Practical 1

Question 1
//Dih Yoong

```
//Get the test scores
List<Integer> studentList = new ArrayList<>();
Integer score=0;



//Display all the test scores.
//Hao Han
for(int i=0; i< studentList.size(); i++){
   System.out.println("num[" + i + "] = " + studentList.get(i));
}


//Find and display the lowest score in the list.
//Jia Jian
 int lowest = studentList.get(0);
        for(int i = 1; i < studentList.size(); i++){
            if(studentList.get(i) < lowest)
                lowest = studentList.get(i);
        }

        System.out.println("The lowest score is" + lowest);


//Find and display the highest score in the list.
//Jun Yan
public static int findHighest(List<Integer> studentList){

        int highestScore = studentList.get(0);

        for(int i = 1; i < studentList.size(); i++){
            if(studentList.get(i) > highestScore){
                highestScore = studentList.get(i);
            }

        }
        return highestScore;
    }


//Compute and display the average of the scores in the list.
//Kah Yee
```

```
private static void findAverage(List<Integer> studentList) {
        double scoreAverage = 0.0;

        for(int i = 0; i < studentList.size(); i++){

            scoreAverage += studentList.get(i);
        }

        scoreAverage /= studentList.size();

        JOptionPane.showMessageDialog(null, "Average: " +
String.format(“%.2f”, scoreAverage), "Average",
JOptionPane.INFORMATION_MESSAGE);
    }
```

Question 2

```
//Choon Peng
//Read name and add a record to List
private List<Runner> runnerList= new ArrayList<>();

private void Button_DisplayActionPerformed(java.awt.event.ActionEvent evt)
{
      // TODO add your handling code here:
      JOptionPane.showMessageDialog(null, formatList());
}

private void Text_NameActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String name = Text_Name.getText();
        Runner runner= new Runner(name);
        Text_Num.setText(""+Runner.getNo());
        runnerList.add(runner);
        Text_Name.setText("");
        Text_Name.grabFocus();
    }
////
private void jtfNumberActionPerformed(java.awt.event.ActionEvent evt) {

        for(int i = 0 ; i < runnerList.size(); i ++){
            if(runnerList.get(i).getNumber() ==
                 Integer.parseInt(jtfNumber.getText())){
                jtfName.setText(runnerList.get(i).getName());
            }
        }
        if(jtfName.getText().isBlank()){
        JOptionPane.showMessageDialog(this,"Please Enter the Valid Data");
        --currentPosition;
       }
```

```
  }

//Joan
//Display Runner Info

 public String formatList() {
    String outputStr = "Marathon Results\n";
    for (int i = 0; i < runnerList.size(); ++i) {
      outputStr += (i + 1) + ". " + runnerList.get(i);
    }
    return outputStr;
  }


private void jbtDisplayActionPerformed(java.awt.event.ActionEvent evt) {
    JOptionPane.showMessageDialog(null, formatList());

  }
```

Question 3

```
//Khor Hui Shuang
for (int i = 0; i < exp.length(); i++)
        {

            char ch = exp.charAt(i);
            if (ch == '(' || ch == '[' || ch == '{')
            {
                stack.push(i);
            }
            else if (ch == ')'|| ch == ']' || ch == '}')
            {
                stack.isEmpty();
            }
        }
        System.out.println("Stack is empty: " + stack.isEmpty());
```

