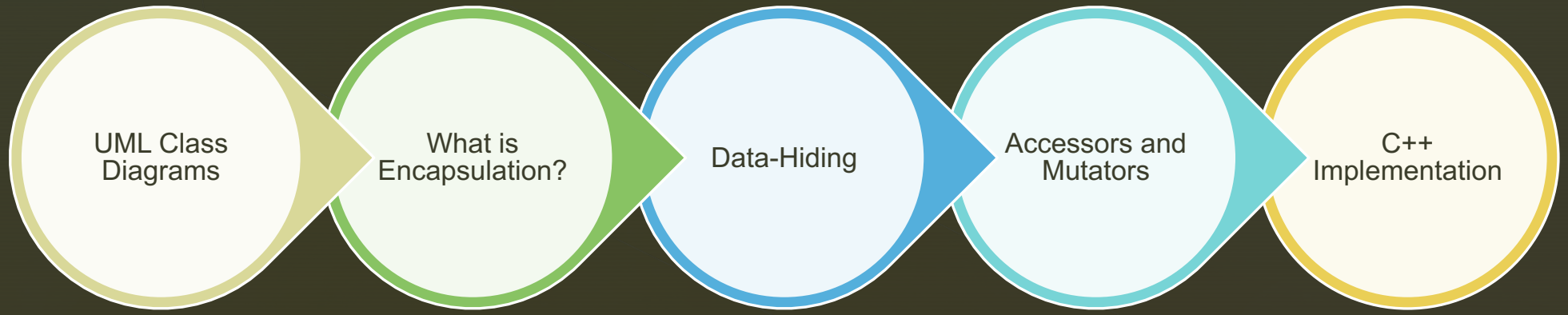
DR. (MRS) T-S.M.A. ADJAIDOO

06
Encapsulation & Data-Hiding

Today's Lesson



Learning Outcomes

1. Be able to design simple class diagrams correctly
2. Appreciate the need for data encapsulation and hiding
3. Learn to use access specifiers and restrict access to data in python

UML Class Diagrams

UNIFIED MODELING LANGUAGE (UML)

- UML is currently the standard notation for documenting object-oriented systems
- It provides a pictorial or graphical notation for documenting the artefacts such as classes, objects and packages that make up an object-oriented system

UML Class Diagrams

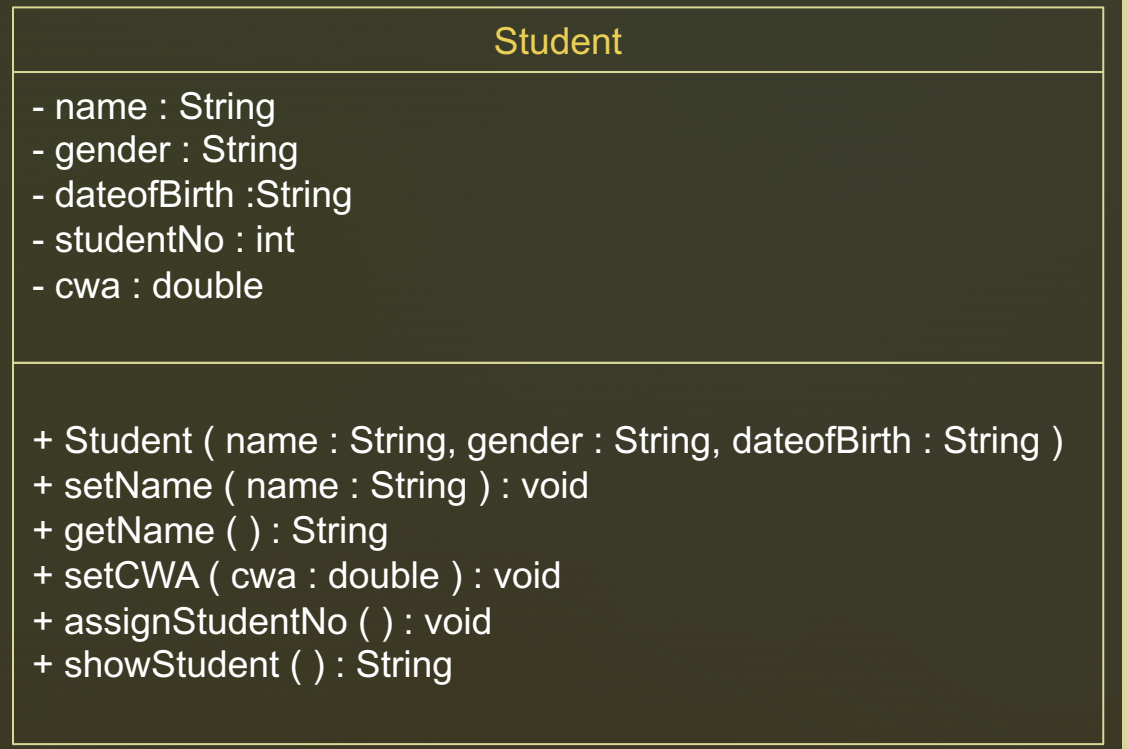
UNIFIED MODELING LANGUAGE (UML)

