Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Object – Oriented Analysis and Design

Object – Oriented Programming

Instructor: Le Thi Ngoc Hanh, Ph.D

ltnhanh@hcmiu.edu.vn

Outline

- Programming Styles
- Why Object – Oriented?
- Object – Oriented Programming
- Reading: [R3]-Ch1

Warm up Problem

If you were to write code to order food in a restaurant, how would it differ if you were telling the system how to cook vs. just telling it what you want?

Programming Styles

In Software Engineering, there are 2 main programming styles:

- Imperative programming styles
- Declarative programming styles

Imperative vs. Declarative

In IMPERATIVE PROGRAMMING

- the programmer describes **how** to solve a problem with a set of instructions that change a program's state.
  - The execution of every instruction is sensitive to the state of the program, and can change it
- **State** = data stored in variables, data structures, or accessible from external sources
- **Program** = collection of instructions: assignments, changes of mutable data (lists, arrays, etc.), etc.

In DECLARATIVE PROGRAMMING

- the programmer describes **what** relationships hold between various entities.
- **Program** = collection of definitions of functions or relations.

Logic Programming Style

- **Facts**: statements that are unconditionally true.
- **Rules:** define logical relationships between facts.
- Query

Logic Programming Style

```
% Facts: Defining who is male and female
male(john).
male(mike).
female(susan).
female(emily).

% Facts: Defining parent relationships
parent(john, mike).   % John is a parent of Mike
parent(susan, mike).  % Susan is a parent of Mike

% Rules: Defining relationships based on the facts

% A mother is a female parent
mother(X, Y) :- parent(X, Y), female(X).

% A father is a male parent
father(X, Y) :- parent(X, Y), male(X).

% query
?- mother(X, mike).
==> X = susan.
```

What is OOP?

OOP is defined as a programming paradigm (not a specific language) built on the concept of objects.

Example: Modeling an Online Banking System

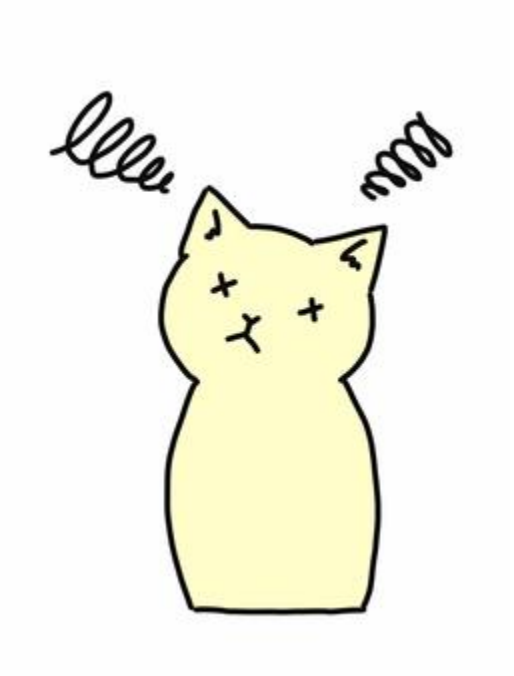
```
# Procedural style
balance = 0

def deposit(amount):
    global balance
    balance += amount

def withdraw(amount):
    global balance
    if balance >= amount:
        balance -= amount
    else:
        print("Insufficient funds")

deposit(100)
withdraw(50)
print("Balance:", balance)
```

```
# OOP style
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient funds")

    def get_balance(self):
        return self.balance

# Creating an instance of BankAccount
account = BankAccount()
account.deposit(100)
account.withdraw(50)
print("Balance:", account.get_balance())
```

