10; Classes and Data Abstraction

Gael Zarco

September 03, 2025

Object-Oriented Design (OOD): Problem solving methodoloy

Object: Combines data and the operations on that data in a single unit

Class: A collection of a fixed number of components

Member: A component of a class

Classes

Three Categories of class members:

- `private` *(default)*

    - Member cannot be accesed outsite the class

- `public`

    - Member is accessible outside the class

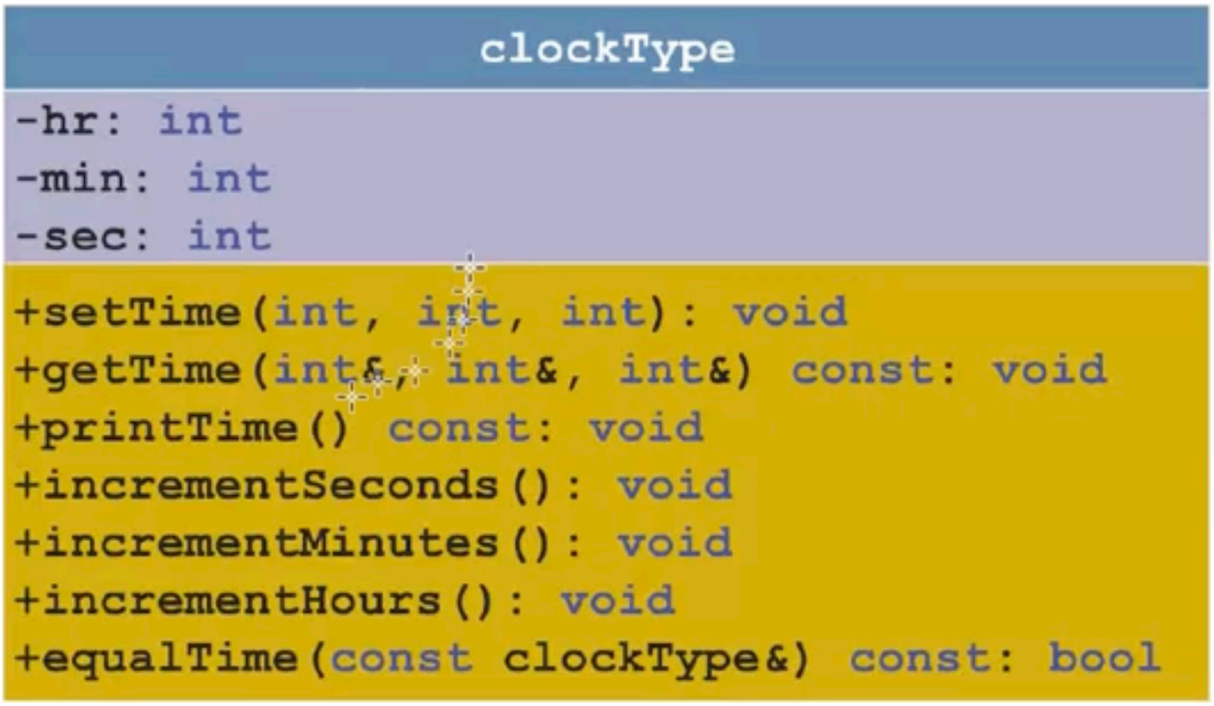
- `protected`

    - Member is accessible within the class and all its subclasses

Unified Modeling Language Class Diagrams (UML)

UML Notation: used to graphically describe a class and its members

- `+` member is `public`
- `-` member is `private`
- `#` member is `protected`



Accessing Class Members

Once an object is declared, it can access the members of the class.

Syntax:

```
classObjectName.memberName
```

- The dot (`.`) is the **Member Access Operator**
- Member functions of a class have access to all members (`public`, `protected`, and `private`) of the same class, irrespective of where an object of the class is

Built-in Operations on Classes

- Arithmetic operators can **NOT** be used on class objects unless the operators are overloaded.
- Relational operators can **NOT** be used to compare two class objects for equality.

Built-in operations that are valid for class objects:

- Member access (`.`)
- Assignment (=)

`class` Scope

A class object can be automatic or static.

Automatic: created when the declaration is reached and destroyed when the surrounding block is exited.

Static: created when the declaration is reached and destroyed when the program terminates.

Reference Params and `class` Objects

Try to pass by reference. Avoid passing by value (creates copy).

Operators

Scope Resolution Operator: `::` used to access functions in a `class`.

