Software Engineering 300:
Introduction to Software Engineering

Sequence Models

- What is sequence modelling?
- How can source code be represented by a sequence diagram?
- When is sequence modelling useful? Or not?

1

What is the sequence of interactions?

```
class A {
  void someMethod() {
    System.out.println("Yo");
  }
}
```

TO HELP, START BY CONSIDERING THE STRUCTURAL MODEL

What is the sequence of interactions?

```
class C {
  void someMethod(D d) {
    d.otherMethod();
  }
}
```

```
class D {
  void otherMethod() {
    System.out.println("Hi");
  }
}
```

Sequence diagrams

- Show sequences of interaction between objects
  - Basic structured programming constructs supported: sequence, choice, loop
- Show messages being passed between objects
- **Issue**: "Object-oriented" source code
  - does not show **objects**!

Objects versus roles

An *object* in a diagram (any UML diagram) represents a specific object.

If we need to model a generic object, we can use a *role.*

Object:

Robert Walker: Instructor

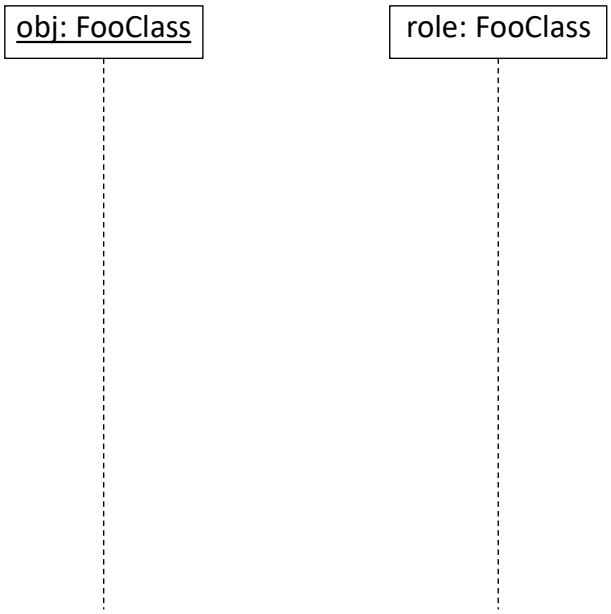
Role:

seng300Instructor: Instructor

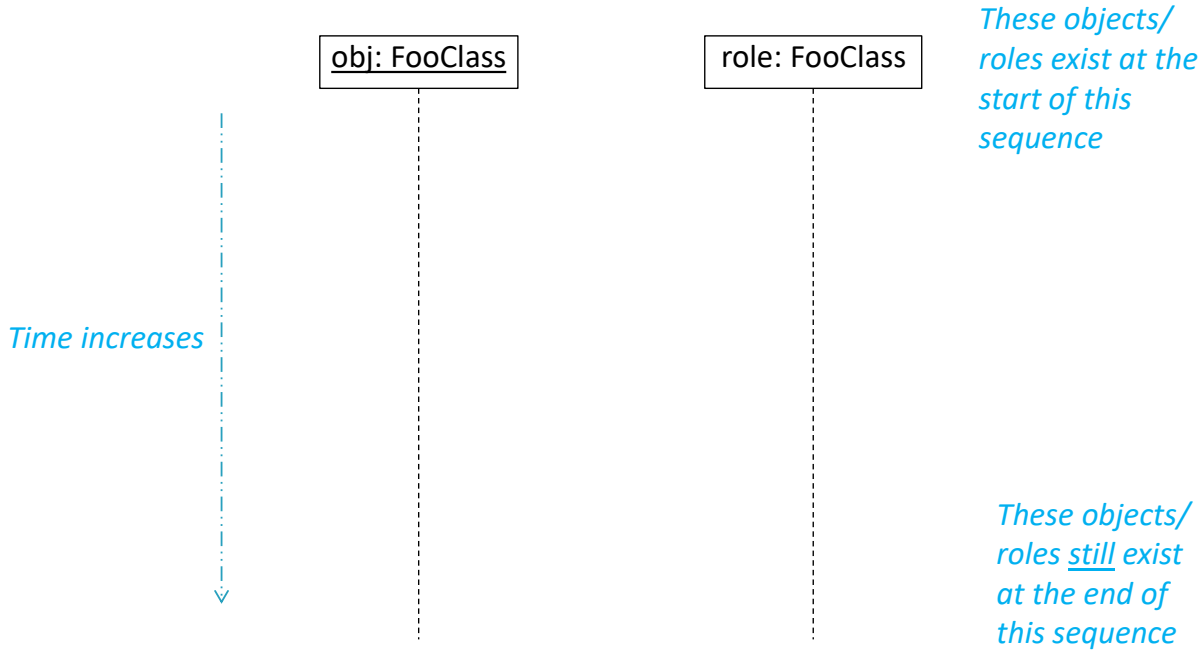
Note that the colon is always present, so these are not classes. The underlining differentiates an object from a role.

(In some unusual cases, classes can be present on a sequence diagram, but this would normally be WRONG.)

