# 01: Introduction to Object-oriented Programming

Programming Technique II

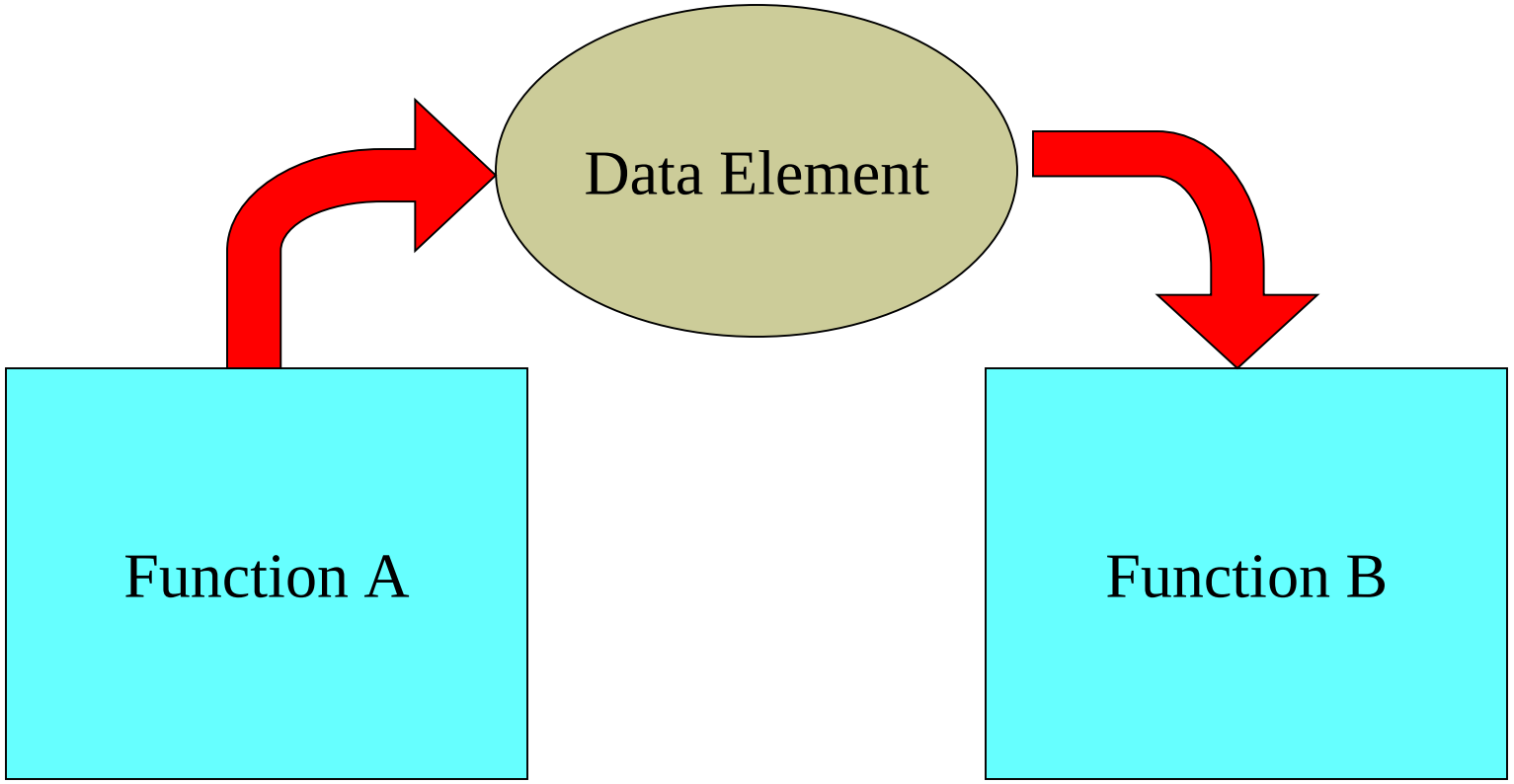
(SCSJ1023)

*Adapted from Tony Gaddis and Barret Krupnow (2016), Starting out with C++: From Control Structures through Objects*

