Jin Wei   CS2103T UML Diagrams

Class Diagrams

ClassName

+ attr1 : type
- attr2 : type = default value

func ( ) : return_type
~ void_ func (...)

optional (even if non-empty)

only need type, not param name

optional
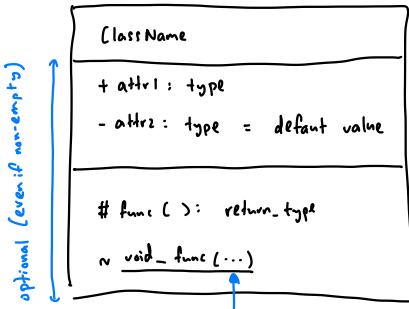+ public
- private
protected
~ package private   (default in Java)

→ typically don't include standard library
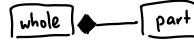
underline class-level attribute/methods
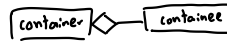aka static

Solid line — association

Solid arrow
→ navigability of association
←→ bi-directional navigation

★ rmb to use composition
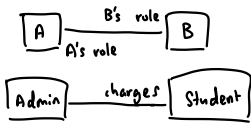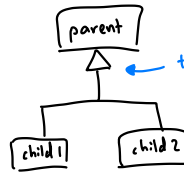
whole ◆— part   composition

container ◇— containee   aggregation

A —B's role— B
A's role
roles   (optional)

Admin —charges— Student

parent
△
child1   child2
triangle, not arrow
inheritance (extends)

A —label ▶— B   label
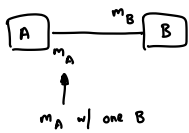
A —m_B— B
m_A
m_A w/ one B
multiplicity

0..1	0 to 1
1	1
*	≥ 0
n..m	n to m

<< interface >>
InterName

x ( )

interface (implements)
→ interface methods are public abstract
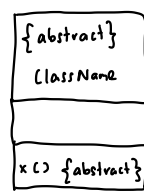→ interface attributes are public static final
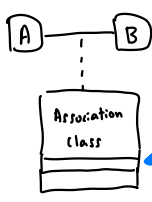
△
X1   X2
x()   x()
need to implement methods

A – – – → B   dependency
(if not captured by other notation)
→ referring to constants
→ method param/return type

A depends on B
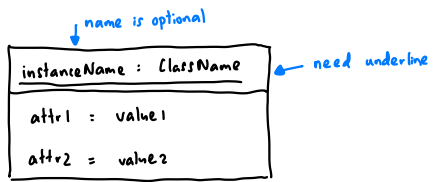
{abstract}
ClassName

x ( ) {abstract}

OR use italics

<< enumeration >>
EnumName

enum1
enum2
...

enumeration

A —— B
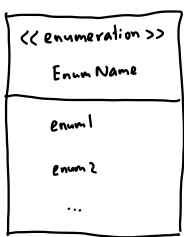
Association Class

association class

don't include A, B

Object Diagrams

name is optional

instanceName : ClassName ← need underline

| attr1 : value1 |
| attr2 = value2 |

→ no multiplicities (always 1)

→ no methods

→ no inheritance

solid line — association

label ▶ — labels

navigation

Sequence Diagrams
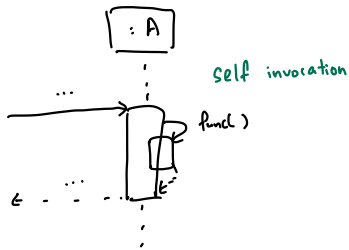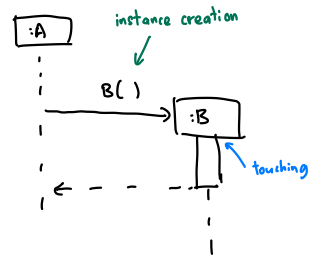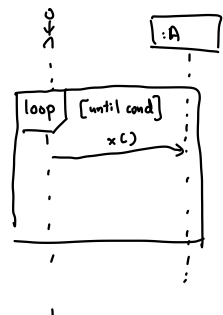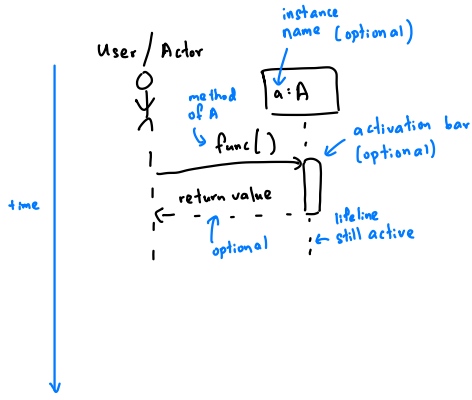
<< class >> A — for calls to static methods

