CoPL
Object Oriented Programming

Crashcourse

Daan Wichmann

Radboud University

02 04 2025

Access Modifiers

Access modifiers

Access modifiers

```python
student = Student("Daan", "Wichmann", "secret", "secret")
```

Access modifiers

```
student = Student("Daan", "Wichmann", "secret", "secret")
print(student.s_number)
>>> "secret"
student.s_number = "different"
print(student.s_number)
>>> "different"
```

Access modifiers: the problem

- However, s-numbers generally shouldn't change!
- And if they do, we want to control how they change
- How do we fix this? **Access modifiers**

Access modifiers

Access modifiers

Access modifiers control the visibility (access) of the methods and data attributes of a class.

Access modifiers

- Access modifiers specify from where we can access our data attributes/methods, meaning:
  - for **methods**: where we can call them
  - for **data attributes**: where we can access them (get them), and where we can change them (set them)

Access modifiers

- We can modify access to our data attributes and methods at three levels:
    - public
    - protected
    - private
- Data attributes and methods are public by default

The different of access

	Inside the class	In derived classes	Outside the class
public	Yes	Yes	Yes
protected	Yes	Yes	No
private	Yes	No	No

The different of access

	Inside the class	In derived classes	Outside the class
public	Yes	Yes	Yes
protected	Yes	Yes	No
private	Yes	No	No

- Remember this table!!!!

How to modify access???

We can change the access modifier of a data attribute/method by changing the name:

- s_number (public)
- _s_number (protected)
- __s_number (private)

How to modify access???

We can change the access modifier of a data attribute/method by changing the name:

- s_number (public)
- _s_number (protected)
- __s_number (private)

We change the number of underscores before the name!

- public = 0 underscores
- protected = 1 underscore
- private = 2 underscores

Lets update our student class

Modifying the access

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
```

Modifying the access

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # We added two underscores
```

Modifying the access

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # We added two underscores

# This part is outside the class
student = Student("Daan", "Wichmann", 21, "s1234567")
print(student.__s_number)
```

- Is s_number a public, protected or private attribute?

Modifying the access

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # We added two underscores

# This part is outside the class
student = Student("Daan", "Wichmann", 21, "s1234567")
print(student.__s_number)
```

- Is s_number a public, protected or private attribute?
- Can we print it here?

Modifying the access

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # We added two underscores

    def talk(self):
        print(f"Student {self.__s_number} is talking")
```

- Can we access it here?

Remember our hierarchy

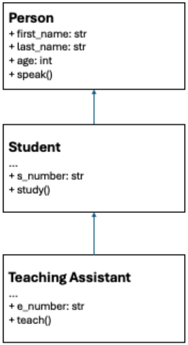

Figure: our hierarchy

Modifying the access

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number

# TeachingAssistant derives from Student
class TeachingAssistant(Student)
    def __init__(first_name, last_name, age, s_number,
    ↪  e_number):
        # Calling the constructor of Student
        super().__init__(first_name, last_name, age,
        ↪  s_number)
        self.__e_number = e_number

    def talk(self):
        print(f"Student {self.__s_number} is talking")
```

- Can we access it here?

Modifying the access again

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self._s_number = s_number  # Removed one underscore

class TeachingAssistant(Student)
    def __init__(first_name, last_name, age, s_number,
    ↪   e_number):
            # Calling the constructor of Student
            super().__init__(first_name, last_name, age,
            ↪   s_number)
            self.__e_number = e_number

    def talk(self):
        print(f"Student {self._s_number} is talking")
```

- Is s_number public, private or protected?

Modifying the access again

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self._s_number = s_number  # Removed one underscore

class TeachingAssistant(Student)
    def __init__(first_name, last_name, age, s_number,
    ↪   e_number):
            # Calling the constructor of Student
            super().__init__(first_name, last_name, age,
            ↪   s_number)
            self.__e_number = e_number

    def talk(self):
        print(f"Student {self._s_number} is talking")
```

- Is s_number public, private or protected?
- Can we access it in TeachingAssistant now?

