Western Cape Government
Education

Learner's Name: _____

INFORMATION TECHNOLOGY

PAPER 1 – PRACTICAL

SEPTEMBER EXAMINATION – 2015

MARKING GRID

Q1	Q2	Q3	Total
[47]	[62]	[41]	[150]

Question 1: Flight Bookings – General Programming

1.1

Button [Display 1.1]

```
String activity = txfActivity1.getText();
activity = activity.toUpperCase();
String agent =
            (String) cmbAgent1.getSelectedItem();
String out = activity + ": " + agent;
lblOutput1.setText(out);
```

Read activity text from text field✓
Turn activity text to upper case✓
Get selected agent's name from combo box✓
Concatenate the 2 strings✓
Output concatenated string to label✓

(5)

1.2(a)

Button [Calculate Fees 1.2a]

```
String input = txfNumPeople2.getText();
int numPeople = Integer.parseInt(input);
fees = 100.00;
if (numPeople > 1) {
   fees += 80.0;       }
if (numPeople > 2) {
   fees += 65.0;       }
if (numPeople > 3) {
    fees = fees + (numPeople - 3) * 50.0;       }
if (chbGroup2.isSelected()) {
   if (fees > 500.0) {
      fees = 500.0;
   }   }
String output = String.format("R%.2f", fees);
      txfOut2A.setText(output);
```

Read input from text field✓
Convert input to integer✓
Set fees to 100 for 1 person (may include a if)✓

Test for 2nd person and add 80✓

Test for 3rd person and add 65✓

Test for more than 3 people
and add 50 for each person over 3✓✓
Test whether group is selected✓
Test whether fees is bigger than 500✓ (also accept numPeople > 9)
Set fees to 500✓
Format output to currency (R, 2 decimals)✓
Output to text field✓

(12)

1.2(b)

Button [I feel that luck is on my side 1.2b]

```
Random ran = new Random();
// OR int ranNum = (int) (Math.random() * 5);
int ranNum = ran.nextInt(5);
double discount = 0.0;
String out = "";
if (ranNum == 1) {

   discount = 3;
} else if (ranNum == 2) {

   discount = 1.5;       }
fees = fees * (100 - discount) / 100;

if (discount > 0.0) {
   out = "You have been lucky, you received a discount of " +
            discount + "%. \nYour fees are now " +
            String.format("R%.2f", fees);
} else {
   out = "Bad luck, try again next time. \nYour fees are still "
            + String.format("R%.2f", fees);    }
txaOut2C.setText(out);
btnActivateLucky2C.setEnabled(false);
```

Random number✓
Range 0 – 4✓ (could also be 1-5, 0-9 or 1-10)

Initialise discount and output variables✓

Test for any ONE of the 5 possible values✓
(or 2 out of 10)
Set discount to 3✓
Test for another ONE of the 5 values (or 2 out of 10)✓
Set discount to 1.5✓
Calculate fees by subtracting discount (percentage of fees)✓
Test if discount is bigger than 0 (could be included in the above)✓

Prepare output string: "lucky", mention discount, fees formatted to currency (R, 2 decimals)✓

OR output string: "no luck", fees formatted to currency (R, 2 decimals)✓
Output to text area✓
Disable button to avoid repeat✓

(13)

1.3	**Button [Calculate Points Needed 1.3]**
	String sKm = "";
	String input1 = txfDeparture4.getText().toUpperCase();
	String input2 = txfDestination4.getText().toUpperCase();
	if (input1.equalsIgnoreCase(input2)) {
	txaOut4.setText("ERROR: You cannot enter the same \nairport twice!")✓;
	} else {✓
	for (int i = 0; i < flightInfo.length; i++) {
	if (flightInfo[i][0].contains(input1) &&
	flightInfo[i][0].contains(input2)) {
	sKm = flightInfo[i][1]; }
	}
	int iKm = Integer.parseInt(sKm);
	double exactPoints = iKm * 321.17;
	int pointsNeeded =
	(int) Math.ceil(exactPoints);
	txaOut4.setText("Exact points = " + exactPoints + "\nPoints needed = " + pointsNeeded); }
		Get 2 airport strings✓
		(toUppercase just as a precaution, NO extra mark)
		Test whether the strings are the same✓
		Warning message✓ (in text area or JOptionPane)
		