# Procedural Programming

- Traditional programming languages were procedural.
  - C, Pascal, BASIC, Ada and COBOL
- Programming in procedural languages involves choosing data structures (appropriate ways to store data), designing algorithms, and translating algorithm into code.
- In procedural programming, data and operations on the data are separated.
- This methodology requires sending data to procedure/functions

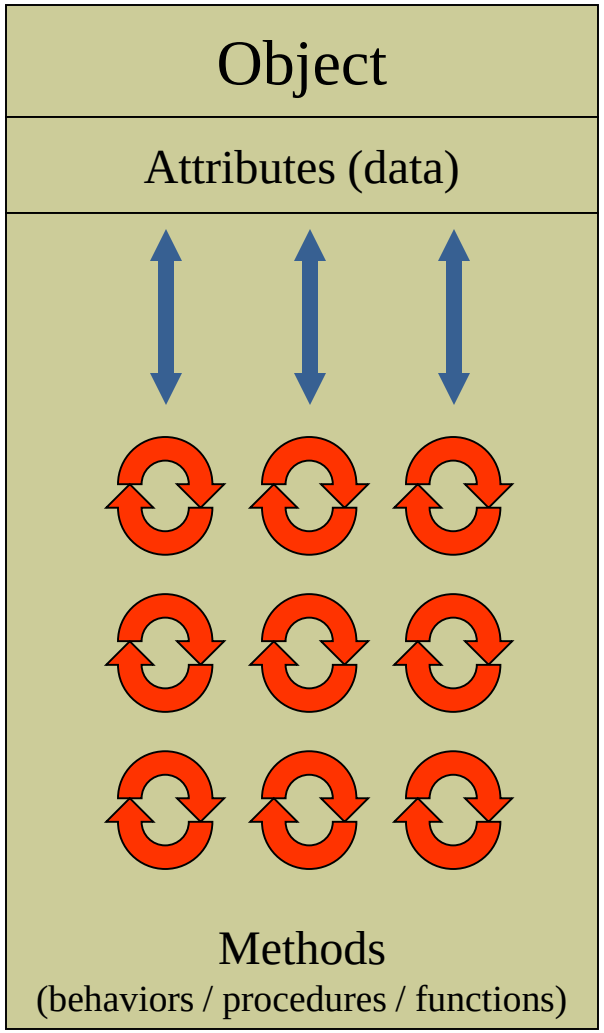
# Procedural Programming



# Object-Oriented Programming

- Object-oriented programming (OOP) is centered on **objects** rather than procedures / functions.
- Objects are a melding of **data and procedures** that manipulate that data.
- Data in an object are known as **properties** or **attributes** .
- Procedures/functions in an object are known as **methods**.