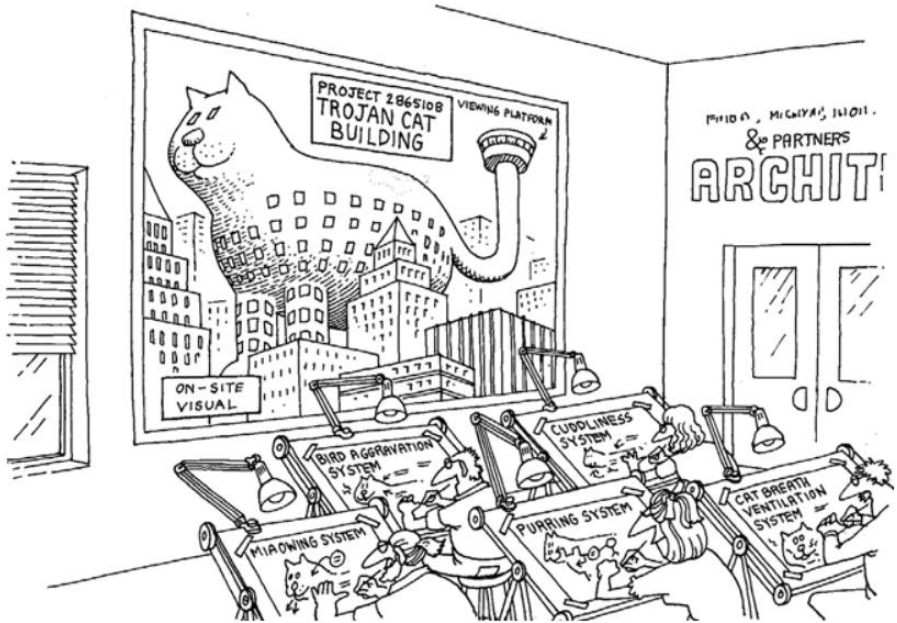
What is OOP?

- OOP is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.
- OOP approach is well suited for software that is large, complex and actively updated or maintained.
- The first stage in OOP is to gather all the objects that a programmer wishes to work with and determine their relationships, a process known as data modeling. Data and functions are combined to create an object from the data structure.

Why Object – Oriented?



Why Object – Oriented?



The architecture of a complex system is a function of its components as well as the hierarchic relationships among these components.

Modeling

In software development, two most common ways to approach a model:

- Algorithmic perspective
- Object – oriented perspective

Algorithmic Decomposition

- Sample functions in online banking system:

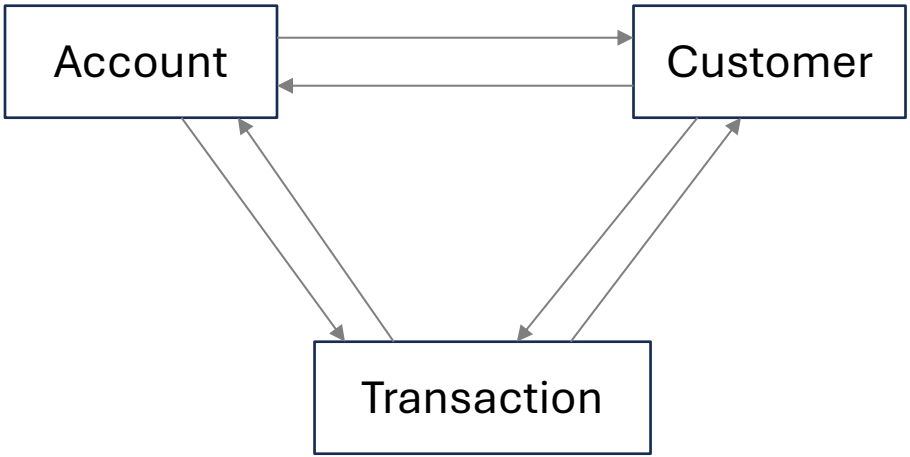
  widthdraw (accountID, amount)

  deposit (accountID, amount)

  checkBalance (accountID)

Object – Oriented Decomposition

- Objects: Account, Customer, Transaction, etc. Each object encapsulates both data (attributes) and behaviors (methods).



Algorithmic vs Object – Oriented?

- Which is the right way to decompose a complex system - by algorithms or by objects?
- We can not use both. We must start decomposing a system either by algorithms or by objects and then use the resulting structure as the framework for expressing the other perspective.

