



<b>MODULE NAME:</b>	<b>MODULE CODE:</b>
<b>PROGRAMMING 1B</b>	<b>PROG6112</b>

<b>ASSESSMENT TYPE:</b>	<b>TAKE-HOME EXAM (PAPER ONLY)</b>
<b>TOTAL MARK ALLOCATION:</b>	<b>120 MARKS</b>
<b>TOTAL TIME:</b>	<b>This assessment should take you 2 Hours to complete, however, you have 21 Hours (midnight to 9PM on the same day) to submit. This additional time has been allocated to allow for the download, completion, and upload of your submission.</b>

*By submitting this assessment, you acknowledge that you have read and understood all the rules as per the terms in the registration contract, in particular the assignment and assessment rules in The IIE Assessment Strategy and Policy (IIE009), the intellectual integrity and plagiarism rules in the Intellectual Integrity Policy (IIE023), as well as any rules and regulations published in the student portal.*

#### **INSTRUCTIONS:**

1. Please **adhere to all instructions**. These instructions are different from what is normally present, so take time to go through these carefully.
2. **Independent work is required**. Students are not allowed to work together on this assessment. Any contraventions of this will be handled as per disciplinary procedures in The IIE policy.
3. **No material may be copied from original sources, even if referenced correctly, unless it is a direct quote indicated with quotation marks.**
4. All work must be adequately and correctly referenced.
5. You should paraphrase (use your own words) the concepts that you are referencing, rather than quoting directly.
6. Marks will be awarded for the quality of your paraphrasing.
7. This is an open-book assessment.
8. Assessments must be typed unless otherwise specified.
9. **Ensure that you save a copy of your responses.**
  - 9.1. Complete your responses in a Word document.
  - 9.2. The document name must be your **Name.student number.Module Code**.
  - 9.3. Once you have completed the assessment, upload your document under the **submission link** in the correct module in Learn.

#### **Additional instructions:**

- Calculators are allowed
- Instructions for assessments including practical computer work:
  - Use of good programming practice and comments in code is compulsory.
  - Save your application in the location indicated by the administrator (e.g., the Z:\ drive or your local drive).
  - Create a folder as follows: use the module code and your own student number and create a folder with a folder name as per the format shown here:

- ***StudentNumber\_ModuleCode\_Exam.*** Save all files (including any source code files, template files, design files, image files, text files, database files, etc.) within this folder.
- *E.g., if your student number is 12345 and you are writing an examination for the module PROG121, create a folder named **12345\_Prog121\_Exam** and use this throughout the session to save all of your files.*
- ***Important:*** Upon completion of your assessment, you must save and close all your open files and double click the ExamLog application on your desktop. You must follow the instructions carefully to ensure that the information about the files that you have submitted for this assessment has been logged on the network. Specify the location of your source code on your question paper.

**GENERAL REQUIREMENTS****(Marks: 10)**

Writing maintainable code in industry is vital, therefore, you are generally required to:

- Follow good programming practices:
  - Variable types correct;
  - Variable scope correct; and
  - Class and method naming standards correct.
- Insert comments to show logic and design for the support of code maintainability.
- Code efficiently. Redundant code must be avoided.
- Spend 15 minutes ensuring that your programmes meet the general criteria below.

Requirement	Maximum Mark	Examiner's Mark	Moderator's Mark
<b>Good Programming Practice</b> <ul style="list-style-type: none"> <li>• Not followed at all = 0</li> <li>• Average – Minor changes required = 1 – 2</li> <li>• Excellent – No changes required = 3</li> </ul>	(3)		
<b>Code Efficiency</b> <ul style="list-style-type: none"> <li>• Poor – Much code is duplicated = 0</li> <li>• Average – Some redundant code = 1</li> <li>• Excellent – Code very maintainable = 2</li> </ul>	(2)		
<b>Comment Statements</b> <ul style="list-style-type: none"> <li>• Poor – No comments = 0</li> <li>• Average – Some comments = 1 – 2</li> <li>• Excellent – All necessary comments given to show logic = 3</li> </ul>	(3)		
<b>Program(s) compile and execute</b> <ul style="list-style-type: none"> <li>• No – Many changes required = 0</li> <li>• Average – Minor changes required = 1</li> <li>• Yes – No changes required = 2</li> </ul>	(2)		
<b>TOTAL MARKS</b>	<b>10</b>		

**Question 1****(Marks: 50)**

Develop a Java application and use a two-dimensional array that will store three property sales for two estate agents for January, February, and March. In your solution, include the total property sales and the 2% commission earned by each estate agent. You are also required to display the top-selling estate agent.

