COMP503/ENSE502/ENSE602: Week 6 – Exercises

➢ Exercise 1: Find the student

Design a **Student** class that

- stores the name of the student and a list of paper codes of the papers in which they are registered (separated by commas).
- has a toString method returning the student's name.
- has a constructor to initialize a Student object.

Design a **StudentList** class the comprises an ArrayList of Student.
Write **enrolledIn(String)** method that returns an array list of all students enrolled in a given paper code (consider the **contains** method in the String class).
For example, the output of the main method is:
[Bob, Jamal]

```
public static void main(String[] args) {
        StudentList studentList = new StudentList();
        studentList.add(new Student("Bob", "COMP503,COMP600"));
        studentList.add(new Student("Robin", "COMP501,COMP600"));
        studentList.add(new Student("Jamal", "COMP503"));
        System.out.println(studentList.enrolledIn("COMP503"));
}
```

COMP503/ENSE502/ENSE602: Week 6 – Exercises

- **Exercise 2: Numbers**

Java's ArrayList maintains a resizeable array. Create the **Numbers** class with the list instance variable.

```
import java.util.ArrayList;
public class Numbers {
        private ArrayList<Integer> list;
}
```

Find the Javadocs and lecture note for the ArrayList class and use its functionality to complete the Numbers class with the following methods

- a constructor that initialise list
- `public void addNumber(Integer i)` that adds a number to list
- `public void clear()`, removing all numbers from the list
- `public void printList()`,that prints the contents of the list to the console
- `public Integer min()`,returns the minimum number in the list
- `public Integer max()`, returns the maximum number in the list
- Write a ***main*** method which tests each of these methods and output them to the console.

- **Exercise 3: Queue Algorithms**

Create the **QueueAlgorithms** class with a main method.

- Write the **reverse** method, which takes as input a ***queue of integers*** and returns a ***queue*** with the elements in reverse order. You can use a Stack as auxiliary data storage.
  - For example,
    - reverse([5, 2, 6, 1]) returns [1, 6, 2, 5]
    - reverse([1,2,3,2,1]) returns [1,2,3,2,1]
- Write the **isPalindrome** method which takes as input a queue of integers and determines whether or not it is a palindrome. A sequence of numbers is a palindrome if it is the same in reverse order.
  - For example,
    - isPalindrome([1, 2, 3, 2, 1]) returns true
    - isPalindrome([]) returns true
    - isPalindrome([2,5]) returns false
- Write two methods **min** and **max** which takes as input a priority queue of integers, returning the minimum and maximum integer respectively. The number should not be removed from the priority queue.
  - For example
    - min([]) returns null;
    - min([1, 2, 6, 100]) returns 1
    - max([1, 2, 6, 100]) returns 100

Exercise 4: Stacks

The stack is a versatile data structure that we have already seen in Programming 2. Java and other programming languages use a stack data structure to store and update a stack of method frames which keeps track of the control flow when your program invokes nested method calls.

In this exercise, our stacks will store Integer and Character data. To use the **Stack** class in Java, we must import the library ***java.util.Stack***.

- Create a new class **StackTest** with a main method that declares and initialises the variable ***Stack<Integer> stack = new Stack<Integer>();***

We can use the stack ***push*** and ***pop*** operations to add and remove numbers from the stack. For example the following commands push two integer objects into the stack and then pop them off. What is the output printed to the console?

```
stack.push(1);
stack.push(2);
Integer i = stack.pop();
Integer j = stack.pop();
System.out.println(i);
System.out.println(j);
```

- Write a method which reads integer values from the console and pushes them onto a stack. When the user types quit the input values should be printed to the console in reverse order.

Challenge yourself😊

Let's consider the following creative problem. Suppose that you are writing a Java program that maintains a stack data type of integers. Your program demands that you determine the maximum and minimum value currently stored on the stack in constant running time. e.g. without iterating through the stack elements.

- Develop a specialised class called **MaxMinStack** which
  - maintains a stack instance variable of type Stack<Integer>,
  - implements the push and pop methods to update the stack,
  - implements getMax and getMin methods to instantaneously determine the current maximum and minimum value on the stack, without iterating over the stack elements.

  Hint: Your class should maintain instance variables ***Stack<Integer> max*** and ***Stack<Integer> min***, updating these stacks when integers are pushed and popped from the stack.

COMP503/ENSE502/ENSE602: Week 6 – Exercises

Here's a more challenging stack problem from *Sedgewick's Algorithms 4th edition book*.

- Write a method **brackets** that takes as input a string of brackets and uses a stack to determine whether its brackets are property balanced.
    - For example, the method should return true for the input array
        - brackets("(())[([])]") returns ***true***
          whereas
        - brackets("[(])]") return **false**.