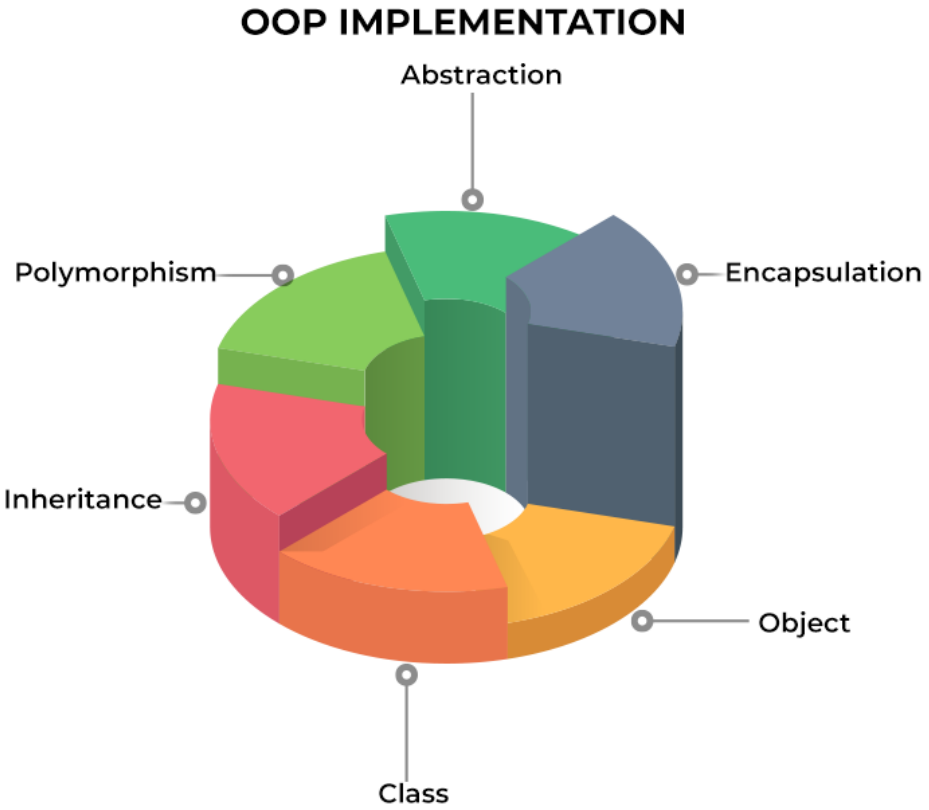


Image credit: Spiceworks

Classes

- A collection of objects that share common properties, attributes, behavior and semantics, in general.
- A collection of objects with the same data structure (attributes, state variables) and behavior (function/code/operations) in the solution space.
- A class defines a new data type for our programs to use.

Classes vs Struct

```
struct BackpackItem {
    int survivalValue;
    int weight;
};

struct Juror {
    string name;
    int bias;
};
```

Definition

struct
A way to bundle different types of information in C++ – like creating a custom data structure.

Classes vs Struct

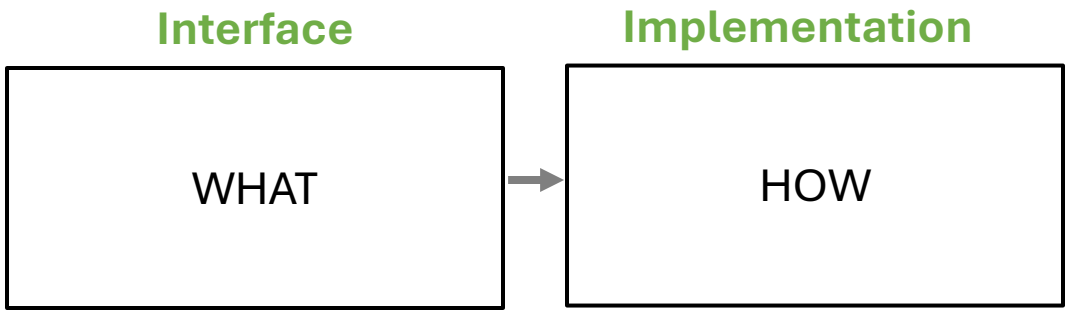
```
GridLocation chosen;
cout << chosen.row << endl;
cout << chosen.col << endl;

chosen.row = 3;
chosen.col = 4;
```

```
GPoint origin(0, 0);
cout << origin.getX() << endl;
cout << origin.getY() << endl;

origin.x = 3;
origin.y = 4;
```

What's the difference in how you use a GridLocation vs. a GPoint?

