Object-oriented programming CS10003

Lecturer: Do Nguyen Kha

Contents

C++ Move Constructor

A move constructor enables the resources owned by an right value object to be moved into an left value without copying.

C++ Move Constructor

```
class MemoryBlock {
private:
  size_t _length; // The length of the resource.
  int* _data; // The resource.
public:
  // Simple constructor that initializes the resource.
  MemoryBlock(size_t length);
  // Copy constructor.
  MemoryBlock(const MemoryBlock& other);
  // Copy assignment operator.
  MemoryBlock& operator=(const MemoryBlock& other);
  // Retrieves the length of the data resource.
  size_t Length() const;
  // Destructor.
  ~MemoryBlock();
};
```

C++ Move Constructor

```
MemoryBlock(MemoryBlock&& other) : _data(nullptr), _length(0) {
  _data = other._data;
  _length = other._length;
  other._data = nullptr;
  other._length = 0;
}
```

C++ Move Constructor

If no user-defined move constructors are provided for a class type, and all of the following is true:

- there are no user-declared copy constructors;
- there are no user-declared copy assignment operators;
- there are no user-declared move assignment operators;
- there is no user-declared destructor.
- Then the compiler will declare a move constructor as a non-explicit inline public member of its class with the signature `T::T(T&&)`.

C++ Move Constructor

The move constructor for class `T` is *trivial* if **ALL** of the following is true:

- it is not user-provided (meaning, it is implicitly-defined or defaulted);
- `T` has no virtual member functions;
- `T` has no virtual base classes;
- the move constructor selected for every direct base of `T` is trivial;
- the move constructor selected for every non-static class type (or array of class type) member of `T` is trivial.

C++ Move Assignment Operator

Move assignment operators typically “steal” the resources held by the argument (e.g. pointers to dynamically-allocated objects, file descriptors, TCP sockets, I/O streams, running threads, etc.), rather than make copies of them, and leave the argument in some valid but otherwise indeterminate state.

C++ Move Assignment Operator

```
MemoryBlock& operator=(MemoryBlock&& other) {
  if (this != &other) {
    delete[] _data;
    // Copy the data pointer and its length from the source object.
    _data = other._data;
    _length = other._length;
    // Release the data pointer from the source object so that
    // the destructor does not free the memory multiple times.
    other._data = nullptr;
    other._length = 0;
  }
}
```

C++ Move Assignment Operator in Move Constructor

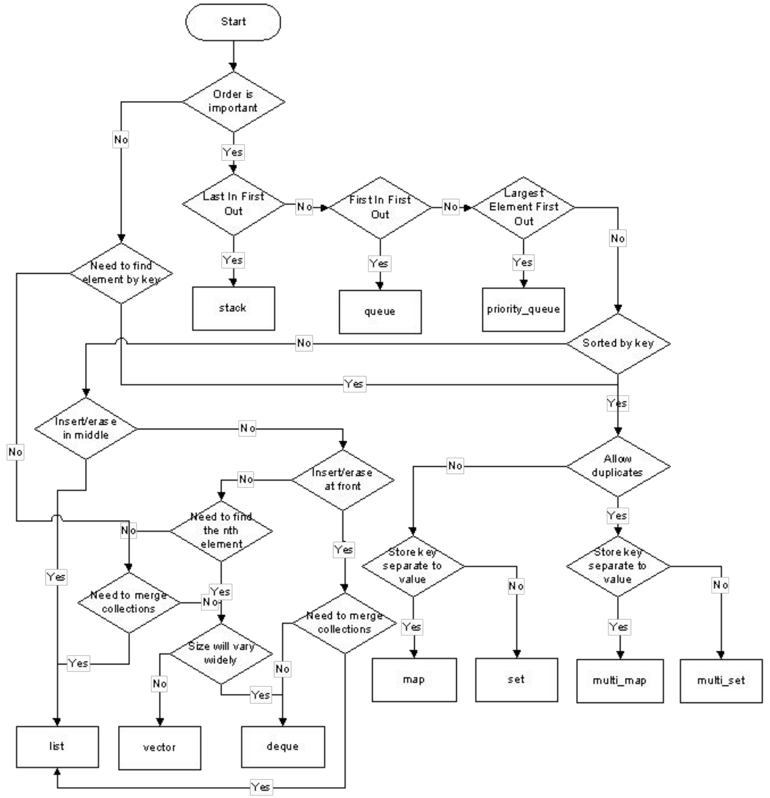
```
// Move constructor.
MemoryBlock(MemoryBlock&& other) : _data(nullptr), _length(0) {
  *this = std::move(other);
}
```

