Software Engineering Patterns

25.01.2013

Motivation

- Concept derived from other engineering fields
- Domain specific
- Reusable solution for commonly occurring problem
- Best practices

Types

- Creational
  - Singleton
  - Factory
  - ...
- Behavioural
  - Observer
  - Template
  - ...
- Structural
  - Composite
  - Adapter
  - ...

Types

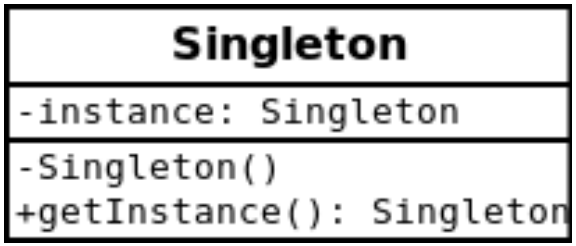
- Creational
  - Singleton
  - Factory
  - ...
- Behavioural
  - Observer
  - Template
  - ...
- Structural
  - Composite
  - Adapter
  - ...

Singleton Pattern - Creational

- One instance of the class
- Global point for accessing the object
- Examples
  - Configuration classes
  - Logger classes

Implementation

- Private constructor
- Eager initialization
- Lazy initialization



Eager Instantiation

```
class Singleton {
    private:
        static Singleton* single;
        Singleton() { /*private constructor*/ }
    public:
        static Singleton* getInstance();
        ~Singleton() { }
};
Singleton* Singleton::single = new Singleton();
Singleton* Singleton::getInstance() {
    return single;
}
```

Lazy Instantiation

```
class Singleton {
    private:
        static bool isInstantiated;
        static Singleton* single;
        Singleton() { /*private constructor*/ }
    public:
        static Singleton* getInstance();
        ~Singleton() { }
};
bool Singleton::isInstantiated = false;
Singleton* Singleton::single = NULL;
Singleton* Singleton::getInstance() {
    if (!isInstantiated) {
        single = new Singleton();
        isInstantiated = true;
    }
    return single;
}
```

Drawbacks

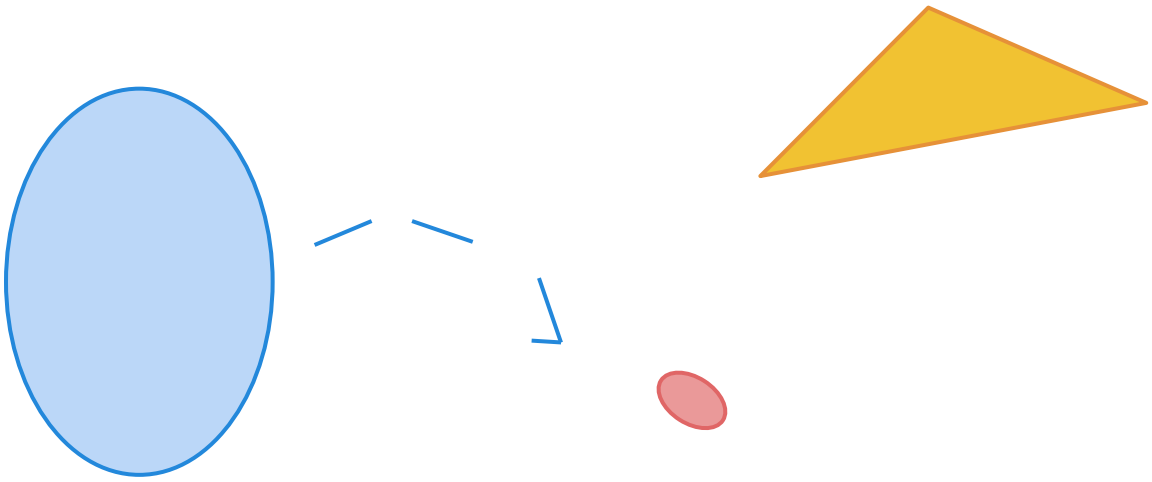
- Multi-threading
  - Less parallelism
- Unit testing more difficult
  - Global state

Composite Pattern - Structural

- Compose objects into tree structures
- Manipulate tree structures (branches and leaves) uniformly

