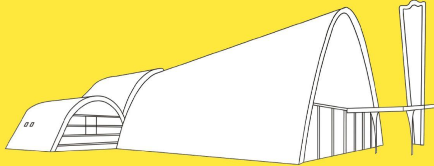
SOFTWARE ENGINEERING

A Modern Approach



MARCO TULIO VALENTE

Chapter 4 - Models

Prof. Marco Tulio Valente

https://softengbook.org

CC-BY: This license enables anyone to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the author.

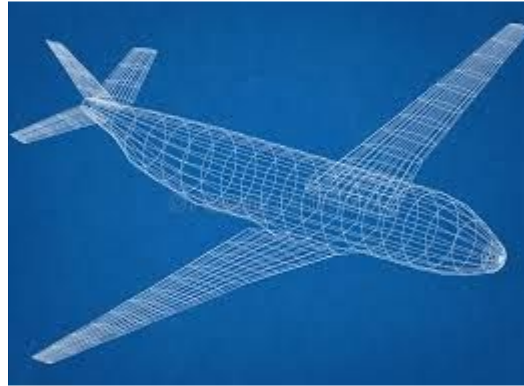
Motivation

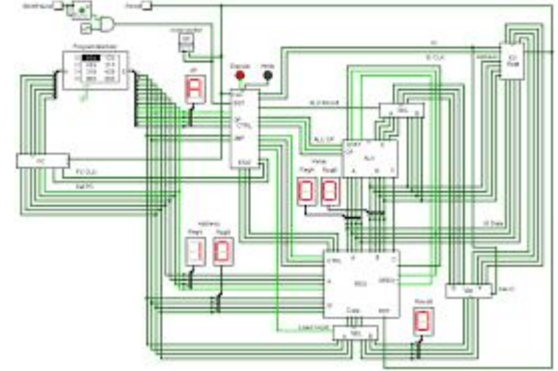
- There is a gap between these two worlds:
    - Requirements: what the system should do
    - Code: how the system implements the requirements

Software Models

- Goal: to fill this gap between requirements and code
- Document a solution to the problem defined by the requirements

Models are common in other engineering fields





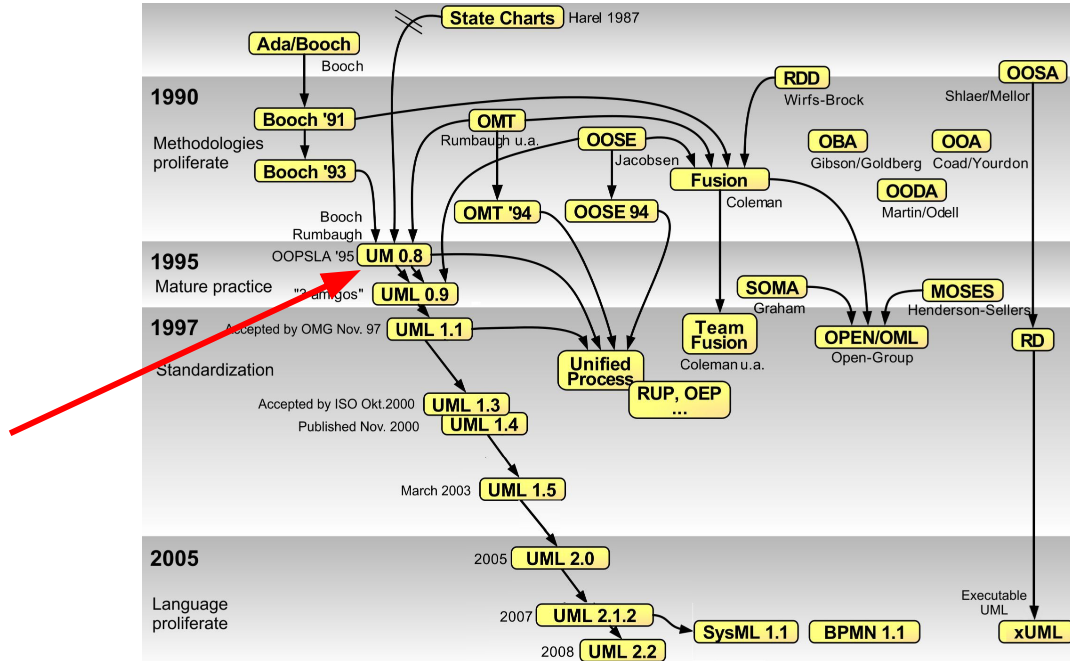
Thus, models were also proposed for software