https://learn.microsoft.com/en-us/cpp/cpp/move-constructors-and-move-assignment-operators-cpp?view=msvc-170

C++ STL Containers

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.

C++ STL Containers



https://stackoverflow.com/questions/471432/in-which-scenario-do-i-use-a-particular-stl-container

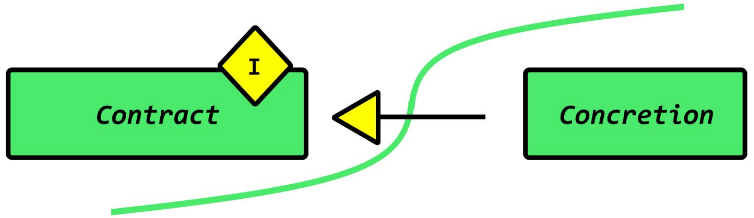
Unicode

```
#include <iostream>
#include<cwchar>
using namespace std;
int main() {
  // wide-char type array string
  wchar_t waname[] = L"geeksforgeeks" ;
  wcout << L"The length of '" << waname
        << L"' is " << wcslen(waname) << endl;
  return 0;
}
```

Unicode

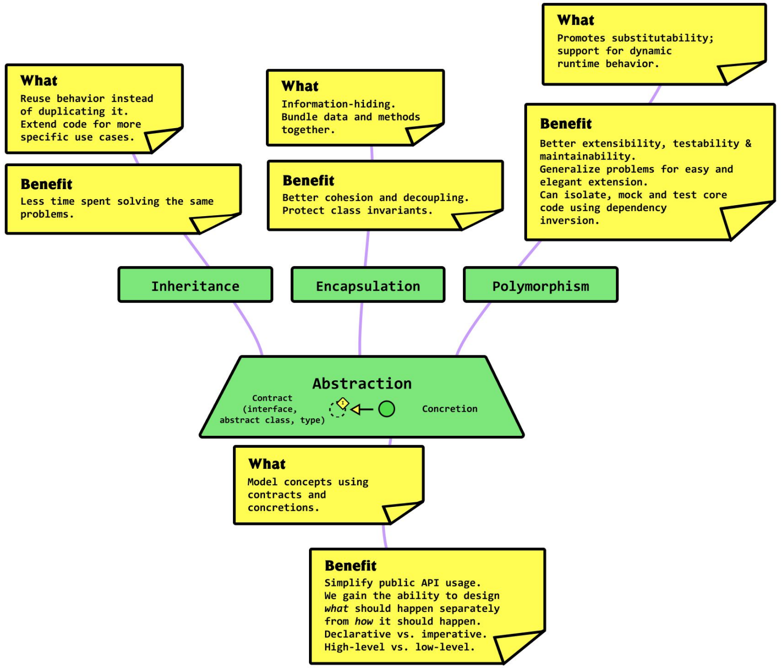
```
#include <iostream>
#include <string>
int main() {
  std::wstring wstr;
  std::wcout << L"Enter a wide string: ";
  std::wcin  >> wstr;
  std::wcout << L"Your wide string is: '" << wstr << L"'\n";
  return 0;
}
```

Contract (interface) vs Concretion



https://khalilstemmler.com/articles/object-oriented/programming/4-principles/

4 Principles



What
Reuse behavior instead of duplicating it. Extend code for more specific use cases.

Benefit
Less time spent solving the same problems.

What
Information-hiding. Bundle data and methods together.

Benefit
Better cohesion and decoupling. Protect class invariants.