		If strings are different (else)✓
		Loop through array✓
		Test for line [i][0]✓ that contains BOTH airports (& - could use nested ifs)✓
		Read the km from info[i][1]✓
		
		Parse km to integer✓
		Calculate exact points (*321.17)✓
		Round UP points to higher integer (Math.ceil or other)✓
		
		Output of BOTH points to text area✓
			(12)

1.4	**Button [Draw 1.4]**
	<u>This is the correct solution:</u>
	txaOut4.setText(" ENTRANCE COCKPIT\n");
	for (int i = 0; i < 33; i++) {
	if (i >= 13 **&&** i <= 15) {
	txaOut4.append(" WING ");
	} else {
	txaOut4.append(" ");
	for (char j = 'F'; j >= 'D'; j--) {
	txaOut4.append(j + " ");
	txaOut4.append(" " + String.format("%2d", **(i + 1)**) + " ");
	for (char j = 'C'; j >= 'A'; j--) {
	txaOut4.append(j + " ");
	if (i >= 13 **&&** i <= 15) {
	txaOut4.append(" WING");
	}
	txaOut4.append("\n");
	}
	txaOut4.append(" TAIL");
		|| changed to **&&**✓
		
		i changed to **(i + 1)**✓
		
		|| changed to **&&**✓
		
		Line of code added in correct position (just before end of big for-loop (<33)✓
		with **append \n**✓ to print lines below each other
		
		Give 5 marks for any sensible solution (not hard-coded) that produces the correct diagram.
			(5)

		Total Question 1 =	**[47]**

Question 2: Hotel Bookings – Object-oriented Programming

2.1 Class Bookings

2.1.1 **Declare the private attributes of the class:**

```
private String firstName;
private String surname;
private String idNum;
private char gender;
private boolean paid = false;
private double dailyRate = 895.0;
```

1 all names as per list and all private✓
2 3 string attributes✓ (IDNum MUST be a String, else candidate will lose this mark)
3 gender a char✓ (nothing else – but we mark their usage of this bearing in mind the data type chosen, no further penalty if used correctly)
4 paid a Boolean (or boolean), set to false✓ (nothing else – but we mark their usage of this bearing in mind the data type chosen, no further penalty if used correctly)
5 dailyRate a double, set to 895.0✓ (accept setting value in constructor)

(5)

2.1.2 **Parameterised Constructor:**

```
public Booking(String firstName,
               String surname,
               String idNum,
               char gender)    {
    this.firstName = firstName;
    this.surname = surname;
    this.idNum = idNum;
    this.gender = gender;
  }
```

1 mark, ONLY awarded if there are NO extra parameters AND if the 4 parameters are assigned correctly to the correct attributes✓

May contain default settings:

```
this.paid = false;
this.dailyRate = 895.0;
```

(1)

2.1.3 Remove the comment symbols from the assessor and mutator methods supplied.

Remove comments (//)✓

(1)

2.1.4

```
public String paymentMade() {
    if (paid) { // or (paid == true)
       return "Paid";
    }
    return "Not paid";    }
```

1 return type String✓
2 test if paid (or if paid == true)✓
3 returning appropriate string correctly (2 options)✓

(3)

2.1.5

```
public String toString() {
    return "Name: " + firstName + " " + surname
          + "\nID Num: " + idNum
          + "\nGender: " + gender
          + "\nPayment: " + paymentMade();    }
```

1 labels as per instruction✓
2 new lines \n✓
3 concatenation of names✓
4 idNum and gender attributes✓
5 using paymentMade() method✓

(5)

2.1.6

```
public double amountPayable(int numDays) {
    return numDays * dailyRate;    }
```

1 return type double int parameter✓
2 correct use of parameter and dailyRate attribute✓
3 multiplication, result returned✓

(3)

2.2	**Class Question2UI**		

2.2.1	**Button [Check customer 2.2.1]**
	File f = new File("BookingsData.txt");
	boolean found = false;
	String guestName =
	(String) cmbName.getSelectedItem();
	if (guestName.equals("Please select a name")) {
	JOptionPane.showMessageDialog(rootPane,
	"No name chosen");
	txaOutput.setText("");
	} else {
	if (!f.exists()) {
	JOptionPane.showMessageDialog(rootPane,
	"File not found");
	}
	try {
	Scanner fileReader = new Scanner(f);
	