note: for the top-left boxes,
note: if X created before Y, X's lifeline starts before Y

`...` this shape w/ the shaved edge!

User / Actor

instance name (optional) — **a : A**

method of A → func()

activation bar (optional)

return value — optional

lifeline still active

time

:A

loop [until cond]
x()

:A

instance creation

B() → :B

touching

---

self invocation

:A

...

func()

...

---

callback

b : B — func(b) — :C

xd()

```
class C {
    func(B b) {
        b.xd();
    }
}
C.func(b);
```
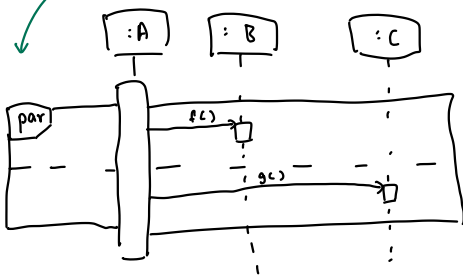
---

delete instance
(Java cannot → just means no more reference
→ garbage collector will clean it)

:A   :B

delete → X

no more lifeline

---

reference to another diagram

ref < ref name >

separate sequence diagram

sd < ref name >

< diagram here >

---

alternative path

:A   :B

alt [cond]
x()

[else]
y()

```
if (cond) {
    b.x();
} else {
    b.y();
}
```

:A   :B

opt [cond] func()

```
if (cond) {
    b.func();
}
```

---

parallel paths (concurrency)

:A   :B   :C

par   f()

g()

OODM

Object - Oriented Domain Models

aka Conceptual Class Diagrams

→ type of class diagram

→ models **problem** NOT **solution**

→ class structure of problem NOT behaviour

→ no navigability

→ no methods

→ can have attributes (types are optional)

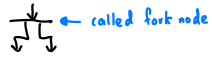## Activity Diagram

X ← ✱ ROUNDED CORNERS   action

→   start

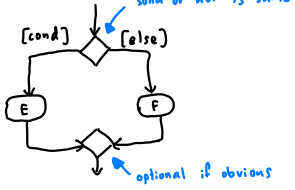← called fork node   fork (to parallel paths)
   ← only for diff threads

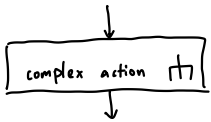← compulsory   join (need ALL parallel paths to complete)

solid or not is same
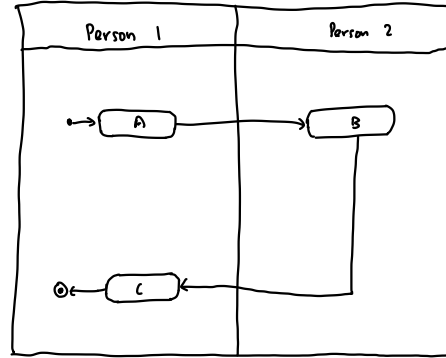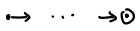[cond]   [else]   branch (exactly one path is taken)

E   F

↓ ← optional if obvious

→◉   end (can have multiple)

complex action ⊞   rake

Activity: complex action

↦ ⋯ →◉

Person 1 | Person 2

→ A → B

◉← C ←

swim lanes