A Calculator

Mahdi Nazari

September 2022

Introduction

In this assignment will a program similar to HP-35 calculator be programmed. Using stack in this calculator is essential. Therefor the static stack and dynamic stack will be presented. The differences between these to types of stacks will be explained in a benchmark. In the last part will the formula for finding the last digit in a personal number be transformed into RPN(Reverse Polish Notation) and then be put in the program to find the last digit.

Static stack

A class of the static stack was implemented. Static stack has a known capacity from the start and those essential operation in a static stack are push, pop and also peak. E.g. code for the function push is as follows:

```
public void push(int item){
        if(pointer < capacity){
            stack[++pointer] = item;
        }
        else{
            System.out.println("Stack is full!");
        }
    }
```

The stack pointer was declared at -1 and when the "push" function is triggered the stack pointer will cerement by one and then the value of argument will be copied to the array.

The calculator is built on RPN principle. To make the whole program work for the calculator the classes Item, ItemType and Calculator was completed. Item class has been completed with som method as getValue, setValue, getType and setType which are going to be used in the program.

Step function in the class Calculator was completed by arithmetic operations as SUB, DIV, MUL and another method which would decide if the the item is an integer(in this case) it should push it on the stacks. This was possible with a switch-case function and enum special class. E.g. MUL function which has been added to the switch-case in the "step" function in the Calculator class is follows:

```
case MUL : {
      int y = stack.pop();
      int x = stack.pop();
      stack.push(x * y);
      System.out.println("The value is pushed: " + (x*y));
      break;
  }
```

Dynamic Stack

A class of dynamic stack was implemented. The main difference between a static stack and a dynamic stack is that in dynamic stack the size of stack can increase or decrease. This mechanism was implemented by methods which will be called as "increase" and "decrease" and these functions were added to the push and pop functions which also was used in the static stack. For example the function increase is as follows:

```
public void increase(){
       int length = size();
       if(length < capacity/2){
           int[] newStack = new int[capacity*2];
           for(int i=0; i<capacity; i++){
               newStack[i] = stack[i];
           }
           stack = newStack;
           capacity *= 2;
       }
   }
```

The function "increase" will call another function which is called "size" which calculate the size of the stack that is free and the function looks if the free space is less than the total capacity divided by 2 then it will create a new array which is 2 times larger and the elements will be copied in the larger array.

Benchmarking

In this part will the runtime of static stack and dynamic stack will be compared to each other. To make it simple a function which is called "benchMark" has bveen created and this function is as follows:

```
public long benchMark(int element, int times){
        long t0 = System.nanoTime();
        for(int i=0; i<times; i++){
            for(int j=0; j<element; j++){
                push(1);
            }
            for(int k=0; k<element; k++){
                pop();
            }
        }
        long t1 = System.nanoTime();
        long t_total = (t1 -t0);
        return (t_total/(times*element*2));
    }
```

This function will calculate the time for a push or a pop in the static stack and dynamic stack. In this function "int times" is a constant with value of 100 and this was add to the code to make sure that the program run many times and then to ensure that the calculated time is more reliable.

Static stack: .
The measurements from the static stack is demonstrated in table 1. The number n is antal push and pop which have been performed and the runtime is in nanoseconds.

Number	**runtime**
10000	6
20000	4
40000	2
80000	1
160000	1
200000	2

Table 1: The measurements from benchmarking of static stack.

The results from measurements shows that the runtime of the static stack is regard to push and pop operations are approximately constant.

The results differs slightly but since it is about a few nanosecond they still can be seen as a constant time.

Dynamic stack: .
The measurements result from the dynamic stack is shown in the table 2. The number n is antal push and pop which have been performed and the runtime is in nanoseconds.

Number	**runtime**
10000	24
20000	12
40000	7
80000	5
160000	4
200000	3

Table 2: The measurements from benchmarking of static stack.

The measurements result shows that the runtime decrease in a dynamic stack when the number of push and pop operations increase and that is not as it was expected.(It should increase by as a log function) The reason is unclear it can be that the java is optimizing and it does not go through the whole code. It could be that there is a bug in the code which I could not find by the time.

Finding last digit in personal number

In this part a formula for the last digit was given and it should be transformed to RPN. Then the RPN version was tested in the Calculator which gave the right last digit of my personal number. The formula converted to RPN is as follows:

```
10(y1)2*´(y2)+(m1)2*´(m2)++(d1)2*´(d2)+(n1)2*´(n2)+++(n3)2*´+ 10 Modulo -
```

My personal number was inseted to the program and my last digit was gotten correctly.

```
10(9)2*´(9)+(0)2*´(2)++(1)2*´(2)+(1)2*´(4)+++(n3)2*´+ 10 Modulo - = 4
```