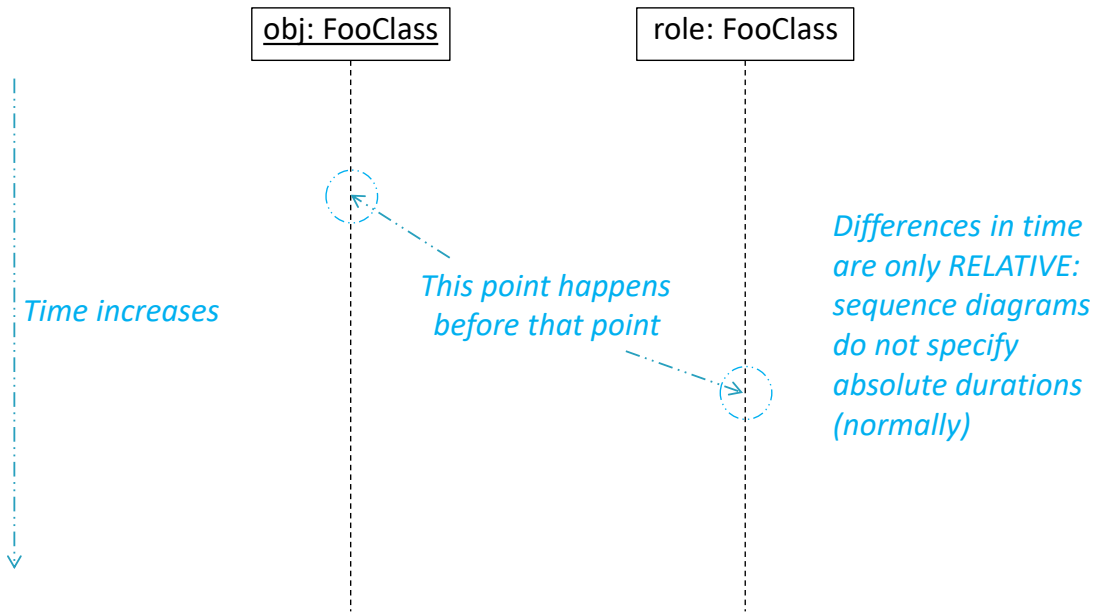
Lifeline



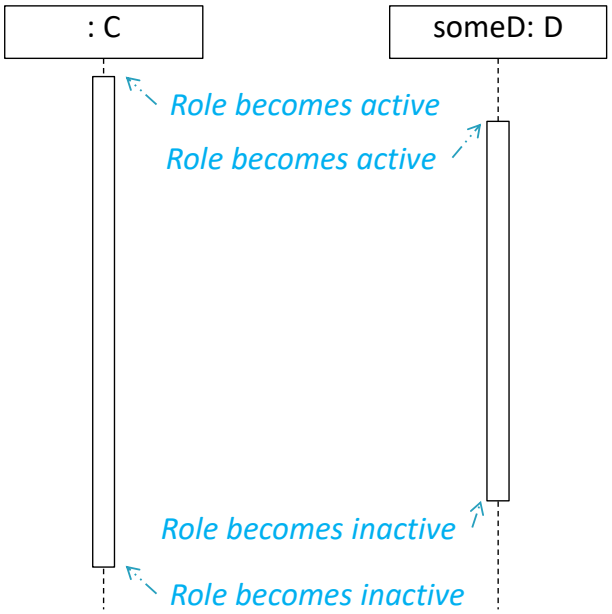
Lifeline



Lifeline



Activation Bars



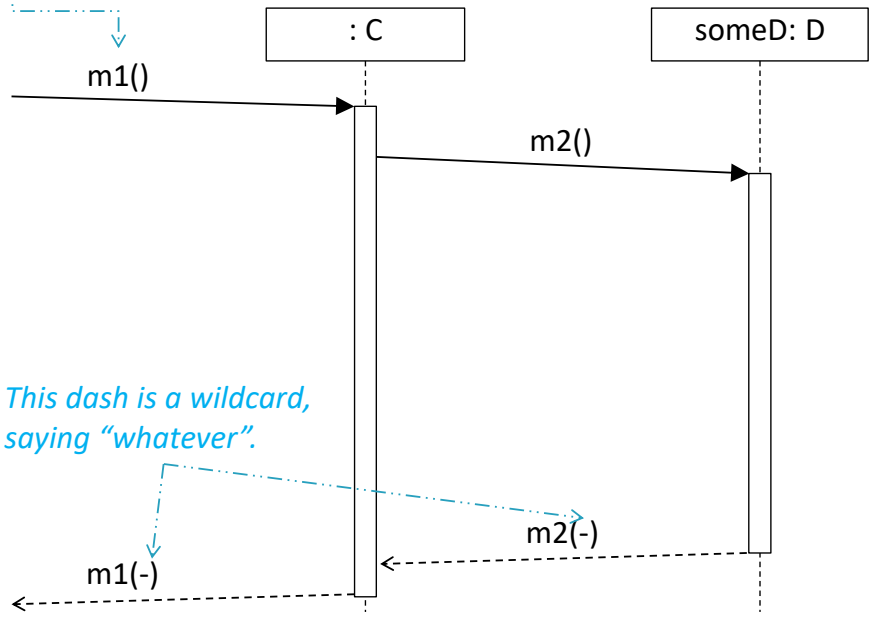
"Active" means a method, for which the object/role is the receiver, is on the call stack.

Equivalently, that method is "doing something" (or waiting for another one to return).

Cause?

Argument Passing and Return Values

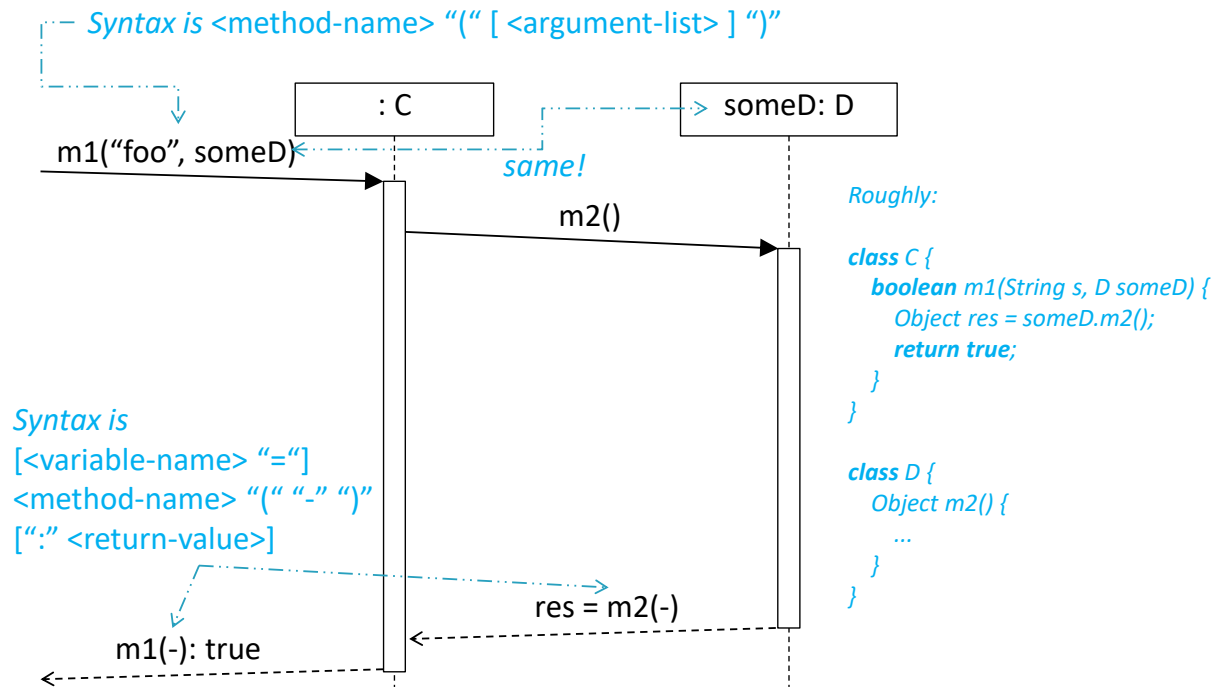
Syntax is <method-name> "(" [<argument-list>] ")"



