CS 246 - F22
Object-Oriented Software Development
Rough Course Summary

With Prof Ross Evans

A very rough list I compiled before studying for the final. Marginally helpful if you don't want to dig for the most important stuff yourself! Not my best work.

Josiah Plett

CS246 STUDY

Stuff I Should Know

① When to include? "Declarations" ▬

② Linker / Static / Templates ▬

③ Casting ▬ static_cast: basic. ‖ reinterpret_cast: basic-er. (bad) ‖ const_cast: const → nonconst ‖ dynamic_cast: (virtual) Big-brain static. gives nullptr.

④ Iterator ▬ for (auto a : vec) !=, (==), *, ++, begin(), end() (List) friend class List

⑤ Big 5 ▬ ☆☆☆☆☆ (see my midterm solutions)

⑥ UML ▬ ◆$\xrightarrow[\text{v \#}]{\text{"owns a"}}$: composition ‖ ◇$\xrightarrow[\text{v \#}]{\text{"has a"}}$: aggregation ‖ ——▷: inherits ‖ *, *: virtual / _v_: static / #: protected ‖ ——: friend

⑦ Exception Safety ▬ basic guarantee: undefined (non-crash) ‖ strong: exception → as if it never happened. ‖ no except: doesn't ever throw

⑧ Decorator Pattern

```
class Component {...}
class ConcComponent1: public Component {/* base functionality */}
class Decorator: public Component {
  protected: Component* nextItem;
}
class ConcDecorator: public Decorator {...}
```

- Add or remove functionality at runtime.

ⓘ functionality → functionality → base (linked list)

⑨ Visitor Pattern

```
class ConcreteSubject { virtual getVisitedBy(Visitor& v) override { v->visit(this); }
class Visitor {
  virtual visit(ConcSubject1& s) = 0;
  virtual visit(concSubject2& s) = 0;
}
```

one dynamic dispatch

← two dynamic dispatch (covariance)

- Specific function calls based on 2 polymorphic types

ⓘ 2 dispatch spots!

⑩ Factory Method Pattern

```
class Level {
  public:
    Level();
    virtual Texture* createTextureBinding();
}
```

Virtual Constructor Pattern →

```
class APGLevel: public Level {public: Texture* createTextureBinding() override; }
```

- Solves if you can't construct something.

ⓘ Another method (virtual) does constructing.

⑪ Template Method Pattern

must be fully in the header

```
class Pizza {
    virtual void addTopping();
    virtual void addUrMom();
  public:
    void pizza()  {addTopping(); ... urMom(); }
}
```

cout, for eg

- Delegating method functionality.

ⓘ Want subclass to adjust *part* of functionality.

⑫ Non-Virtual ImplementationMethod

```
class Video {
    virtual void doPlay() = 0;
  public:
    void play() { ... doPlay(); ... };
}
```

- To enforce the implementation doing certain things
- To do extra stuff.

ⓘ *All implementations* (in interface) are non-virtual.

⑬ Observer Pattern

~Subject() defined outside.

```
class Subject {
    vector<Observer*> observers;
  public:
    virtual void attach(Observer* ob) = 0;
    virtual void detach(Observer* ob) = 0;
    virtual void notifyObserver() = 0;
}
```

- Subject "has-a" observers
- Concrete Observer has 'notify()' pure virtual.

ⓘ *Many-to-One* relationship, like Twitter notifications.

⑭ Bridge Pattern (pImpl)

```
class OilPlane {
    Oil Engine* e;
    Plane* p;
}
```

← multiple pImpls.

- Solving the stupidity of inheritence

ⓘ Multiple inheritence is solved.

⑮ Coupling + Cohesion ▬ Coupling: reliance + interdependence ‖ Cohesion: SRP + does just one thing

⑯ Model + View + Controller ▬ Model: data

can be together { View: accessing the data
Controller: client ↔ us, + manipulation

☆ **If you have a virtual method, Destructor MUST be virtual.**

⑰ Polymorphic Copy/Move

```
class Book {
  // define the Big 5
}
class Text: public Book {/* defined outside */}
```

★ Make sure Book is abstract!

Otherwise, we either have partial assignment or mixed assignment.

copy ctr

```
Text::Text(const Text& other): Book{other}, topic{other.topic} {};
```

move ctr

```
Text::Text(Text&& other): Book{std::move(other)}, topic{std::move(other.topic)} {}
```

assignment

(copy ass. is same but ~~w/~~ std::move)

```
Text& Text::operator=(Text&& other) {
    Book::operator=(std::move(other));
    topic = std::move(other.topic);
    return *this;
}
```