

Tishk International University
Department of Computer Engineering
Object Oriented Programming
Week 6
Fall 2023-24
November 6, 2023



Modifiers, Inheritance and Method Overriding

Wisam Abdulaziz Qadir
Wisam.abdulaziz@tiu.edu.iq

Outline



- Modifiers
- Encapsulation (brief review)
- Inheritance
- Protected
- Object Class
- Method Overriding

Objectives



- Learning the difference between different types of modifiers in Java.
- Learning how to use inheritance.
- Learning how to method Overriding.

Modifiers



We divide modifiers into two groups:

- **Access Modifiers**: controls access level
- **Non-Access Modifiers**: provides some functionality

Access Modifiers



For classes

Modifier	Description
<code>public</code>	The class is accessible by all other classes
default	<ul style="list-style-type: none">• The class is only accessible by classes in the same package.• This is used when you don't specify a modifier.

Access Modifiers



For **attributes**, **methods** and **constructors**

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
default	<ul style="list-style-type: none">• The code is only accessible in the same package.• This is used when you don't specify a modifier.
protected	The code is accessible in the same package and subclasses .

Non-Access Modifiers



For classes

Modifier	Description
final	The class cannot be inherited by other classes
abstract	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class.

Non-Access Modifiers



For **attributes** and **methods**

Modifier	Description
final	Attributes and methods cannot be overridden/modified
static	Attributes and methods belong to a class, rather than an object
abstract	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example abstract void run(); . The body is provided by the subclass (inherited from).

Basics of OOP



- What is it?



A PIE

Abstract
Polymorphism
Inheritance
Encapsulation

- Each of these 4 are huge subjects and together form what is known as OOP.

Encapsulation



- Encapsulation refers to the bundling (packaging) of fields and methods inside a single class.
- A class has ***Attributes and Methods***.
- Encapsulation keeps them together.

Area
Width Length
area()

Encapsulation



- In Encapsulation, **Data Hiding** can be used to make sure that "sensitive" data is hidden from users.
- To achieve this, we must:
 - declare attributes as **private**.
 - provide public **get()** and **set()** to access value of **private** attributes.

Why Encapsulation?



- Better control of class attributes and methods.
- Class attributes can be made **read-only** (if you only use the **get** method), or **write-only** (if you only use the **set** method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

Inheritance



- In Java, it is possible to inherit attributes and methods from one class to another.
- We group the "inheritance concept" into **two** categories:
 - **superclass** (parent class) - the class being inherited from.
 - **subclass** (child class) - the class that inherits from another class.
- To inherit from a class, use the **extends** keyword.

Inheritance



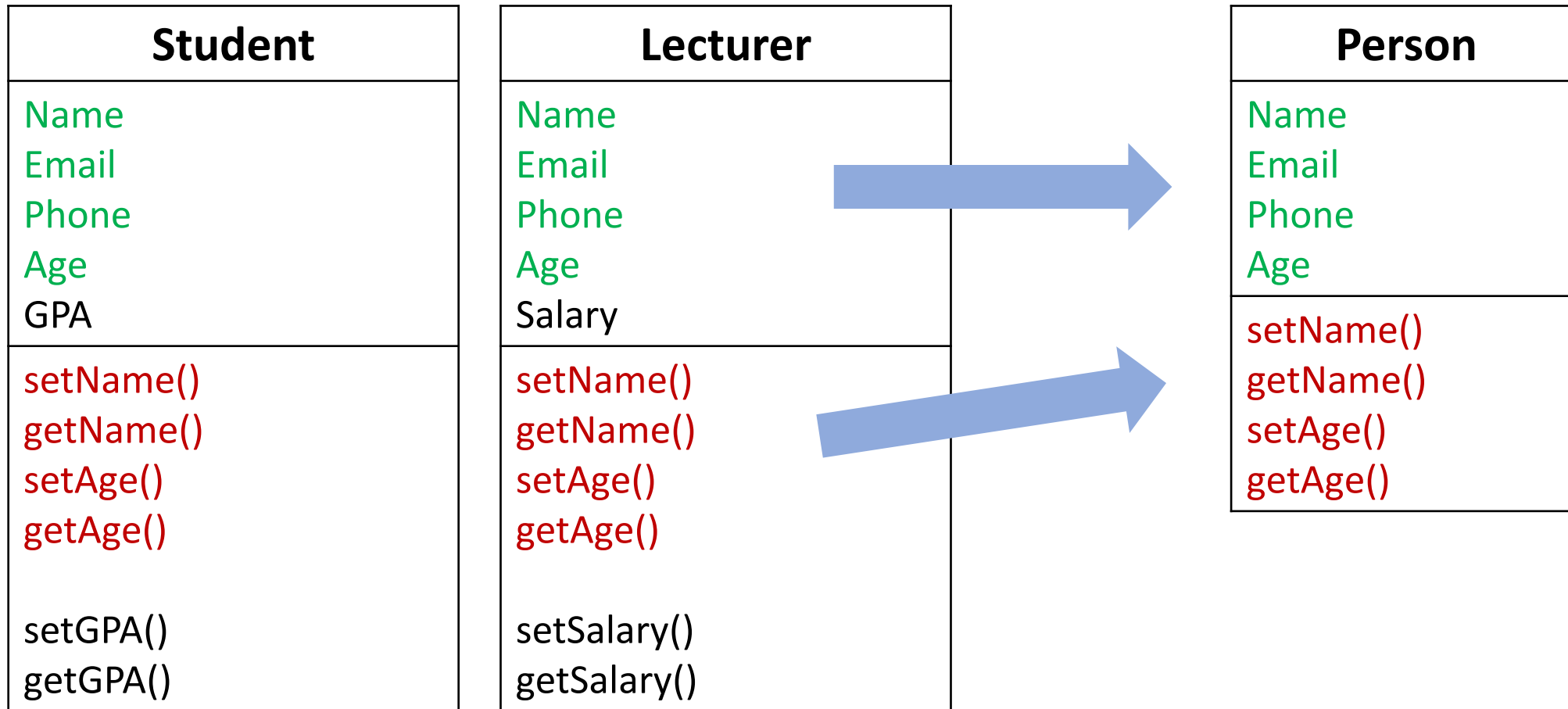
- Inheritance hierarchy (Inheritance relationships):
 - A class becomes
 - **Superclass**
 - When it supplies members to other classes
 - OR
 - **Subclass**
 - When it inherits members from other classes