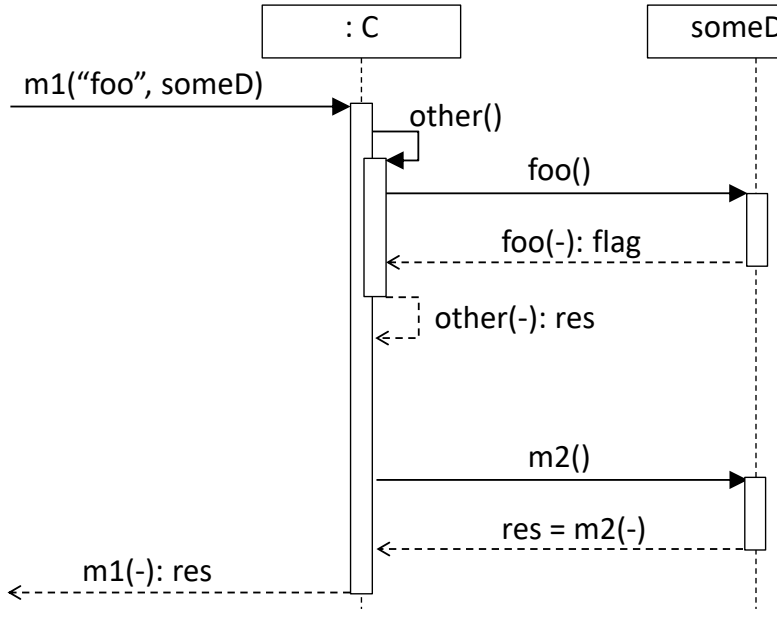
Roughly:

```
class C {
   D someD;
   void m1() {
      someD.m2();
   }
}

class D {
   void m2() {
      ...
   }
}
```

Method Call and Method Return



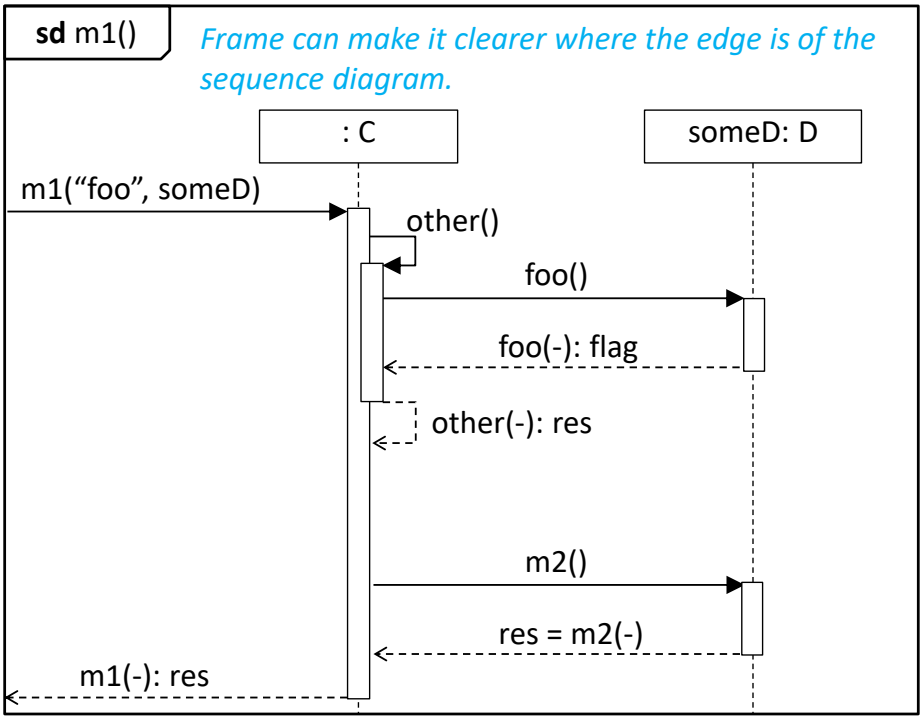
Self-Call



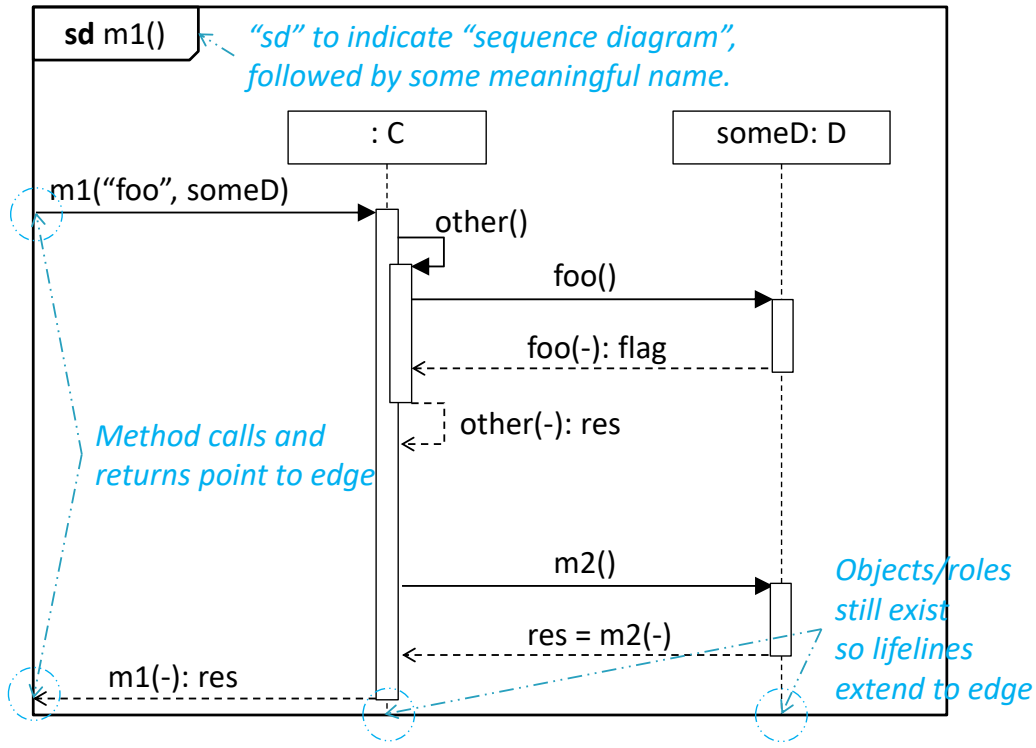
Roughly:

```
class C {
   D someD;
   boolean m1(String s, D someD) {
      this.someD = someD;
      return other();
   }

   private boolean other() {
      return someD.m2();
   }
}

class D {
   boolean m2() {
      ...
   }
}
```

