Comp 2511

- Compile time Error: 可以直接找出 syntax (语法) 的错误. 在编译的时候. (有些语法不符合 java 的特性.).
- run-time Error: 在编译的过程中体现不出来. 需要运行程序才可以显示出来的错误. 例如 Null-pointer 之类的.

- Abstraction ← 如何把一个东西转换为数据结构.

强化 ↑ Control Aucss Rights

事物 + definition → class → define data structure.

事物的实例 instance → object ← create.

- Encapsulation → 将 class 内部的 Attributes 封装起来.

↓ 那么要访问 Attribute 的一定要通过方法.

→ 可以理解为. Instance 这个实例其实是 pointing 向具体的 class 的. ← 但不想暴露内部结构

带来的好处:

(1) State Control. (只有自己可以控制) Method.

getter setter → (2) Usability: provide consistent contact with invoker, keep data private. ← 对外来说访问者只能通过一定方法来访问. 使结构清晰.

可以带来 Low coupling 的好处 → (3) Lead to Abstraction, Reduce dependency. 若想要将某个 Attribute 设为不可改变 = Immutable ↑ 若要改变需创新的 instance. 就 Remove Setters.

- Relationship → 只要有任何关系. 都是 Associate 的 →

- Inheritance 继承关系: extend, override. → Abstract class 里只是定义了 data type 或 Method 但需子类来 override 以便完整

- Association 相关 可以有多对多. 一对多 多对一

  - Aggregation 聚合 → Course 有很多 students. 但 student 和 course 独立存在. 并无任何关系. (contains relationship). Eg. 成员变量
  - Composite 组合 → 相反. → 是组成的一部分. death relationship. Eg. 局部变量

- **Polymorphism** → 只有在run time的时候才有这种情况.
  - (1) Instance can change to different types / definition.
    - 可以变成不同的type ← 不一定是class. 也可能interface.
    - → class 可以 extend 一个 class. Eg. class extends ~~A.B.~~
    - → interface 可以有 multiple implementation.
    - Eg. class A implements B.C.D.E... ✓
  - (2) Instance cannot refer to Abstract class ← 只可被继承.
  - (3) Instance type { check in compile time ← 第(1)条. / determined in run-time. — Dynamic Binding
    - ↑ Eg. class A extends B.

B ↑ A

B b = new B();

type → A a = new B(); ← type 依旧是 A.

- design-tool:

Domain Model → 其实是一个 Analysis requirement的过程. { 对内 → programming / 对外 → user }

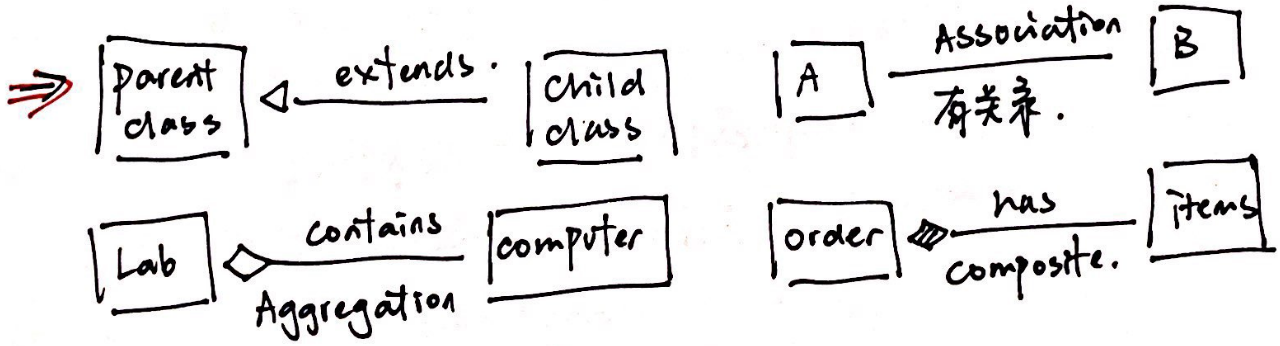
→ Verb. Noun Analysis.

→ CRC - Name + Knows/does + Colaborator (其他的Noun)

UML → "-" private. "+" public "#" protected.

可以表示 class / object / Abstract class / interface → 第一部分写名. 图

斜体的className 只能被implement



Java Features. 访问权限

(1) Public → 所有人.

(2) Protected → Same Class + Sub class + Same Package.

(3) Default → Same Class + Same package.

(4) Private → Same class.

- Method Override

(1) 限制条件 limitation.

- Access Rights - 总要比 parent 的大. Eg. Parent 中的为 public 那子类中也不可为 private.
- Final Method — 是不可以被 override 的.
  → 可以 Assign value 的. 但 assign 了之后就不能被改的.
  → 但 static Method. 是完全不可以改变的. 一开始就 set 好了的.
- Constructor 也不可以被 override
  → 可以重新 define Constructor 或者使用 overload.
  → 重新 define 后需 super() 拿到父类的 constructor.
- Return type <= Parents return type.
  Eg. B ← A. 若一个父类 return type 是 B, 子类也可 A 可 B.

Eg. 在 A 类中有方法 Method1() 为 static.
在 B 类可以直接运行 A.Method1()
↑
哪怕叫同样的名字也是不同的 Method.

(2) Static Method → Cannot be override. instead. they are different methods. reference is determined in runtime

- Generate type.

```
public class Box<T> {
    private T t.
    public T get() return t;
    public void set(T t) this.t = t;
}
```

<T> → 可以是任何的 data type.

可能会造成 run-time Error.
↓
若不定义则以下两条都存在. → Eg. Box<String> b = new B<>();

Eg. Box<String> b = new B<>();
b.set("hello!") ← ✓
~~b.g~~ b.set(10) ← × 10 is not String

- Exception → 错误的处理方法. ← catch 到就可以了.
  → 导致程序无法继续下去的时候. → throwable.

无法处理 → Error (no need to handle)
可以被 handle → Exception (need to be handled)
的. try... catch. If not handle → unhandable exception error.

使用 → Exception 是一个 abstract class 所以是需要被 extend 的.

→ functions can throw Exceptions.

```
public void fn() throws xException, YException {
    String s = input().
    if s.contains ("x") {
        throws new xException().
        throws new YException()
    } else ...
}
```

②

⇒ 做一个 catch 的动作

```
try {
    fn()
} catch (Exception e) {
    deal with e
} finally ...
```

①

可以接收任何 Exception. 也可定义. Eg. xException xe

处理中..

所有会call的函数. 使用①来处理②

call.

Eg.
```
public void dosomething (File file)
    try {
        fileReader fr = new FileReader(file)
    catch (IoException ie)
        system.out.println ("file doesn't exists")
```

← 不存在时会触发 IOException

handle 若不存在的情况

→ Program Contract

```
/**
 * deposite Amount into Bank Amount.
 * @Param an Amount to be deposited to account
 * @Pre Amount > 0.
 * @post Balance = Balance + Amount.
 */
```

在Method之前定义限制条件.

@pre → 输入的条件.

@post → 作为@pre的input后. 所会得到的output.