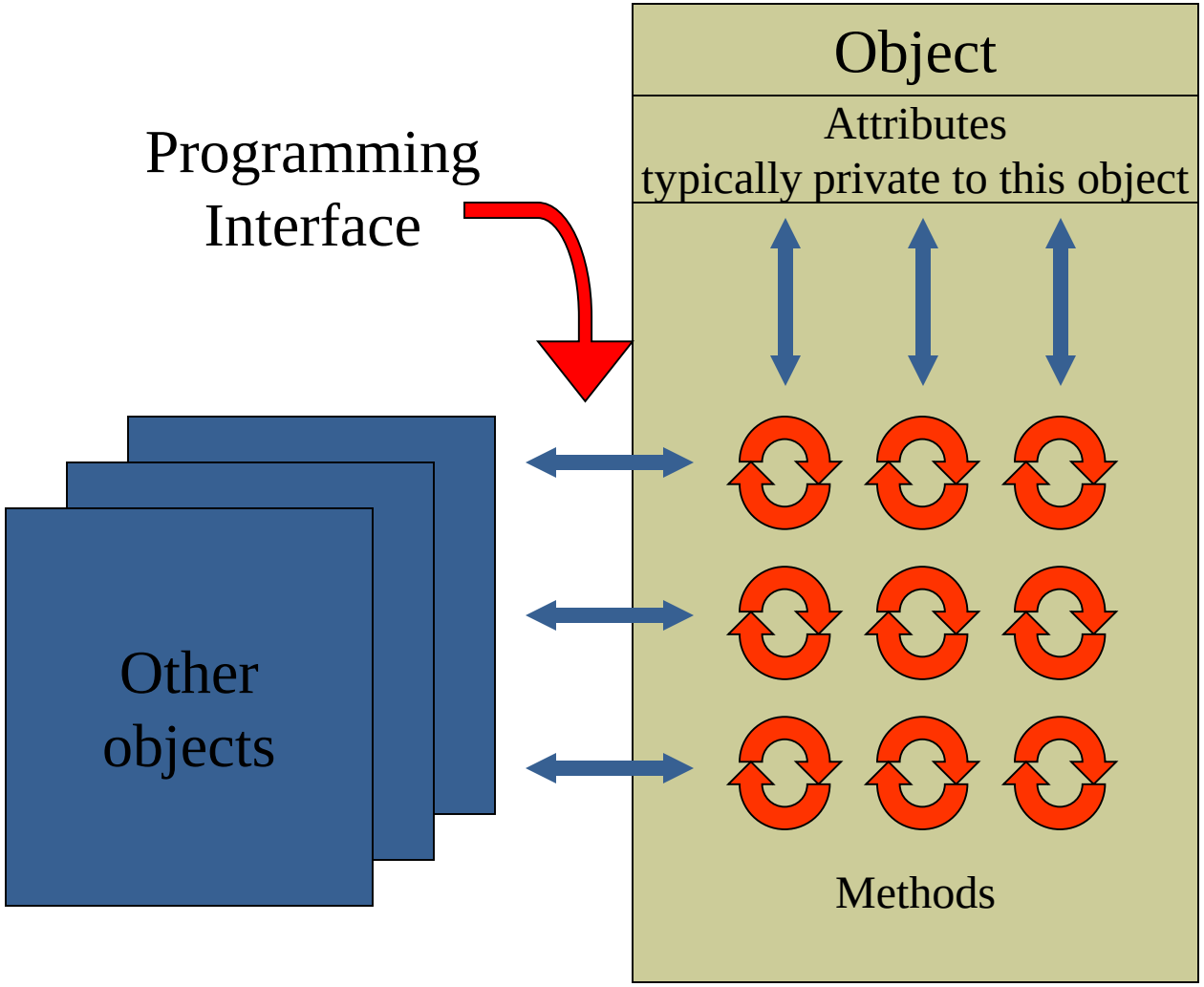
# Object-Oriented Programming



# Object-Oriented Programming

- Object-oriented programming combines data and methods via **encapsulation**.
- **Data hiding** is the ability of an object to hide data from other objects in the program
- Only object's methods should be able to directly manipulate its attributes
- Other objects are allowed to manipulate object's attributes via the object's methods.
- This indirect access is known as a **programming interface**

# Object-Oriented Programming



# Object-Oriented Programming Languages

- Pure OO Languages
  - Smalltalk, Eiffel, Actor, Java

- Hybrid OO Languages
  - C++, Objective-C, Object-Pascal

# OOP Principles: Classes

- A class is the **template** or mould or blueprint from which objects are actually made.

- A class **encapsulates** the attributes and actions that characterizes a certain type of object.

# OOP Principles: Objects

- Classes can be used to **instantiate** as many objects as are needed.
- Each object that is created from a class is called an **instance** of the class.
- A program is simply a collection of objects that interact with each other to accomplish a goal.

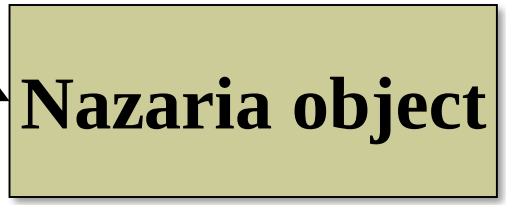
# Classes and Objects

The *Car* class defines the attributes and methods that will exist in all objects that are instances of the class.



The Kancil object is an instance of the Car class.

The Nazaria object is an instance of the Car class.



# OOP Principles: Encapsulation

- **Encapsulation** is a key concept in working with objects: **Combining attributes and methods** in one package and hiding the implementation of the data from the user of the object.