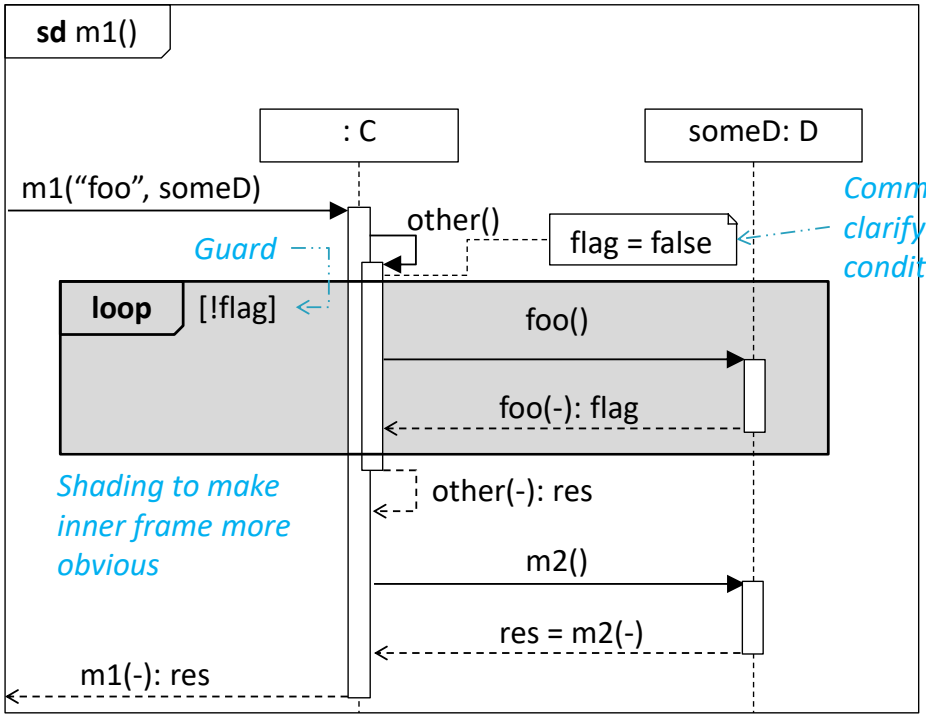
Interaction Frame



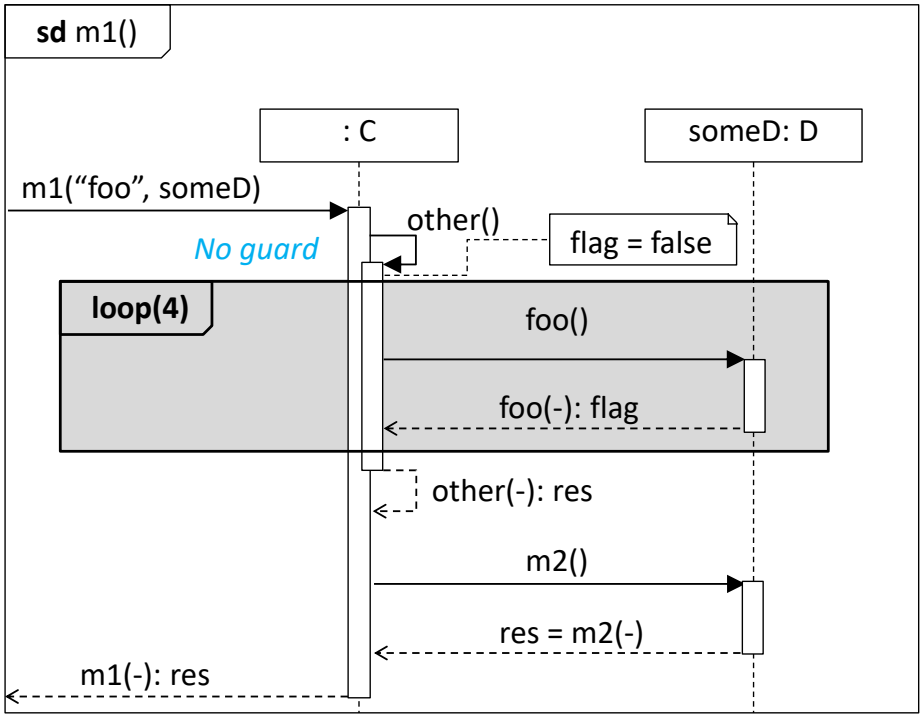
Interaction Frame



Loop Fragment

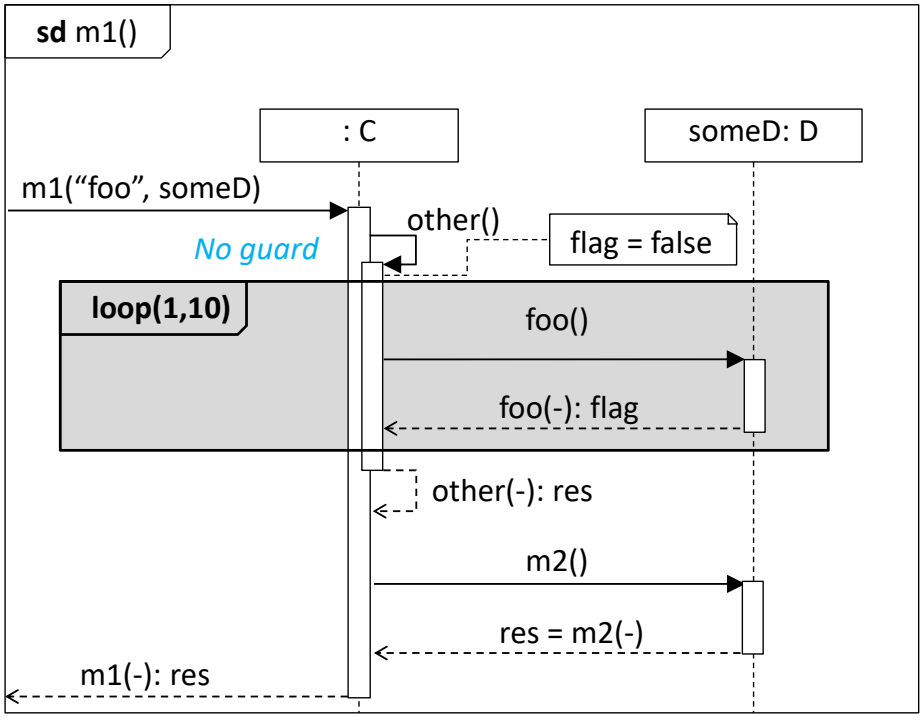


