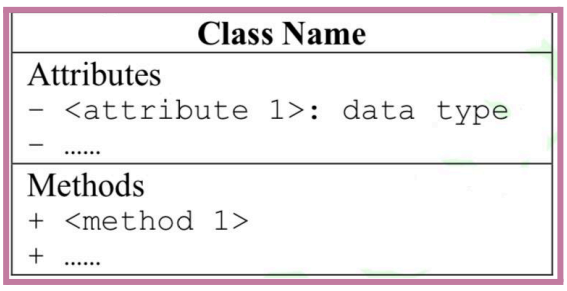
Object-Oriented Programming

definitions are highlighted!

Objects

- In OOP, the world inside a program is viewed as a collection of interacting objects, which can represent a real-world entity or something abstract.
- All processing in the program is performed through objects.
- Each object is responsible for:
    - Managing its own data (**attributes**)
        - *Attribute — a piece of data stored in an object*
        - An attribute is also known as: property, data member of the class
    - Handing its own actions (**methods**)
        - *Methods are actions that an object can perform.*
        - e.g. A linked list can add a node, delete a node, display data etc.
- An object also has a **state**, which describes its current condition.
- Objects communicate with one another by sending messages and receiving responses.

Classes

- A class defines the attributes and behaviours that its objects will have.
    - The attributes can hold different values for different objects.
- Class diagram format:
    - 

- Class Name is written as one word (e.g. AccountHolder).
- Symbols used:
  - "+": Public — can be accessed from anywhere
  - "-": Private — can only be accessed inside the class; cannot be accessed directly
  - "#": Protected — can be accessed by the class and its child classes
- In our syllabus, we focus on public methods and private attributes.
  - Attributes are private to protect the data inside the class.
  - Methods are public so that other classes can interact with the object safely.
    - They allow safe reading (get) or updating (set) of data.

- Pseudocode for making a class:

```
CLASS AccountHolder

    PRIVATE HolderName: STRING
    PRIVATE Address:    STRING
    PRIVATE Profession: STRING
    PRIVATE Birthday:   STRING

    PUBLIC PROCEDURE new(name,address,profession,birthday: STRING)
        (instructions to set initial values)
    ENDPROCEDURE

    PUBLIC PROCEDURE ChangeResidence (newAddress: STRING)
        (instructions)
    ENDPROCEDURE

    PUBLIC PROCEDURE ChangeProfession (newProfession: STRING)
        (instructions)
    ENDPROCEDURE

ENDCLASS
```

  - "new" is the name of the constructor (explained below).

Instantiation (object creation)

- Once we create a class, we can use it to create objects.
- When we instantiate an object, the object is given its own copy of attributes and methods defined by the class.
- *Constructor — a special method that runs when an object is created*
    - Sets up a new object, often by assigning initial values to its attributes
    - The class name is used like a function to create the object (refer to example below).
    - Each object created by a class is called an **instance**.
        - All objects from the same class share the same structure (attributes and methods), but have unique data (attribute values).
    - Pseudocode example:

```
acct1= new AccountHolder("Ronald Lee","21 Kallang","Teacher","12/12/1988")
```

        - acct1 is a reference type variable — it holds a reference (aka pointer) to the object stored in memory.
- Some languages have destructors to clean up an object when it is no longer needed.
    - Because Python handles memory automatically, we usually do not need destructors.