Creating Classes without Using Inheritance

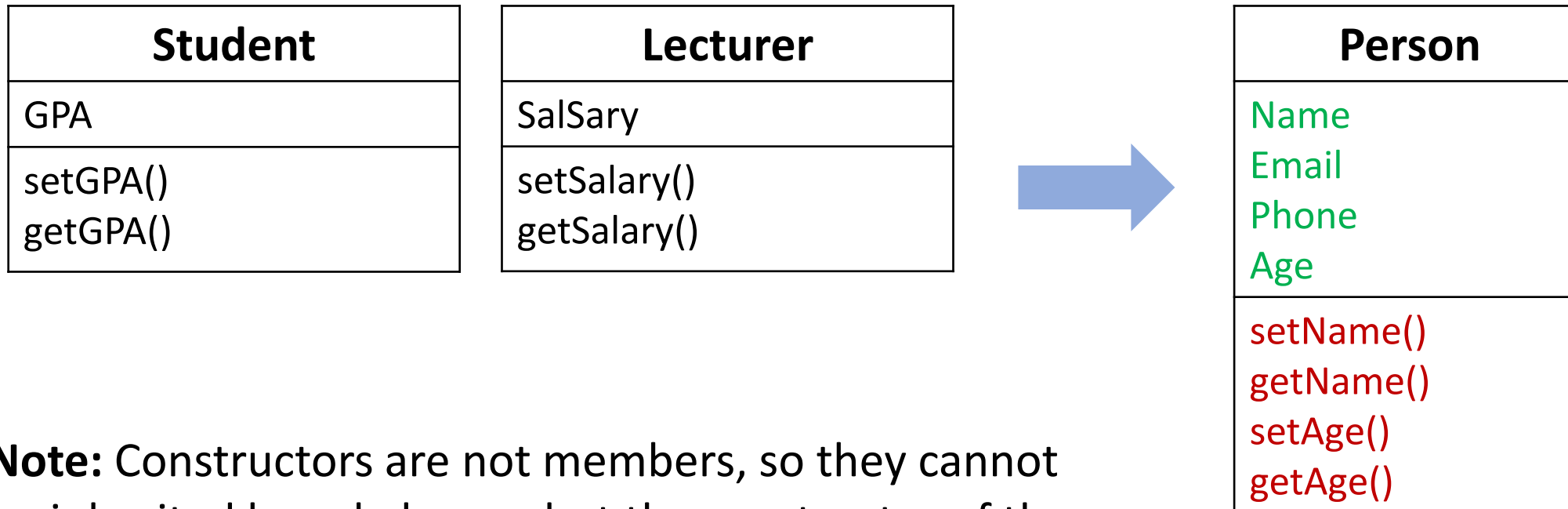


Student	Lecturer
Name Email Phone Age GPA	Name Email Phone Age Salary
setName() getName() setAge() getAge() setGPA() getGPA()	setName() getName() setAge() getAge() setSalary() getSalary()

