KADİR HAS UNIVERSITY

CE 343 Object Oriented Programming Languages

2018-2019 Fall

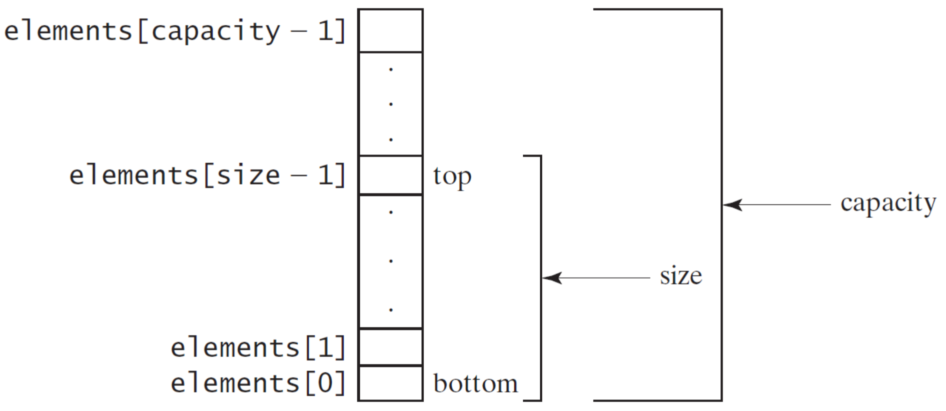
HW 2 – Objects and Classes

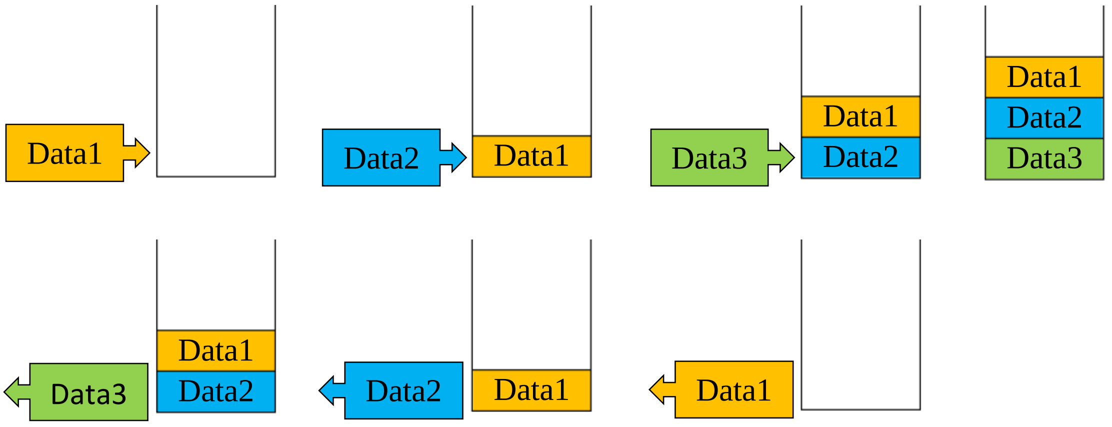
Due Date: Tuesday 06/11/2018 23:59

Submit your java **source** files (**.java**) via BlackBoard before the due date.
Maximum 2 students can work together. The file should contain the name of group members.
You are expected to provide **compile-able** and **executable** source code.
Any type of **shared** work with different groups will be considered **cheating**. Thus, do **not** share your work.

Task#1 StackBottom

StackBottom	
-elements: int[]	An array to store integers in the stack.
-size: int	The number of integers in the stack.
+StackBottom()	Constructs an empty stack with a default capacity of 4.
+StackBottom(capacity: int)	Constructs an empty stack with a specified capacity.
+empty(): boolean	Returns true if the stack is empty.
+peek(): int	Returns the integer at the **bottom** of the stack without removing it from the stack.
+push(value: int): int	Stores an integer into the **bottom** of the stack.
+pop(): int	Removes the integer at the **bottom** of the stack and returns it.
+getSize(): int	Returns the number of elements in the stack.
+toStringAll(): String	Returns a String representation of the **whole** stack.
+toString(): String	Returns a String representation of the **elements** in the stack.





Design a class named **StackBottom**.

Note that we push an integer into the **bottom** of the stack, which **shifts** elements to the top by one.

During insertion, if the capacity is **full**, we **double** the capacity of the stack.

During pop, we remove the integer at the **bottom** of the stack, which **shifts** elements to the bottom by one.

Below is a sample main method and its output:

```
    public static void main(String[] args) {
        StackBottom stack = new StackBottom();
        System.out.println(stack.toStringAll());
        for (int i = 11; i < 17; i++) {
            stack.push(i);
            System.out.println(stack.toStringAll());
        }
        while (!stack.empty()) {
            stack.pop();
            System.out.println(stack.toStringAll());
        }
        System.out.println("");
        for (int i = 11; i < 17; i++) {
            stack.push(i);
            System.out.println(stack.toString());
        }
        while (!stack.empty()) {
            stack.pop();
            System.out.println(stack.toString());
        }
    }
```

OUTPUT:

```
0       0       0       0
11      0       0       0
12      11      0       0
13      12      11      0
14      13      12      11
15      14      13      12      11      0       0       0
16      15      14      13      12      11      0       0
15      14      13      12      11      0       0       0
14      13      12      11      0       0       0       0
13      12      11      0       0       0       0       0
12      11      0       0       0       0       0       0
11      0       0       0       0       0       0       0
0       0       0       0       0       0       0       0

11
12      11
13      12      11
14      13      12      11
15      14      13      12      11
16      15      14      13      12      11
15      14      13      12      11
14      13      12      11
13      12      11
12      11
11
```

