***SECTION A***

***GitHub Link :*** <https://github.com/abdulahicarus-cell/Question1-Accommodation-Area.git>

***GitHub Link***: <https://github.com/abdulahicarus-cell/EliteSACCO.git>

***GitHub Link :*** <https://github.com/abdulahicarus-cell/MirrorFocalLength.git>

(a) **Purpose of Access Modifiers in OOP Languages**

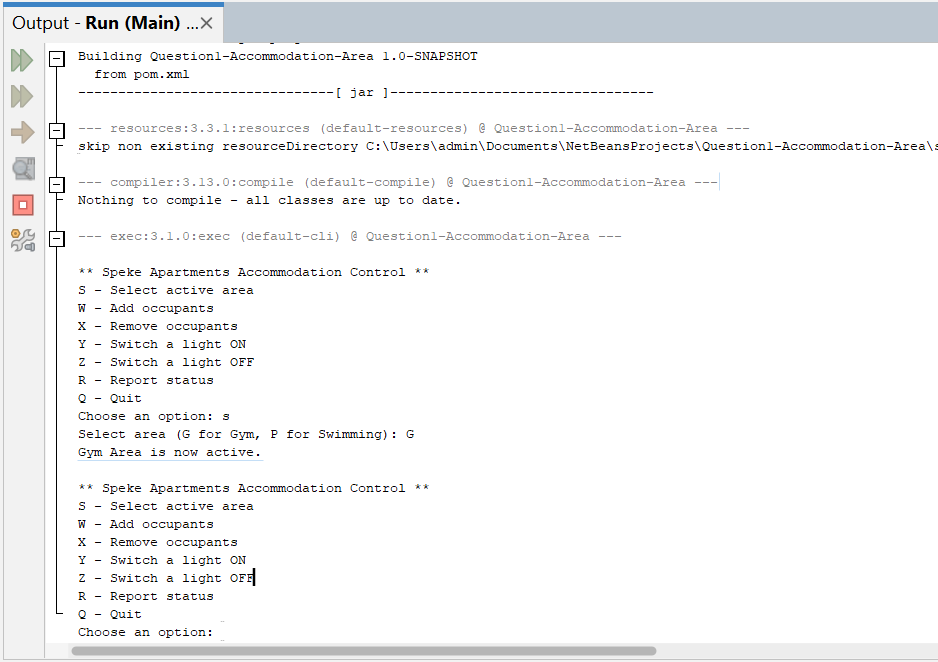
Access modifiers are object-oriented programming terms that are employed to specify the visibility (accessibility) of classes, methods, constructors, and other members. Access modifiers are crucial to Encapsulation since they help a class control what information is exposed to other classes, hiding internal implementation details and protecting the integrity of the data.

.