Classes

- Structure of class: 2 parts



Identifying classes

- We want our class to be a grouping of conceptually-related state and behaviour
- One popular way to group is using grammar
  - Noun → Object
  - Verb → Method

"A quiz program that asks questions and checks the answers are correct"

Objects

- Object is an instance of a class that holds data (values) in its variables. Data can be accessed by its functions.
- Methods are functions that objects can perform. They are defined inside a class that describe the behaviors of an object.
- Attributes represent the state of an object. In other words, they are the characteristics that distinguish classes.

Objects and Classes

Attributes:
Name, height, age

Behaviors:
Speak, listen, eat, run, walk

Object person 1

Attributes:
Name: Tomy
Height: 175cm
Age: 22

Behaviors:
Speak, listen, eat, run, walk

Object person 2

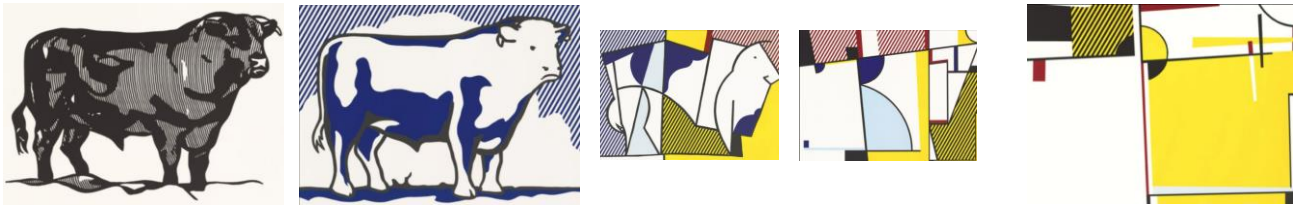
Attributes:
Name: Rose
Height: 163cm
Age: 21

Behaviors:
Speak, listen, eat, run, walk

Abstraction

- Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.
- Design that hides the details of how something works while still allowing the user to access complex functionality.
- Abstraction can help developers more easily make additional changes or additions over time.

Abstraction



abstracting

Definition

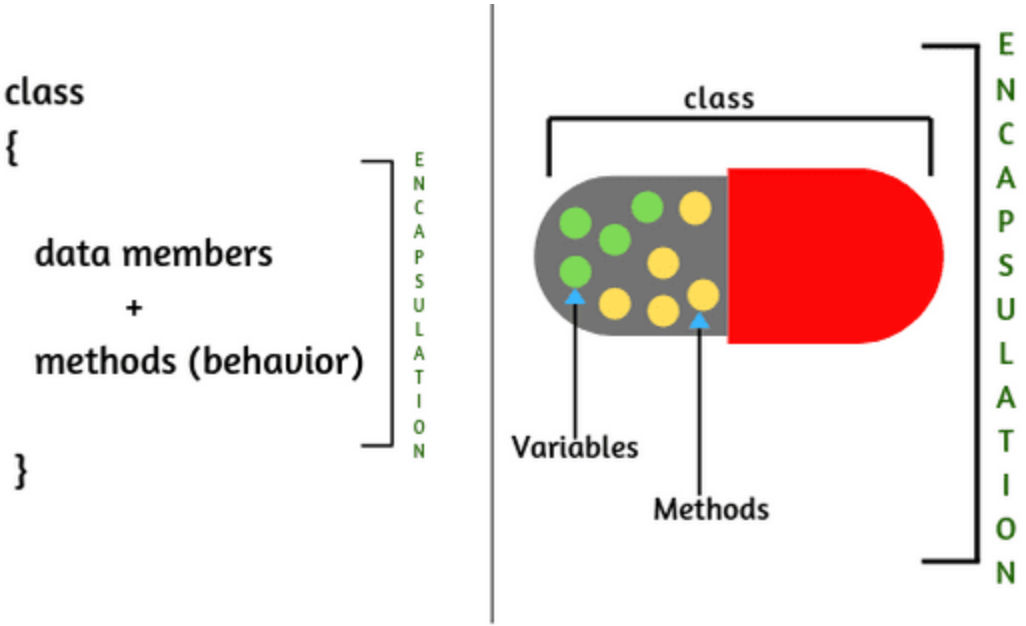
abstraction
Design that hides the details of how something works while still allowing the user to access complex functionality