Back to being private

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # It's private again!

class TeachingAssistant(Student)
    def __init__(first_name, last_name, age, s_number,
    ↪  e_number):
            # Calling the constructor of Student
            super().__init__(first_name, last_name, age,
            ↪  s_number)
            self.__e_number = e_number

    def change_student_number(self, new_student_number):
        self.__s_number = new_student_number
```

- Can we do this?

Back to being private

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # It's private again!

class TeachingAssistant(Student)
    def __init__(first_name, last_name, age, s_number,
    ↪  e_number):
            # Calling the constructor of Student
            super().__init__(first_name, last_name, age,
            ↪  s_number)
            self.__e_number = e_number

    def change_student_number(self, new_student_number):
        self.__s_number = new_student_number
```

- Can we do this?
- Not being able to access it, means we also shouldn't change it!

Python being Python

Python being Python

Access modifiers in Python (only for technical details)

In Python, access modifiers are not heavily enforced. Meaning, that if we wanted to, we can still access private/protected data attributes in places we shouldn't. However, it is the convention that you shouldn't!

- Other languages like Java, do heavily enforce their access modifiers.

Why are we doing this again?

- Why are we making our data attributes and methods private or protected?

Why are we doing this again?

- Why are we making our data attributes and methods private or protected?
- **Encapsulation**

Encapsulation

The problem

Why do we want to make our data attributes protected or private?

The problem

Why do we want to make our data attributes protected or private?

- Sometimes they contain information that shouldn't be known outside of the class

The problem

Why do we want to make our data attributes protected or private?

- Sometimes they contain information that shouldn’t be known outside of the class
- **We want to control how they are changed**

The problem

Why do we want to make our data attributes protected or private?

- Sometimes they contain information that shouldn’t be known outside of the class
- **We want to control how they are changed**
    - E.g., a student number should always start with a ’s’ and have 7 numbers.

The problem

Why do we want to make our data attributes protected or private?

- Sometimes they contain information that shouldn't be known outside of the class
- **We want to control how they are changed**
    - E.g., a student number should always start with a ’s’ and have 7 numbers.
    - A password should contain certain special characters

The problem

Why do we want to make our data attributes protected or private?

- Sometimes they contain information that shouldn't be known outside of the class
- **We want to control how they are changed**
  - E.g., a student number should always start with a 's' and have 7 numbers.
  - A password should contain certain special characters
- Sometimes you want a data attribute to be able to be read, but not be able to be set. And the other way around.

Solution to this problem

Solution: we make methods that change our protected and private data attributes (**setters**) and/or make methods that return the value of our data attribute (**getters**).

Solution to this problem

Solution: we make methods that change our protected and private data attributes (**setters**) and/or make methods that return the value of our data attribute (**getters**).

- We ’**encapsulate**’ our data attributes with *getters* and/or *setters*.

Solution to this problem

Solution: we make methods that change our protected and private data attributes (**setters**) and/or make methods that return the value of our data attribute (**getters**).

- We '**encapsulate**' our data attributes with *getters* and/or *setters*.
- **This is encapsulation!**

Let’s see how we do this in Python!

Making getters and setters

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # Private data attribute
```

Making getters and setters

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # Private data attribute

    @property  # This is a decorator that indicates its a gette
```

Making getters and setters

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # Private data attribute

    @property  # This is a decorator that indicates its a gette
    def s_number(self):  # Note the name
```

Making getters and setters

```
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # Private data attribute

    @property  # This is a decorator that indicates its a gette
    def s_number(self):  # Note the name
        return self.__s_number
```

Making getters and setters

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # Private data attribute

    @property  # This is a decorator that indicates its a gette
    def s_number(self):  # Note the name
        return self.__s_number

    @s_number.setter  # Decorator for setter
```

Making getters and setters

```python
class Student(Person):
    def __init__(first_name, last_name, age, s_number):
        super().__init__(first_name, last_name, age)
        self.__s_number = s_number  # Private data attribute

    @property  # This is a decorator that indicates its a getter
    def s_number(self):  # Note the name
        return self.__s_number

    @s_number.setter  # Decorator for setter
    def s_number(self, new_s_number):  # Note the name
        if new_s_number.startswith("s"):
            self.__s_number = new_s_number
        else:
            raise ValueError
```