Inheritance



Inheritance



Note: Constructors are not members, so they cannot be inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

Inheritance

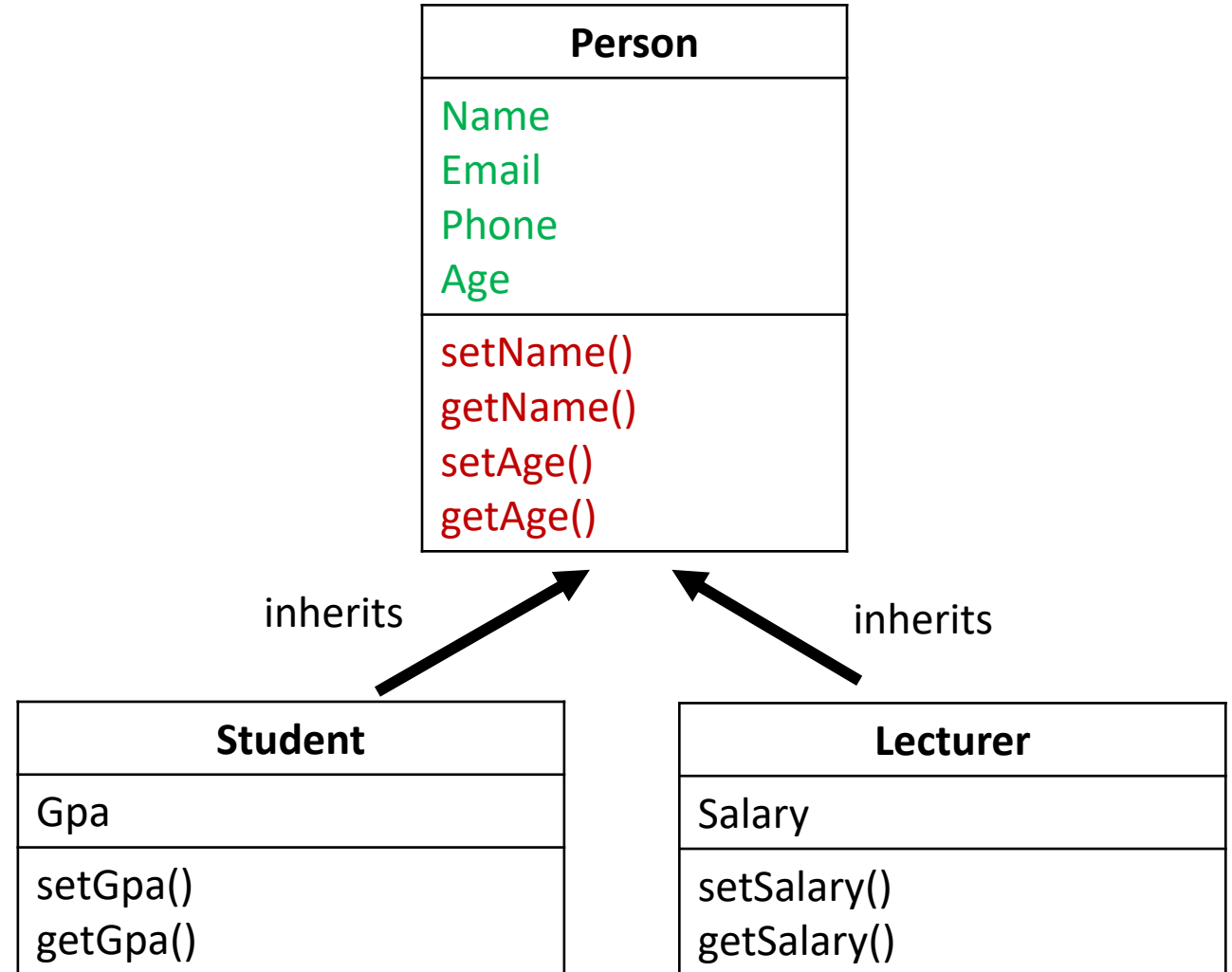


Superclass:

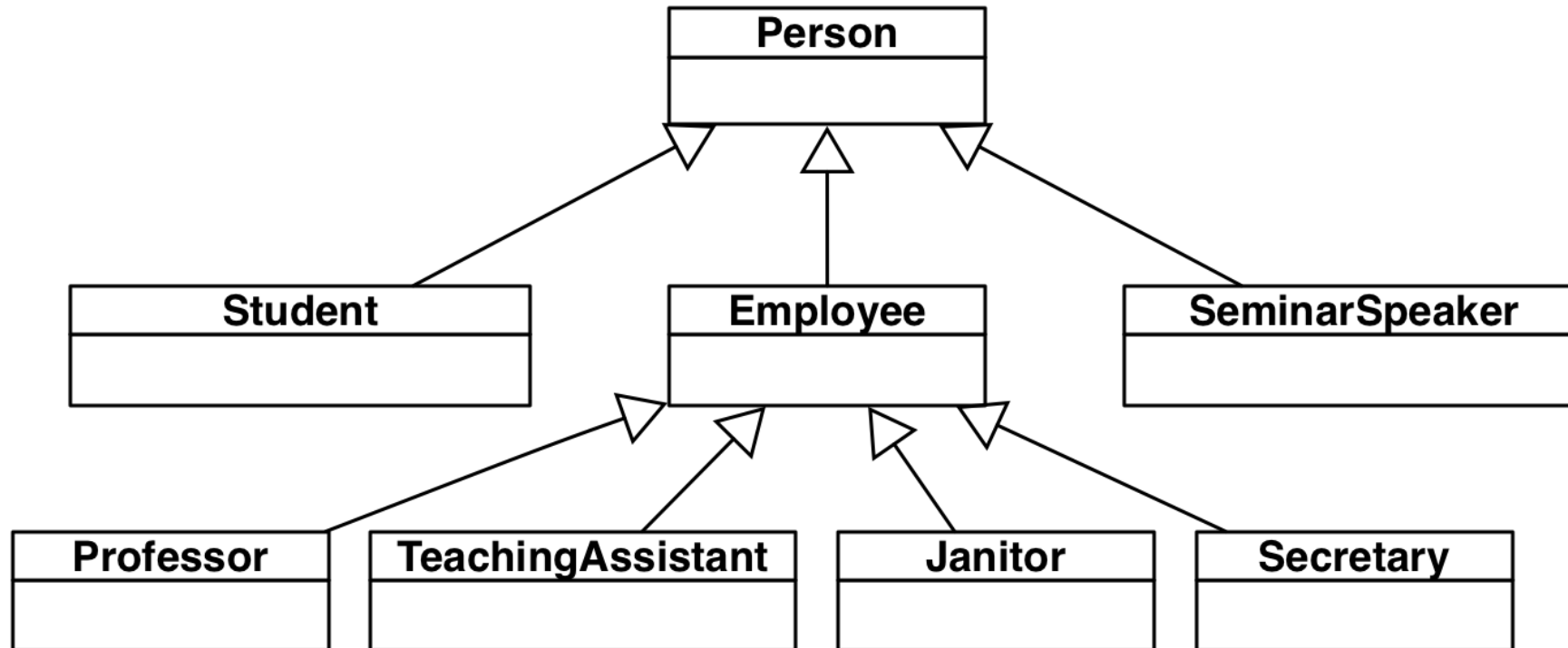
- Person

Subclass:

- Student
- Teacher



Inheritance





```
public class Person
{
    private String name;
    private int age;

    public void setName(String n)
    {
        name = n;
    }

    public String getName()
    {
        return name;
    }

    public void setAge(int a)
    {
        age = a;
    }

    public int getAge()
    {
        return age;
    }
}
```



```
public class Student extends Person
{
    private double gpa;

    public void setGpa(double g)
    {
        gpa = g;
    }

    public double getGpa()
    {
        return gpa;
    }
}
```

```
public class Lecturer extends Person
{
    private double salary;

    public void setSalary(double s)
    {
        salary = s;
    }

    public double getSalary()
    {
        return salary;
    }
}
```



```
public class MainClass
{
    public static void main(String[] args)
    {
        Student s001 = new Student();
        s001.setName("Dara");
        s001.setAge(20);
        s001.setGpa(3.5);

        System.out.println(s001.getName() + " : " + s001.getAge() + " : " + s001.getGpa());

        Lecturer l001 = new Lecturer();
        l001.setName("Dara");
        l001.setAge(20);
        l001.setSalary(3.5);

        System.out.println(l001.getName() + " : " + l001.getAge() + " : " +
l001.getSalary());
    }
}
```

Protected



- **Protected** access is Intermediate level of protection between **public** and **private**.
- **protected** members are accessible by
 - Class members in the **same package**
 - **Subclass** members

Protected



- Difference between **Public**, **Private** and **Protected**:
 - The **public** modifier specifies that the member can be accessed in **all packages**.
 - The **private** modifier specifies that the member can be accessed in its **own class only**.
 - The **protected** modifier specifies that the member can be accessed within its **own package only**, and by its subclasses in other packages.