Member Access Operator: `.` used to access members in a `class`,

Accessor and Mutator Functions

Accessor Function: Member function that only accesses the value(s) of member variable(s).

Mutator Function: Member function that modifies the value(s) of member variable(s).

Constant Function:
- Member function that cannot modify member variables of that class.
- Member function heading with `const` at the end.

Order of `public` and `private` Members of a Class

No defined order, however follow the book's convention of `public` first.

Constructors

Use constructors to guarantee that member variables are initialized.

Two types:
- Without params (`default` constructors)
- With params

Other properties:
- Name of constructor is the same as the class.
- A constructor has no type.

`default` Constructor Example:

```
class StudentType {
  public:
```

```
    StudentType();                          // default constructor
    std::string getName() const;
    void setName(std::string name);

  private:
    std::string name;
    std::string id;
    std::string phone;
}

int main() {
  StudentType stu1;
}

// Default constructor implementation
StudentType::StudentType() {
  name = "";
  id = "000";
  phone = "702-000-0000";
}
```

Parameterized Constructor Example:

```
class StudentType {
  public:
    StudentType(std::string name, std::string id, std::string phone);
    std::string getName() const;
    void setName(std::string name);

  private:
    std::string name;
    std::string id;
    std::string phone;
}

int main() {
  StudentType stu1("Gael", "5006289777", "702-426-8371");
}
```

Parameterized Constructor with Default Arguments Example:

```
class StudentType {
  public:
    StudentType(std::string name = "", std::string id = "000", std::string
phone =
    "702-000-0000") {
      name = name;
      id = id;
      phone = phone;
    }
    std::string getName() const;
    void setName(std::string name);
```

```
  private:
    std::string name;
    std::string id;
    std::string phone;
}

int main() {
  StudentType stu1("Gael", "5006289777", "702-426-8371");
}
```

Classes **CAN** have more than 1 constructor.
- Each must have a different formal parameter list (function signature).

Costructors execute automatically when a class object enters its scope.
- They cannot be called like other functions.
- Which constructor executes depends of the types of values passed to the `class` object when the `class` object is declared.

Constructor Precautions

C++ provides a default constructor if a class does not have one.
- Likely to be uninitialized if in-line initialization is not used.

HOWEVER, if a class includes constructor(s) with param(s), but not a `default`:
- C++ does **NOT** provide the `default`.
- Appropriate args must be included when obj is declared.

*If you define any constructor, you **must always** define a `default` constructor.*

In-Line Initialization of Data Members and the `default` constructor

C++14 standard allows member initializations in class declarations.
- Called in-line initialization of the data members.

When an object is declared without params, then the object is initialized with the in-line initialized values.
- If declared with params, then the default vals are overriden

Example:

```
class ClockType {
  public:
    // ...
  private:
    int hr = 0;
    int min = 0;
    int sec = 0;
}
```

Member-Initialized Lists

Used to initialize class member variables when a constructor is invoked.
- Use it in the constructor definition.

Syntax:

```
className(params): member1(value1), member2(value2) {
  // Constructor body...
}
```

Example:

```
StudentType::StudentType(): name(""), id("000"), phone("702-000-0000") {

}
```

Member variables are initialized before the constructor body executes.

Useful for:
- Initializing `const` or `reference` members.
- Efficiently intialzing non-default-constructible objects.

Example:

```
class clockType {
  public:
    clockType(int hour, int min, int sec)
    : hr(hour), m(min), s(sec){}
  private:
    int hr;
    int m;
    int s;
}
```

Arrays of Class Objects (Variables) and Constructors

If you declare an arr of class obects, the class should have a default constructor.
- Typically used to intialize each (array) class object.
- Classes should **ALWAYS** have a default constructor.

If a class has a user-defined constructor, the default constructor is not implicitly declared.

Destructors

Destructors are functions without any types.
- A class can have only one destructor (has no params).
- The name of the destructor is `~className`.
- Automatically executes when the class object goes out of scope.

- Should **NEVER** be invoked directly.

Data Abstraction, Classes, and Abstract Data Types

Abstraction: Separating design details from usage
- Separates logical properties from the implementation details.

Abstract Data Type (ADT): Data type that separates the logical properties from the implementation details.

Three things associated with an ADT:

1. **Type Name**: The name of the ADT
2. **Domain**: The set of values belonging to the ADT.
3. Set of **Operations** on the data.

`struct` VS `class`

`struct`	`class`
members are `public` by default	members are `private` by default

Both have the same capabilities.