(B) **Access Modifiers Table**

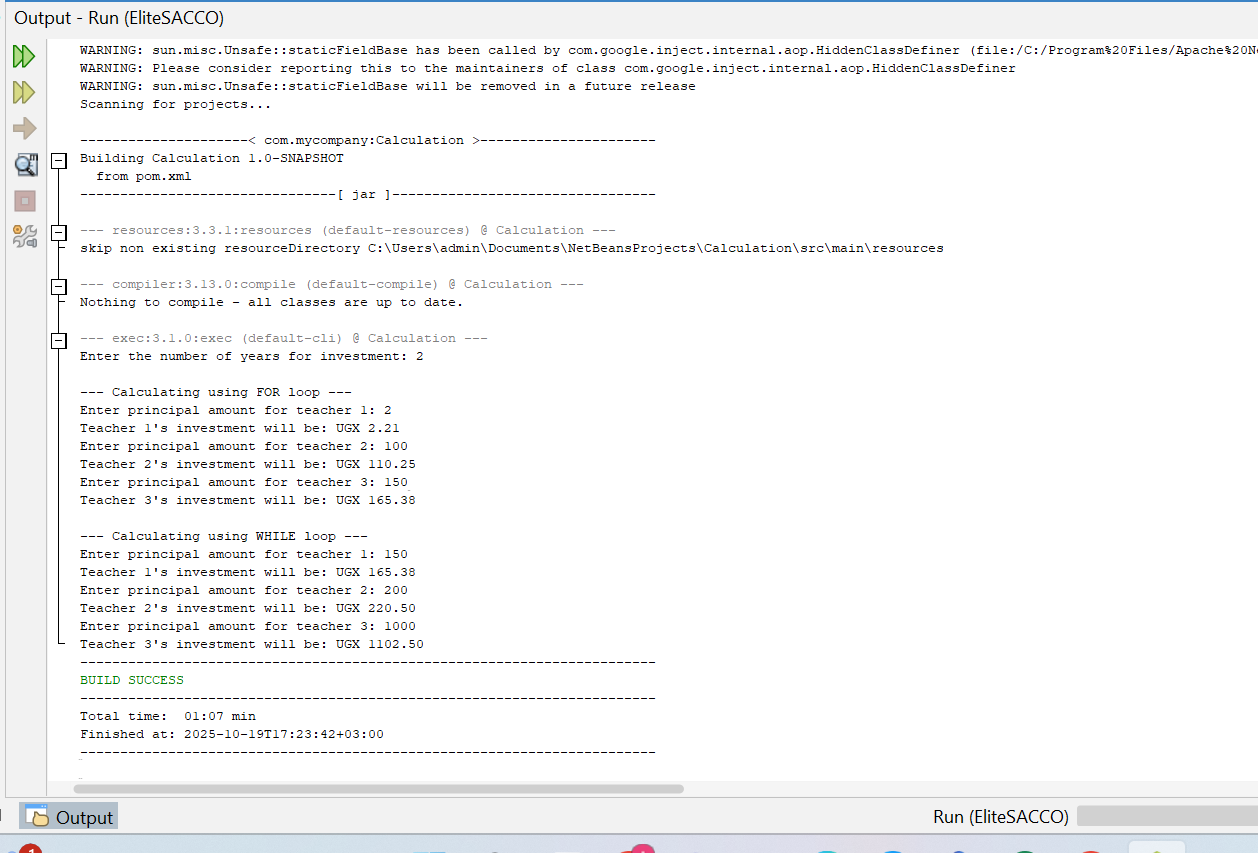
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | Defining Class | Class in Same Package | Subclass in Different Package | Non-Subclass in Different Package |
| public | yes | yes | yes | yes |
| protected | yes | yes | yes | no |
| default (no modifier) | yes | yes | no | no |
| private | yes | no | no |  |