Your program must:

- Q.1.1** Contain a two-dimensional array to contain three property sales for January, February, and March for two different estate agents.

ESTATE AGENTS		JAN	FEB	MAR
Joe Bloggs		R 800 000	R 1 500 000	R 2 000 000
Jane Doe		R 700 000	R 1 200 000	R 1 600 000

- Q.1.2** Calculate and print the total property sales for each estate agent.

- Q.1.3** Calculate and print the total 2% commission earned by each estate agent.

**Sample screenshot**

```

ESTATE AGENTS SALES REPORT

                JAN                FEB                MAR
-----
Joe Bloggs      R 800000.0        R 1500000.0        R 2000000.0
Jane Doe        R 700000.0        R 1200000.0        R 1600000.0

Total property sales for Joe Bloggs = R 4,300,000
Total property sales for Jane Doe = R 3,500,000

Sales Commission for Joe Bloggs = R 86,000
Sales Commission for Jane Doe = R 70,000

Top performing estate agent: Joe Bloggs
  
```

**Q.1.4** Make use of a class named EstateAgent that contains methods to calculate estate agent sales, estate agent commission, and the top-performing estate agent. The EstateAgent class must implement an IEstateAgent interface that contains the following:

```
public interface IEstateAgent {
    double EstateAgentSales(double[] propertySales);
    double EstateAgentCommission(double propertySales);
    int TopEstateAgent(double[] totalSales);
}
```

**Q.1.5** You are required to write unit tests for the application. Create a test package within the application you created, which will contain the necessary unit tests.

You are required to write the following unit tests:

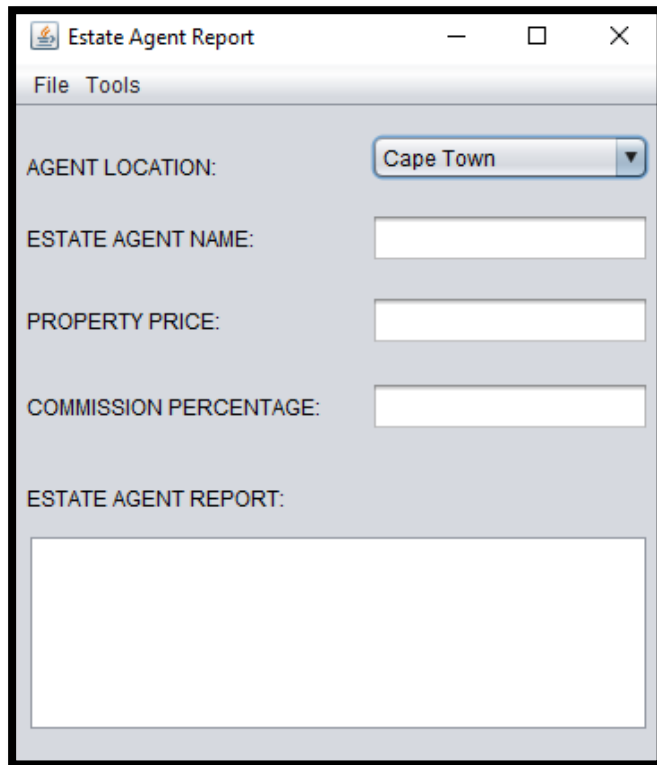
Test Name	Test Purpose
CalculateTotalSales_ReturnsTotalSales()	To supply the property sales to the estate agent sales method. The test will determine that the correct total sales value is returned from the estate agent sales method.
CalculateTotalCommission_ReturnsCommission()	To supply a total property sales amount to the estate agent commission method. The test will determine that the correct estate agent commission has been calculated.
TopAgent_ReturnsTopPosition()	To supply total sales for each estate agent to the top estate agent method. The test will determine the top selling total property sales for the estate agent.

Question 1 Mark Allocation	Levels of Achievement				Feedback
	Excellent	Good	Developing	Poor	
	Score Ranges Per Level (½ marks possible)				
2D Array created to store the property sales prices	10	5-9	1-4	0	
	2D Array created to store the property sales	Minor changes required	Major changes required	Not provided	
Calculate the total property sales for each estate agent	5	3-4	1-2	0	
	Calculate the total property sales for each estate agent	Minor changes required	Major changes required	Not provided	
Calculate the commission for each estate agent	5	3-4	1-2	0	
	Calculate the commission for each estate agent	Minor changes required	Major changes required	Not provided	
Correct printing of the monthly property sales, total property sales and commission for each estate agent	5	3-4	1-2	0	
	Correct printing of required report items	Minor changes required	Major changes required	Not provided	
IEstateAgent interface class created	5	3-4	1-2	0	
	IEstateAgent interface class created.	Minor changes required	Major changes required	Not provided	

