***ANSWER***

***SECTION A***

***GitHub Link:*** https://github.com/Dibaa112/Question1-Accommodation-Area.git

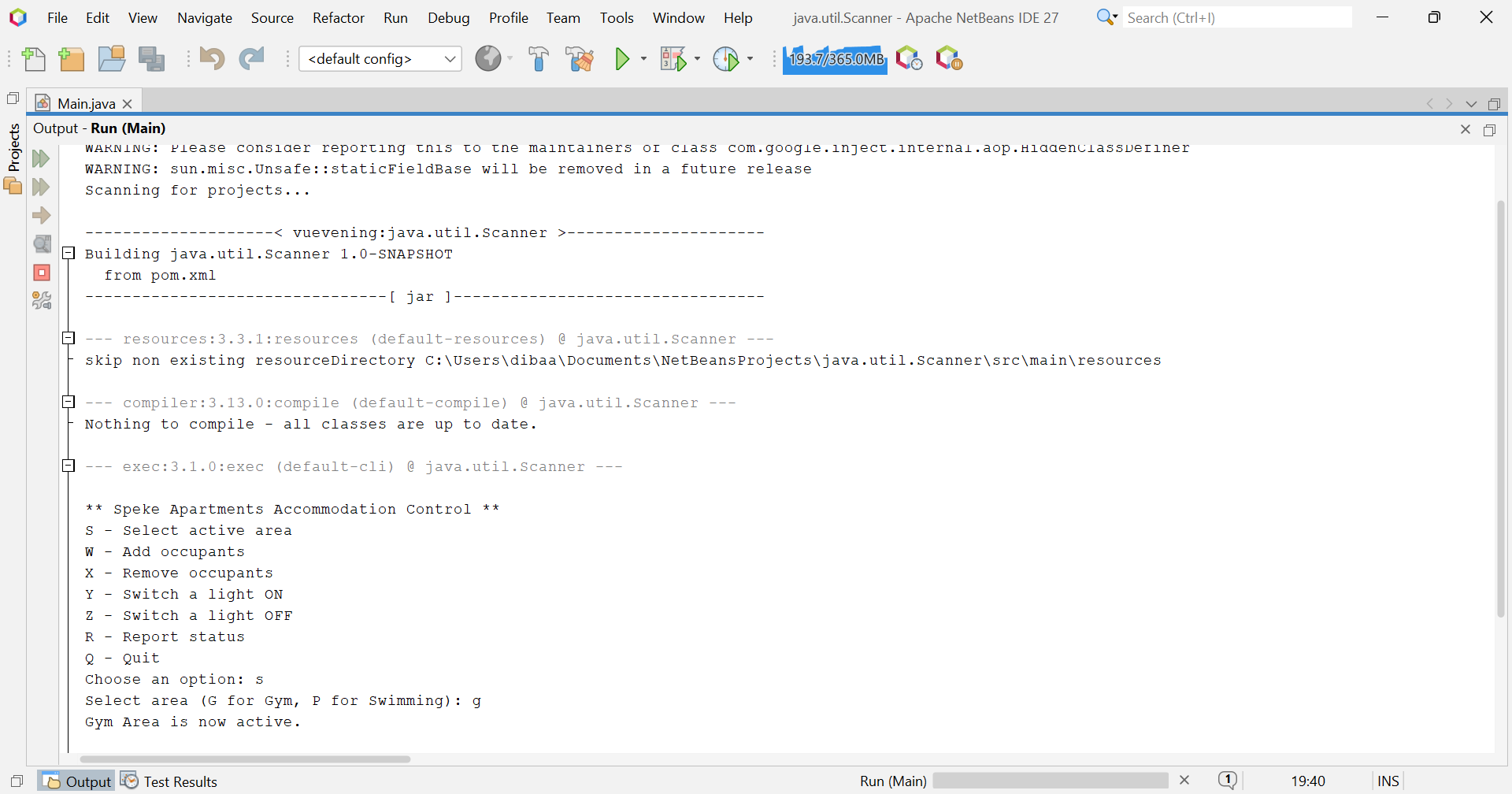
**1 . (a) Purpose of access modifiers in OOP languages**

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.