Types of Software Models

- Formal: less common; will not be studied here
- Graphical: UML is the most common notation

UML: Unified Modeling Language

- Proposed in 1995 to unify other notations



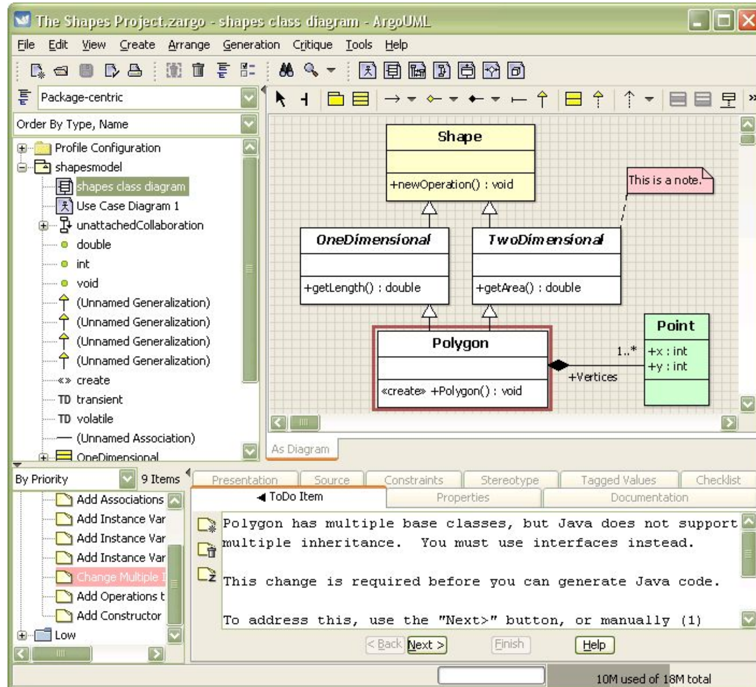
Source: Wikipedia

UML & RUP

- Most common process at the time: RUP
- Detailed documentation and planning
- Code written after months of specification and modelling

CASE (Computer-Aided Software Engineering) Tools

- Equivalent to CAD tools, but for Software Engineering



Main uses of UML

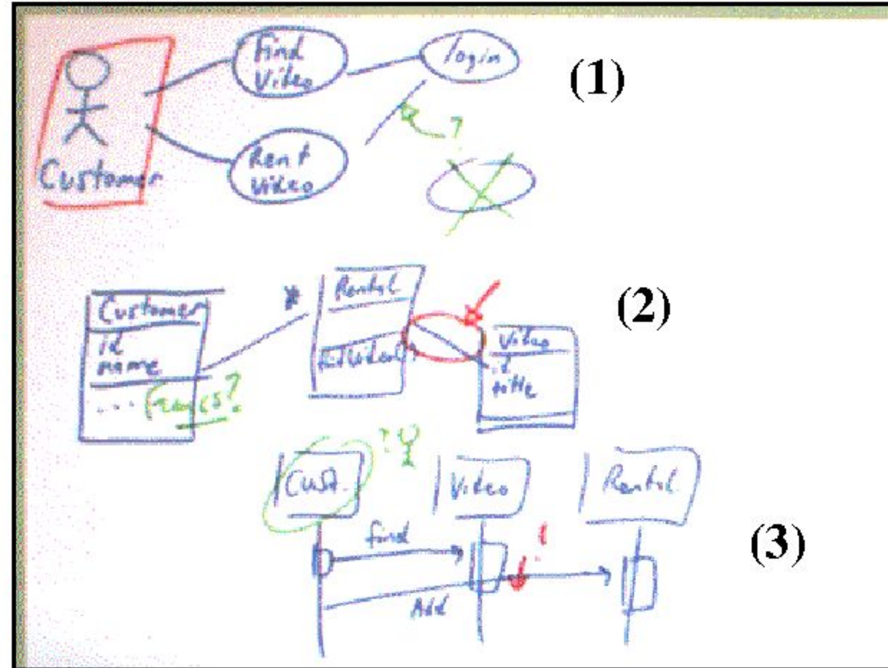
- As blueprint (detailed plans)
- As sketch (drafts, outlines)

