Topic: Applicable modifiers in classes declarations.
OOP concepts involved: Classes, Class Attributes, Class Access Modifiers, Inheritance.
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{}
```

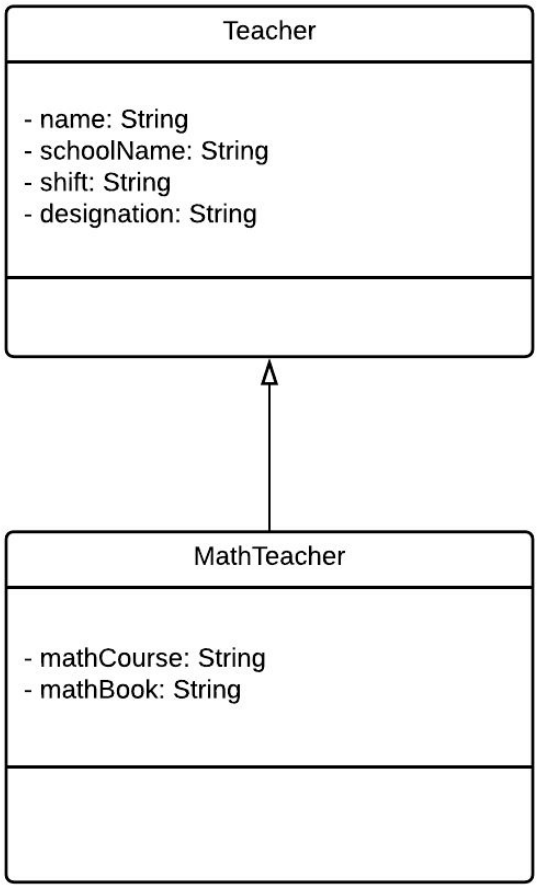
What this does, is it allows the class extending (also known as subclass or child class) to act as is it had attributes and methods from the class being extended (also known as the parent class). The subclass also has the option to override methods in the parent class, changing the contents in them,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

➢ Statement

A school wants to register all its teachers into a database, and we need to know the name, school name and shift of each teacher. Also, all math teachers must also include their course and book used for it into the database.

Develop a program that manages the registration of all General and Math Teachers while using the extended modifier. Use access and final modifiers to manage the methods and variables used in the parent and child class.

➢ **Class design (UML)**



➢ **Program Code**

Teacher.java

```
public class Teacher {
	//Defining object attributes (Its state)
	private String designation;
	private String name;
	private String schoolName;
	private String shift;

	// Constructor Method
	public Teacher(){
		designation = "Teaching";
	}
```

```
	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getSchoolName() {
		return schoolName;
	}

	public void setSchoolName(String schoolName) {
		this.schoolName = schoolName;
	}

	public String getShift() {
		return shift;
	}

	public void setShift(String shift) {
		this.shift = shift;
	}

	public String getDesignation() {
		return designation;
	}
}
```

MathTeacher.java

```
public class MathTeacher extends Teacher{
	//By making the class final, we prevent any subclass from inheriting from this class
		private String mathCourse;
		private String mathBook;

		// Constructor Method
		public MathTeacher(){
			super(); //This line of code calls the constructor of the parent class
		}

		public void printInfo(){
			System.out.println("Information about the Teacher:");
			System.out.println("Name: " + this.getName());
			System.out.println("Works at: " + this.getSchoolName());
			System.out.println("Doing: " + this.getDesignation());
			System.out.println("mathCourse: " + this.getMathCourse());
			System.out.println("Book used: " + this.getMathBook());
			System.out.println("Shift: " + this.getShift() + "\n");
```

```
	}

	//Getters and Setters
	public final void setMathCourse(String mathCourse){
		this.mathCourse = mathCourse;
	}

	public final void setMathBook(String mathBook){
		this.mathBook = book;
	}

	public final String getMathCourse(){
		return mathCourse;
	}

	public final String getMathBook(){
		return mathBook;
	}

}
```

Main.java

```
import java.util.Scanner;

public class Main {

	public static void main(String[] args) {
		// Declaring an object based on the MathTeacher class
		MathTeacher mt= new MathTeacher();

		// 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 to the object
		System.out.println("What's the Math Teacher's name?");
		mt.setName(sc.nextLine());
		System.out.println("What's the School's name?");
		mt.setSchoolName(sc.nextLine());
		System.out.println("What's the Teacher's work shift?");
		mt.setShift(sc.nextLine());
		System.out.println("What's the Teacher's Math course?");
		mt.setMathCourse(sc.nextLine());
		System.out.println("What's the Book used in the course?");
		mt.setMathBook(sc.nextLine());
		mt.printInfo(); // Printing the MathTeacher Info

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

```
        }
}
```

➢ Program execution

The objective of the program is to create an extended class MathTeacher that inherits all of the Teacher's class attributes while adding the extra attributes necessary for Math Teachers. The program also uses use of the Final modifier to avoid the subclass from being Inherited.

At the end of the program, we see the data we've collected for the teacher.

```
What's the Math Teacher's name?
Paul Raegan
What's the School's name?
DULS
What's the Teacher's work shift?
Morning
What's the Teacher's Math course?
Differential Calculus
What's the Book used in the course?
Differential Calculus for Engineers
Information about the Teacher:
Name: Paul Raegan
Works at: DULS
Doing: Teaching
mathCourse: Differential Calculus
Book used: Differential Calculus for Engineers
Shift: Morning
```

➢ Conclusions

All modifiers are great for limiting what others can do with certain variables, methods or classes. This gives the programmer control with the data and how it can be accessed, protecting the information.

While some of them, like **extends**, allow us to inherit attributes and methods from a class to create a subclass and have more control with our classes and save time re-coding what we already have.