(C) **Java Application for Speke Apartments**

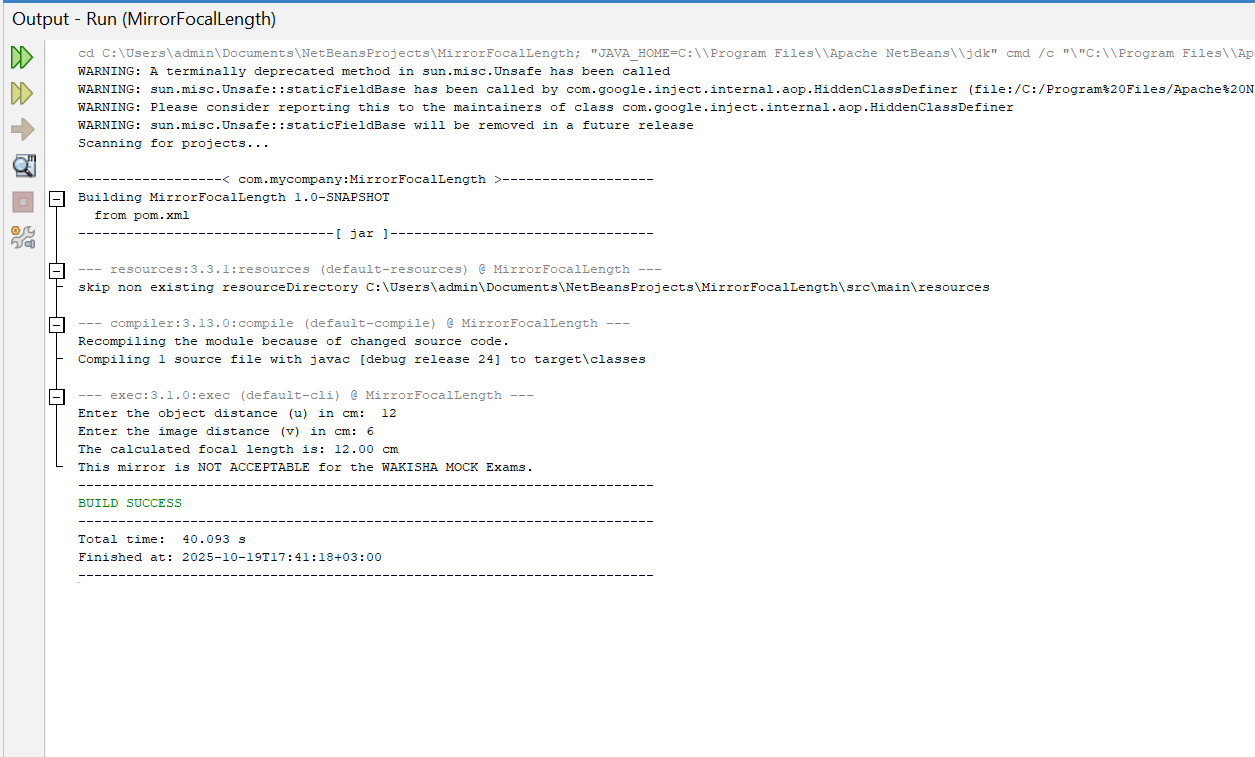


**QUESTION 2 (A)**

**Investment Calculation using Loops**



B) **Concave Mirror Focal Length Verification**



**C)** **Code Analysis and Output**

The provided code is a simple loop that prints numbers from 1 to 4

• int i = 1; initializes the loop counter.

• The condition i < 5 means the loop will run as long as i is less than 5 (i.e., 1, 2, 3, 4).

• System.out.print(i + " "); prints the value of i followed by a space.

• i++; increments i by 1 after each iteration.

Output:

1 2 3 4

***SECTION B***

***GitHub Link :*** [***https://github.com/abdulahicarus-cell/GradeCalculator.git***](https://github.com/abdulahicarus-cell/GradeCalculator.git)

***GitHub Link:*** [***https://github.com/abdulahicarus-cell/Question1-Accommodation.git***](https://github.com/abdulahicarus-cell/Question1-Accommodation.git)

**3. (a) Distinctions**

**(i) High-level vs Machine-level Programming Languages**

|  |  |  |
| --- | --- | --- |
| Aspect | High-Level Programming Language | Machine-Level Programming Language |
| Definition | A language that is closer to human language and abstracts hardware details. | A low-level language consisting of binary (0s and 1s) understood directly by a computer. |
| Ease of Use | Easy to write, read, and understand. | Difficult to read and write. |
| Portability | Platform-independent (can run on different machines with JVM). | Platform-dependent (specific to hardware). |
| Example (Java) | System.out.println("Hello, World!"); | Machine language would be binary instructions like 10111000 00000001, which isn't readable or used directly in Java. |

**(ii) Difference between a for loop and a while loop**

|  |  |  |
| --- | --- | --- |
| Aspect | for loop | while loop |
| Usage | Used when the number of iterations is known. | Used when the number of iterations is not known in advance. |
| Structure | Initialization, condition, and increment in one line. | Initialization is separate; only condition is in loop declaration. |
| Readability | More compact for counting loops. | More flexible for conditional loops. |
| Example | See below. | See below. |

**(b) Importance of Comments in a Programming Workplace (Java or any programming language)**

Comments are non-executable lines in a program used to describe code. They are ignored by the compiler/interpreter, but read by humans. Here's why they're important in a workplace**:**

* **Improve Code Readability**

Comments explain what the code does, making it easier for others (or yourself later) to understand.

