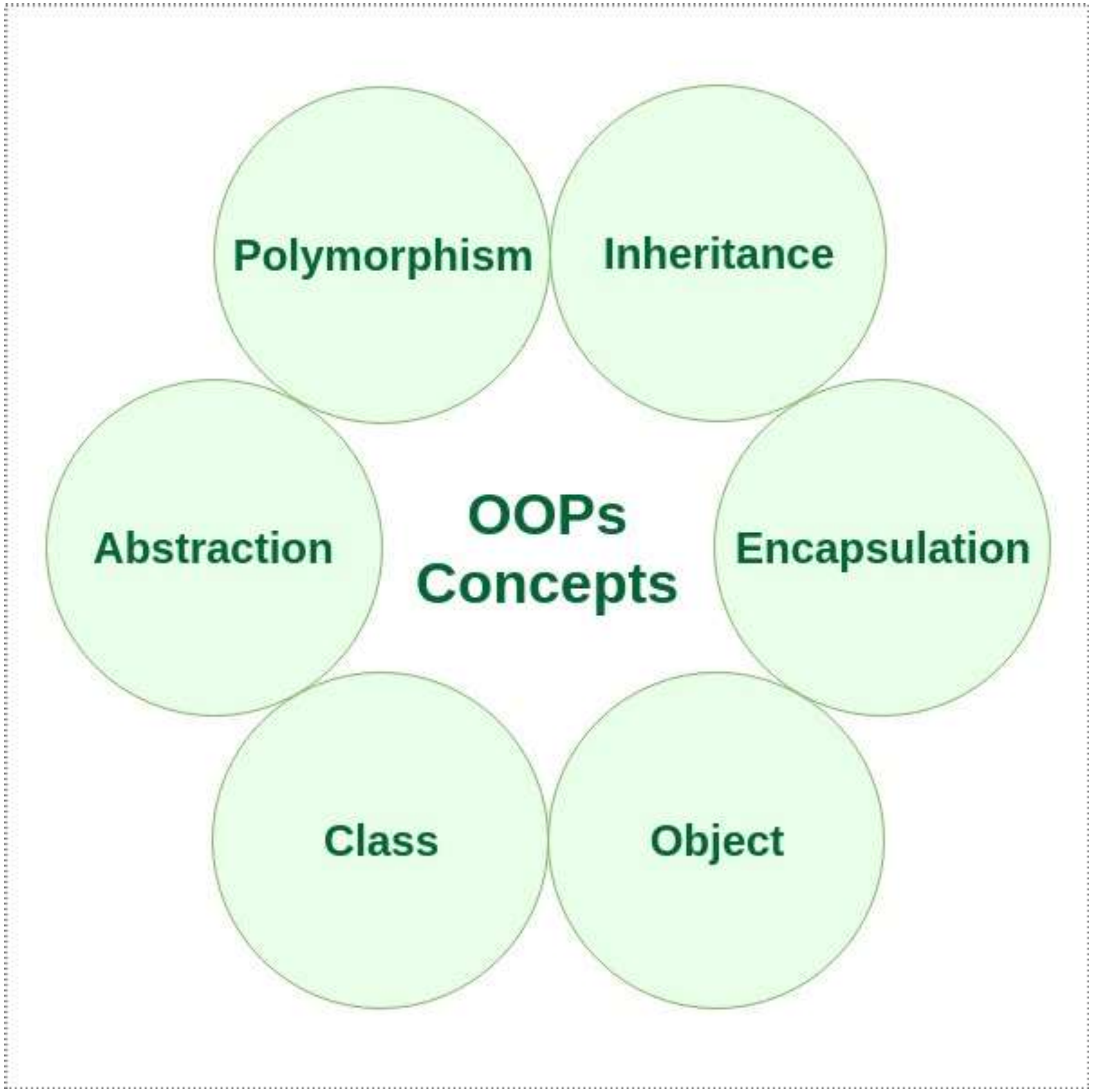
Object Oriented Programming

Course Instructor: Dr. R. Shathanaa

What is OOP?

- *Object-oriented programming (OOP)* is a programming paradigm that allows you to package together data states and functionality to modify those data states, while keeping the details hidden away. As a result, code with OOP design is flexible, modular, and abstract. This makes it particularly useful when you create larger programs.

- Let’s consider a physical object: a light bulb. A light (usually) has two possible states: on and off. It also has functionality that allows you to change its state: you can turn it on and you turn it off. Thankfully, you don’t need to know electrical engineering to use the light! You only need to know how to interact with it.


Polymorphism
Inheritance
Abstraction
OOPs Concepts
Encapsulation
Class
Object

Why So Many Languages?

- Bring the language “closer” to the problem.
- Many advanced languages are typically focused on specialized domains (e.g., relational databases).
- We want a language that is general purpose, yet can easily be “tailored” to any domain.