What
Promotes substitutability; support for dynamic runtime behavior.

Benefit
Better extensibility, testability & maintainability. Generalize problems for easy and elegant extension. Can isolate, mock and test core code using dependency inversion.

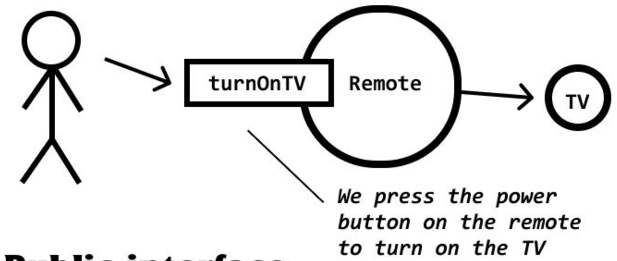
What
Model concepts using contracts and concretions.

Benefit
Simplify public API usage. We gain the ability to design *what* should happen separately from *how* it should happen. Declarative vs. imperative. High-level vs. low-level.

https://khalilstemmler.com/articles/object-oriented/programming/4-principles/

Abstraction

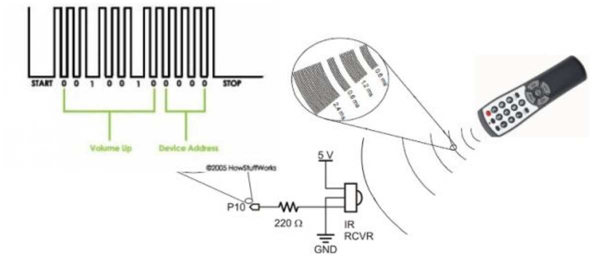
Abstraction example: Using a remote control to turn on a TV



