Jelina Alexandrine Concha

CSE210 – Week 07

13 August, 2025

**Explain Programming with Classes**

# Abstraction

1. Briefly define the principle.

Abstraction is the concept of hiding complex implementation details and showing only the essential features of an object. It allows you to create a simple, high-level interface for a complex system.

1. How did you use that principle in one of your programs.

Like using a car, you interact with the steering wheel, brake, and gas pedal. You don't need to know how the engine or transmission works internally to drive it. The car's controls are an abstraction of its complex mechanical systems. Similar in programming, a class is an abstraction. It defines a set of public methods (the interface) that other parts of the program can use, without needing to know the complex internal logic or data structures hidden inside. This simplifies my code, makes it easier to use, and helps manage complexity.

1. How did using that principle help that program become more flexible for future changes?

Using abstraction helped my program become more flexible for future changes by decoupling the public interface from the implementation details. This means I can change how an object works internally without affecting any of the other code that uses that object.

## Encapsulation

1. Briefly define the principle.

Encapsulation is the principle of bundling an object's data (attributes) and the methods that operate on that data into a single unit, which is a class. The main idea is to hide the object's internal state from the outside world.

1. How did you use that principle in one of your programs.

Like a smartphone, all phones’ have complex internal parts— battery, processor, and circuits—they’re safely sealed inside its case. I can't directly manipulate them. Instead, I interact with the phone through its public interface (the screen and buttons). In programming, a class uses encapsulation to protect its data. By making data "private,"

I can only access or modify it through the class's public methods. This prevents other parts of the code from accidentally corrupting the data, making the program more robust and easier to maintain.

1. How did using that principle help that program become more flexible for future changes?

Encapsulation helped my program become more flexible for future changes by protecting an object's internal data and providing a controlled interface for interaction. This allows me to modify the internal workings of a class without having to rewrite any of the external code that uses it.

## Inheritance

1. Briefly define the principle.

Inheritance is a fundamental principle of object-oriented programming that allows a new class to inherit the attributes and methods of an existing class. This establishes a "is-a" relationship between them.

1. How did you use that principle in one of your programs.

Similar to a child inheriting traits from their parents, a Child class could inherit from a Parent class. The Parent class might have general attributes like eye color and methods like language(). The Child class would automatically get these features, so you wouldn't have to write them again. It could then add its own specific methods, like speech().

This process promotes code reuse, making your program more organized and easier to maintain. The new class (the child or subclass) can extend or specialize the existing class (the parent or superclass) without starting from scratch.

1. How did using that principle help that program become more flexible for future changes?

Inheritance made my program more flexible for future changes by allowing me to easily extend the codebase with new functionality and object types without altering the existing, proven code. This principle is a cornerstone of code reuse and organized structure.

## Polymorphism

1. Briefly define the principle.

Polymorphism is a core principle in object-oriented programming that means "many forms." It allows objects of different classes to be treated as objects of a common parent class. This enables a single method call to behave differently depending on the specific object it's called on.

1. How did you use that principle in one of your programs.

Let’s use an example of an Animal parent class with a speak() method. A Dog subclass might implement speak() to say "Woof!", while a Cat subclass might implement it to say "Meow!".

The power of polymorphism is that I can have a list of different animal objects and loop through them, calling the same speak() method on each one. The program automatically knows to call the specific speak() method for each animal type. This allows for clean, flexible, and scalable code that can handle new object types without requiring changes to the existing program logic.

1. How did using that principle help that program become more flexible for future changes?

Polymorphism made my program more flexible for future changes by allowing me to add new classes with their own unique behaviors without having to modify the existing code that interacts with them. This is a cornerstone of creating extensible and scalable software.