Task#2 MyBigNumber

We arrange a positive number $D_nD_{n-1}... D_2D_1$ in an integer array of length n as $\{ D_1, D_2, ..., D_{n-1}, D_n \}$.
The i^{th} digit of the number is stored in the i^{th} integer of the array.
Note that the least significant digit of the number, which is the digit in the right-most position, is stored in the first entry of the array. Therefore, the number 1465 is stored in an integer array of length 4 as { 5, 6, 4, 1 }:

	5	6	4	1
Indexes:	0	1	2	3

The class **MyBigNumber** represents a positive number, which is stored as an array of integers, as explained above.
Define a class named **MyBigNumber** that contains:

- private data field **digits** of type int[], is the list storing digits of the number. Each number is stored as an array of integers described above.
- **MyBigNumber** (int[]) is the constructor initiating a MyBigNumber object based on the digits given as parameter to the constructor. The integers given in the parameter, which represent the digits of the number, should be copied into a new array pointed by **digits** of the object. A MyBigNumber object should **not** contain redundant zeros. However, the integer list given as parameter can contain redundant zeros at the beginning (e.g., 000651), which should be removed. For example:
    - new MyBigNumber(new int[]{1, 5, 6}); represents number 651.
    - new MyBigNumber(new int[]{1, 5, 6, 0, 0, 0}); represents 651, therefore **digits** should **only** contain {1, 5, 6}.
    - new MyBigNumber(new int[]{0, 0, 5, 6, 0, 0, 0}); represents 6500.
    - new MyBigNumber(new int[]{0, 0, 0, 0, 0}); represents 0.
- public boolean **equals**(MyBigNumber) compares the MyBigNumber object with the one given as parameter and returns **true** if both numbers are equal, **false** otherwise.
- public boolean **greater**(MyBigNumber) compares the MyBigNumber object with the one given as parameter and returns **true** if the MyBigNumber object is greater than the one given as parameter, **false** otherwise.
- public String toString() returns a String representation of the number based on its digits. For example,
  (new MyBigNumber(new int[]{0, 1, 5, 6})).toString() returns "6510".
  (new MyBigNumber(new int[]{0, 0, 5, 6, 0, 0, 0})).toString() returns "6500".
- public MyBigNumber **addition**(MyBigNumber) returns a new MyBigNumber object which represents the sum of the MyBigNumber object and the one given as parameter. For example,
  (new MyBigNumber(new int[]{8, 2, 3})).addition(new MyBigNumber(new int[]{1, 4, 7, 9})) returns a new MyBigNumber object representing 328 + 9741 = 10069. The addition of two numbers should be realized digit by digit. The result of the addition is stored in an integer array, which is used to create a new MyBigNumber object.
- public MyBigNumber **absolute_difference** (MyBigNumber) returns a new MyBigNumber object which represents the absolute difference of the MyBigNumber object and the one given as parameter. The subtraction of two numbers is done digit by digit. The method always subtracts the smaller number from the bigger one to obtain a positive number. The result is stored in an integer array, which is used to create a new MyBigNumber object.

Below is a sample main method and its output:

```
public static void main(String[] args) {
   int[] n1 = {3, 4, 5};
   int[] n2 = {3, 4, 5, 6};
   int[] n3 = {0, 0, 5, 6, 0, 0, 0};
   int[] n4 = {0, 0, 0, 0, 0};
   MyBigNumber num1 = new MyBigNumber(n1);
   MyBigNumber num2 = new MyBigNumber(n2);
   MyBigNumber num3 = new MyBigNumber(n3);
   MyBigNumber num4 = new MyBigNumber(n4);
   System.out.println("num1 : " + num1);
   System.out.println("num2 : " + num2);
   System.out.println("num3 : " + num3);
   System.out.println("num4 : " + num4);
   System.out.println(num1 + " == " + num1 + " : " + num1.equals(num1));
   System.out.println(num1 + " == " + num2 + " : " + num1.equals(num2));
   System.out.println(num1 + " > " + num2 + " : " + num1.greater(num2));
   System.out.println(num2 + " > " + num1 + " : " + num2.greater(num1));
   System.out.println(num1 + " + " + num2 + " = " + num1.addition(num2));
   System.out.println(num2 + " + " + num1 + " = " + num2.addition(num1));
   System.out.println("| " + num1 + " - " + num2 + " | = "
                                       + num1.absolute_difference(num2));
   System.out.println("| " + num2 + " - " + num1 + " | = "
                                       + num2.absolute_difference(num1));
   System.out.println("| " + num3 + " - " + num2 + " | = "
                                       + num3.absolute_difference(num2));
   System.out.println("| " + num1 + " - " + num1 + " | = "
                                       + num1.absolute_difference(num1));
}
```

```
OUTPUT:
num1 : 543
num2 : 6543
num3 : 6500
num4 : 0
543 == 543 : true
543 == 6543 : false
543 > 6543 : false
6543 > 543 : true
543 + 6543 = 7086
6543 + 543 = 7086
| 543 - 6543 | = 6000
| 6543 - 543 | = 6000
| 6500 - 6543 | = 43
| 543 - 543 | = 0
```