Making getters and setters

```
class Student(Person):
    ...
    @property  # This is a decorator that indicates its a gette
    def s_number(self):
        return self.__s_number

    @s_number.setter  # Decorator for setter
    def s_number(self, new_s_number):
        if new_s_number.startswith("s"):
            self.__s_number = new_s_number
        else:
            raise ValueError

student = Student("Daan", "Wichmann", "Secret", "s1234567")
# Because of the 'property' decorator we can use the getter
# as if s_number is just a normal data attribute
print(student.s_number)
>>> "s1234567"
```

Making getters and setters

```python
class Student(Person):
    ...
    @property  # This is a decorator that indicates its a gette
    def s_number(self):
        return self.__s_number

    @s_number.setter  # Decorator for setter
    def s_number(self, new_s_number):
        if new_s_number.startswith("s"):
            self.__s_number = new_s_number
        else:
            raise ValueError

student = Student("Daan", "Wichmann", "Secret", "s1234567")
student.s_number = 1234567
print(student.s_number)
```

What will happen?

Making getters and setters

```python
class Student(Person):
    ...
    @property  # This is a decorator that indicates its a gette
    def s_number(self):
        return self.__s_number

    @s_number.setter  # Decorator for setter
    def s_number(self, new_s_number):
        if new_s_number.startswith("s"):
            self.__s_number = new_s_number
        else:
            raise ValueError

student = Student("Daan", "Wichmann", "Secret", "s1234567")
student.s_number = 1234567
print(student.s_number)
```

What will happen?

```
>>> ValueError
```

Making getters and setters

```python
class Student(Person):
    ...
    @property  # This is a decorator that indicates its a getter
    def s_number(self):
        return self.__s_number

    @s_number.setter  # Decorator for setter
    def s_number(self, new_s_number):
        if new_s_number.startswith("s"):
            self.__s_number = new_s_number
        else:
            raise ValueError

student = Student("Daan", "Wichmann", "Secret", "s1234567")
student.s_number = s1020425
print(student.s_number)
```

Making getters and setters

```
class Student(Person):
    ...
    @property  # This is a decorator that indicates its a gette
    def s_number(self):
        return self.__s_number

    @s_number.setter  # Decorator for setter
    def s_number(self, new_s_number):
        if new_s_number.startswith("s"):
            self.__s_number = new_s_number
        else:
            raise ValueError

student = Student("Daan", "Wichmann", "Secret", "s1234567")
student.s_number = s1020425
print(student.s_number)
>>> "s1020425"
```

Encapsulation is important

- Encapsulation is one of the most important concepts of OOP!

Encapsulation is important

- Encapsulation is one of the most important concepts of OOP!
- Make sure to study it from the material, before the exam.

Encapsulation is important

- Encapsulation is one of the most important concepts of OOP!
- Make sure to study it from the material, before the exam.

Encapsulation (definition from the material)

Hiding attributes from clients is called encapsulation. As the name implies, the attribute is "enclosed in a capsule". The client is then offered a suitable interface for accessing and processing the data stored in the object.

Encapsulation is important

- Encapsulation is one of the most important concepts of OOP!
- Make sure to study it from the material, before the exam.

Encapsulation (definition from the material)

Hiding attributes from clients is called encapsulation. As the name implies, the attribute is "enclosed in a capsule". The client is then offered a suitable interface for accessing and processing the data stored in the object.

- Client := your program
- The suitable interface that is referred to are the getters and setters

Seperation of Concerns

Separation of Concerns

Separation of Concerns

Separation of concerns is a design principle for separating a computer program into distinct sections such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program.

Separation of Concerns

- One of the most important concepts of OOP! (together with encapsulation).
- Classes separate our program into different parts
- This makes it so that one part of the program does not have to be **concerned** with another part of our program.
- **Encapsulation** facilitates this by hiding the implementation of our classes behind well defined interfaces (e.g. getters and setters).