In this course, we will study UML as Sketch

UML as Sketch

- Most common with agile methods
- To discuss or document parts of the code or design
- Lightweight and informal use of the notation
- The goal is not having a complete model (blueprint)

UML as Sketch



Q. Chen, J. Grundy, J. Hosking: SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. Software Practice and Experience, 2008

UML sketches are useful in

Forward and in Reverse Engineering

Forward Engineering

- Sketches are used to discuss design alternatives
- Before any line of code is implemented

Reverse Engineering

- Sketches are used to explain an existing code
- Context: software maintenance and evolution

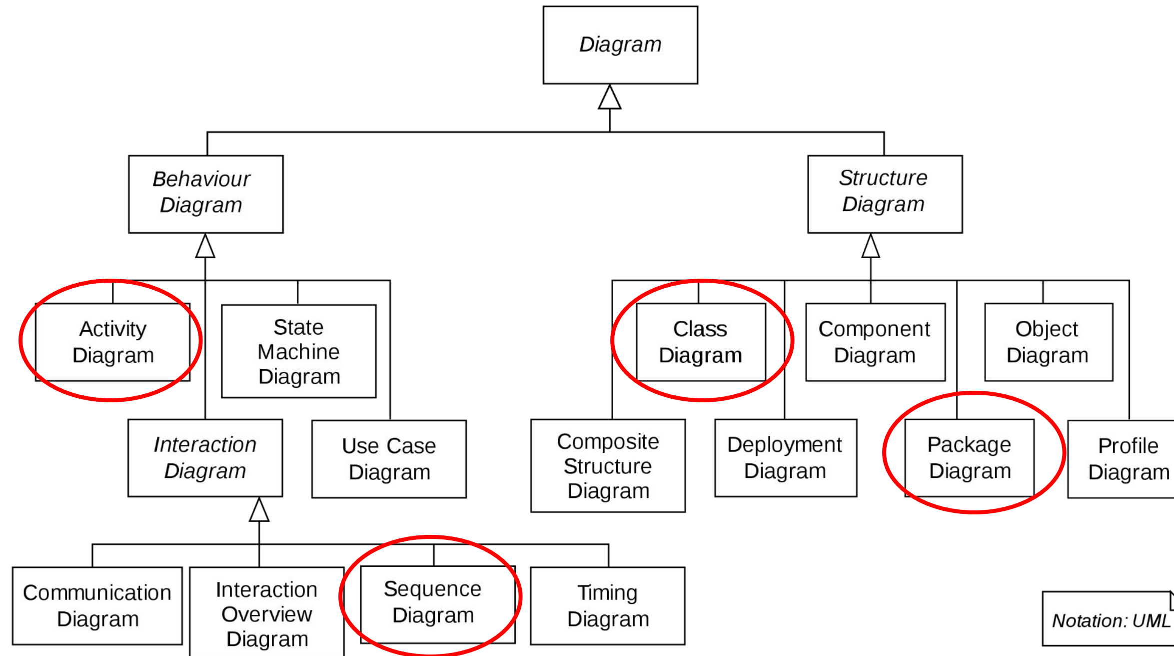
UML Diagrams

UML Diagrams

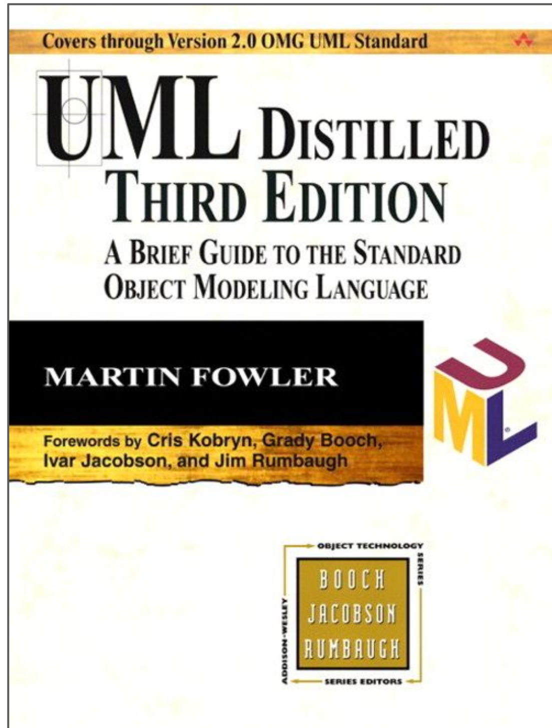
- Static Diagrams: model the structure of the code
- Dynamic Diagrams: model the execution of the code (the behavior of the system)

UML Diagrams

In red, the diagrams that we will study



We will use the UML version described in this book

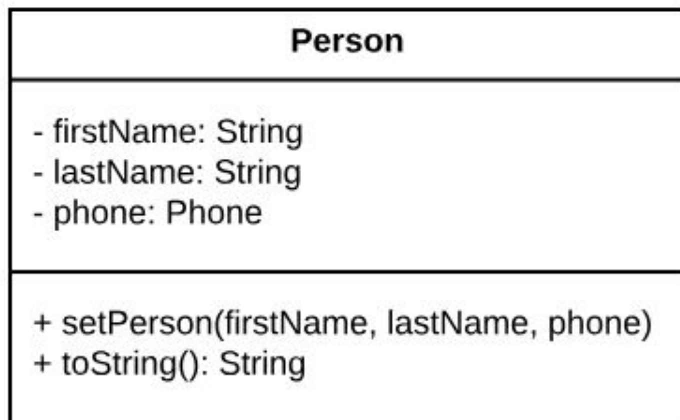


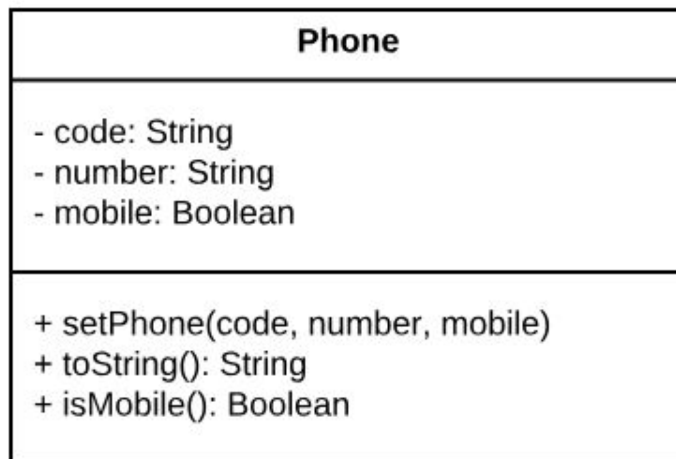
Class Diagrams

Generic format

[class name]
[attributes]
[methods]

