### **Assignment 3**

### **1. Classes and Objects**

* **Definition:**A class is a blueprint for creating objects. Objects are instances of a class that hold data and behaviors defined by that class.
* **How it's used in the project:**
  + **Classes:**
    - Task: Represents a basic task with a title, description, and status.
    - PriorityTask: Inherits from Task and adds a priority attribute.
    - TaskList: Manages a collection of tasks, offering functionality to add, remove, list, search, and mark tasks as complete.
  + **Objects:**
    - In the main() function, an object of TaskList is created to store and manage all tasks.

### **2. Methods**

* **Definition:**Methods are functions defined within a class that describe the behaviors of an object.
* **How it's used in the project:**
  + Each class has its own methods.
  + For example, the Task class includes the mark\_complete() method to update a task's status.
  + The \_\_str\_\_() method in both Task and PriorityTask is used to provide a user-friendly string representation of the task.
  + The TaskList class has methods like add\_task(), add\_priority\_task(), remove\_task(), list\_tasks(), and find\_task\_by\_title() to manage tasks.

### **3. Method Overloading**

* **Definition:**Method overloading is the ability to have multiple ways to call a method with different parameters.
* **How it's used in the project:**
  + In the TaskList class, the add\_task() method uses a default parameter for description ("No description provided") so that you can call it with or without a description.

def add\_task(self, title, description="No description provided"):

new\_task = Task(title, description)

self.tasks.append(new\_task)

print("Task added successfully.")

### **4. Method Overriding**

* **Definition:**Method overriding allows a subclass to provide a specific implementation for a method that is already defined in its parent class.
* **How it's used in the project:**
  + The \_\_str\_\_() method is defined in the Task class and then overridden in the PriorityTask class to include the additional priority attribute.

def \_\_str\_\_(self):

return (f"Title: {self.\_title}\n"

f"Description: {self.\_description}\n"

f"Priority: {self.\_priority}\n"

f"Status: {self.\_status}")

### **5. Inheritance**

* **Definition:**Inheritance allows one class (child) to inherit attributes and methods from another class (parent).
* **How it's used in the project:**
  + The PriorityTask class inherits from the Task class, meaning it gets all the attributes and methods of Task but also adds its own attribute (\_priority).

class PriorityTask(Task):

def \_\_init\_\_(self, title, description, priority):

super().\_\_init\_\_(title, description)

self.\_priority = priority

### **6. Encapsulation**

* **Definition:**Encapsulation is the practice of hiding the internal state of an object and requiring all interaction to be performed through an object's methods.
* **How it's used in the project:**
  + Attributes like \_title, \_description, and \_status are prefixed with an underscore to indicate that they are intended for internal use within the class.
  + Users of the class interact with tasks through methods like mark\_complete() rather than directly modifying these attributes.

### 

### 

### **7. Polymorphism**

* **Definition:**Polymorphism allows objects of different classes to be treated as objects of a common superclass. In this project, it means that both Task and PriorityTask can be used interchangeably when managing tasks.
* **How it's used in the project:**
  + The list\_tasks() method in the TaskList class prints tasks using the \_\_str\_\_() method. Regardless of whether the task is a Task or a PriorityTask, calling print(task) will use the appropriate overridden \_\_str\_\_() method.

for task in self.tasks:

print(task)