PRACTICAL TASK 1
Data Structures

Question 1: ArrayList - Search and Delete Records

Analyse the data structure used in the task. Was the data structure fit for the task? Explain your answer. Is there another data structure that could be used in place of array list to complete the task?

ArrayList is the Array with dynamic size. The main reason to use this data structure for this task is that we don't know how many records there are in the file with patients. We can't use Array for this task, because Array has to have a fixed size.

Also, I noticed that we create the Object Patient, but instead of storing the objects in ArrayList we make string from the object and store strings. Seems like we could skip this step and store objects in our ArrayList, as it gives us more opportunities for searching, deleting and changing our records, and we always can make a string from a record using toString() method.

Question 2: Stack – Validate an Expression

Analyse the data structure used in the task. Was the data structure fit for the task? Explain your answer. Is there another data structure that could be used in place of stack to complete the task?

Stack is data structure which let the developer to add and take elements from the top. It works for this task. But because we don't really care *which* element we add (in this task we add to the stack only opening brackets), we could use just a counter instead - add 1 when we meet the opening bracket, subtract 1 when we meet closing bracket. On each step we would check a counter and if it goes below 0 we would realize that expression is not valid. We would check counter after we check all elements in the expression and if it is 0 the expression would be valid.

```
int brCounter = 0;
for (int i=0; i < expression.length(); i++){
 Character thisChar = expression.charAt(i);//read the symbol from the
expression
 if (thisChar.equals(leftBracket)){
     brCounter += 1;//add 1 if opening bracket
 }
 if (thisChar.equals(rightBracket)){
     brCounter -= 1;// subtract 1 if closing bracket
       if (brCounter < 0){ //if counter goes below zero - break the loop
                  break;
       }
 }
          }
 if (brCounter == 0) { //if counter equals 0 after the loop the
expression is valid
 System.out.println ("YAY!!! Your expression has the same number of "
             + "opening and closing brackets! We should celebrate!");
  }
  else {
  System.out.println ("Expression is not valid. Sorry :(");//in other
case the expression is not valid
       }
```

Using stack would make much more sense if we try to validate the expression with different kind of brackets, for example

(34kl)(There is funny picture {of cats})[34+049]

In this case it would matter *which* kind of element we put or take from the stack.

Question 3: Queue – Implement ticketing system in the shop.

Analyse the data structure used in the task. Was the data structure fit for the task? Explain your answer. Is there another data structure that could be used in place of queue to complete the task?

Queue works for this task. But, again like with the previous task, we could use two counters instead of it. One counter for the new tickets , second counter for the number of ticket which sales assistant is going to serve. There is only numbers, so we don't need to use complex data structures.

```
public class TicketingSystemTEST {

    static int ticketNumberLAST = 1;//create one counter
    static int ticketNumberFIRST = 1;//create second counter
```

```java
public static void main(String[] args) {
        Timer timer = new Timer();
        timer.schedule(new SalesAssisstantTask(), 0, 5000); //run code in the SalesAssistantTask run() method every 5 seconds
        timer.schedule(new CustomerInLine(), 0, 3000);//run code in the CustomerInLine run() method every 3 seconds
    }

 public static class CustomerInLine extends TimerTask { //inner class
     @Override
     public void run() {
       System.out.println("Customer with ticket " + ticketNumberLAST + " is added to the queue."); //theLine.add(ticketNumber);//add customer to the end of line
       ticketNumberLAST++;// increase ticket number counter
     }
 }

    public static class SalesAssisstantTask extends TimerTask { //inner class
      @Override
      public void run() {
        System.out.println("\nSales Assistant is ready to see the next customer.");
            if (ticketNumberLAST-1 == 0) { //if the line is empty
               System.out.println("There are no customers to see.");}
            else{
                System.out.println("The customer with ticket number " + ticketNumberFIRST + " will be seen");
                ticketNumberFIRST++;
            }
        }
    }
}
```