UML diagrams can be divided into three categories.

- Structure diagrams: show the static architecture of the system irrespective of time.
  - For example, structure diagrams for a university system may include diagrams that depict the design of classes such as Student, Faculty, etc.
- Behaviour diagrams: depict the behaviour of a system or business process.
- Interaction diagrams: show the methods, interactions and activities of the objects.
  - For a university system, a possible behaviour diagram would show how a student registers for a course.

UML Class Diagrams

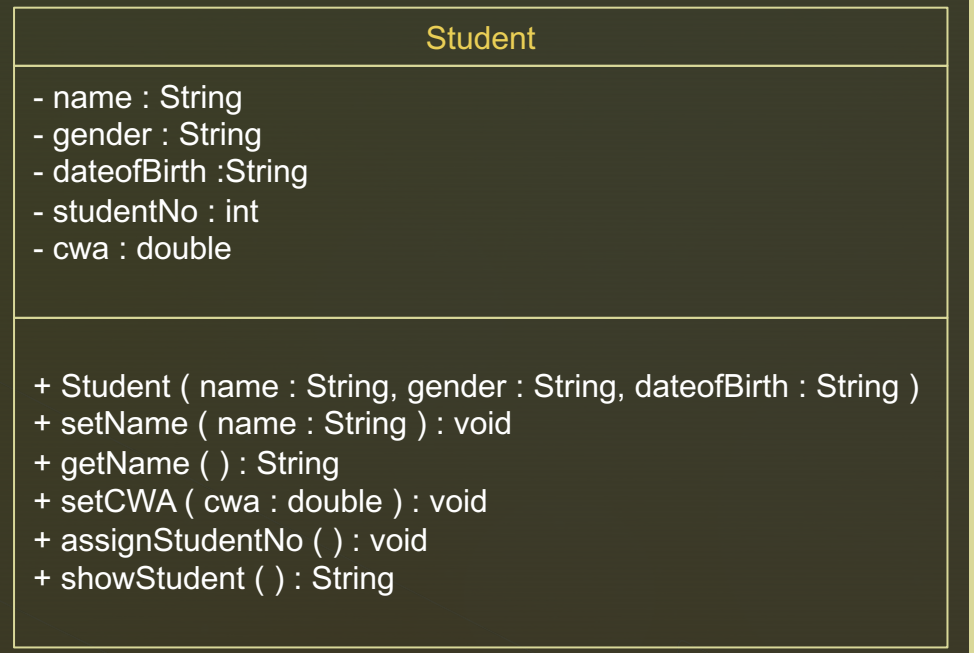
- Class diagrams are structured diagrams which show the classes, their methods and fields
- We will use UML class diagrams to illustrate the classes that we build in this course so let's take a look at how we draw class diagrams
- As we delve more deeply into OO design, these class diagrams will get much more sophisticated and convey much more information on how the different classes interact with each other.