Separation of Concerns

- One of the most important concepts of OOP! (together with encapsulation).
- Classes separate our program into different parts
- This makes it so that one part of the program does not have to be **concerned** with another part of our program.
- **Encapsulation** facilitates this by hiding the implementation of our classes behind well defined interfaces (e.g. getters and setters).
  - So other classes don't have to be concerned with how a variable is changed or read.
- **In a nutshell:** a class should only do what it is designed to do.

Why you should use OOP!

Why you should know about seperation of concerns and encapsulation.

Why you should use OOP!

Let’s say you have a group project, to create a phone book app.

Why you should use OOP!

Let’s say you have a group project, to create a phone book app.

- You create a **PhoneBook** class that stores phone numbers.

Why you should use OOP!

Let’s say you have a group project, to create a phone book app.

- You create a **PhoneBook** class that stores phone numbers.
- You want to be able to add and see phone numbers via the terminal.

Why you should use OOP!

Let’s say you have a group project, to create a phone book app.

- You create a **PhoneBook** class that stores phone numbers.
- You want to be able to add and see phone numbers via the terminal.
- (Following seperation of concerns) you create a different class **PhoneBookApplication** that handles the input, output, and the loop of your program.

Why you should use OOP!

Let’s say you have a group project, to create a phone book app.

- You create a **PhoneBook** class that stores phone numbers.
- You want to be able to add and see phone numbers via the terminal.
- (Following seperation of concerns) you create a different class **PhoneBookApplication** that handles the input, output, and the loop of your program.
- You also want to maintain a database with all phone numbers (e.g. txt file, CSV file, or MySQL database).

Why you should use OOP!

Let’s say you have a group project, to create a phone book app.

- You create a **PhoneBook** class that stores phone numbers.
- You want to be able to add and see phone numbers via the terminal.
- (Following seperation of concerns) you create a different class **PhoneBookApplication** that handles the input, output, and the loop of your program.
- You also want to maintain a database with all phone numbers (e.g. txt file, CSV file, or MySQL database).
- So, you create another class that interacts with a database **DatabaseHandler**

Why you should use OOP!

- Let’s assume person A made the PhoneBook class, person B made the PhoneBookApplication class, and person C made the DatabaseHandler class.

Why you should use OOP!

- Let's assume person A made the PhoneBook class, person B made the PhoneBookApplication class, and person C made the DatabaseHandler class.
- By creating different classes you can divide the work nicely!

Why you should use OOP!

- Let’s assume person A made the PhoneBook class, person B made the PhoneBookApplication class, and person C made the DatabaseHandler class.
- By creating different classes you can divide the work nicely!
- Lets assume all classes have well defined methods.

Why you should use OOP!

- Let’s assume person A made the PhoneBook class, person B made the PhoneBookApplication class, and person C made the DatabaseHandler class.
- By creating different classes you can divide the work nicely!
- Lets assume all classes have well defined methods.
- For example, the DatabaseHandler has a method save_to_database(phonebook).

Why you should use OOP!

- Let’s assume person A made the PhoneBook class, person B made the PhoneBookApplication class, and person C made the DatabaseHandler class.
- By creating different classes you can divide the work nicely!
- Lets assume all classes have well defined methods.
- For example, the DatabaseHandler has a method save_to_database(phonebook).
- Only person C has to be concerned with how this method works, and has to make sure that it saves to a database.

Why you should use OOP!

- Let’s assume person A made the PhoneBook class, person B made the PhoneBookApplication class, and person C made the DatabaseHandler class.
- By creating different classes you can divide the work nicely!
- Lets assume all classes have well defined methods.
- For example, the DatabaseHandler has a method save_to_database(phonebook).
- Only person C has to be concerned with how this method works, and has to make sure that it saves to a database.
- Person B needs to use this method, but does not need to be **concerned** with how it works! The implementation is **encapsulated** behind the method.

Why you should use OOP!

- Separation of Concerns and encapsulation makes creating big projects (in groups) way easier!

That is it!
(make sure to still look at the material!)

NB

I did not cover **class attributes**. Please study this in your own time, because it is still fairly important! (Part 9.5 of the material)

Extra info

If you are interested in programming, I highly recommend following: Object Orientend Programming (NWI-IPI005)

You can find these slides and the code from the presentation on my github: `https://github.com/D21W12`