We press the power button on the remote to turn on the TV

Public interface

Public
Declarative
Human-centered
Highest level of abstraction

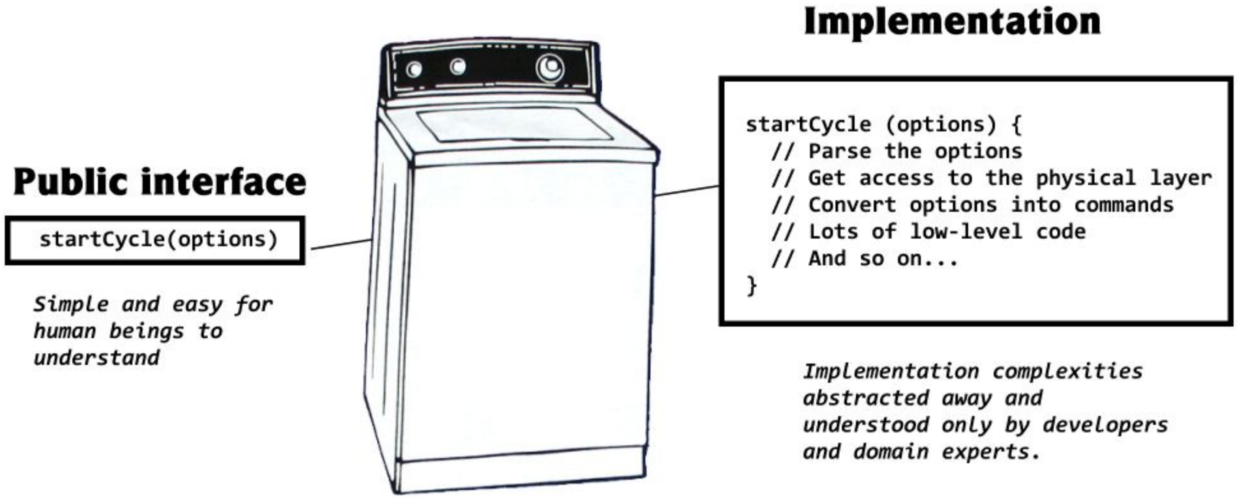


Implementation

Private
Imperative
More technical
Lower levels of abstraction

https://khalilstemmler.com/articles/object-oriented/programming/4-principles/

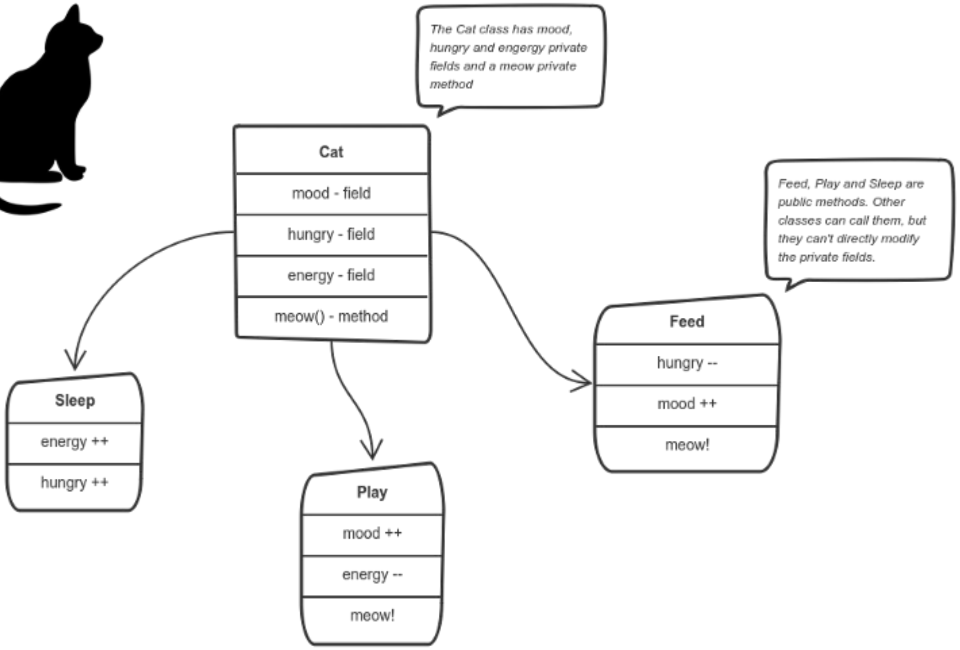
Abstraction


Public interface
startCycle(options)
Simple and easy for human beings to understand
Implementation
Implementation complexities abstracted away and understood only by developers and domain experts.

```
startCycle (options) {
  // Parse the options
  // Get access to the physical layer
  // Convert options into commands
  // Lots of low-level code
  // And so on...
}
```

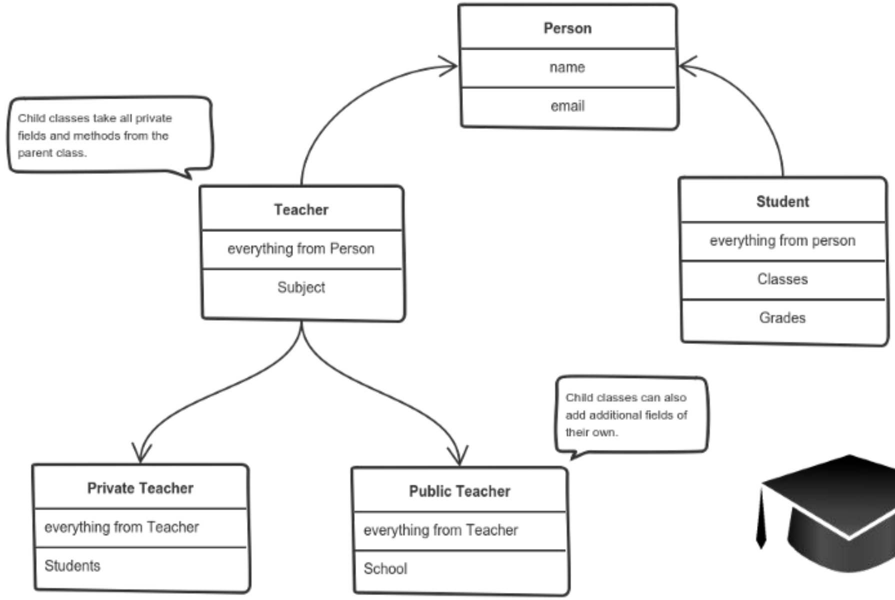
https://khalilstemmler.com/articles/object-oriented/programming/4-principles/

Encapsulation


The Cat class has mood, hungry and engergy private fields and a meow private method
Cat
mood - field
hungry - field
energy - field
meow() - method
Feed, Play and Sleep are public methods. Other classes can call them, but they can't directly modify the private fields.
Sleep
energy ++
hungry ++
Play
mood ++
energy --
meow!
Feed
hungry --
mood ++
meow!