Example

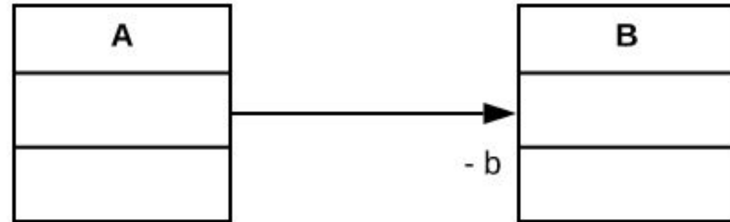




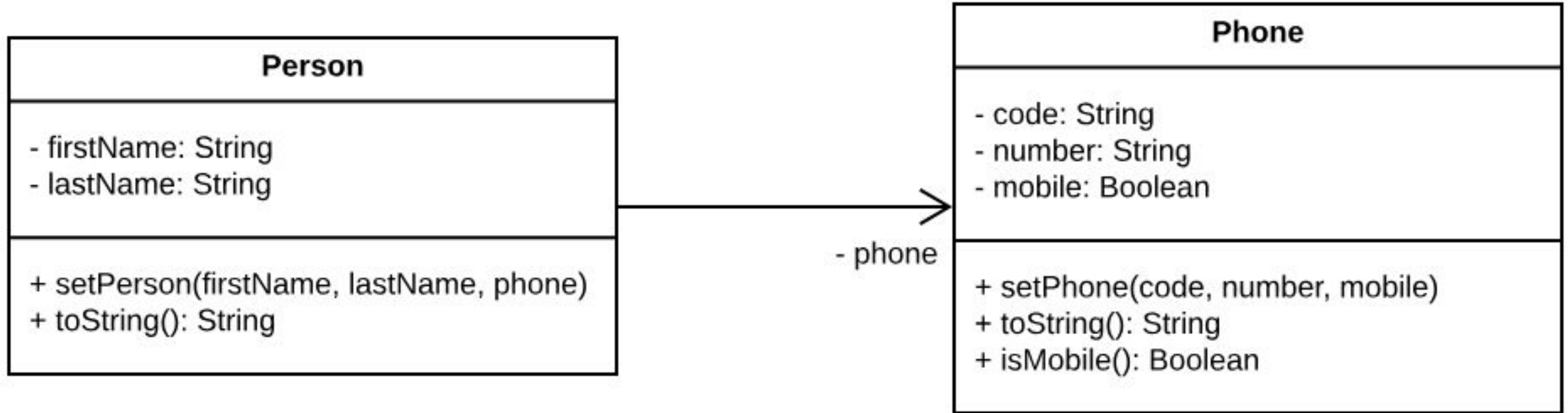
- : private
+: public

Association

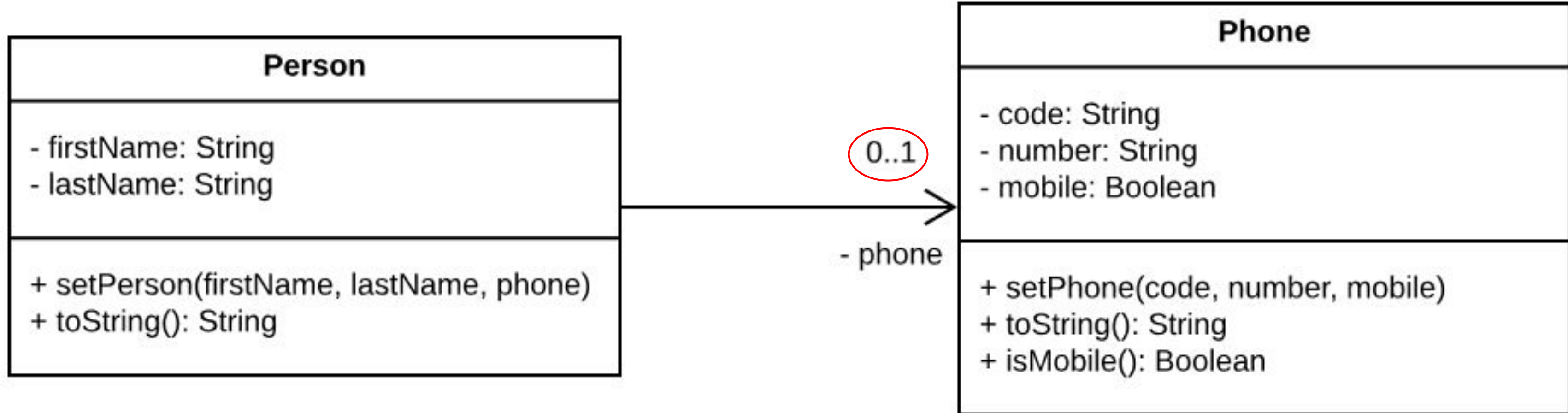
```
class A {
    ...
    private B b;
    ...
}


class B {
    ...
}
```