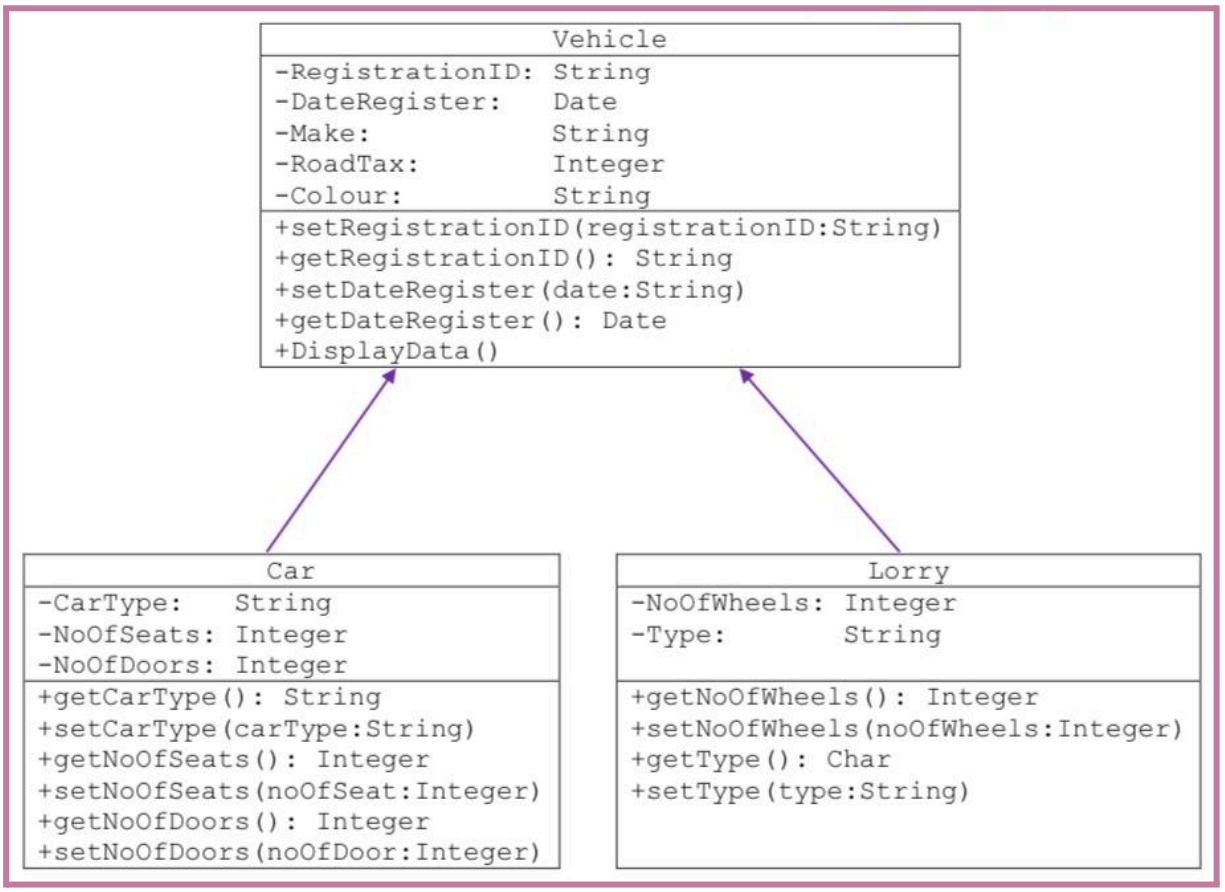
Encapsulation

- *Grouping an object's attributes (data) and methods (actions) together, hiding the internal details from the outside world*
- Protects an object's data by controlling how it is accessed and changed

    - **Controlled access:** Data is only accessible through public methods (getters and setters). This helps to keep the data accurate and consistent.
        - Protects the object from bad data
        - Getters and setters:
            - Getter methods are used to read attribute values.
            - Setter methods are used to update attribute values safely.
                - Checks can be added inside these to help to ensure that only valid data is stored.
    - **Accident prevention:** Since other parts of the program cannot directly touch the data, it is less likely that bugs or errors will occur from accidental changes.
    - **Easier teamwork:** Encapsulation makes it easier for different programmers to work on separate parts of a program without interfering with each other. Since each object hides its internal details, others only need to know how to use it, not how it works inside.
- Leads to implementation independence, which means that we can change how an object works internally (e.g. changing the data structure) without affecting the rest of the program as long as the interface (method) stays the same
- The technique of hiding the inner workings of an object and showing only what is necessary is called information hiding.
- Keeps the code modular and reusable

Inheritance

- A derived class (aka subclass, child class) can reuse and extend the attributes and methods of a base class (aka superclass, parent class), allowing us to create new classes quickly, without rewriting existing code.
    - Derived classes may have their own specific attributes and methods. (e.g. refer to class diagram)
- "Is-a" rule:
    - Before using inheritance, always ask yourself: Is a [derived class] truly a kind of [base class]?
    - The inheritance is only appropriate if the "is-a" relationship makes sense.
- Class diagram:



    - **Note:** The arrow points upward to the base class.

- Benefits:
    - **Code reuse:** Methods do not need to be rewritten; instead, they are inherited directly from the base class.
        - Reduces code duplication
    - **Easier maintenance:** Fixes or changes made in the base class automatically apply to all derived classes, reducing errors and saving time.
        - Makes programs easier to maintain and expand
    - **Logical structure:** Models real world relationships between objects clearly.
    - **Code generalisation:** Allows a derived class to inherit general attributes and methods from a base class, while also adding its own specialised features
        - Lays the groundwork for polymorphism, where derived classes can override methods to behave differently while still being used in the same general way as their base class

Polymorphism

- *Ability to process objects differently depending on their class*
- Allows different objects to respond differently to the same action (method)
    - Even when different classes share the same method name, the behaviour can change based on the object's type.
    - This makes programs flexible, easier to maintain and more powerful.
- Builds on code generalisation by allowing subclasses to override the same method in their own unique way
- Method overriding
    - Happens when a subclass redefines a method it inherited from its parent class

        - The method name and input parameters stay the same.
        - The behaviour is different.
    - Ensures that each class can have its own version of the method without needing to modify existing code
    - The correct method is automatically called based on the object type.
- Benefits:
    - Simplified code
        - There is no need to manually check an object's type before calling a method, reducing the need for complex if-else statements.
        - When you call the method, the correct version will automatically run based on the object's type.
    - Greater flexibility — different objects can be treated similarly while allowing them to behave differently
        - Allows programs to adapt their behaviour dynamically based on the actual object type at runtime, making them more flexible and robust
    - Easier to extend programs — new object types can be added without changing existing code
        - As long as the new class inherits from the same base class and implements the necessary methods, it can fit into the system naturally.
        - This makes the programme easier to expand and maintain over time.

@star.notes