Encapsulation

- Enclosing data by containing it within an object. In OOP, encapsulation forms a barrier around data to protect it from the rest of the code.

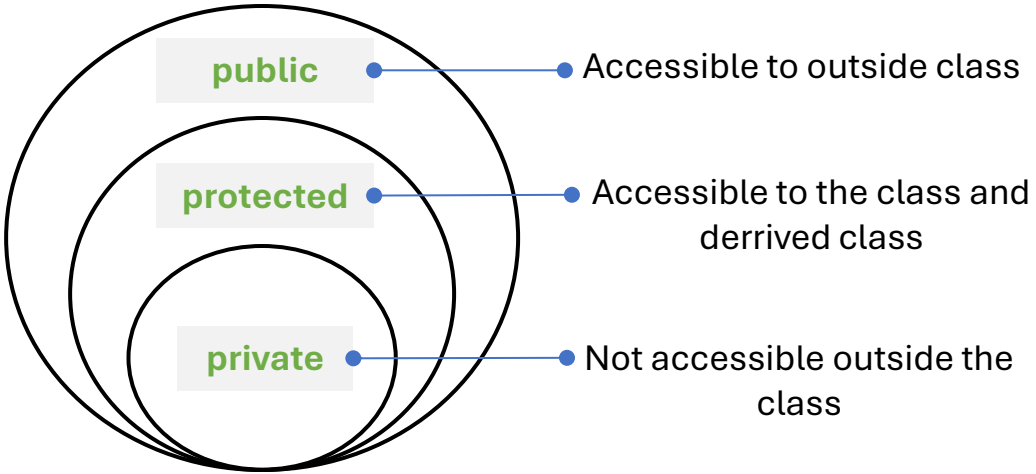
- Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods.

Encapsulation



Source: Internet

Encapsulation - How



Types of Encapsulation in OOP

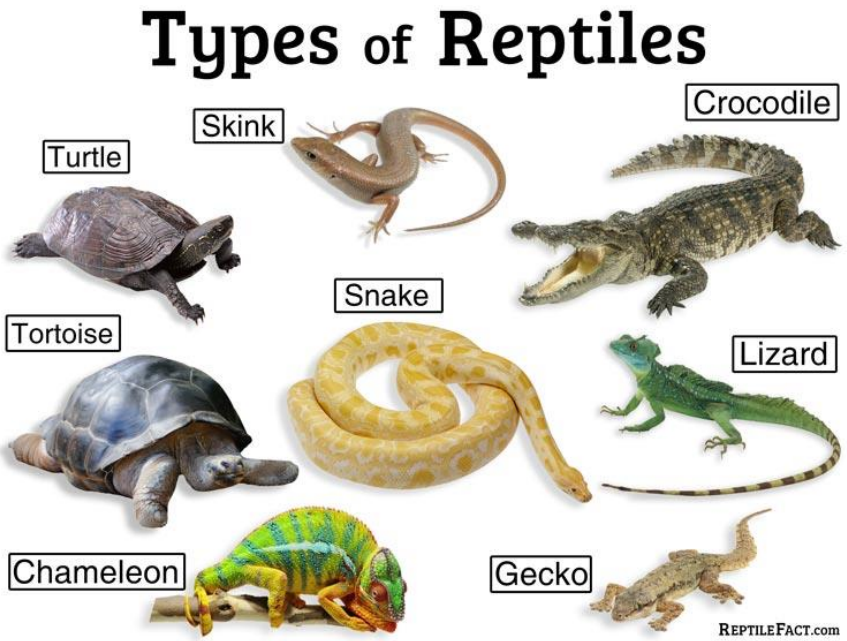
There are 3 basic techniques to implement encapsulation in OOP:

