CBS Department

OOP Principles: Inheritance and Polymorphism

Soma Soleiman Zadeh

Object-Oriented Programming (CBS 215)

Fall 2025 - 2026

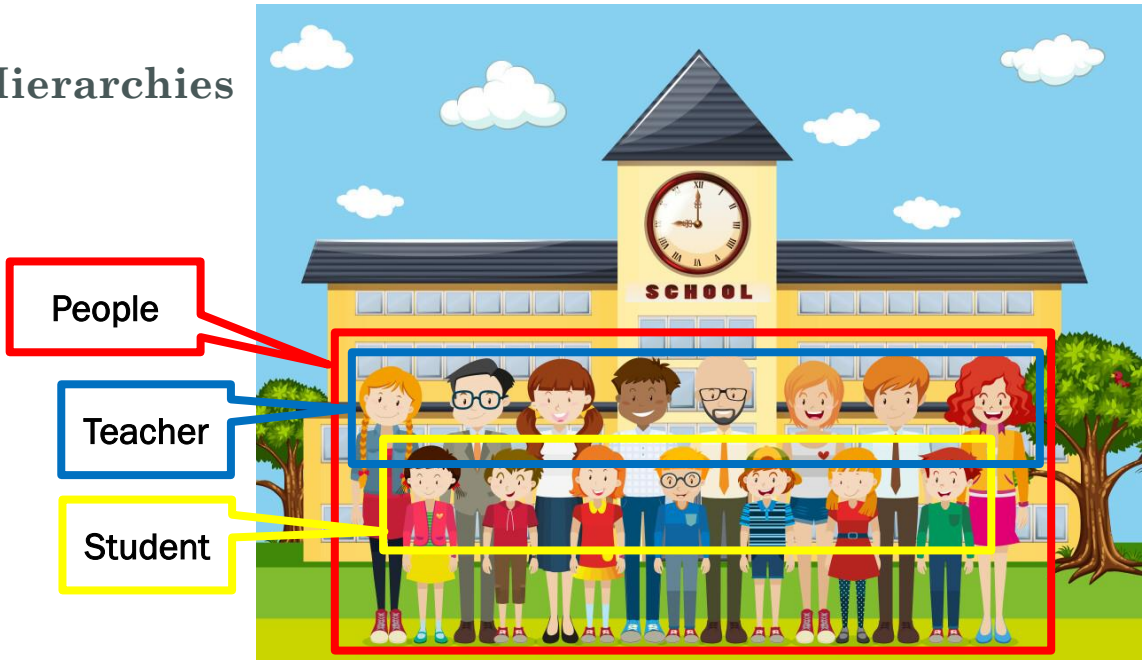
Week 9 – Week 10

December 3 – 10, 2025

Outline

- **OOP Concepts (**Encapsulation, Abstraction**, Inheritance, Polymorphism)**
- **Inheritance**
  - **Parent Class (Superclass)**
  - **Child Class (Subclass)**
- **Polymorphism**

Hierarchies



Necessity of Inheritance

- Sometimes you come across a situation where you have already defined a class, but then realize you need special behaviors in some, but not all, objects of the class.
- Then again, sometimes you realize you've defined two very similar classes with only minor differences.
- As programmers, we aim to always repeat ourselves as little as possible.
- So how can we have different implementations of similar objects?

Example of Similar Objects

```python
class Student:
    def __init__(self, name, id, email, credits):
        self.name = name
        self.id = id
        self.email = email
        self.credits = credits
```

```python
class Teacher:
    def __init__(self, name, email, room, teaching_years):
        self.name = name
        self.email = email
        self.room = room
        self.teaching_years = teaching_years
```

Example of Similar Objects

- Imagine the school's email address changed. All addresses would have to be updated.
- Two separate versions of the same function:

```python
def update_email_student(self):
    self.email = self.email.replace(".com", ".edu")
```

```python
def update_email_teacher(self):
    self.email = self.email.replace(".com", ".edu")
```

Solution?

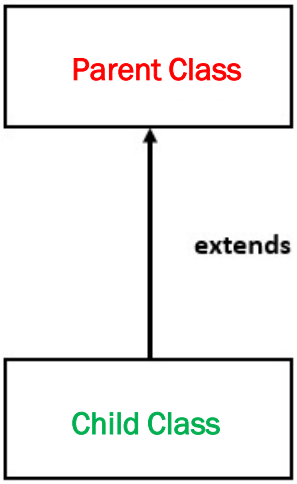
- Unnecessary Repetition of Attributes and Methods
- Combining all attributes and all methods of both classes in a **single class**.
  - **Is it a good solution?**

Is there a better solution?

Inheritance

- OOP languages usually have a technique called **inheritance**.
- A class can inherit the **attributes** and **methods** of another class.
- In addition to these inherited attributes and methods, a class can also contain attributes and methods that are unique to it.

Parent Class and Child Class



- Child class inherits all attributes and methods of the parent class.
  - Child class can add more attributes,
  - Child class can add more methods,
  - Child class can override methods of the parent class.

Person Class

```
class Person:
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def update_email(self,newDomain):
        oldDomain = self.email.split("@")[1]
        self.email = self.email.replace(oldDomain, newDomain)
```

Child Classes (Student and Teacher)

```
class Student(Person):
    def __init__(self, name, id, email, credits):
        Person.__init__(name, email)
        self.id = id
        self.credits = credits
```

```
class Teacher(Person):
    def __init__(self, name, email, room, teaching_years):
        Person.__init__(name, email)
        self.room = room
        self.teaching_years = teaching_years
```

Let's Create Objects and Try Accessing Attributes and Calling Methods

```
# Creating Student and Teacher Objects
s1 = Student('Amir', 'amir@yahoo.com',151024001,20)
t1 = Teacher('Kamal', 'kamal@icloud.com',328, 5)

# Accessing Attributes, Calling Methods
print("The old email address of ",s1.name, "is", s1.email)

s1.setEmail('gmail.com')

print("The new email address of ",s1.name, "is", s1.email)
```

Example

- Add a method to **Student** class to return student's stage based on their credits.

credits	stage
0 – 19	1
20 – 39	2
40 – 59	3
60 – 80	4

- Add a method to **Teacher** table that calculate and return the total additional payment to the teacher(***additional payment is $20 per each teaching year experience***).

Polymorphism

- **Polymorphism** is an OOP core component.
- **Polymorphism** means "multiple forms".
- **Polymorphism** → Using methods/functions/operators with the same name that can be executed on many objects or classes.

Function Polymorphism

- The same **function** can operate on different types of data and behave differently.
- Example: **len()** Function

```
l1 = len('Hello')

l2 = len([1,7,5])

l3 = len((9,6,3,7,1))
```

Operator Polymorphism

- The same operator can operate on different types of data and behave differently.

```
print(4 + 6)

print(2.6 + 4.1)

print([4,5] + [1,9])

print("Cybersecurity" + " " + "Department")
```

Class Polymorphism

- The **same method** in different classes has **different implementations**.
- **Class Polymorphism** can be used:
  - in parent class and child classes.
  - in different classes.

Method Overriding

- **Method overriding** in Python happens when a child class defines a method that has the **same name** and **parameters** as a method in its parent class.
- The **child class method overrides or replaces the parent class method** when called on an object of the child class.

Class Polymorphism

```python
class Animal:
    def speak(self):
        return "Generic Sound"

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"
```