Object-Oriented Languages

- Smalltalk, C++, Java, etc…
- How different from procedural languages?
  - Not different at all: Every (reasonable) language is “Turing complete”
  - Very different: Make expression easier, less error-prone

O-O Languages

- Everything is an object.
- A program is a bunch of objects telling each other what to do, by sending messages.
- Each object has its own memory, and is made up of other objects.
- Every object has a type (class).
- All objects of the same type can receive the same messages.

Introduction to Object Technology

- Objects, or more precisely, the classes objects come from, are essentially reusable software components.
    - There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
    - Almost any noun can be reasonably represented as a software object in terms of attributes (e.g., name, color and size) and behaviors (e.g., calculating, moving and communicating).

Introduction to Object Technology (Cont.)

- The Automobile as an Object
  - Let’s begin with a simple analogy.
  - Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*.
  - Before you can drive a car, someone has to *design it*.
  - A car typically begins as engineering drawings, similar to the *blueprints that describe the design of a house*.
  - Drawings include the design for an accelerator pedal.
  - Pedal *hides from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel “hides” the mechanisms that turn the car.*

Introduction to Object Technology (Cont.)

- Enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
- Before you can drive a car, it must be *built* from the engineering drawings that describe it.
- A completed car has an *actual* accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must press the pedal to accelerate the car.

Introduction to Object Technology (Cont.)

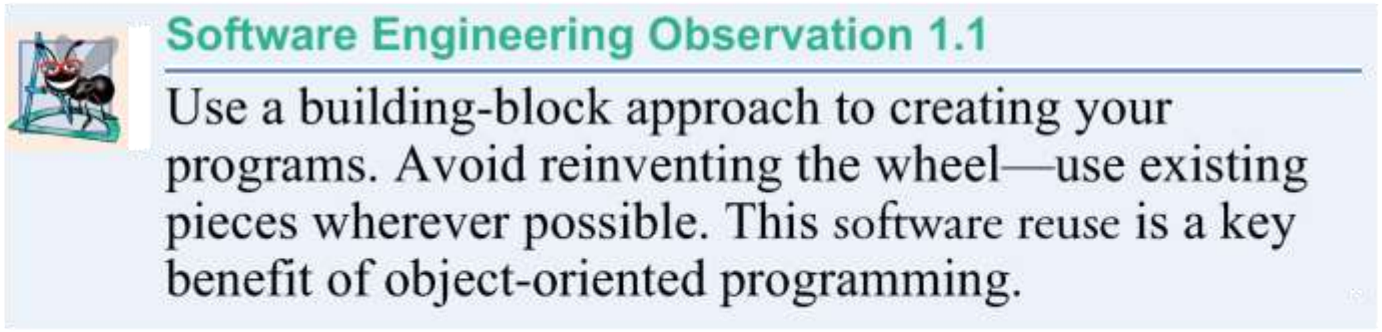
- Methods and Classes
  - Performing a task in a program requires a method.
  - The method houses the program statements that actually perform its tasks.
  - Hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster.
  - In Java, we create a program unit called a class to house the set of methods that perform the class's tasks.
  - A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

Introduction to Object Technology (Cont.)

- Instantiation
  - Just as someone has to build a car from its engineering drawings before you can actually drive a car, you must build an object of a class before a program can perform the tasks that the class's methods define.
  - An object is then referred to as an instance of its class.

Introduction to Object Technology (Cont.)

- Reuse
  - Just as a car's engineering drawings can be *reused* many times to build many cars, you can reuse a class many times to build many objects.
  - Reuse of existing classes when building new classes and programs saves time and effort.
  - Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive *testing, debugging and performance tuning*.
  - Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that has been spurred by object technology.



Software Engineering Observation 1.1

Use a building-block approach to creating your programs. Avoid reinventing the wheel—use existing pieces wherever possible. This software reuse is a key benefit of object-oriented programming.

Introduction to Object Technology (Cont.)

- Attributes and Instance Variables
  - A car has *attributes*
  - Color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
  - The car's attributes are represented as part of its design in its engineering diagrams.
  - Every car maintains its *own attributes.*
  - Each car knows how much gas is in its own gas tank, but *not* how much is in the tanks of other cars.

Introduction to Object Technology (Cont.)

- Messages and Methods Calls
  - When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster.
  - Similarly, you send messages to an object.
  - Each message is implemented as a method call that tells a method of the object to perform its task.

Introduction to Object Technology (Cont.)

- An object, has attributes that it carries along as it's used in a program.
- Specified as part of the object's class.
- A bank account object has a *balance* attribute that represents the amount of money in the account.
- Each bank account object knows the balance in the account it represents, but *not* the balances of the other accounts in the bank.
- Attributes are specified by the class's instance variables.

Introduction to Object Technology (Cont.)

- Encapsulation
  - Classes encapsulate (i.e., wrap) attributes and methods into objects—an object's attributes and methods are intimately related.
  - Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves.
  - Information hiding, as we'll see, is crucial to good software engineering.

Introduction to Object Technology (Cont.)

- Inheritance
  - A new class of objects can be created quickly and conveniently by inheritance—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own.
  - In our car analogy, an object of class "convertible" certainly *is an* object of the more general class "automobile," but more specifically, the roof can be raised or lowered.