- Data member encapsulation
- Method encapsulation
- class encapsulation

Inheritance

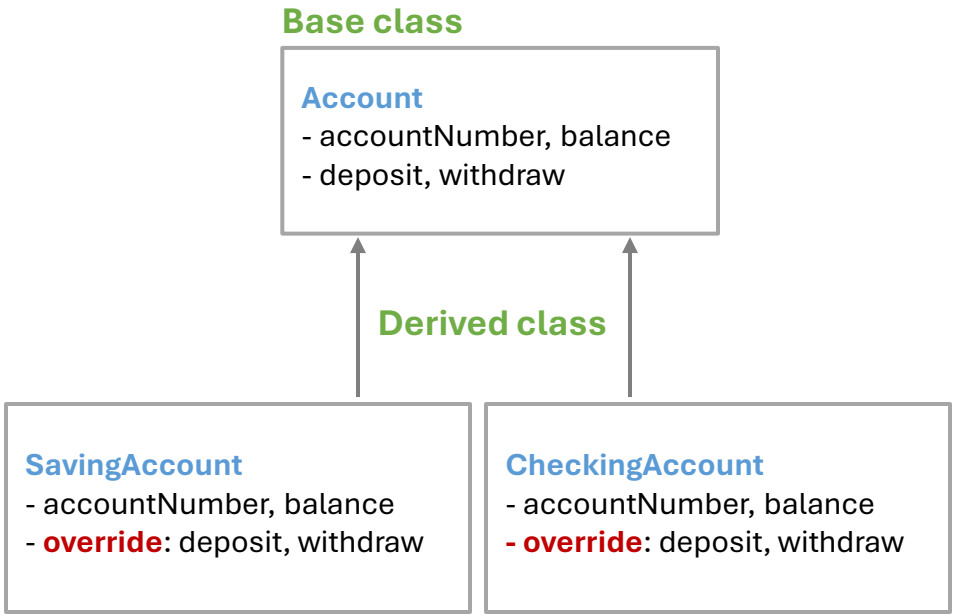
- Classes can reuse code and properties from other classes.
- If a class has a parent class, it means the class has inherited the properties of the parent. The child class can also modify or extend the behavior of its parent class. Inheritance allows you to reuse code without redefining the functions of a child class.
- Use inheritance when there is a clear "is-a" relationship between classes.

Inheritance



Snake is a reptile

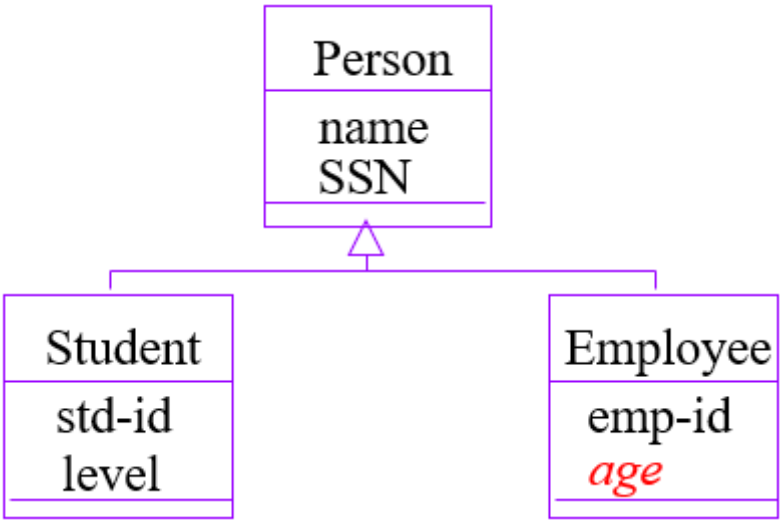
Inheritance



Inheritance

- Specialization: The act of defining one class as a refinement of another.
- Subclass: A class defined in terms of a specialization of a superclass using inheritance.
- Superclass: A class serving as a base for inheritance in a class hierarchy
- Inheritance: Automatic duplication of superclass attribute and behavior definitions in subclass