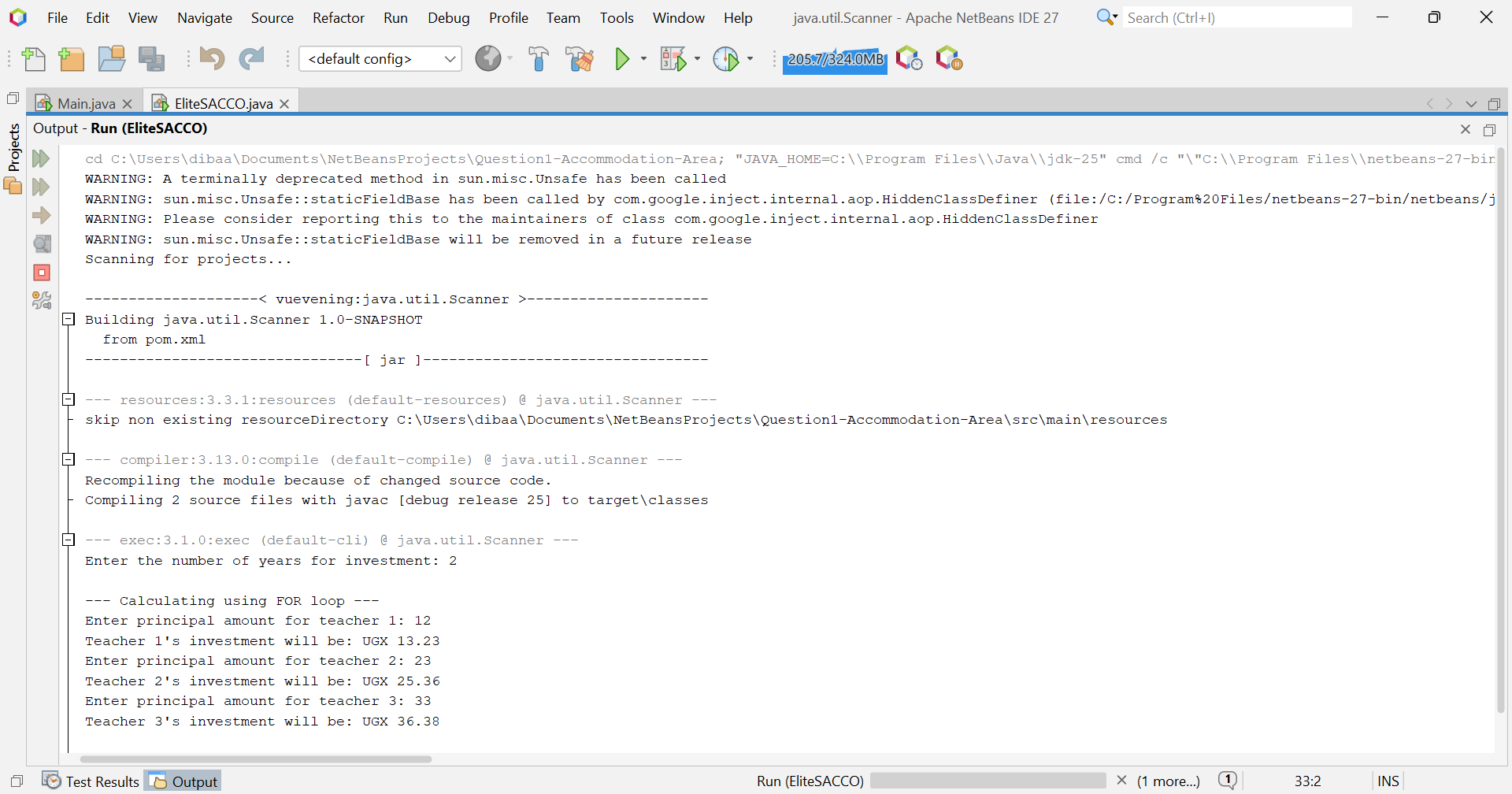
**(b) Access Modifiers Table**

| Access Modifiers | 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 | No |

