Classes and Encapsulation

Today's Objectives

- Classes
  - What are they?
  - How do we use them in Object Oriented Programming (OOP)?
- Proper class definition
- Create and call Constructors
- Access modifiers: public vs private
- Create an instance of a class
- Overloading, as it relates to classes

Three Fundamental Principles of OOP

- **Encapsulation**: the concept of hiding values or state of data within a class, limiting the points of access

- **Polymorphism**: the ability for our code to take on different forms

- **Inheritance**: the practice of creating a hierarchy for classes in which descendants obtain the attributes and behaviors from other classes

Benefits of OOP

- A natural way to express real-world objects in code
- Modular and reliable, allowing changes to be made in one part of the code without affecting another
- Discrete units of reusable code
- Units of code can communicate with each other by sending and receiving messages and processing data

Classes

- A **class** is a blueprint to create an object
    - Specifies **state**/variables
    - Defines **behavior**/methods
- Class Naming
    - Use singular nouns, not verbs
    - Class must match the file name
    - Use Pascal casing
    - A Fully Qualified Name is unambiguous and includes the package and class name

What is Pascal Case?

- A subset of Camel Case where the first letter is capitalized.
    - Camel Case: **`userAccount`**
    - Pascal Case: **`UserAccount`**

- Use Camel Case for variable names.

- Use Pascal case for Class names and Constructors.

Instance Variables

Instance variable represent the **properties** of a class.

- Each instance of a class will have its own instance variables that represent its internal **state**.

- Instance variables are declared with **access modifiers**.
    - **`public`**
        - Can be accessed by any other object.
    - **`private`**
        - Can only be accessed by the current instance of a class.

Encapsulation Using Instance Variables

Encapsulation is the concept of hiding data and controlling access to it.

- Letting other code modify data in an instance can be dangerous because it means an instance is not in control of its internal state.
- Hiding code implementation allows other classes to use a class without knowing anything about how it works
- By declaring instance variables **`private`**, we prevent other code from accessing instance variables directly.
- We use **getters** and **setters** to provide access to internal data to external code.

Goals of Encapsulation

Encapsulation is the concept of hiding data and controlling access to it.

- Encapsulation makes code extendable.
- Encapsulation makes code maintainable.
- Encapsulation promotes "loose coupling."
  - A **loosely coupled** system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components.