- Background:
  - When creating a user-defined data type using C++ **`class`**, one would typically consider providing 4 "generic" categories of supporting functions:
    - Constructors, destructor and assignment operator.
      - Big 4 (every **`class`** must have): *default constructor* + "gang of 3" (*copy constructor*, *destructor*, and *assignment operator*)
        - To support default construction, copy construction, copy assignment and destruction of object.
      - Other ("non-special") constructors where appropriate, such as conversion constructors.
    - Accessors (each has **`const`** appearing immediately after closing parenthesis).
      - Can only inspect but *not* change the state (values of data members) of the invoking object.
      - The **`const`** protects ONLY *the invoking object* and NOT *any other object*.
    - Mutators.
      - Can inspect and change the state (values of data members) of the invoking object.
    - Helpers
      - For use by developer (when implementing other member functions), not for client.
      - Typically made **`private`** (not **`public`**).
- C++ support for supporting functions:
  - When planning a supporting function, one may have several design options, including:
    - Use of member function (method).
    - Use of non-member, non-**`friend`** function.
    - Use of non-member, **`friend`** function.
      - Should use only sparingly and where warranted (don't want to *break encapsulation* unless unavoidable).
    - Use of operator: need to know how to do *operator overloading* and may have several options to overload operator, including:
      - Use of member function.
      - Use of non-member, non-**`friend`** function.
      - Use of non-member, **`friend`** function.
  - When implementing a binary operation using member function, the *invoking object is the LHS operand*.
  - When implementing a binary operation using non-member function, the *first parameter is the LHS operand*.
- More on operator overloading:
  - Operators provided automatically for use with objects: **`sizeof`**, address-of (**`&`**), dot (**`.`**), assignment (**`=`**).
    - Automatic assignment operator does only *shallow copying* and thus may not be adequate.
  - Couple of exceptions:
    - Assignment operator (**`=`**) can only be overloaded using *member* function.
    - Stream input (extraction) and output (insertion) operators (**`>>`** and **`<<`**) can only be overloaded using *non-member* functions (unless the stream libraries are modified).
  - When overloading a binary operator using member function, the *invoking object is the LHS operand*.
  - When overloading a binary operator using non-member function, the *first parameter is the LHS operand*.

- Background:
  - Every C++ **`class`** must have the "Big 4":
    - *default constructor* + "gang of 3" (*copy constructor*, *destructor*, and *assignment operator*)
  - If user doesn't provide custom "Big 4", automatic versions are supplied by the compiler: *automatic default constructor*, *automatic copy constructor*, *automatic destructor*, and *automatic assignment operator*.
    - Compiler *won't* provide *automatic default constructor* if user decides to write one or more constructors (but doesn't cover the default constructor).
      - Will flag a "class does not have a default constructor" compilation error.

      NOTE: A constructor that has parameters but has *default value* specified for *each and every one* of the parameters (through *function-with-default-arguments* feature) will cover the default constructor.
    - Compiler will always provide the *automatic copy constructor* if user doesn't write custom version for it.
    - Compiler will always provide the *automatic destructor* if user doesn't write custom version for it.
    - Compiler will always provide the *automatic assignment operator* if user doesn't write custom version for it.
  - Automatic (compiler-supplied) versions of "Big 4" have limitations:
    - *Automatic default constructor* will do nothing (besides calling the default constructors of **`class`** data members that have default constructors).
    - *Automatic copy constructor* will do only *shallow* (memberwise, bit-by-bit) *copying*.
    - *Automatic destructor* will do nothing (besides calling the destructors of **`class`** data members).
    - *Automatic assignment operator* will do only *shallow* (memberwise, bit-by-bit) *copying*.
- For any **`class`**, one would usually want to write the *default constructor* so an object of the **`class`** constructed using the default constructor will be in a *consistent state* right after construction.
- For a simple **`class`**, *i.e.*, one that *doesn't* involve *acquisition of non-automatically-managed resources* (such as *dynamic memory allocated at run time*), the automatic (compiler-supplied) versions of "gang of 3" are usually adequate.
- For a more involved **`class`**, *i.e.*, one that *does* involve *acquisition of non-automatically-managed resources* (such as *dynamic memory allocated at run time*), the automatic (compiler-supplied) versions of "gang of 3" are usually *not* adequate and the user will usually have to write custom versions for them.
  - Implementation templates for the "gang of 3" for a **`class`** named **`someClass`**:

copy constructor (invoking object is the object being constructed)	`someClass::someClass(const someClass& src)`
	`// add/use initializer list wherever possible`
	`{`
	`  // do deep copying of src (to invoking object)`
	`}`

destructor (invoking object is the object being destroyed)	`someClass::~someClass()`
	`{`
	`  // free non-automatically-managed resources`
	`}`

assignment operator (invoking object is the left-hand-side object)	`someClass& someClass::operator=(const someClass& rhs)`
	`{`
	`  if (this != &rhs) // if not self-assignment`
	`  {`
	`  // do deep copying of rhs (to invoking object)`
	`  }`
	`  return *this;`
	`}`