Loop Fragment



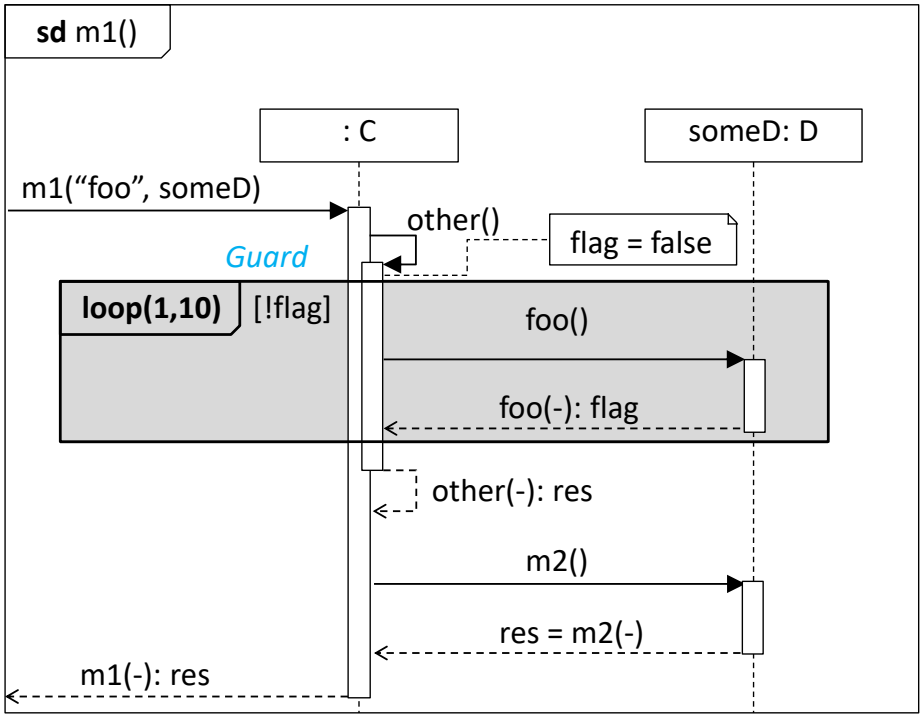
Loop exactly four times.

Loop Fragment



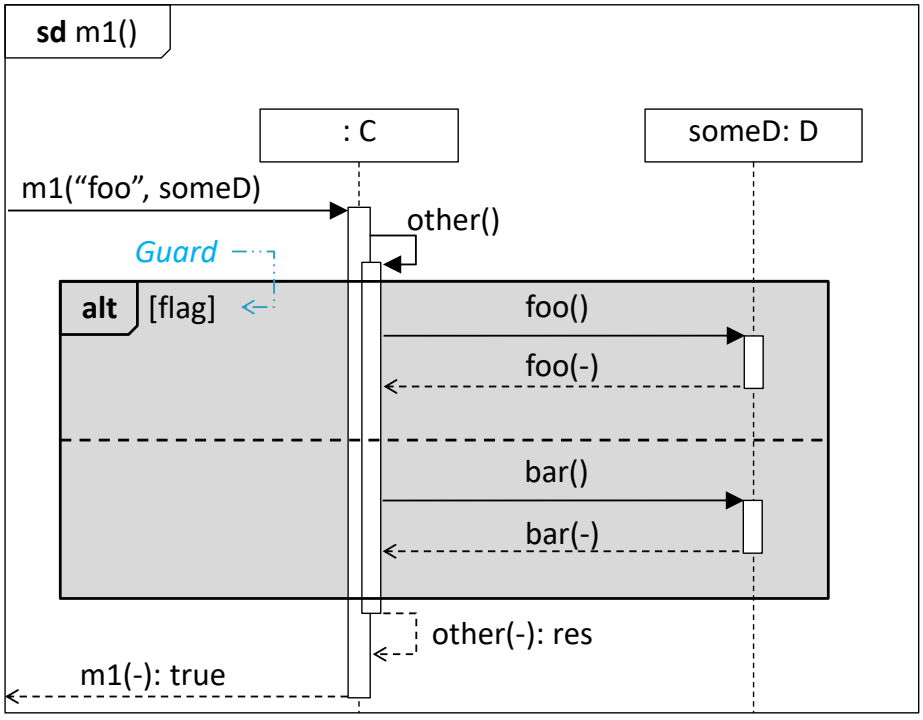
Loop at least once but maybe up to 10 times.