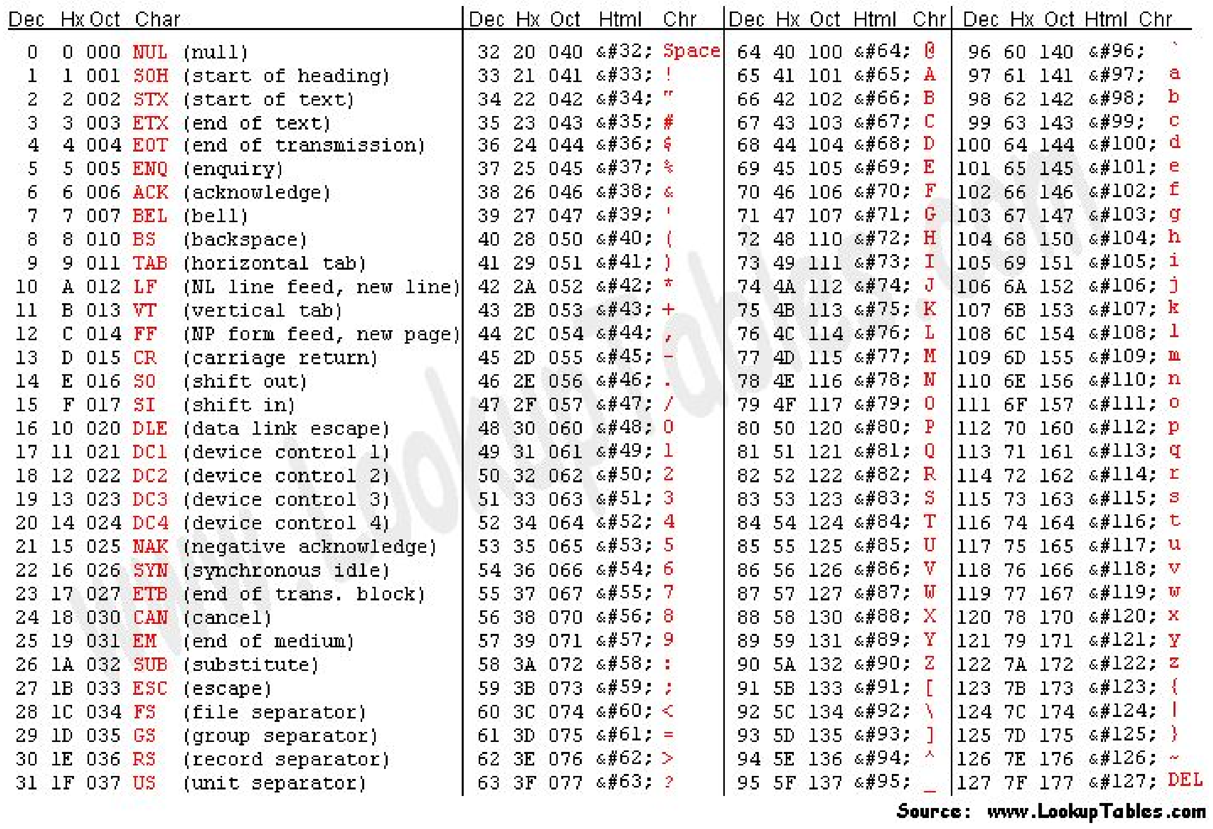
Question 4

```
//Loh Kean Min - operand

for(int i = 0; i < exp.length(); i++){
            char c = exp.charAt(i);

            //If the char is operand, push it to the stack
            if(Character.isDigit(c))
                stack.push(c - '0');
//Since my exp is String, so it need to be converted to int(c - ‘0’)
```

#exp store the equation, by using for loop to get the operand and push it to the stack



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```
Integer.parseInt()
int x =Integer.parseInt("9");

//Lim Kuan Xian - operator
for (int i = 0; i < postFix.length(); i++){
            char check = postFix.charAt(i);

            switch(check){
                    case '+' :
                        stk.push(stk.pop() + stk.pop());
                        break;
                    case '-' :
                        stk.push(stk.pop() - stk.pop());
                        break;
                    case '/' :
                        stk.push(stk.pop() / stk.pop());
                        break;
                    case '*' :
                        stk.push(stk.pop() * stk.pop());
                        break;
            }
}


//Lee Ling - evaluate the postfix expression
```

```
public static int evaPostfix(String postfixExpression){
        Stack<Integer> stack = new Stack<>();
        char current;

        for(int i = 0; i < postfixExpression.length(); i++){
            current = postfixExpression.charAt(i);

            if(Character.isDigit(current))
               stack.push(current - '0');
            else{
                int num1 = stack.pop();
                int num2 = stack.pop();

                switch(current){
                    case '+':
                        stack.push(num2 + num1);
                        break;

                    case '-':
                        stack.push(num2 - num1);
                        break;

                    case '*':
                        stack.push(num2 * num1);
                        break;

                    case '/':
                        stack.push(num2 / num1);
                        break;
                }
            }
        }
        return stack.pop();
    }
```

Question 5
//Lim Ming Yeu

```
public String reverse(String inputString) {

        Stack<Character> stack = new Stack<>();
        Queue<Character> queue = new LinkedList<>();

        for (int i = 0; i < inputString.length(); ++i) {
            if(inputString.charAt(i) != ' '){
                stack.push(inputString.toLowerCase().charAt(i));
                queue.add(inputString.toLowerCase().charAt(i));
            }
        }

        StringBuilder reversedString = new StringBuilder();
        while (!stack.empty()) {
            reversedString.append(stack.pop());
        }

        for (int i = 0; i < reversedString.length(); ++i) {
            if(reversedString.charAt(i) != ' '){
                stack.push(reversedString.charAt(i));
                queue.add(reversedString.charAt(i));
            }
        }

        StringBuilder inputStr = new StringBuilder();
        while (!stack.empty()) {
            inputStr.append(stack.pop());
        }

        if(reversedString.toString().equals(inputStr.toString())){
            return "palindrome";
        }
        else {
            return "not palindrome";
        }
  }
```

Question 6