***Encapsulation:***

Attributes/data
+
Methods/functions = Class

**Example:**
a car has attributes and methods below.

| Car |
| --- |
| *Attributes:*<br>model,<br>cylinder capacity |
| *Methods:*<br>move,<br>accelerate |

# OOP Principles: Data Hiding

- **Data hiding** ensures methods **should not directly access** instance attributes in a class other than their own.

- Programs should interact with object attributes only through the object's methods.

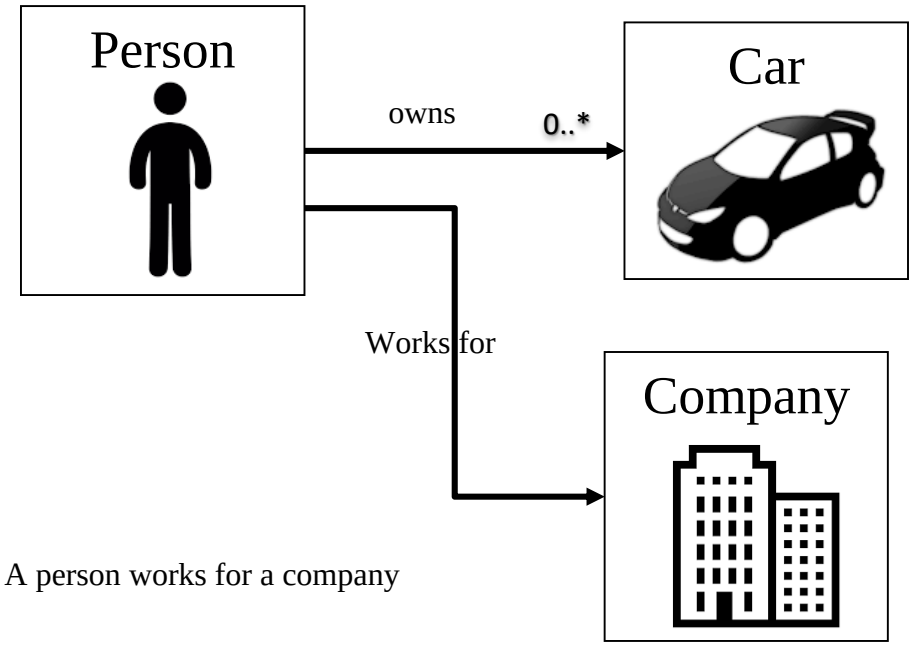
- Data hiding is important for several reasons.
  - It protects of attributes from accidental corruption by outside objects.
  - It hides the details of how an object works, so the programmer can concentrate on using it.
  - It allows the maintainer of the object to have the ability to modify the internal functioning of the object without “breaking” someone else's code.

# OOP Principles: Associations

- **Association**: relates classes to each other through their objects.
- Association can be, one to one, one to many, many to one, or many to many relationships.