Example of a class diagram

UML Class Diagrams

- Each class is represented by a box, which is divided into three rectangles.
- The name of the class is given in the top rectangle.
- The attributes are shown with their names and their types in the second box.
- The third box shows the methods with their return types and parameters (names and types)
  - The access specifier for each field and method is given just in front of the field name or method name.
    - − (minus) sign indicates private access
    - + (plus) stands for public access and
    - # (hash) is used for protected access



What is Encapsulation?

- One of the primary advantages of using objects is that the object need not reveal all its attributes and behaviours.
- Encapsulation is defined by the fact that objects contain both the attributes and behaviours. A class itself is an example of encapsulation as it captures attributes and behaviours in objects
- Data hiding is a major part of encapsulation

Data-Hiding

- Data-hiding involves hiding data or information from users
- Object details (data members and implementations) that are not relevant to the use of an object are hidden from other objects
- This makes the program more robust and secures
- For instance, in our school management program, a student object has a method which generates a studentNo. A course object requesting for the studentNo does not need to have access to how that number is generated. So that aspect is hidden from the course object
- To achieve data-hiding we use access specifiers

Data-Hiding

Access Specifiers

- Most O-O programming languages control access to class members using access modifiers
- There are fundamentally three types of access specifiers used:
  - Public Access Specifier
  - Private Access Specifier
  - Protected Access Specifier

Data-Hiding

Public Access Specifier

- Class members which are specified as public are accessible to all members of their class and also to objects outside their class
- They can be accessed easily for any part of the program
- By default all class members are public unless specified otherwise
- This is the least secure specifier
- In C++ public members are written under the label, private:

Data-Hiding

Private Access Specifier

- Class members which are declared as private are accessible to only members of their class
- They cannot be accessed from without the class
- This is the most secure type of access specifier
- In C++ private members are written under the label, private:

Data-Hiding

Protected Access Specifier

- Class members which are declared as protected are accessible to only members of their derived classes (We will learn about derived classes in the next chapter)
- In C++, protected members are written under the label, protected:

Accessors and Mutators

- Recall the getX() and setX() methods we mentioned in the previous chapter
- We have just discussed the need to keep internal data of objects private
- This means there must be some methods in the class that allow for controlled access to object data
- These methods are referred to as Accessors and Mutators

Accessors and Mutators

Accessor Methods

- These grant access to internally hidden data in an object for use.
- This method cannot be used to change the data values
- They are commonly called getters
- The methods are prefixed with the word get
- Eg: getName( ), getId( )

Accessors and Mutators

Mutator Methods

- These grant access to change or modify the values of internally hidden data in an object.
- This method cannot be used to retrieve the data values
- They are commonly called setters
- The methods are prefixed with the word set
- Eg: setName(nameValue), getId(idValue)

C++ Implementation

Let's look at encapsulation in this example.

Pay attention to the access specifiers

```
#include <iostream>
using namespace std;

class smallobj //define a class
{
    private:
        int somedata; //class data
    protected:
        void somefunct(){
            cout << "Hello Parent"<< endl;
        }
    public:
        void setdata(int d) //member function to set data
            { somedata = d; }
        void showdata() //member function to display data
            { cout << "Data is "<< somedata << endl; }
};
```

C++ Implementation

The derived class accesses the protected method

```
class smallobjA : public smallobj //define a class
{
    private:
        int somedataA; //class data

    public:
        void setdataA(int d) //member function to set data
            { somedataA = d; }
        void showdataA() //member function to display data
            {   somefunct();
                cout << "Data is "<< somedataA << endl;
            }
};
```

C++ Implementation

```
int main()
    {
        smallobj s1; //define two objects of class smallobj
        s1.setdata(23);
        s1.showdata(); //call member function to display data
        smallobjA s2;
        s2.setdataA(100);
        s2.showdataA();
        return 0;
}
```

```
Data is 23
Hello Parent
Data is 100
```

Class Activity

- Draw class diagrams for the following program:
  - An application that lets a user send and receive messages from other users of the same application
- What objects would you need?
- What attributes and methods would be required by the objects to accomplish the task?

Any Questions?

The End

Contact: tsadjaidoo@knust.edu.gh

Office: Caesar Building, Room 413