Chapter One: Introduction to Object-Oriented Concepts

Object oriented languages are defined by the following: **encapsulation**, **inheritance**, and **polymorphism**. Thus, if a language does not implement all of these, it is generally not considered completely object oriented.

In its basic definition, an **object** is an entity that contains both data and behavior.

The data stored within an object represents the state of the object. In object oriented programming terminology, this data is called **attributes**. Employee attributes could be Social Security numbers, date of birth, gender, phone number, and so on.

The **behavior** of an object represents what the object can do. In procedural languages, the behavior is defined by procedures, functions, and subroutines. In object oriented programming terminology, these behaviors are contained in **methods**, and you invoke a method by sending a message to it.

The concept of getters and setters supports the concept of data hiding. Because other objects should directly manipulate data within other object, the getters and setters provide controlled access to an object's data. Getters and setters are sometimes called accessor methods and mutator methods, respectively.

A **class** is a blueprint for an object.

An object cannot be instantiated without a class.

Classes can be thought of as the template for objects.

When a data type or method is defined as **public**, others objects can directly access it. When a data type of method is defined as **private**, only that specific object can access it

One of the primary advantages of using objects is that the objects need not reveal all its attributes and behaviors.

For data hiding to work, all attributes should be declared as private. Thus, attributes are never part of the interface. Only the public methods are part of the class interface. Declaring an attribute as public breaks the concept of data hiding.

Only the public attributes and methods are considered the interface. The user should not see any part of the internal implementation- interacting with an object solely through class interfaces. Thus, anything defined as private in inaccessible to the user and considered part of the class's internal implementation.

Inheritance allows a class to inherit the attributes and methods of another class. This allows creation of brand-new classes by abstracting out common attributes and behaviors.

The **superclass**, or **parent** class (sometimes called **base** class), contains all the attributes and behaviors that are common to

classes that inherit from it.

The **subclass**, or **child** class (sometimes called **derived** class), is an extension of the superclass.

In most recent object oriented languages (such as java, .NET, and Objective C), a class can only only a single parent class; however, a class can have any child classes. Some languages, such as C++, can have multiple parents. The former case is called **single-inheritance**, and the latter is called **multiple-inheritance**.

Polymorphism is a Greek word that literally means many shapes. Although polymorphism is tightly coupled to inheritance, it is often cited separately as one of the most powerful advantages to object-oriented technologies.

Overriding basically means replacing an implementation of a parent with one from a child.

A **stack** is a data structure that is a last-in, first out system. It is like a coin changer, where you insert coins at the top of the cylinder and, when need a coin, you take one off the top, which is the last one you inserted. **Pushing** an item onto the stack means that you are adding an item to the top. **Popping** an item off the stack means that you are taking the last item off the stack.

Objects are often built, or composed, from other objects: This is **composition**.

Conclusion:

- **Encapsulation**-- Encapsulating the data and behavior into a single object is of primary importance in Object Oriented development. A single object contains both its data and behavior and can hide what it wants from other objects.

- **Inheritance**-- A class can inherit from another class and take advantages of the attributes and methods defined by the superclass.

- **Polymorphism**-- Polymorphism means that similar objects can respond to the same message in different ways. For example, you might have an system with many shapes. However a circle, a square, and a star are each drawn differently: Using polymorphism, you can send each of these shapes the same message (for example, draw), and each shape is responsible for drawing itself.
- **Composition**-- Composition means that an object is built from other objects.