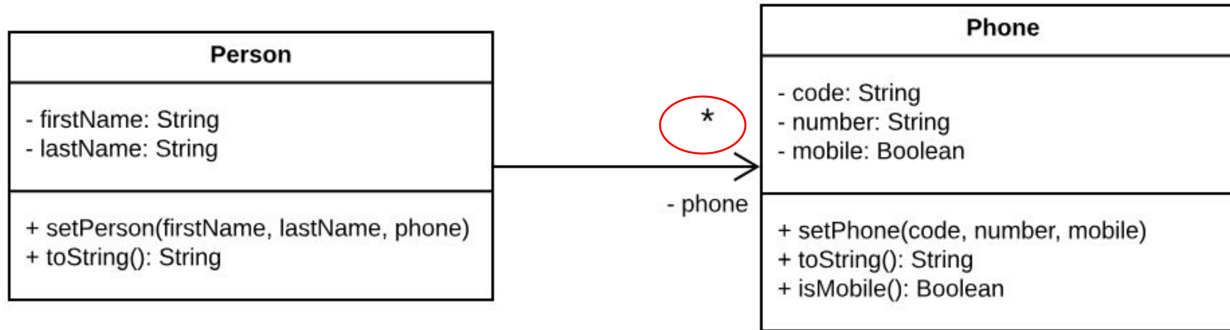
Association



Multiplicity



Multiplicity

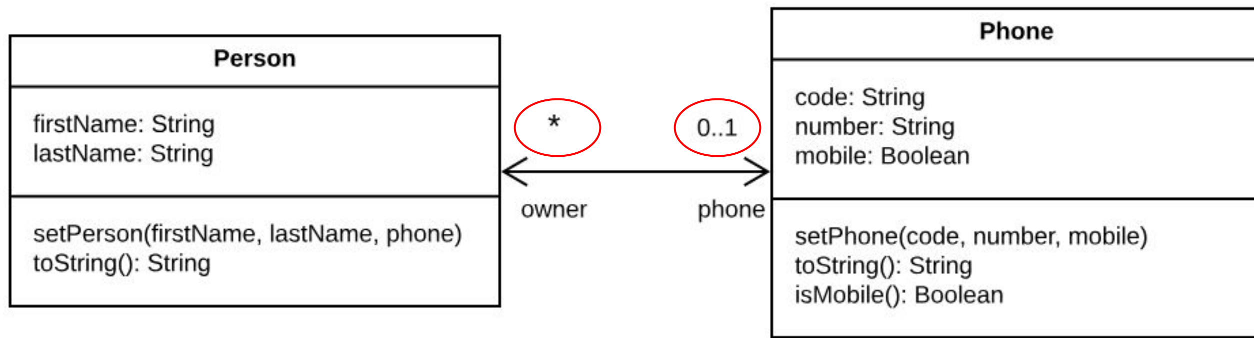



```
class Person {
    private Phone[] phone;
    ...
}

class Phone {
    ...
}
```