**Example:**

A person can own several cars



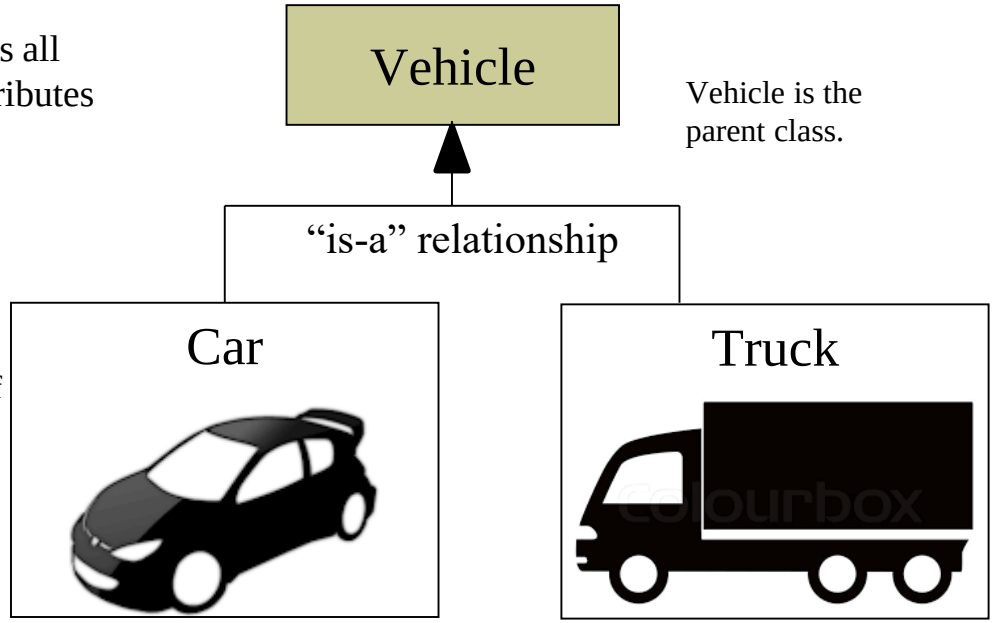

A person works for a company

# OOP Principles: Inheritance

- Inheritance is the ability of one class to **extend** the capabilities of another.
  - it allows code defined in one class to be reused in other classes

**Example:**

Vehicle represents all of the generic attributes and methods of a vehicle.



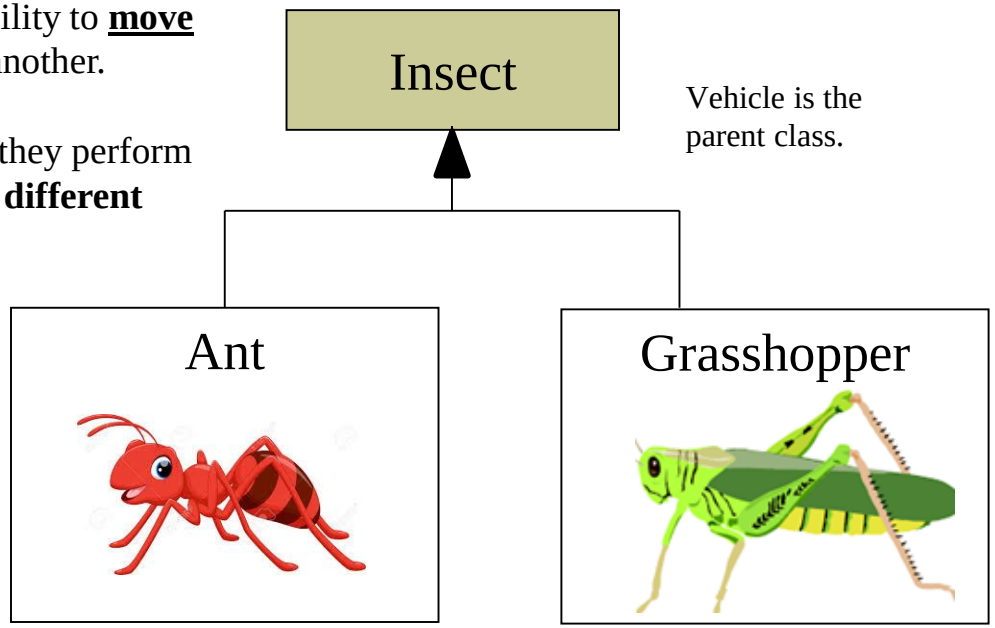
# OOP Principles: Polymorphism

- Polymorphism is the ability of objects **performing the same actions differently**.

**Example:**

Insects have the ability to **move** from one point to another.

However, the way they perform their **movement is different**



# Self-test: Introduction to Object Oriented Programming

- State the differences between procedural programming and Object Oriented Programming.
- What is an Object and what is a Class? What is the difference between them?
- What is an Attribute?
- What is a Method?
- What is encapsulation? How it relates to data hiding?
- What is association?
- What is inheritance? How it relates to polymorphism?

# The Unified Modeling Language

Programming Technique II

(SCSJ1023)

*Adapted from Tony Gaddis and Barret Krupnow (2016), Starting out with C++: From Control Structures through Objects*

