Topic: Applicable modifiers in classes declarations.
OOP concepts involved: Classes, Class Attributes, Class Access Modifiers.
Programming generic concepts involved: Variable declaration, Value assignment to variable, Primitive data types, Data capture.

➢ Theoric introduction

CLASS MODIFIERS

In Java, **modifiers** are keywords that can be utilized when defining a class which will affect its runtime behaviour. There are multiple modifiers that can be used when declaring a class, some of which are:

- Access Modifiers
- Final
- Abstract
- Static

Not all modifiers are allowed on all classes, for example an interface cannot be final and an enum cannot be abstract. A class can be declared using one or more Modifiers using this basic form:

```
[Access Modifier] [Modifier(s)] class identifier{}
```

ACCESS MODIFIERS

Access Modifiers, like the name suggests, are modifiers used to restrict the scope of a class, that is to say, who can access and modify the values in it. This type of modifiers aren't exclusive to classes: constructors, variables, methods and data members can use this modifier too.

There are four types of Access Modifiers, each of whom, allow access to a certain level.

- Default : When no access modifier is specified for a class, constructor, variable, method or data member, it is said to be having the **default** access modifier. The classes, variables, methods or data members with this access modifier, are only accessible within their own package.
- Private: Any method, variable or data member declared as **private** will only be accessible within the same class. Any other class in the same package will not be able to access these members. Classes and interfaces cannot be declared **private**.
- Protected: The methods or data members declared as **protected** are accessible within the same package or subclasses in different packages.
- Public: This access modifier has the widest scope among all of them. Classes, methods, variables or data members declared as **public** are accessible from everywhere in the program. There is no restriction in who can access these members.

Visibility	Public	Protected	*Default*	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, *through inheritance*	No	No
From any non-subclass class outside the package	Yes	No	No	No

FINAL MODIFIER

The final modifier can be used on a Class, Method or Variable to prevent its value from being changed, depending on what it is used, it will work in a different way.

- Final **variables**: These variables can only be initialized once, and cannot be reassigned to refer to a different object, however, this does not prevent the data within the object from

being changed. In other words, the state of the object can be changed, but not the reference.

- ❏ Final **methods**: Final methods cannot be overridden by any subclasses, this makes it so that any class that inherits from a class with a final method, cannot change the contents of it.
- ❏ Final **classes**: These kind of classes are used when want to prevent a class from being subclassed. If a class is declared as final, no other class can inherit anything from it.

EXTENDS

When creating a class, we can use the keyword **extends**, to inherit all the attributes and methods defined in another class, this is a process called Inheritance. Below is an example of the basic form of a class extending another.

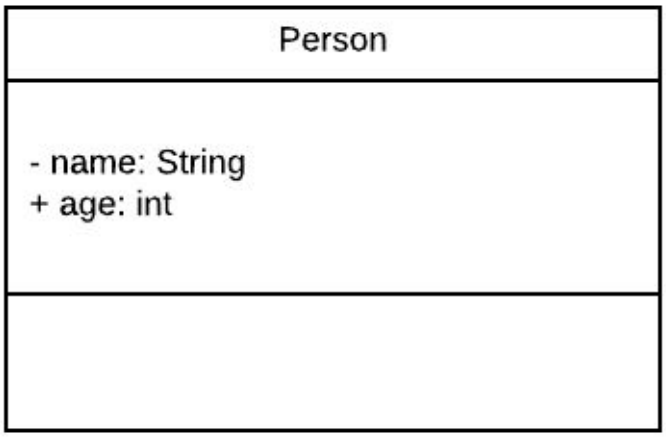
```
[Access Modifier] [Modifier(s)] class identifier extends identifier{}
```

What this does, is it allows the class extending (also known as subclass or child class) to act as is it had attributes and methods from the class being extended (also known as the parent class). The subclass also has the option to override methods in the parent class, changing the contents in them, unless the final modifier is used.

➢ Statement

Make use of access modifiers to create a simple relationship between two classes. Use public and private modifier to experiment with the difference in access between those two.

➢ Class design (UML)



➢ Program Code

Person.java

```
public class Person {
	// Defining object attributes (Its state)
	private String name; // private access modifier
	int age; // default access modifier

	// Constructor methods

	// Empty Class Constructor
	public Person(){}

	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}

	// Getters and Setters Methods
	// Since we can't access the attribute directly, we must create public methods from inside the object that return the value in it to outside objects or set the value using an outside input.
	public void setName(String name){
		this.name = name;
	}
	public String getName(){
		return this.name;
	}
}
```

Main.java

```
import java.util.Scanner;
import java.lang.Integer;

public class Main {

	public static void main(String[] args) {
		// Attributes of the class
		String name;
		int age;
		Person jeff; // Declaring an object based on the Person class

		// Declaring the object to allow data reading predefining object
		// System.in for standard data input from the keyboard
		Scanner sc = new Scanner(System.in);

		// Capturing data through the console and assigning it inside Person object
attributes
		System.out.println("What's the Person's name?");
		name = sc.nextLine();
		System.out.println("What's the Person's age?");
		age = Integer.parseInt(sc.nextLine());

		jeff = new Person(name,age); // Initializing Person object passing name and
age as parameters

		// Printing in console the information
		System.out.println("Information about the Person:");
		System.out.println("Name: " + jeff.getName());
		System.out.println("Age: " + jeff.age);

		sc.close(); // Closing data flow
	}

}
```

➢ **Program execution**

The program functionality is to capture a series of data into the attributes of the Person object created. The objective is to show the difference when accessing attributes while utilizing Access Modifiers.

```
What's the Person's name?
Jeff Bezos
What's the Person's age?
52
Information about the Person:
Name: Jeff Bezos
Age: 52
```

➢ **Conclusions**

Access Modifiers can completely change the way we access information and method from other classes, this is why it is important to correctly define the Access Modifiers for each Attribute and Class depending on what we want to accomplish with them and who we want to access them.