Other multiplicities

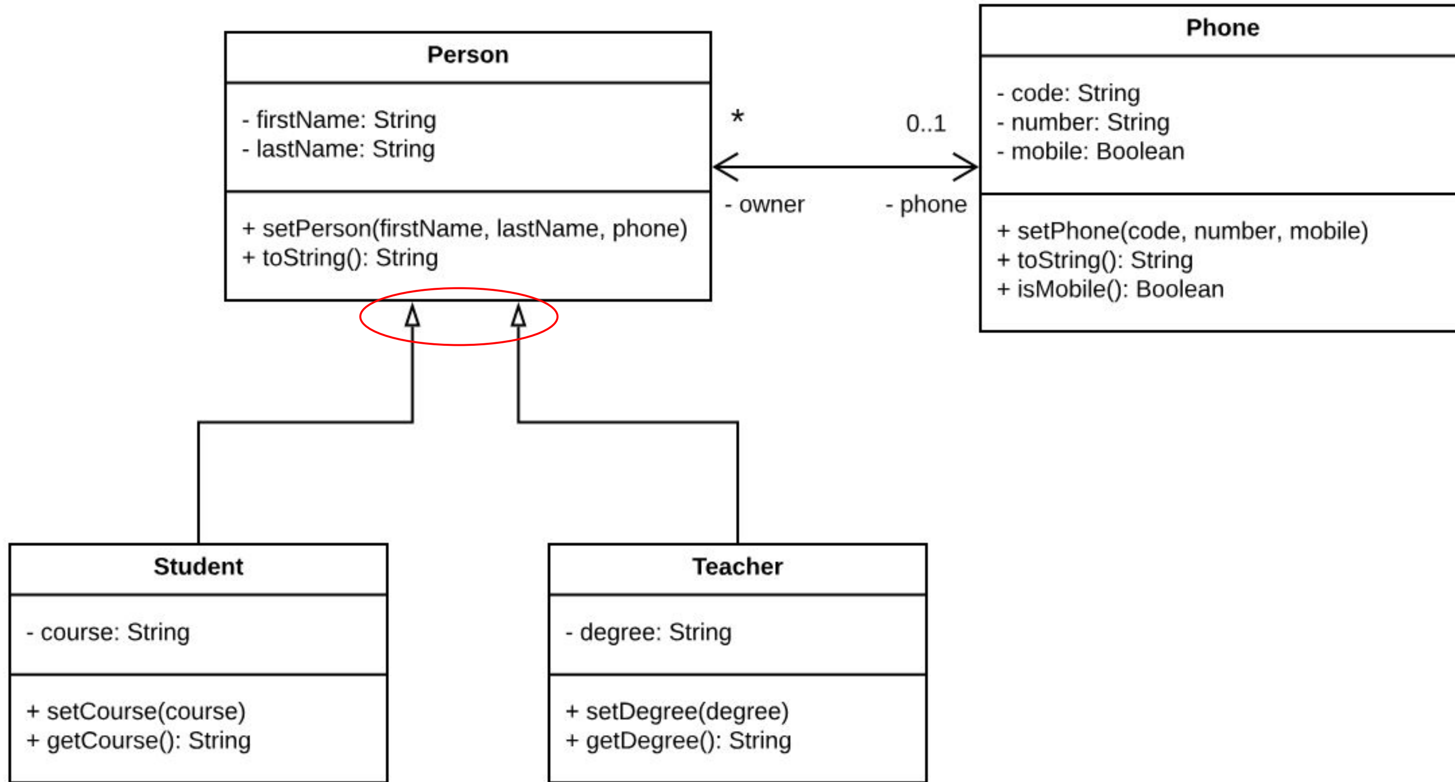
0..1
1
*
0..*
1..*
n

Bidirectional Associations




```
class Person {
    private Phone phone;
    ...
}

class Phone {
    private Person[] owner;
    ...
}
```

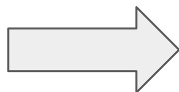
Inheritance

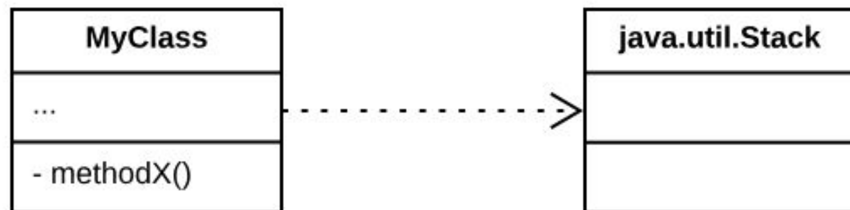


Dependencies (dashed arrows)