Loop Fragment



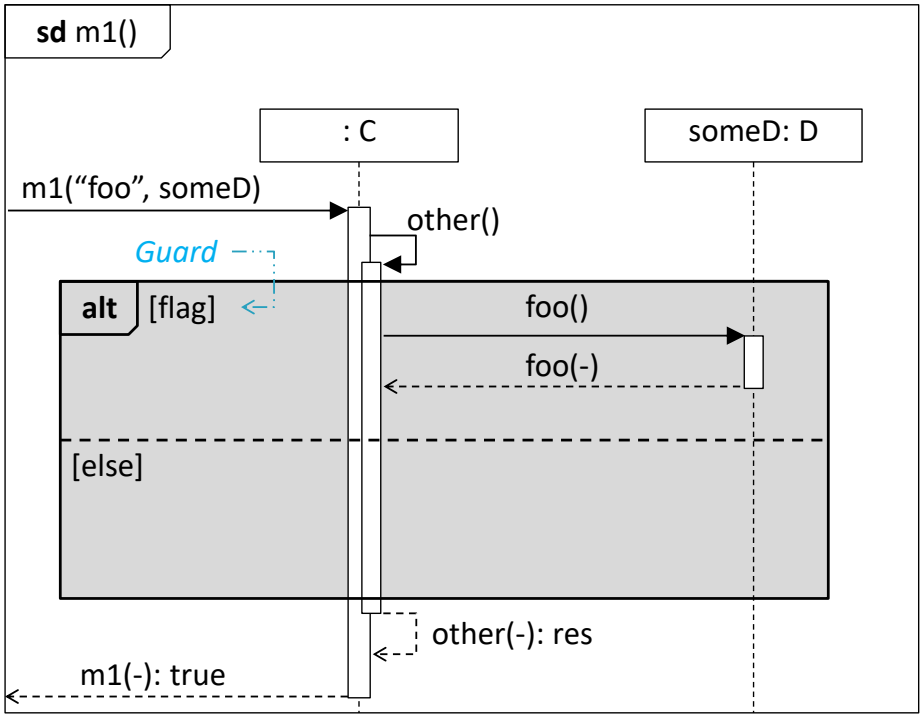
Loop at least once but maybe up to 10 times; after 1 loop the loop stops unless the guard is true.

Alternative Fragment



If the guard is true, the top compartment executes; otherwise, the bottom.

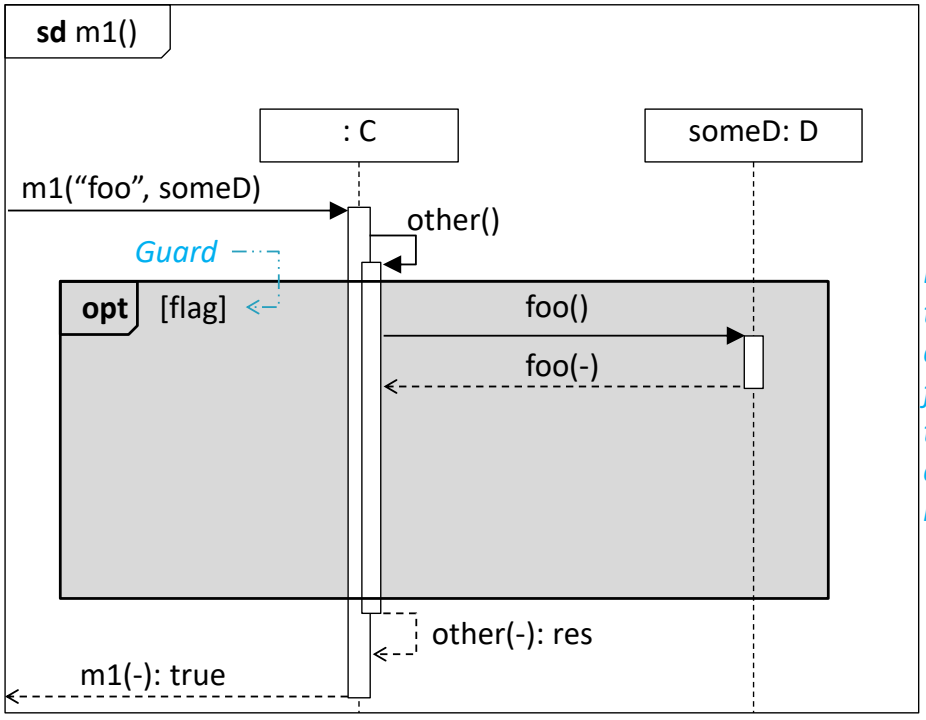
Alternative Fragment



Compartments can be empty.

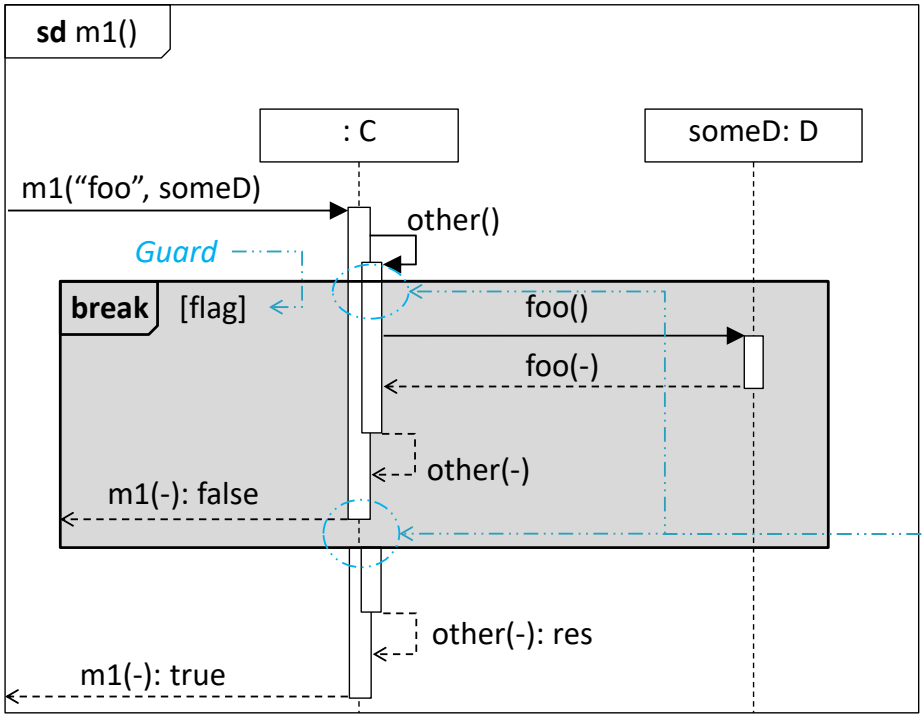
The "else" guard is optional; with or without it, the meaning is identical.

