UFAZ / Strasbourg University

Object Oriented Programming

Year 1 – Common curriculum

Tutorial / Lab session #5: Exceptions & class diagram

Exercise 1 – Improving our parametrized array

In this exercise, we are going to keep working with the class `MyArray<E>` that we created and implemented in the last tutorial.

1. Modify the method `void add(E e)` so that it throws an exception `MaxCapacityException`.
2. Modify the method `void get(int index)` so that it throws an exception `InvalidIndexException`.
3. Write a method `int find(E e)` that returns the index of the object `e` given as argument. If the element is not found, the method throws an exception `ElementNotFoundException`.
4. Create the classes
   a. `MaxCapacityException`
   b. `InvalidIndexException`
   c. `ElementNotFoundException`
   d. What class these 3 classes must extend?

Exercise 2 – Class diagram

1. Using UML, represent the class diagram of `MyArray<E>`. Don't forget to include the exceptions in your diagram!
2. Using UML, represent the class diagram of the classes created in tutorials 3 and 4 (arithmetic expressions)

Exercise 3 – Parenthesis matching

In this exercise, our objective is to create a Java program that will check if parentheses in a string are balanced. Here are a few examples of strings we want to be able to analyze:

1. `((a+b)-(c+d)) // correct`
2. `a+(b+(c+d)) // correct`
3. `((a+b)+c // incorrect`
4. `((a+b)+c)) // incorrect`
5. `)a+b( // incorrect`

A very common way to perform such analysis is by using a stack. The algorithm is quite simple and goes as follows:

For each character in the string to analyze **do**
  If the current character is an opening parenthesis **then**
    Push the character on the stack
  Endif
  If the current character is a closing parenthesis **then**
    If the stack is empty then
      Return false // this is an extra closing parenthesis
    Endif
    Pop the last entry of the stack
    If the parenthesis doesn't match then //the last entry is not an opening parenthesis)
      Return false
    Endif
  Endif
End **loop**

Algorithm 1 - Parenthesis matching using a stack

1. Modify the class `MyStack` that we created during tutorial 2:
    a) Make the class generic (`MyStack<E>`) and adapt the code of the class accordingly.
    b) In the method `void push(E e)` of your class, deal with the problem of having reached the capacity of the stack using the exception `MaxCapacityException` defined previously.
    c) Create a class `EmptyStackException`. This exception must be thrown by the methods `Object pop()` and `Object peek()`.
    d) Override the method `String toString()` so you can print a stack (the stack itself, as well as its capacity and the current top)
2. Create a class `ParenthesisMatching`. Instance variables of this class are a stack of character and a string to parse (don't forget the getters and setters).
3. Write the constructor of the class: the constructor should take a string as argument (the string we want to parse). The capacity of the stack will be the length of that string.
4. Write the private method `static boolean bracketsAreMatching(char c1, char c2)` that returns true if characters c1 and c2 are matching parenthesis (i.e. if c1 is a '(' and c2 is a ')').
    a) What does "static" mean?
5. Write the method `boolean parse()` that implements the algorithm 1.
6. Add the class `ParenthesisMatching` to the diagram you draw in exercise 2.1.