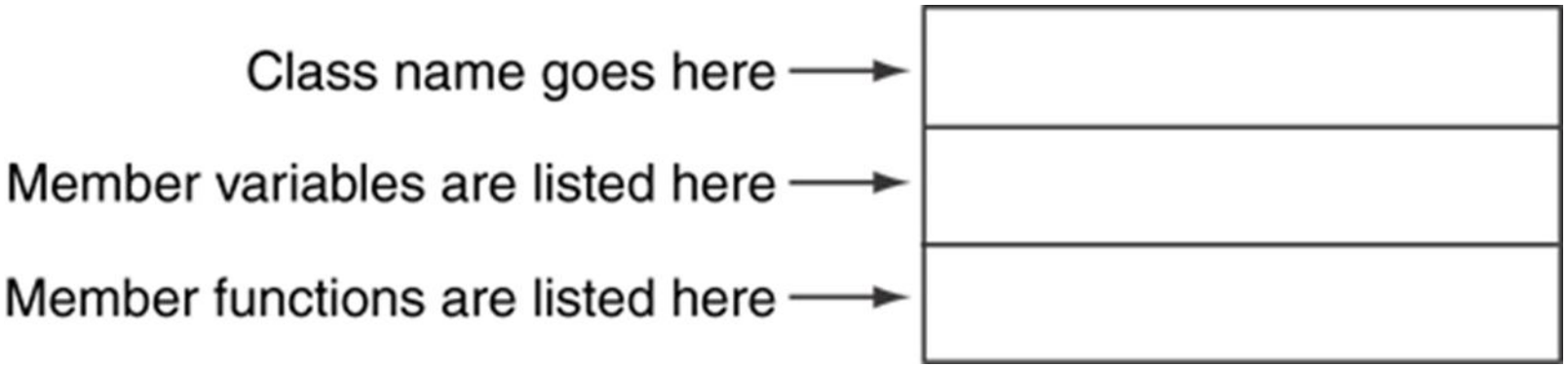
# The Unified Modeling Language

- UML stands for Unified Modelling Language.

- The UML provides a set of standard diagrams for graphically depicting object-oriented systems

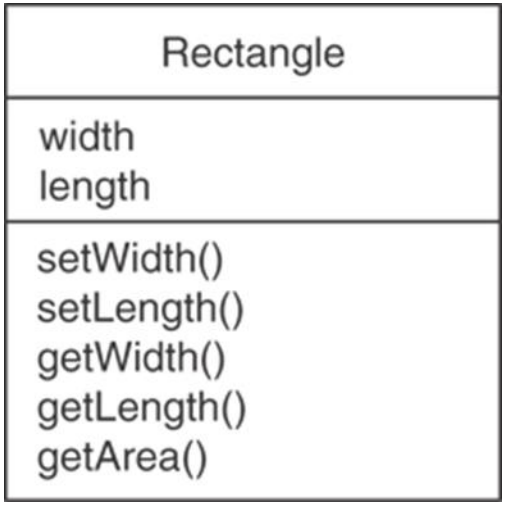
# UML Class Diagram

- A UML diagram for a class has three main sections.



# Example: A Rectangle Class

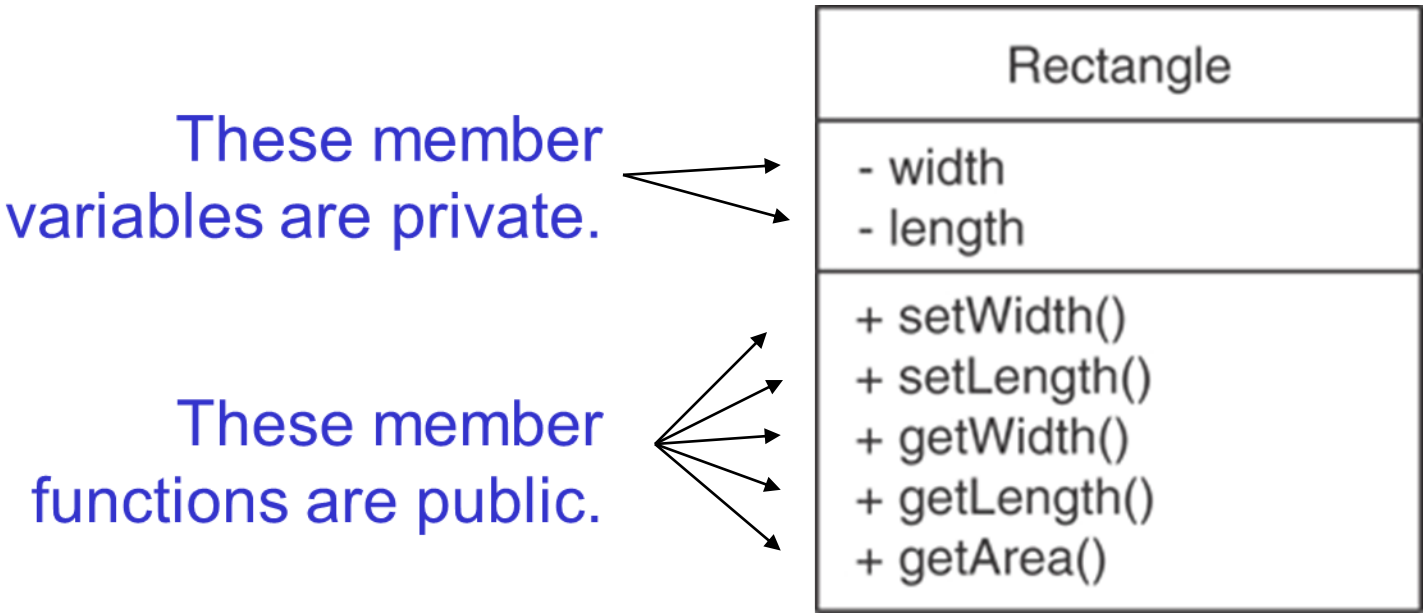
- A UML diagram for a class has three main sections.




```
class Rectangle
{
   private:
      double width;
      double length;
   public:
      bool setWidth(double);
      bool setLength(double);
      double getWidth() const;
      double getLength() const;
      double getArea() const;
};
```