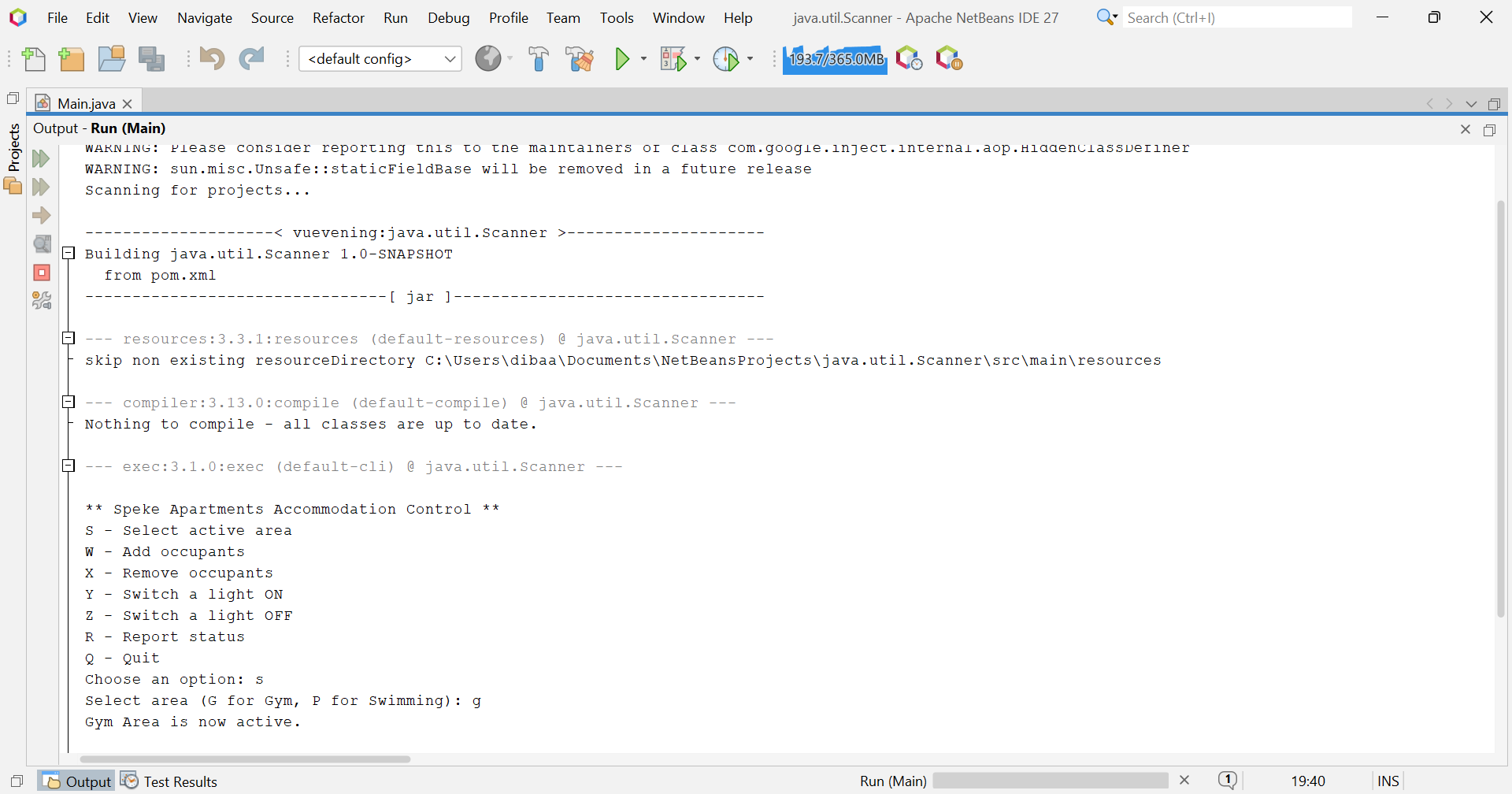
**(c) Java Application for Speke Apartments**

****

**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***

**3. (a) Distinctions**

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **High-Level Language (e.g., Java, Python)** | **Machine-Level Language (Binary/Machine Code)** |
| **Abstraction** | **Close to human language (English, math).** | **Consists of only 0s and 1s. Directly understood by the CPU.** |
| **Portability** | **Portable (platform-independent). Code written once can run on any machine with an interpreter/compiler.** | **Not portable. Tied to a specific computer architecture.** |
| **Readability** | **Easy to read, write, and maintain.** | **Very difficult for humans to read or write.** |
| **Execution** | **Requires a compiler or interpreter to translate into machine code.** | **Executed directly by the CPU.** |