Optional Fragment



Exactly equal to an alternative fragment where the bottom compartment is empty.

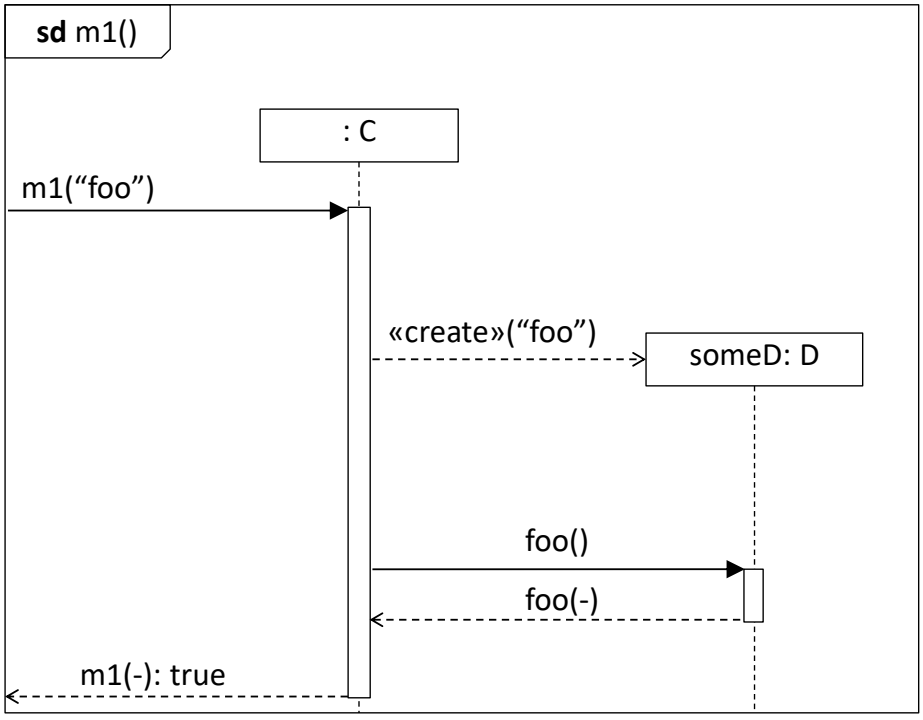
Break Fragment



The body of the break fragment executes only if the guard is true. At the end of the break, the outer fragment is stopped.

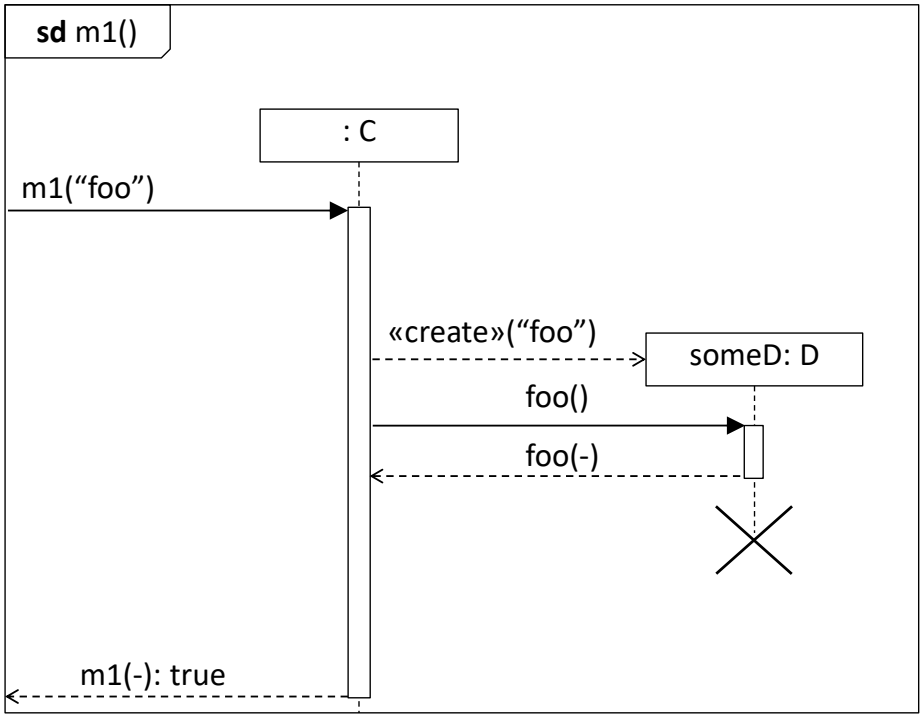
If the break is skipped, the activation bars continue.

Object Creation



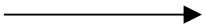
"someD" did not exist until created; box is offset from top.

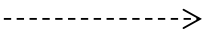
Object Destruction

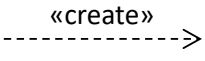


"someD" is garbage collected once we are through with it. Its lifeline is ended with a big X.

Message kinds

Synchronous call 

Return 

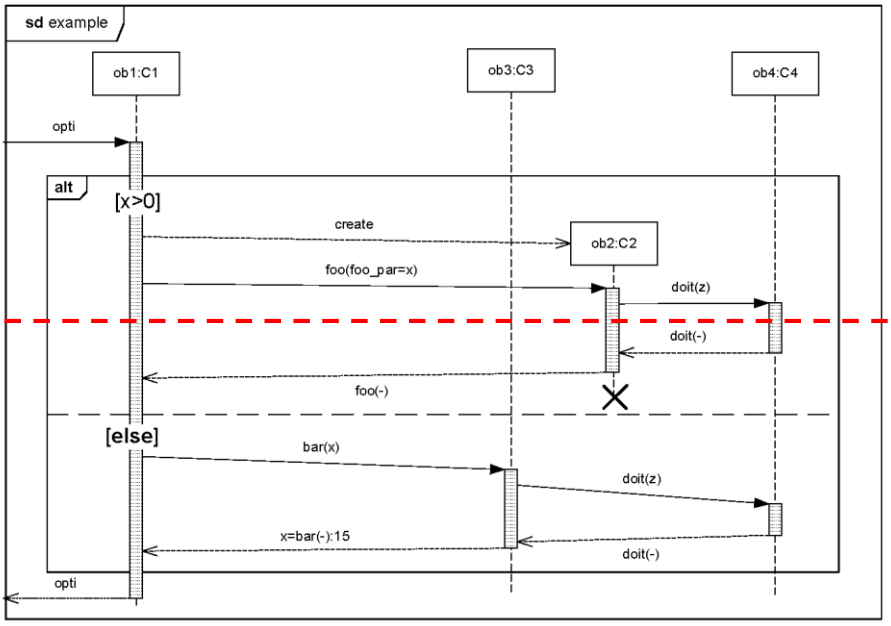
Object creation 

Object destruction

implicit (no message is sent) 