* **Help in Team Collaboration**

In workplace projects, multiple developers work on the same codebase.

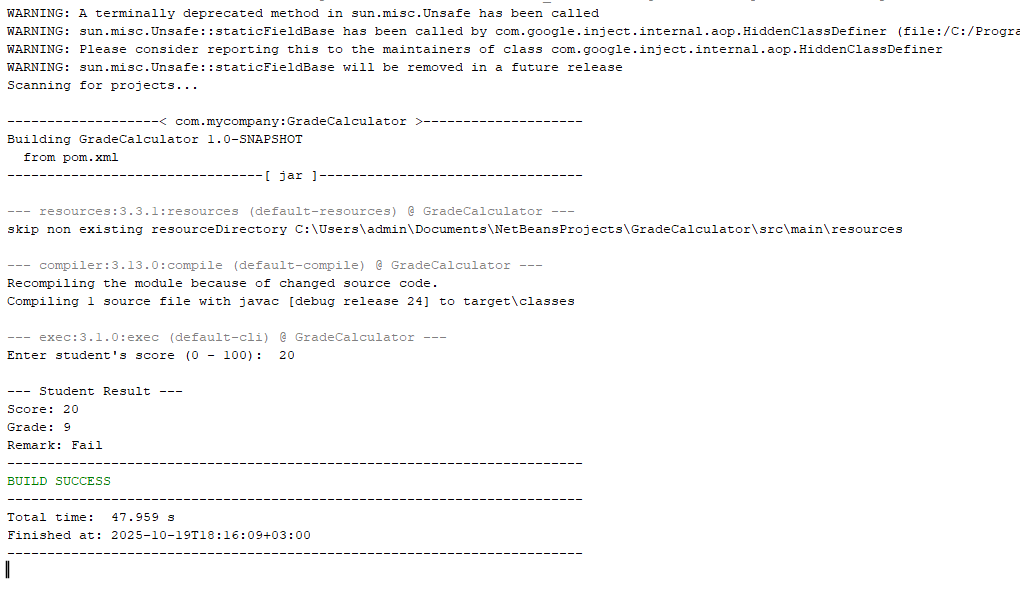
Comments help team members quickly understand each other's logic**.**

* **Aid in Debugging and Maintenance**

Developers can use comments to temporarily disable code for testing or debugging.

Comments indicate why certain logic was written in a specific way.

**(c) & (d) A-Level Grading Program**

****

**5. (a) Object-Oriented Concepts**

**(i) Polymorphism**

Polymorphism means "many forms". It allows us to perform a single action in different ways. In Java, this is often achieved by a parent class reference referring to a child class object.

**Example:**

Imagine a superclass Animal with a method sound(). Subclasses like Dog and Cat override the sound() method

Animal myAnimal = new Animal();

Animal myDog = new Dog();

Animal myCat = new Cat();

myAnimal.sound();

myDog.sound();

myCat.sound();

The same sound() method behaves differently depending on the actual object being referred to.

**(ii) Interface**

An interface is a completely "abstract class" used to group related methods with empty bodies. It defines a contract that implementing classes must follow.

**Example:**

An Vehicle interface can define methods like startEngine() and stopEngine(). Different vehicles like Car and Motorcycle can then implement this interface and provide their own specific implementations for these methods

interface Vehicle {

void startEngine();

void stopEngine();

}

class Car implements Vehicle {

public void startEngine() {

System.out.println("Car engine started with a key.");

}

public void stopEngine() {

System.out.println("Car engine stopped.");

}

}

This ensures that all vehicles have start and stop functionality, but each can implement it differently.

**(b) (c) (d)**