	String hotelName = fileReader.nextLine();
	lblHotelName.setText(hotelName);
	
	while (fileReader.hasNext()) {
	String line = fileReader.nextLine();
	if (line.contains(guestName)) {
	
	found = true;
	Scanner lineReader =
	new Scanner(line).useDelimiter("#");
	String fName = lineReader.next();
	String sName = lineReader.next();
	String id = lineReader.next();
	char gender =
	lineReader.next().charAt(0);
	bookingItem = new Booking
	(fName, sName, id, gender);
	
	txaOutput.setText(bookingItem.toString());
	btn2.setEnabled(true);
	}
	}
	} catch (Exception e) {
	JOptionPane.showMessageDialog(rootPane,
	"Error reading file");
	}
	if (!found) {
	JOptionPane.showMessageDialog(rootPane,
	guestName + " not found in text file");
	btn2.setEnabled(false);
	} }
		File object declared✓
		Flag declared and default value✓
		Read selected name from combo box✓
		
		Test if name equal to fist line (i.e. no name selected)✓
		also accept test for selectedIndex == 1
		Error message✓
		Clear text area (no mark)
		If a name has been selected (can use else)✓
		Test if file exists✓
		Error message if file does not exist✓
		
		Try…catch for file reading✓
		File reader object (Scanner or BufferedReader)✓
		Read hotel name from first line of file (outside loop!)✓
		Set hotel name to label✓
		Conditional (while) loop through file✓
		Read a line from file✓
		Test if line contains guest's name✓
		If guest's name is found
		Set flag true✓
		Declare line reader object (or split) using #✓
		Any way to deal with the comma (another split, replace)✓✓
		
		Read 3 strings (fname, sname & id) ✓
		
		Read 4th string and use 1st char (gender)✓
		
		Use existing bookingItem variable to declare new object✓
		With 4 parameters✓
		Output to text area using toString method✓
		Enable next button (and disable if not found *)✓
		
		Test if flag has not changed (i.e. not found)✓
		Message to this effect✓ using guest's name✓
		
		Disable next button (with previous mark *)
			(28)

2.2.2	**Button [Check payment 2.2.2]**
	if (!bookingItem.isPaid()) { //OR ==false
	int days = Integer.parseInt(JOptionPane.showInputDialog("How many days is guest staying?"));
	double toPay = bookingItem.amountPayable(days);
	String input = JOptionPane.showInputDialog("Guest should pay " + String.format("R%.2f", toPay) + "\nHow much is guest paying?");
	double payment = Double.parseDouble(input);
	
	if (payment >= toPay) {
	bookingItem.setPaid(true);
	txaOutput.setText("Payment made in full.\n\n");
	
	} else {
	txaOutput.setText("Some money still outstanding.\n\n"); }
	txaOutput.append(bookingItem.toString()); }
		Test if guest has paid or not:
		(if✓ object with correct method✓)
		Obtain days (parsed to int✓) from dialog box✓
		
		Obtain amount✓ by calling object.correct method✓
		with days as parameter✓
		Use dialog box to display how much guest must pay✓
		
		Use dialog box to obtain amount guest has paid✓
		(May use one combined dialog)
		Parse amount to double✓
		Test if payment is = or more than amount owed✓
		If payment is more
		Use method of object to set paid to true✓
		Display that payment has been made in text area✓
		If payment is less (else)✓
		Display that payment is insufficient in text area✓
		Display status of object using toString✓
			(16)

		Total Question 2 =	**[62]**

3.	**Question 3: Booking Engine – Problem-solving Programming**		
	NB: In this question the use of methods/object(s) is marked in 3.0 (1 object containing 3 methods = 4 marks). Thereafter (3.1 – 3.3) you must see whether the "job" gets done (here presented in separate sections, "job description" in the section headings which lend themselves to methods, esp. as some things – like the display – gets repeated several times). Please search for relevant code and mark it whether it is in separate methods or not. Make sure that you mark each "job" only once (i.e. the loops for displaying only get marked once, thereafter the fact that the room allocation is displayed again gets only 1 mark).		
3.0	**The addition of an output component and the use of good programming techniques and modular design:** Has added JTextArea OR JTable to GUI, renamed Has coded using own methods and/or object(s)	**Add component for output:** JTextArea OR JTable to GUI✓, renamed✓ **Uses modular design:** At least 3 own methods (1 each up to 3)✓✓✓ Using parameter(s) with at least ONE of the three methods✓	(6)