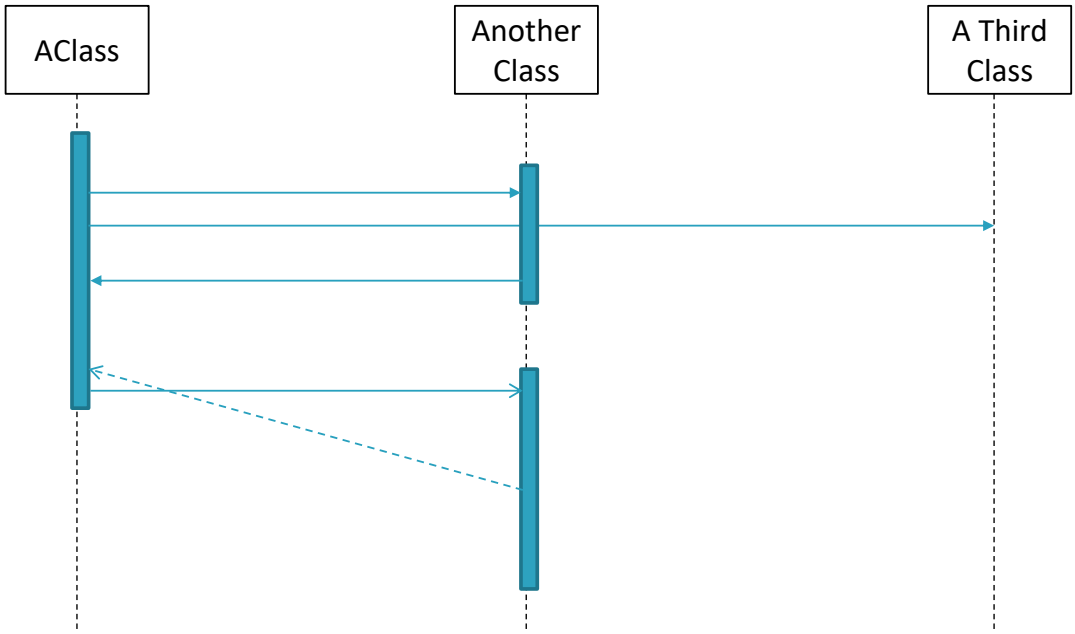
Other kinds exist, but beyond scope of course

Sequence diagram example



WHAT IS HAPPENING HERE?

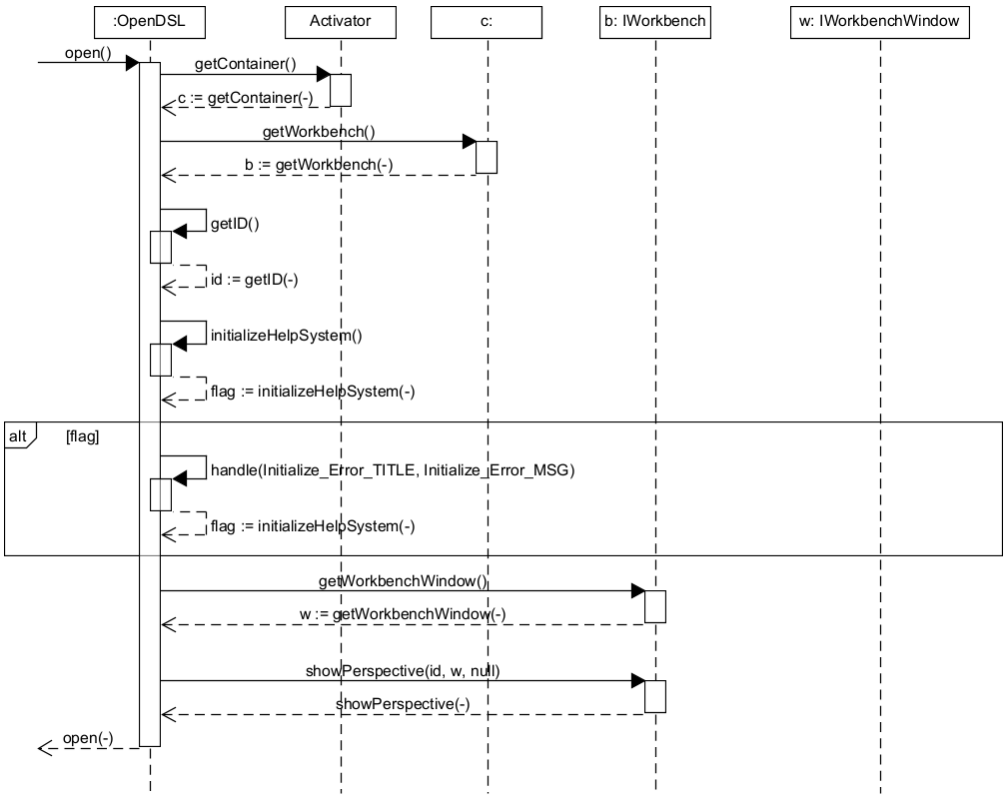
Some common errors



Exercise

- Model this:

```
public class OpenDSL {
   public void open() {
      IWorkbench bench = Activator.getContainer().getWorkbench();
      String id = getID();

      if( initializeDSLHelpSystem() ) {
          handle(Initialize_Error_TITLE, Initialize_Error_MSG);
      }

      IWorkbenchWindow window = bench.getWindow();
      IAdapter adapter = null;
      bench.showPerspective(id, window, adapter);
   }
}
```



Next time

- State Machine Models