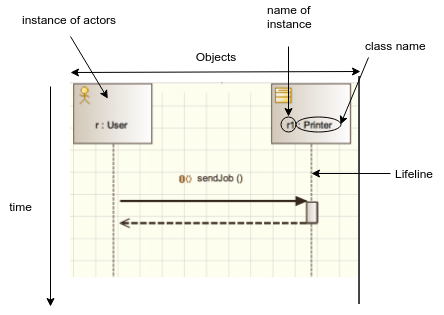
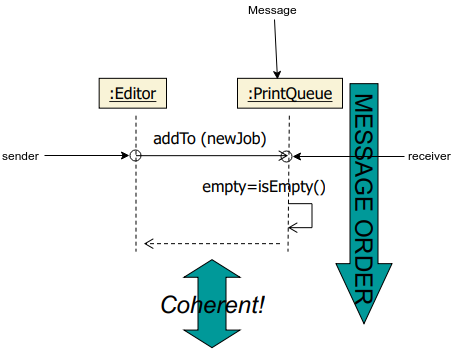
# Behaviour; week 5

### 1/ Sequence diagram

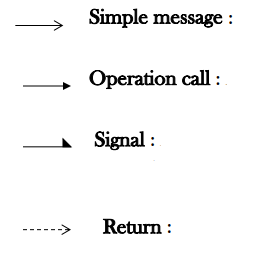
* **inter-object view** : we try to specify how objects interact with each other. All classes
* specify the algorithms
* **participant :** entities participating in the interaction
* **messages :** communication between objects.
* 2 axes :
  + **horizontal :** which object/participant is acting
  + **vertical :** time

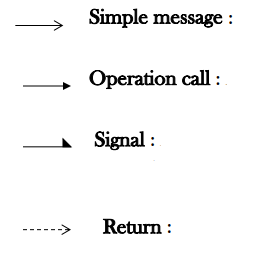


### 2/ Message

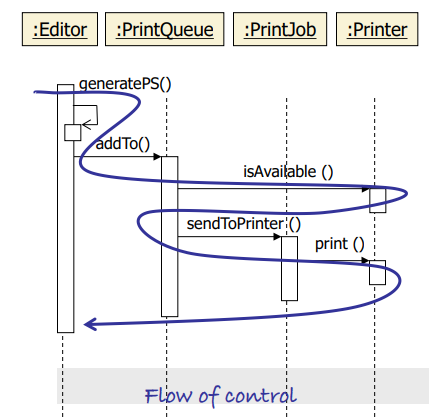


* It’s a call of methods.
* The receiver is the one calling the message
  + **synchronous :** blocks the caller while the called method executes. The sender can’t do anything until the method didn’t finish.
    - you implement it normally: method1(); method2();
  + **asynchronous :** doesn’t block the caller. Caller executes // with the method. You have to use threads to implement it.
* notations :

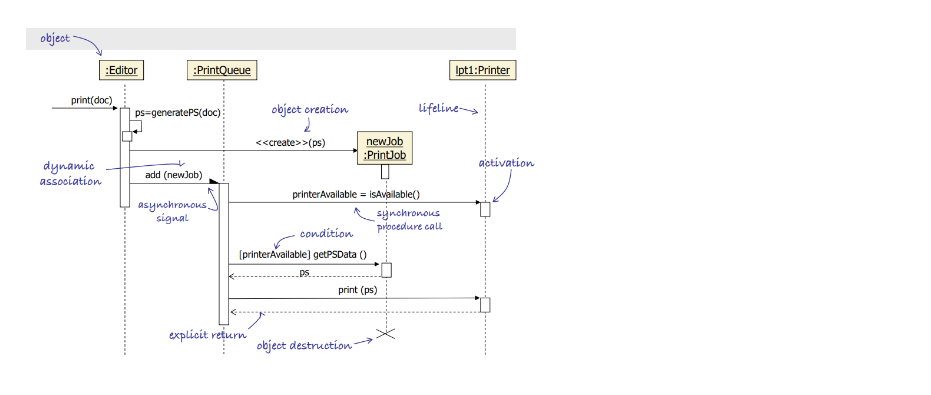




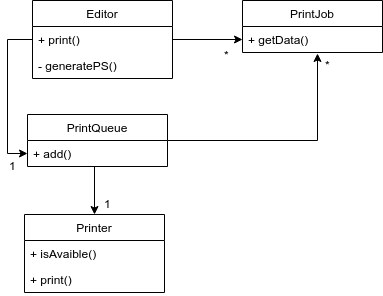
* deletion message : you delete it from memory. There is garbage collector that does it automatically in Java. But it C++, you use free().
* activation : when an object is executing a method or an operation, it becomes ‘active’.
* using activation, you can follow the flow of control



### **Example : a printer**



* ps : post script. self called, so it can be private.



class : collection of objects inside the application that represent the entity.