3.1	**Button [Start new room allocation and display 3.1]**
	clearRoomAllocation();
	displayRoomAllocation();
	btnAllocate.setEnabled(true);
		Use of methods marked before (don't mark again, look that the code features described below are present, even if in one long program!)
		Enable next button✓ 1	

	private void clearRoomAllocation() {
	for (int i = 0; i < allocations.length; i++) {
	for (int j = 0; j < allocations[i].length; j++) {
	allocations[i][j] = "-"; } } }
		Clearing the arrays and setting to "-":
		For-loop through 2D-array (may use 16)✓
		Nested for-loop to 10✓
		Setting each data item to "-"✓ 3	

	private void displayRoomAllocation() {
	String out = "Day "
	for (int i = 0; i < 10; i++) {
	out += String.format("%-4s %-5d", "Room", (i + 1)); }
	out += "\n";
	for (int i = 0; i < allocations.length; i++) {
	out += String.format("%-3d", (i + 1));
	for (int j = 0; j < allocations[i].length; j++) {
	out += String.format("%-10s", allocations[i][j]);
	}
	out += "\n"; }
	txaOutput.setText(out); }
		Displaying room allocation:
		Build long string or use append – both with \n✓
		Loop for headings ("Room x")✓
		Displaying Room X in neat columns✓
		
		Loop to 16✓
		Display day number✓
		Loop to 10✓
		Display content of data items✓
		in neat columns✓
		Display in text area✓ 9
			(13)

3.2	**Button** [Allocate existing bookings 3.2]
	
	allocateBookings();
	displayRoomAllocation();
	btnNewGuest.setEnabled(true);
		Use of methods marked before (don't mark again, look that the code features described below are present, even if in one long program!)
		Display bookings after allocation✓
		Enabling next button✓ 2	

	private void allocateBookings() {
	for (int i = 0; i < septemberBookings.length; i++) {
	String string = septemberBookings[i];
	String[] part = string.split(",");
	int numDays = Integer.parseInt(part[0]);
	int startDate = Integer.parseInt(part[1]);
	String name = part[2];
	int roomNum = getFreeRoomNum(numDays, startDate);
	for (int j = startDate - 1; j < startDate - 1 + numDays; j++) {
	allocations[j][roomNum] = name;
	} } }
		Allocating bookings:
		Loop through given data array✓
		
		Separating info in each string✓
		
		Obtaining the 2 numbers and parsing✓
		
		Finding a free room for date and days required✓
		
		Looping through data structure✓
		Adding guest name into correct slots✓ 6	

	private int getFreeRoomNum(int numDays, int start) {
	int r = 0;
	int d = start - 1;
	int daysFree = 0;
	boolean found = false;
	while (!found) { // or similar
	while (allocations[d][r].equals("-") && d < 15
	&& r < 10) {
	daysFree++;
	d++; }
	if (daysFree >= numDays) {
	found = true;
	} else {
	r++; } }
	return r; }
		Finding a free room for date and days required:
		(May be incorporated into previous method)
		
		Looping through data structure✓
		Checking correct data items✓ for free space✓ within the correct limits✓
		Keeping track of number of days available✓
		
		Checking if days available match days required✓
		
		If not found enough available days (else)
		Incrementing room number to check next room✓
		7
			(15)

3.3	**Button [Allocate new booking 3.3]**
	
	String name = txfSurname.getText();
	int numDays =
	Integer.parseInt(txfNumNights.getText());
	int start = Integer.parseInt(txfStartingDate.getText());
	int room = getFreeRoomNum(numDays, start);
	
	for (int j = start – 1; j < start – 1 + numDays; j++) {
	allocations[j][room] = name;
	}
	displayRoomAllocation();
		Read new guest's requirements, allocate room and display:
		Read guest's surname (name) from textfield✓
		
		Read and parse 2 numbers from textfields✓
		
		Use some means (previously written method optional) to get a room that is open for the required number of days✓✓
		Looping through the 2D array✓
		Entering guest's name in all fields applicable (previously written method optional)✓
		Displaying room allocation once again✓
			(7)

		Total Question 3 =	**[41]**