# UML Access Specification Notation

- In UML you indicate a private member with a minus (-) and a public member with a plus(+).



# UML Data Type Notation

- To indicate the data type of a member variable, place a colon followed by the name of the data type after the name of the variable.

```
- width : double
- length : double
```

# UML Parameter Type Notation

- To indicate the data type of a function's parameter variable, place a colon followed by the name of the data type after the name of the variable.

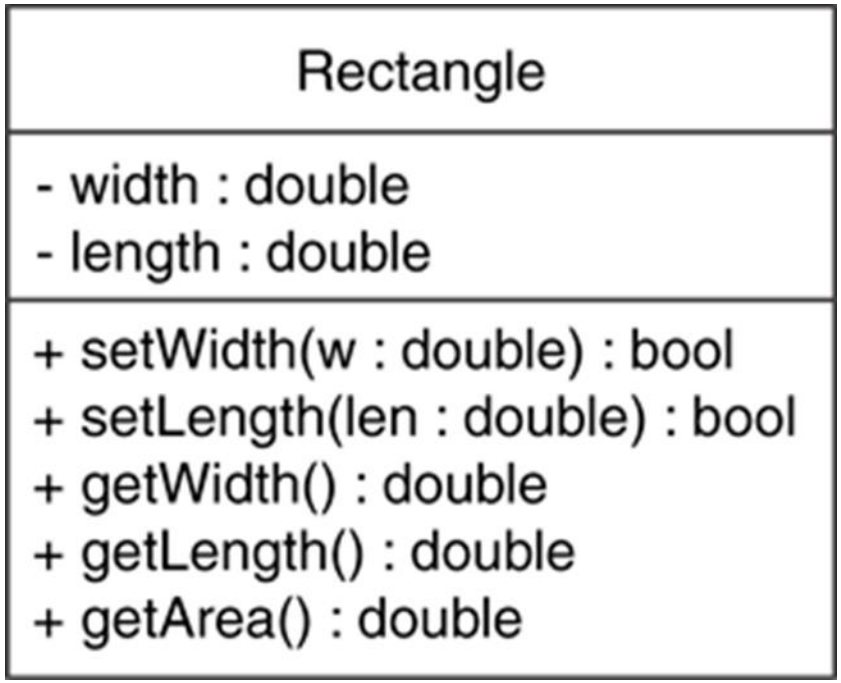
```
+ setWidth(w : double)
```

# UML Function Return Type Notation

- To indicate the data type of a function's return value, place a colon followed by the name of the data type after the function's parameter list.

```
+ setWidth(w : double) : void
```

# The Rectangle Class



# Showing Constructors and Destructors

*No return type listed for constructors or destructors*