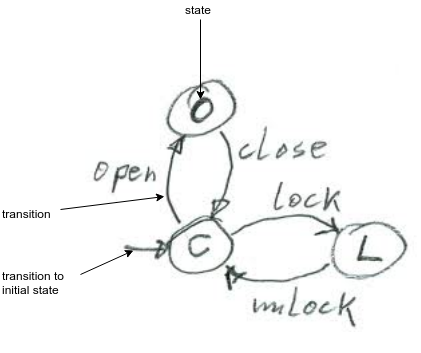
actor : outside the application. It’s an extended physical entity, it interacts with the system.

### 3/ communication diagram

* pretty same as sequence diagram but the execution order is described by numbers.

### 4/ State machine

* **intra-object view :** how one object behaves; one class
* you don’t use it to describe passive objects (that don’t change states)
* initial state : mandatory
* final state : not mandatory
* transition : describes how an object moves from one state to another



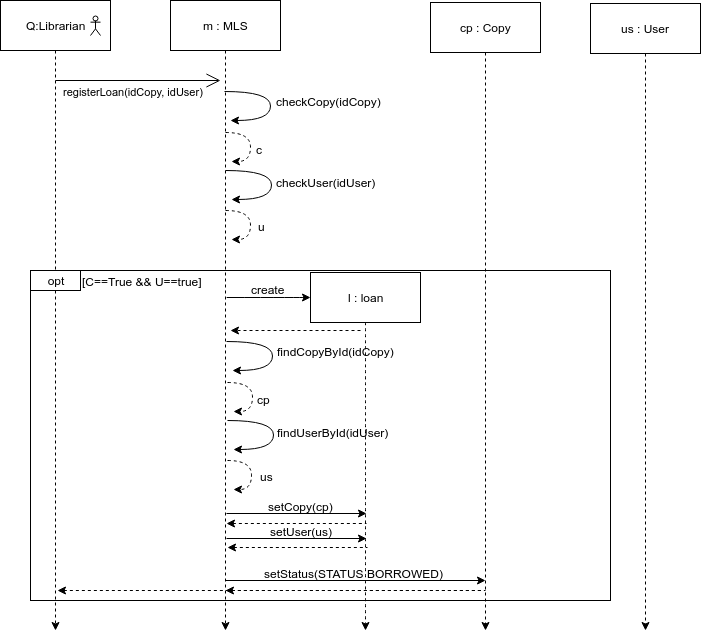
* timeout : event that is related to time
* composite/super state : gather a set of substates into a composite state => used to factorise some transitions

|  |  |
| --- | --- |
| **Without superstate** | **With composite state** |
|  |  |

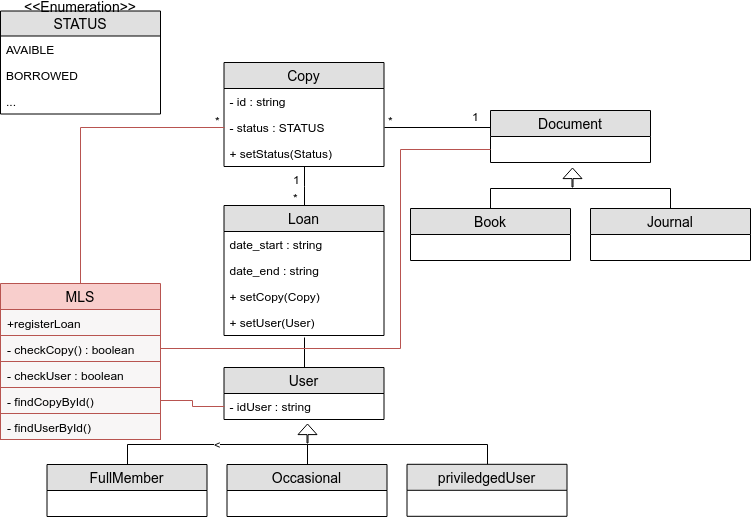
H : when you define a composite state, you have to add a h-state (history state); its purpose is to allow you to leave and to return to the superstate.

MLS example :

Sequence diagram:



The class diagram is modified in consequences :



state machine of copy