<b>EstateAgentClass created that implements the IEstateAgent class with EstateAgentSales, EstateAgentCommission and TopEstateAgent methods</b>	<b>5</b>	<b>3-4</b>	<b>1-2</b>	<b>0</b>	
	EstateAgentClass created that implements the IEstateAgent class with required methods	Minor changes required	Major changes required	Not provided	
<b>CalculateTotalSales ReturnsTotalSales unit test created that tests the required functionality</b>	<b>5</b>	<b>3-4</b>	<b>1-2</b>	<b>0</b>	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	
<b>CalculateTotalCommission ReturnsCommission unit test created that tests the required functionality</b>	<b>5</b>	<b>3-4</b>	<b>1-2</b>	<b>0</b>	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	
<b>TopAgent ReturnsTopPosition unit test created that tests the required functionality</b>	<b>5</b>	<b>3-4</b>	<b>1-2</b>	<b>0</b>	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	

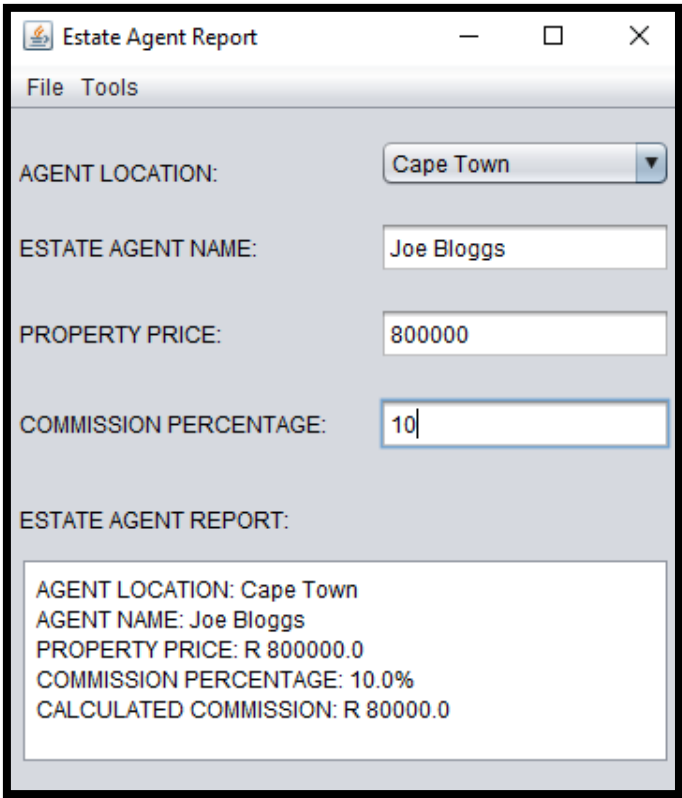
**Question 2****(Marks: 60)**

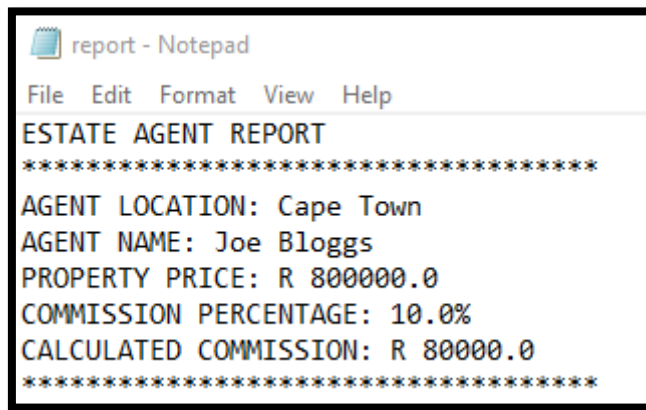
Create a Java GUI application that will capture the commission earned by an estate agent for the sale of a property. The application must capture all the required input and display the estate agent commission report.