**ii) For Loop vs. While Loop**

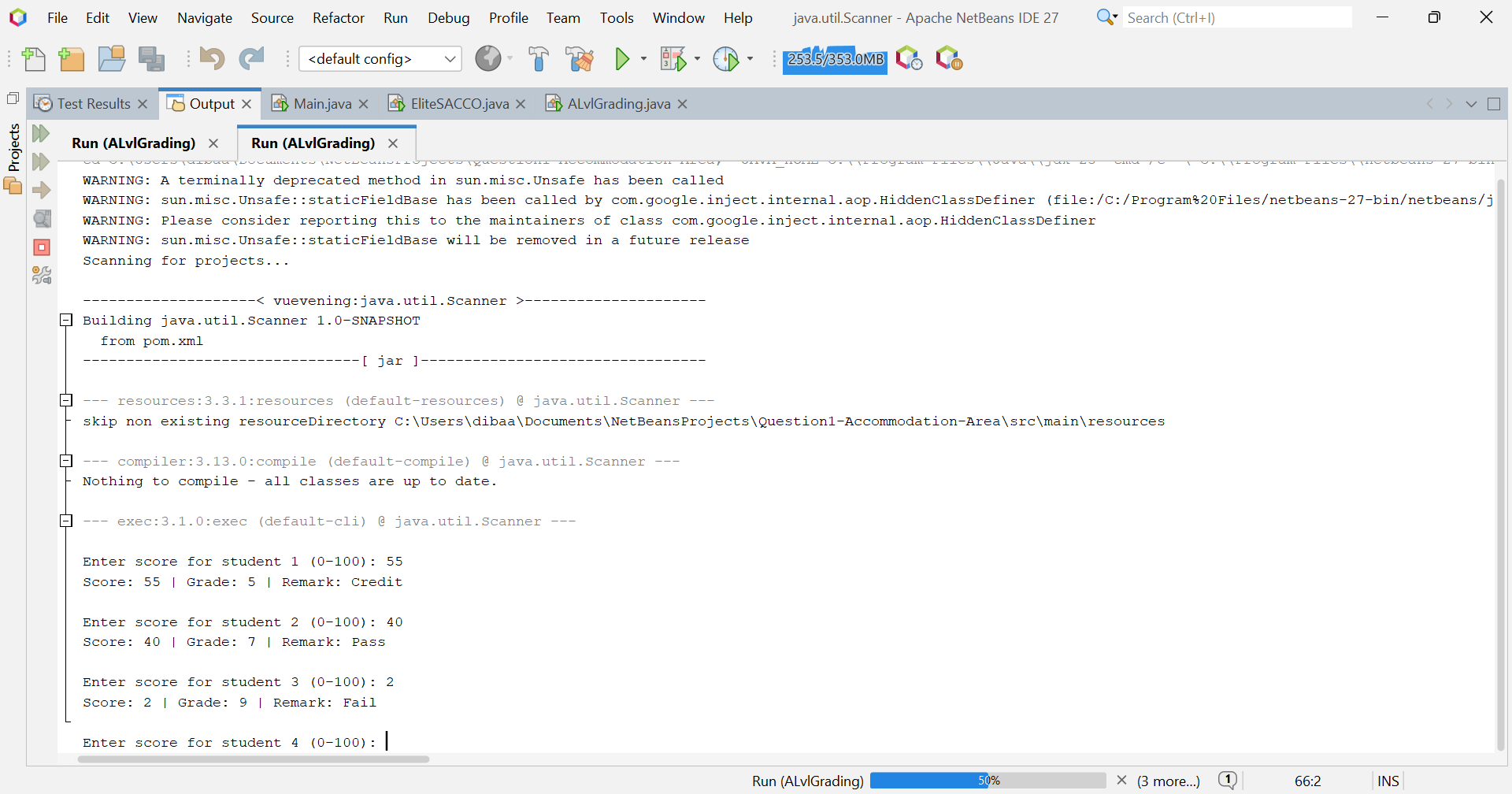
|  |  |  |
| --- | --- | --- |
| **Feature** | **For Loop** | **While Loop** |
| **Usage** | **Used when the number of iterations is known beforehand.** | **Used when the number of iterations is not known, and depends on a condition.** |
| **Initialization** | **The loop variable is initialized within the loop syntax itself (for(int i=0; ...)).** | **The loop variable is initialized before the loop statement.** |
| **Structure** | **for(init; condition; update) { ... }. All three parts are in one line.** | **while(condition) { ... }. Initialization and update are done separately.** |

**(b) Importance of Comments**

Comments are crucial in the workplace because:

1. **Documentation:** They explain the "why" behind complex code, making it easier for others (or yourself in the future) to understand the logic and purpose.
2. **Maintainability:** They significantly reduce the time and cost of maintaining and debugging code.
3. **Collaboration:** In team environments, comments are essential for communication between developers, ensuring everyone is on the same page.

**(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)**

