CS202: Programming Systems

Week 9a - Design Patterns

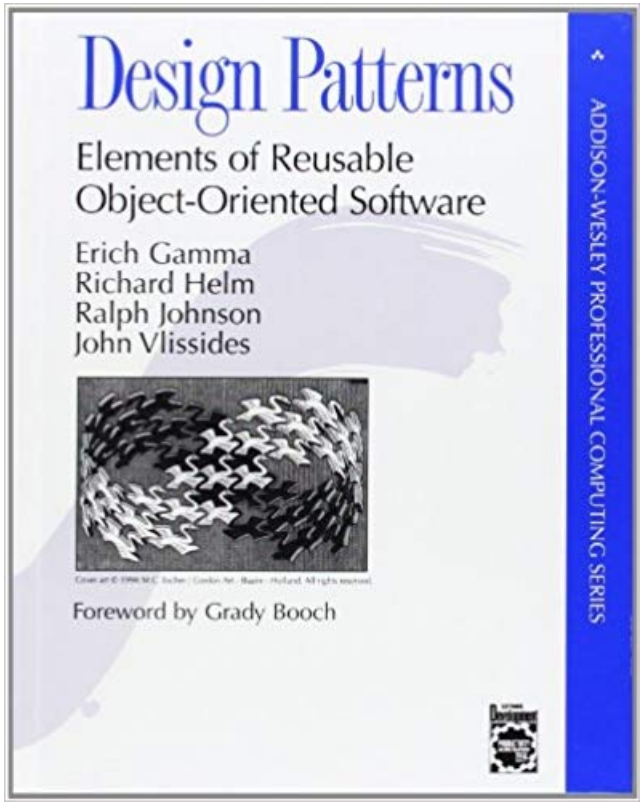
11/2024

Design Patterns

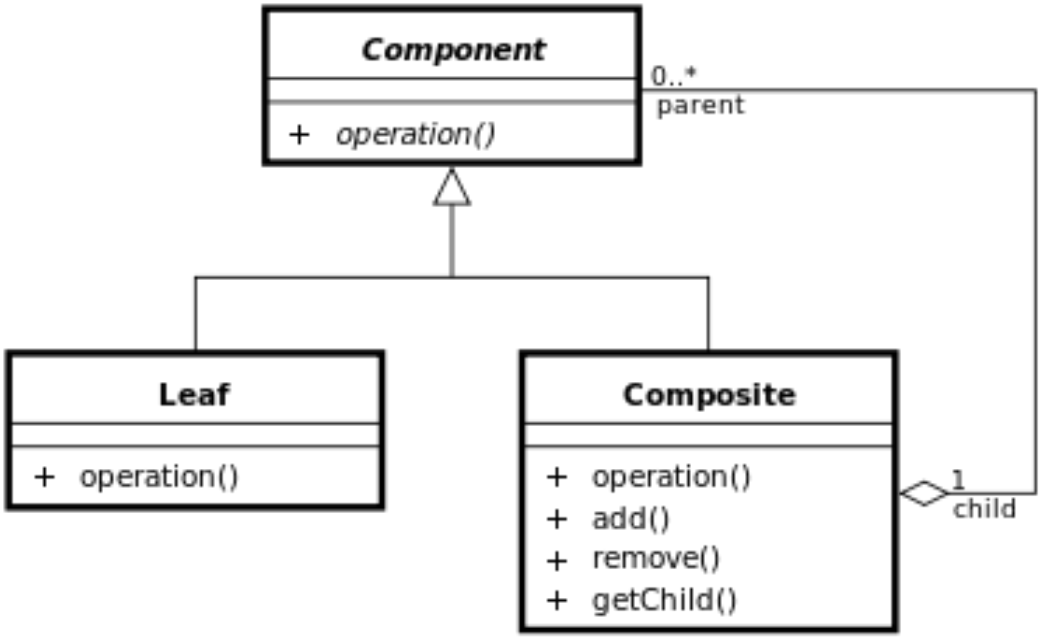
- In software development, a design pattern is a general and reusable solution to a commonly occurring problem.
- A design pattern can solve many problems by providing a framework for building an application.
- With design patterns, the design process is cleaner and more efficient.

Book from the “Gang of Four”

- 23 patterns:



Composite Pattern



(from Wiki)

Exercise

- Design and implement the File&Folder structure using the Composite Pattern

Design pattern: Singleton

- The singleton pattern is a design pattern used to restrict the instantiation of a class to only one object.
- It is very useful when exactly one object is needed for the system.

Singleton

```
class Singleton {
public:
  static Singleton* Instance()   {
       if (!singleton) singleton = new Singleton;
       return singleton;
 }
private:
 static Singleton* singleton;
 Singleton() {};
 Singleton(const Singleton&);// prevent copy-construction
 Singleton& operator=(const Singleton&); //prevent =
};
Singleton* singleton = nullptr;
```

Singleton: attempt to delete?!

```
int main() {
        Singleton* pS = new Singleton;
        //...
        delete pS;
        //...
        return 0;
}
```

Singleton: avoid the deletion

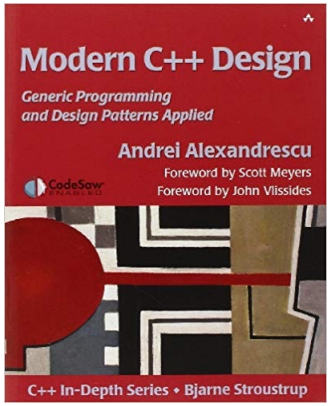
```
class Singleton {
public:
  static Singleton& Instance()    {
       if (!singleton) singleton = new Singleton;
       return *singleton;
 }
private:
 static Singleton* singleton;
 Singleton() {};
 Singleton(const Singleton&);// prevent copy-construction
 Singleton& operator=(const Singleton&);  //prevent =
};
Singleton* singleton = nullptr;
```

Return a reference

Design pattern: Singleton (cont.)

- Still, there are several problems that we need to consider carefully when we implement the Singleton Pattern:
  - When and how to destroy the Singleton?
  - How can we ensure it will be de-allocated lastly comparing to other objects?
  - Multi-processing/multi-threading

Reference: Modern C++ Design, chapter 6.



Singleton: Meyers singleton

```
class Singleton {
public:
  static Singleton& Instance()   {
       static Singleton singleton;
       return singleton;
 }
private:
 Singleton() {};
 Singleton(const Singleton&);// prevent copy-construction
 Singleton& operator=(const Singleton&); //prevent =
};
```