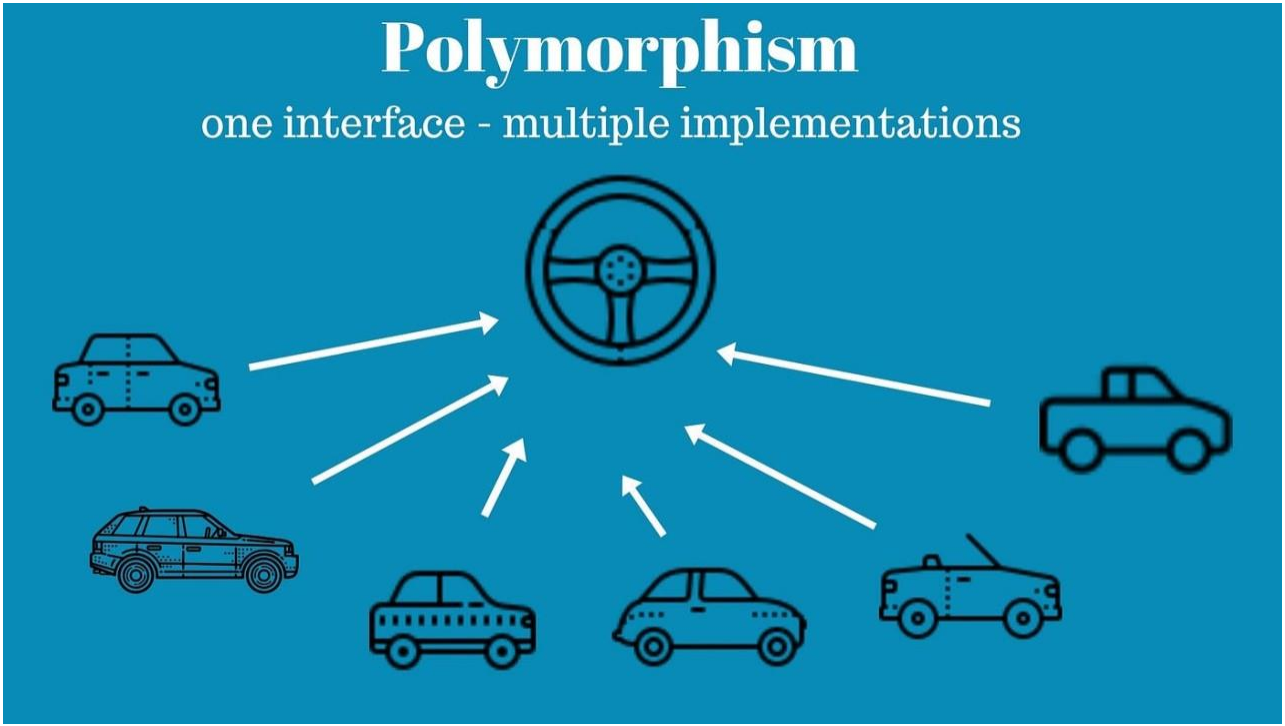
Inheritance



Polymorphism

- Objects are designed to share behaviors, and they can take on more than one form. A child class is then created, which extends the functionality of the parent class. Polymorphism enables different types of objects to pass through the same interface.

Polymorphism



Conclusion - Why OOP?

- **Modularity:** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
- **Reusability:** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
- **Productivity:** Programmers can construct new programs quickly through the use of multiple libraries and reusable code.
- **Easily upgradable and scalable:** Programmers can implement system functionalities independently.

Conclusion - Why OOP?

- **Interface descriptions.** Descriptions of external systems are simple, due to message-passing techniques that are used for object communication.
- **Security.** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and internet protocols are protected.
- **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

Conclusion - Why OOP?

- **Code maintenance.** Parts of a system can be updated and maintained without needing to make significant adjustments.
- **Lower cost.** Other benefits, such as its maintenance and reusability, reduce development costs.