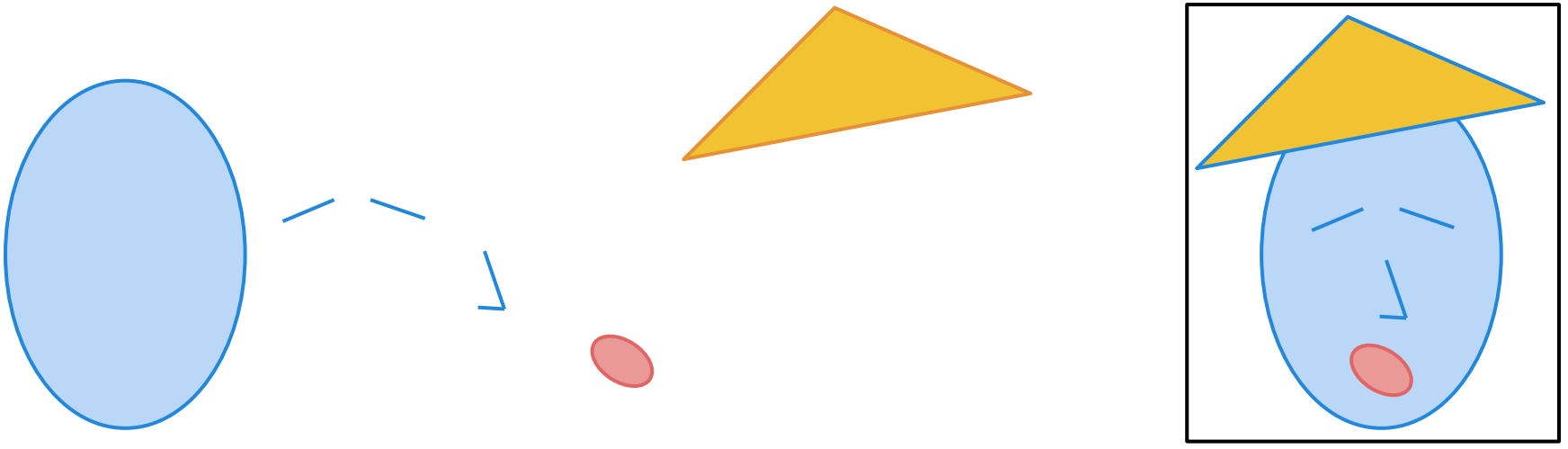
Composite Pattern - Structural

- Example: graphics drawing editor
  - Basic components: lines, triangles, ellipses
  - Complex figures made of basic components



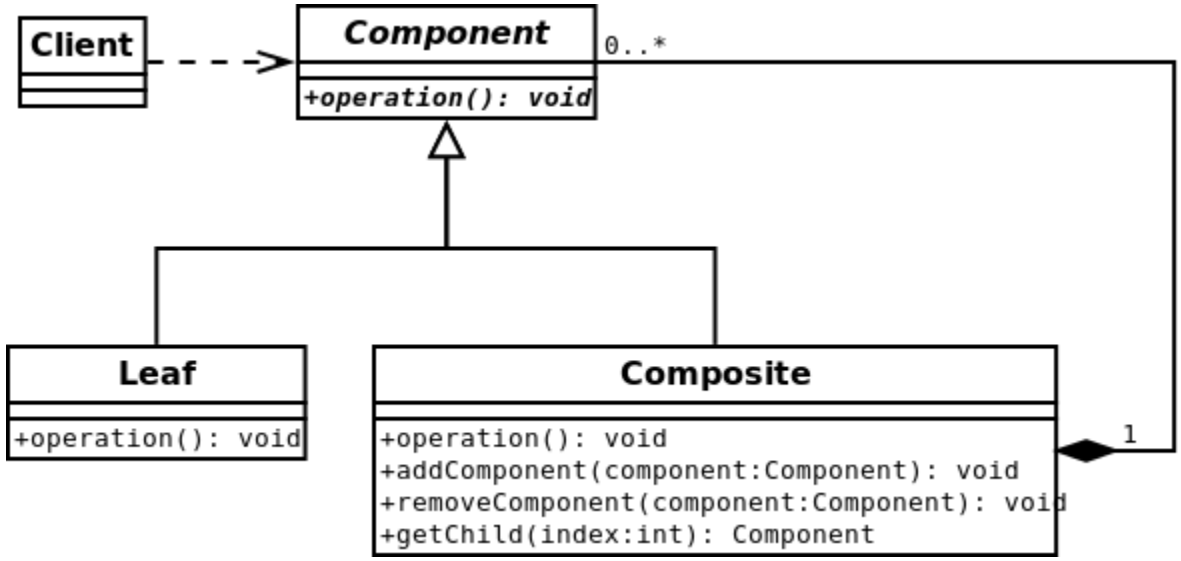
Composite Pattern - Structural

- Example: graphics drawing editor
  - Basic components: lines, triangles, ellipses
  - Complex figures made of basic components



Implementation

- Component: abstraction for leaves and composites
- Leaves: implement the service
- Composite: store child components
- Client: manipulate objects through Component abstraction



Drawbacks

- Runtime checks
- Might be difficult to restrict the components of tree to only particular types

Thank you for your attention

Questions are welcome