**Sample Screenshot**

- |              |  |
|--------------|--|
| <b>Q.2.1</b> | On the form, create one combo box for the estate agent location. The estate agent locations include Cape Town, Durban, and Pretoria. You are also required to create three text fields to capture the estate agent name, property price, and commission percentage. Also, create a text area to display the estate agent report. |
| <b>Q.2.2</b> | <p>A menu system is required for the user to interact with. The following menu items are required:</p> <p><b>File</b></p> <ul style="list-style-type: none"> <li>• Exit</li> </ul> <p><b>Tools</b></p> <ul style="list-style-type: none"> <li>• Process Report</li> </ul>  |



	<ul style="list-style-type: none"> <li>• Clear</li> <li>• Save Report</li> </ul>
<b>Q.2.3</b>	When the Exit menu item is clicked, close the application.
<b>Q.2.4</b>	<p>When the Process Report menu item is clicked, capture the selected agent location, estate agent name, property price, and commission percentage, and display the estate agent report. The commission is calculated by multiplying the property price by the commission percentage.</p> <p><b>Sample Screenshot</b></p> 
<b>Q.2.5</b>	<p>When the Save Report menu item is clicked, save the estate agent report to a text file named report.txt.</p> <p><b>Sample Screenshot</b></p>



**Q.2.6** When clicking the Clear Menu item, set the text fields and text area to the default state.

**Q.2.7** Make use of a class named EstateAgent that contains methods to calculate the estate agent commission and validate the data. Only valid data should be captured in the application; the validation rules are:

VALIDATION	RULE
Agent Location	It cannot be empty.
Agent Name	It cannot be empty.
Property Price	Cannot be less than or equal to zero (0)
Commission Percentage	Cannot be less than or equal to zero (0)

The EstateAgent class must implement an IEstateAgent interface that contains the following:

```
public interface IEstateAgent {
    double CalculateCommission(String propertyPrice, String agentCommission);
    boolean ValidateData(Data dataToValidate);
}
```

**Q.2.8**

You are required to write unit tests for the application. Create a test package within the application you created, which will contain the necessary unit tests.

You are required to write the following unit tests:

Test Name	Test Purpose
CalculateCommission_CalculatedSuccessfully()	To supply the property price and agent commission percentage to the Calculate Commission method. The test will determine that the correct commission value is returned.
CalculateCommission_CalculatedUnSuccessfully()	To supply the property price and agent commission percentage to the Calculate Commission method. The test will determine if there is a calculation error.
Validation Test	You must only write one unit test to prove that the validation performs as expected. Examples can include validating location is valid, validating name is valid or validating property price is valid.

Question 2 Mark Allocation	Levels of Achievement				Feedback
	Excellent	Good	Developing	Poor	
	Score Ranges Per Level (½ marks possible)				
GUI form created with all objects	5	2-4	1	0	
	GUI form created with all objects	Minor changes are required.	Major changes are required.	Not provided.	
Populating the combo box with the correct values	5	2-4	1	0	
	Populating the combo box with the correct values	Minor changes are required.	Major changes are required.	Not provided.	
Creating the menu system with required menu items	5	3-4	1-2	0	
	Creating the menu system with required menu items	Minor changes are required.	Major changes are required.	Not provided.	
Report produced as per sample	5	2-4	1	0	
	Report produced as per sample	Minor changes required	Major changes are required.	Not provided.	
File saved correctly with the report contents	5	2-4	1	0	
	File saved correctly	Minor changes are required.	Major changes are required.	No Output	

<b>IEstateAgent class created</b>	<b>5</b>	<b>2-4</b>	<b>1</b>	<b>0</b>	
	IEstateAgent class created	Minor changes required.	Major changes required.	No Output	
<b>Estate agent class created that implements IEstateAgent with required methods</b>	<b>5</b>	<b>2-4</b>	<b>1</b>	<b>0</b>	
	Estate agent class created with required methods	Minor changes are required.	Major changes are required.	No Output	
<b>Estate agent commission calculated correctly</b>	<b>5</b>	<b>2-4</b>	<b>1</b>	<b>0</b>	
	Estate agent commission calculated correctly.	Minor changes are required.	Major changes are required.	No Output	
<b>Validation performed correctly</b>	<b>5</b>	<b>2-4</b>	<b>1</b>	<b>0</b>	
	Validation performed correctly	Minor changes are required.	Major changes are required.	No Output	
<b>CalculateCommission CalculatedSuccessfully unit test created that tests the required functionality</b>	<b>5</b>	<b>3-4</b>	<b>1-2</b>	<b>0</b>	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	

<b>CalculateCommission CalculatedUnsuccessfully unit test created that tests the required functionality</b>	<b>5</b>	<b>3-4</b>	<b>1-2</b>	<b>0</b>	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	
<b>Any validation test to prove that the data captured is valid</b>	<b>5</b>	<b>2-4</b>	<b>1</b>	<b>0</b>	
	Validation unit test created that tests the required functionality	Minor changes required.	Major changes required.	No Output	

**END OF PAPER**