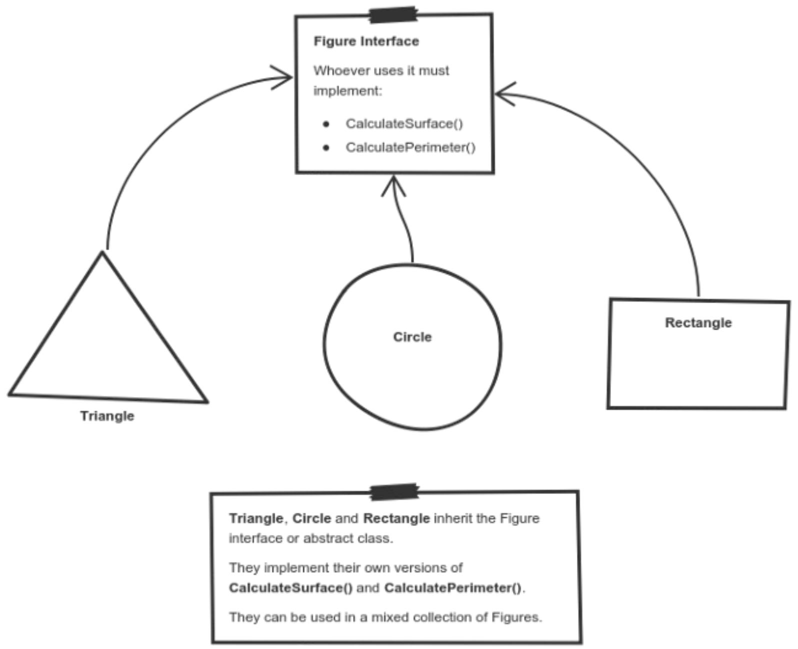
https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/

Inheritance



https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/

Polymorphism



Triangle, **Circle** and **Rectangle** inherit the Figure interface or abstract class.

They implement their own versions of **CalculateSurface()** and **CalculatePerimeter()**.

They can be used in a mixed collection of Figures.

https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/

Team Project: Milestone 3 (deadline 01/01/2024)

- Implement `path`
- Implement `linearGradient`
- Implement `radialGradient` is optional due lack of support of graphics libary, but fallback to solid (average) color or simulate using `linearGradient`
- Implement `viewBox`
- Post showcase on Facebook prior to deadline
- Test cases:
  - Apple: https://upload.wikimedia.org/wikipedia/commons/f/fa/Apple_logo_black.svg
  - Chrome: https://www.google.com/chrome/static/images/chrome-logo.svg
  - Firefox: https://upload.wikimedia.org/wikipedia/commons/a/a0/Firefox_logo%2C_2019.svg
  - Instagram: https://upload.wikimedia.org/wikipedia/commons/e/e7/Instagram_logo_2016.svg
  - HCMUS: TBD
  - …

Team Project: Assessment

- Milestone 1 & 2 (2.0 points each):
  - Implementation (1.0 point)
  - Report + Video demo (0.5 point)
  - Github collaboration (0.5 point)
- Milestone 3 (6.0+ points):
  - Implementation (2.0 points): 4 assessment test cases (0.5 point/case)
  - Report + Video demo (2.0 points)
  - Source code quality (1.0 points)
  - Github collaboration (1.0 points)
  - Optional features (up to 1.0 point)
- Individual (10.0) = Contribution % x Team Assessment x Size of team
- Member contribution should be reported in Milestone 3 submission