# Class Terminology

## Class Overview

**Abstract Data Type:** Abbreviated as ADT. A type where the users do not have access to the ways detailing how the operations (member functions) and values (variables) are implemented.

**Encapsulation:** A fundamental of Object-Oriented Programming. Taking variables and functions and putting them all into one container (into an object of a class). This is used to hide the value or state of the class. Therefore, this restricts direct access to some of an object's components.

**Inheritance:** A fundamental of Object-Oriented Programming. When a class (child class) is constructed from another class (parent class). The child class takes behaviors and attributes from the parent class and uses polymorphism.

**Polymorphism:** A fundamental of Object-Oriented Programming. The ability to have one function with different meanings when using inheritance. The function will have the same name, formal parameters, and return type but will have different implementations in the parent class and all of the child classes. Pure virtual functions will not have an implementation in the parent class.

## Class Types

**Class:** A data type that the user defines to contain various pieces of data and member functions and whose variables are called objects or instances.

**Abstract Class:** Class with at least one pure virtual function. This class cannot have any objects associated with it.

**Parent Class:** Also known as a base class or a superclass. The class that declares virtual functions and member variables for its own class as well as child classes. This class also declares pure virtual functions, but those are for child classes to implement. Note, the term parent will be used in the **class** section of this guide.

**Child Class:** Also known as a derived class or a subclass. The class that is made from a parent class, inherits from the parent class, and implements virtual/pure virtual functions. It may also have its own member functions and member variables. Note, the term child will be used in the **class** section of this guide.

## Class Sections

**Interface:** Informs the user on how to use the class (ADT).

**Implementation:** The aspect of the class where the member functions are defined and this aspect includes the private section of the class. The user does not need to know anything about this portion of the code.

**Public:** Constructors, destructor, member functions, and overloaded operators that are available for objects to utilize in any function of the program.

**Private:** Member variables and helper functions that cannot be accessed directly outside of a class. They are accessed only via member functions of the class.

**Protected:** Member functions, overloaded operators, and member variables that are visible to the class they are in as well as its derived classes. Used during inheritance. All other areas of the program will have the protected section act as a private section of the class (cannot access directly).

## Class Functions

**Behavior:** What the objects do and are defined by the member functions. These member functions may alter the state of the object or simply access the member variables.

**Member Function/Method:** A function within a class. This function is usually implemented as **public** and performs operations on the member variables as well as performs operations on the objects in general. Note, the term member function will be used in the **class** section of this guide.

**Accessor Function:** A member function that accesses member variables.

**Mutator Function:** A member function that changes the state (value) of member variables.

**Friend Function:** A function that is not a member function of a class (it is a global function). This function, however, has access to the private section of the class.

**Overloaded Operator:** A usage of an operator, such as **+ * - << >> = == < [] ()** That allows (an) object(s) to be utilized/manipulated as you would a primitive base type, such as **int**, **char**, or **string**.

**Helper Function:** A member function that is **private** to aid in the behavior and state of a class that is not invoked using an object in other functions of the program. This function is only used in other member functions of the class. This function may be also known as a private member function.

**Constant Parameter:** The keyword **const** that is put at the end of a member function to signify that member function will not alter the contents of the class.

**Scope Resolution Operator:** :: operator used after a class name during the definition of a member/helper function.

**Type Qualifier:** The class name that comes before the scope resolution operator.

## Class Data

**Member Variables/Attributes:** The variables within a class. These variables are implemented as **protected** or **private** and can only be accessed through member variables. Note, the term member variable will be used in the **class** section of this guide.

**State:** The condition of the member variables (what the member variables currently are, or assigned to).

## Class Constructors

**Constructor:** A member function that is invoked when you declare an object of a class. Used to initialize the member variables of the object. This member function has to be the same name as the class with no return type.

**Copy Constructor:** A member function that is invoked when you declare an object of a class, but this member function takes an argument of another object of the same class. The member variables of the argument object will then be copied over to the newly declared object.

**Initializer List:** Also known as initialization list. Replaces the initialization within curly brackets. It is the section after the constructor name and starts with a : (colon). Each member variable (in order) is listed and separated by a comma and the default value is put inside parenthesis that append each member variable. This method is also able to be used in the copy constructor where the argument variables are used in place of default values.

**Destructor:** A member function for the deallocation of dynamically allocated data.

## Class Objects

**Object/Instance:** The variable of a class. Each object contains unique data separate from other objects but utilizes the same member functions and member variables. Note, the term object will be used in the **class** section.

**Dot Operator:** Used after an object to invoke member functions or member variables for said object.

## Class Inheritance

**Redefining Function:** A function that is declared and defined in a parent class that may also be declared and defined in its child classes. The functions are all the same in terms of formal parameters, return type, and function name. However, you cannot utilize polymorphism using this method.

**Virtual Function:** A function that is declared and defined in a parent class with type **virtual** that may also be declared and defined in its child classes. The functions are all the same in terms of formal parameters, return type, and function name. Must use a virtual function to take part in the aspects of polymorphism.

**Pure Virtual Function:** A function that is declared but not defined in a parent class with type **virtual** that is declared and defined in every child class. The function in the parent class ends with a **=** 0.

**Overriding:** The act of changing a virtual function definition in a child class versus using the definition in the parent class. May use the keyword **override** at the end of the function in the child class as a safety check to make sure the compiler knows it is a function that is supposed to be taking place of the parent's virtual function.

**Late Binding:** Run-time implementation to figure out which virtual function to use. Also known as dynamic binding.

**Slicing Problem:** Act of assigning a child class object to a parent class object which slices off the specific member functions and member variables of said child class.