```
//a) generate a sequence number - Chin Wai Kian

public class PostOfficeSim extends javax.swing.JFrame {
    private JTextField[] jtfDisplayRowArr = new JTextField[4];
    private String callString = " --> Counter ";
    private CounterListener counterListener = new CounterListener();
    private static int nextNumber = 1001;
    private int currentNo = nextNumber - 1;
    private Queue<Customer> q = new ArrayBlockingQueue<Customer>(100);
    private ArrayList<Customer> serviceList = new ArrayList<Customer>();
    private String counterStr = "Counter ";
    private int counterNoIndex = counterStr.length();

    /**
     * Creates new form PostOfficeSim
     */
    public PostOfficeSim() {
        initComponents();
        initializeDisplay();
    }

    private void initializeDisplay() {
        jtfDisplayRowArr[0] = jtfRow1;
        jtfDisplayRowArr[1] = jtfRow2;
        jtfDisplayRowArr[2] = jtfRow3;
        jtfDisplayRowArr[3] = jtfRow4;
    }

    private void announceNumber(Customer s) {
        int sleepTime = 700;
        String numStr = String.valueOf(s.getSeqNo());
        try {
            for (int i = 0; i < numStr.length(); ++i) {
                Thread.sleep(sleepTime);
                int num = numStr.charAt(i) - '0';
                audioClips.get(num).play();
            }

            Thread.sleep(sleepTime);
            audioClips.get(audioClips.size() - 1).play();
            Thread.sleep(sleepTime);
            audioClips.get(s.getCounter()).play();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    private void updateDisplay(Customer s) {
        for (int i = jtfDisplayRowArr.length - 1; i > 0; i--) {
            jtfDisplayRowArr[i].setText(jtfDisplayRowArr[i -
1].getText());
        }
        jtfDisplayRowArr[0].setText(s.getSeqNo() + callString +
s.getCounter());
        currentNo++;
    }

    private class CounterListener implements ActionListener {

      @Override
      public void actionPerformed(ActionEvent e) {

            if (!q.isEmpty()) {
                int counterNo =
Integer.parseInt(e.getActionCommand().substring(counterNoIndex));
                Customer s = q.poll();
                s.setServeTime(new GregorianCalendar());
                s.setCounter(counterNo);
                serviceList.add(s);
                updateDisplay(s);
                announceNumber(s);
            }
        }

    }



//b) Wong Jung Hao

if(!q.isEmpty()){
                serviceList.add(q.remove());
                int serviceSeq = serviceList.size() - 1;
                serviceList.get(serviceSeq).setServeTime(new
GregorianCalendar());

                if(e.getSource() == jbtCounter1){
                    serviceList.get(serviceSeq).setCounter(1);
                }else if(e.getSource() == jbtCounter2){
                    serviceList.get(serviceSeq).setCounter(2);
                }else if(e.getSource() == jbtCounter3){
                    serviceList.get(serviceSeq).setCounter(3);
                }else if(e.getSource() == jbtCounter4){
                    serviceList.get(serviceSeq).setCounter(4);
                }else if(e.getSource() == jbtCounter5){
                    serviceList.get(serviceSeq).setCounter(5);
                }

                updateDisplay(serviceList.get(serviceSeq));
                announceNumber(serviceList.get(serviceSeq));



//c) Kow Yann Tang
private void jbtReportActionPerformed(java.awt.event.ActionEvent evt) {
        JTextArea jtaReport = new JTextArea(50, 200);
        String str = String.format("%70s\n", "Service Analysis Report");
        str += String.format("%-5s %-10s %-20s %-20s %-15s %-15s\n",
                "No", "Seq. No", "Arr. Time (ms)", "Serve Time(ms)",
"Counter", "Waiting Time(s)");

        int totlaWaitingTime = 0;

        for(int i = 0; i < serviceList.size(); i++){
            str += String.format("%-10s", (i + 1)) + serviceList.get(i);

            totlaWaitingTime += serviceList.get(i).getWaitingTime();
        }

        str += "\n" + "Total customers served : " + serviceList.size();
        str += "\n" + "Average waiting time     : " +
(totlaWaitingTime/serviceList.size() + " s");

        Font reportFont = new Font("Arial", Font.BOLD, 14);
        jtaReport.setText(str);
        jtaReport.setEditable(false);
        jtaReport.setFont(reportFont);
        JFrame reportFrame = new JFrame();
        reportFrame.add(jtaReport);
        reportFrame.setSize(600, 400);
        reportFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        reportFrame.setLocationRelativeTo(null);
        reportFrame.setVisible(true);
    }
```