Protected



Date.java

```
public class Date
{
    protected int day;
    protected int month;
    protected int year;

    public void displayDate()
    {
        System.out.println(day+ "/" + month + "/" + year);
    }
}
```

MainClass.java

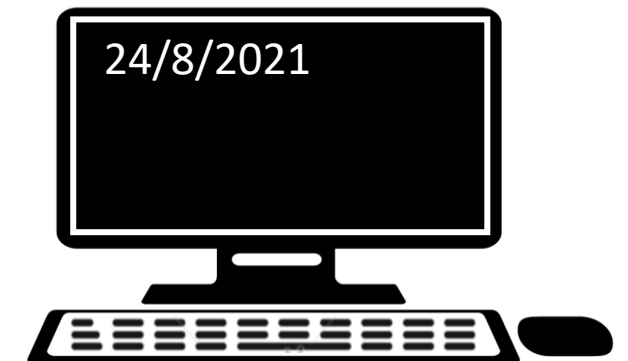
```
public class MainClass {

    public static void main(String[] args) {

        Date d01 = new Date();
        d01.day = 24;
        d01.month= 8;
        d01.year = 2021;

        d01.displayDate();
    }
}
```

Note: with protected, we don't need to use set() and get().



Object Class



- Extending **Object** class
 - Every class in Java **extends** an existing class, except **Object class**
 - Every class inherits methods of Object class.
 - Each new class **extends Object class** automatically which will be declared by the Java compiler, if it **does not extend any other classes**.

Method Overriding



- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.
- **Why method overriding?**
 - If a subclass should have some additional implementation of the method that has been declared by its Superclass.

Method Overriding



- Rules for Java Method Overriding?
 - The method must have the same **access modifier**, **return type**, **method name** and **parameter list**.
 - There must be an IS-A relationship (inheritance).

```
public class Person {  
  
    public void show(){  
        System.out.println("person");  
    }  
}
```

```
public class Student extends Person  
{  
    public void show(){  
        System.out.println("student");  
    }  
}
```

```
public class Lecturer extends Person  
{  
    public void show(){  
        System.out.println("lecturer");  
    }  
}
```

Method Overriding



```
public class MainClass {  
    public static void main(String[] args) {  
        Person p01 = new Person();  
        p01.show();  
  
        Student s01 = new Student();  
        s01.show();  
  
        Lecturer l01 = new Lecturer();  
        l01.show();  
    }  
}
```


```
public class Person {
    private String name;
    private int age;

    public void setName(String n)
    {
        name = n;
    }

    public String getName()
    {
        return name;
    }

    public void setAge(int a)
    {
        if(a >= 0)
            age = a;
        else
            age = 0;
    }

    public int getAge()
    {
        return age;
    }
}
```



```
public class Student extends Person
{
    private double gpa;
    private int age;

    public void setGpa(double g)
    {
        gpa = g;
    }

    public double getGpa()
    {
        return gpa;
    }

    public void setAge(int a)
    {
        if(a >= 18)
            age = a;
        else
            age = 18;
    }

    public int getAge()
    {
        return age;
    }
}
```

```
public class Lecturer extends Person
{
    private double salary;
    private int age;

    public void setSalary(double s)
    {
        salary = s;
    }

    public double getSalary()
    {
        return salary;
    }

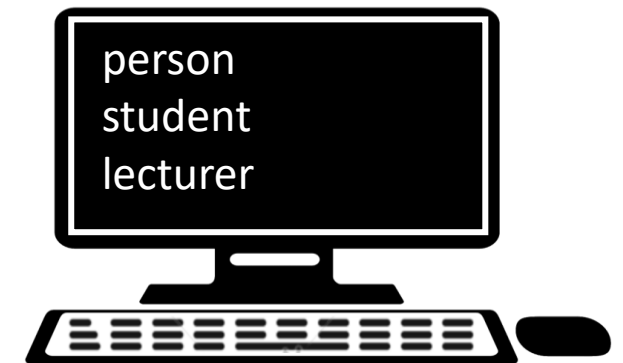
    public void setAge(int a)
    {
        if(a >= 25)
            age = a;
        else
            age = 25;
    }

    public int getAge()
    {
        return age;
    }
}
```

Method Overriding



```
public class MainClass {  
  
    public static void main(String[] args) {  
        Student s01 = new Student();  
        s01.setAge(19);  
        System.out.println(s01.getAge());  
  
        Lecturer l01 = new Lecturer();  
        l01.setAge(26);  
        System.out.println(l01.getAge());  
    }  
}
```





Thank You