uma lista de Permutações de Operandos simples, onde vai permutar os quatro valores, ou seja, vai retornar uma lista com todas as combinações possíveis da posição dos números.

Depois é criada uma lista de operadores que vai chamar o método "createListOperators()", este vai declarar um array de Operadores das 4 operações aritméticas, e uma lista de permutações de Operadores que vai conter o array dos Operadores declarados e onde é feita todas as combinações possíveis dos operadores (onde serão usados apenas três operadores, isto porque cada expressão com quatro números sem repetição só haverá 3 operações possíveis).

Ainda no mesmo método é declarado uma lista de Operandos Complexos que vai conter as expressões criadas pelo método "CreateExpressions()", passando como argumento as listas criadas anteriormente neste mesmo método. O "CreateExpressions()" vai juntar os operandos e os operadores tudo numa mesma lista, quatro em quatro operandos e três em três operadores, isto tudo com todas as combinações possíveis já anteriormente feitas e guardadas nessas listas que foram passadas para argumento, e chama o método "createExpressionsOrder()" para ordenar a ordem das operações e verificar se é o resultado esperado.

Agora é declarada uma lista de String que vai conter todas as expressões cujo resultado está certo, isto tudo em String para ser enviado para a consola para o utilizador, e para isso vai ser chamado o método “expressionsToStrings()” que vai receber como argumento uma lista com as expressões, ou seja, uma lista de operandos complexos, e vai retornar uma lista de Strings.

Agora é chamado método “removeDuplicates()” que passa como argumento a lista de Strings. Este método vai criar uma lista de HashSet que vai receber todas as expressões contidas na lista de Strings, esta lista como não aceita repetições vai conter as expressões menos as que estão a mais. É feito um “clear()” na lista de Strings e finalmente adiciona-se as expressões guardadas no HashSet para a lista de Strings e agora sem nenhuma repetição, e retorna o mesmo.

Para finalizar este método “play()” é usada a classe “Collections” (a classe de todas as colecções/listas), e é chamado o método “sort()” para ordenar as expressões na lista. A condição “if” que vem a seguir servirá para ver se foi possível criar expressões para o resultado que queremos, caso não haja, retornará na consola um “void”. Caso passar esse teste será feito um “for each” da lista de Strings com a expressão para escrever na consola as expressões obtidas e válidas para os números escritos na consola.