Relationship between classes, but not due to association or inheritance

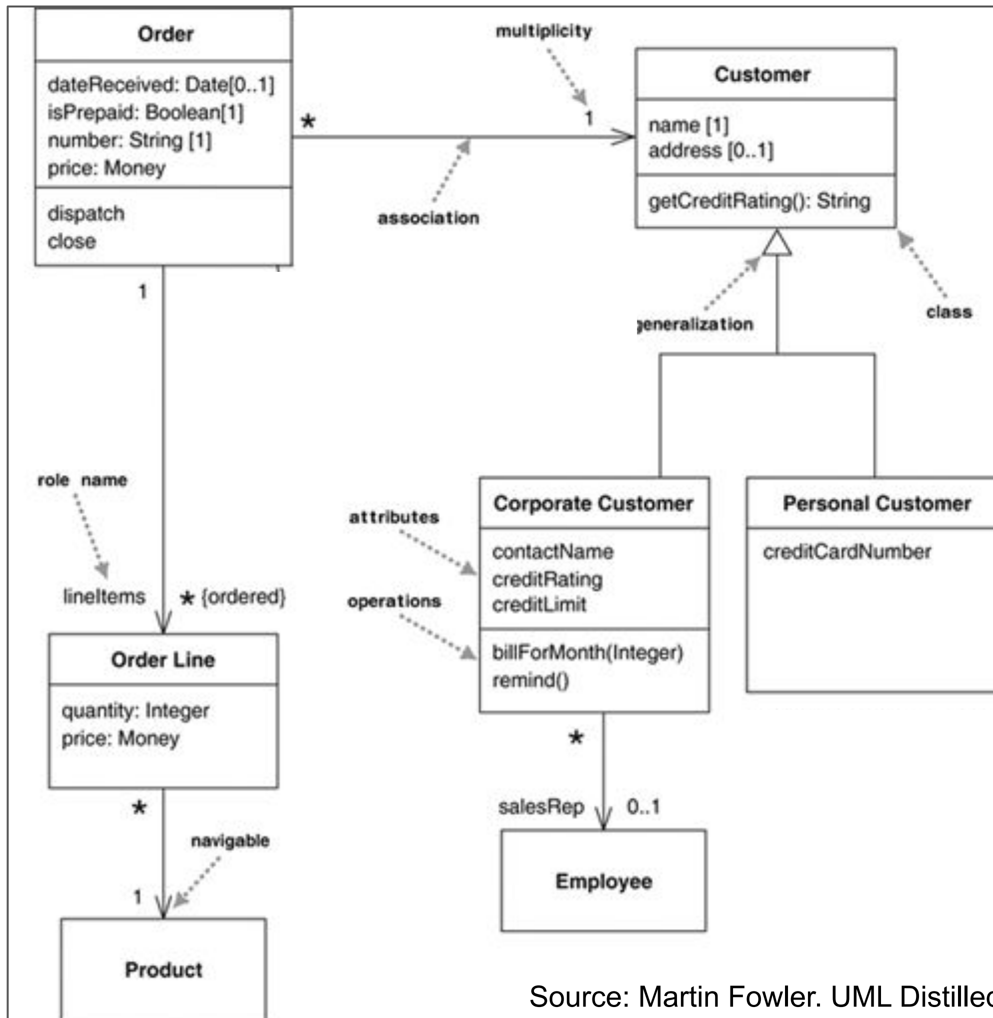
```
import java.util.Stack;

class MyClass {
   ...
   private void methodX() {
      Stack stack = new Stack();
      ...
   }
}
```





Dependencies do not have multiplicity information

Exercises



1. Study and try to understand this class diagram.

2. Model using class diagrams. The classes are in a different font.

- `BankAccount` has exactly one `Customer`. But a `Customer` can have several `BankAccount`, with bidirectional navigation.
- `SavingsAccount` and `SalaryAccount` are subclasses of `BankAccount`.
- The `BankAccount` code declares a local variable of type `Database`.
- An `OrderItem` refers to a single `Order` (without navigation). An `Order` can have several `OrderItem` (with navigation).
- The `Student` class has attributes `name`, `ID`, `course` (all private); and methods `getCourse()` and `cancelEnrollment()`, both public.

3. Create class diagrams for the following programs:

(a)

```
class HelloFrame {
   public static void main(String[] args) {
     JFrame frame = new JFrame("Hello!");
     frame.setVisible(true);
   }
}
```

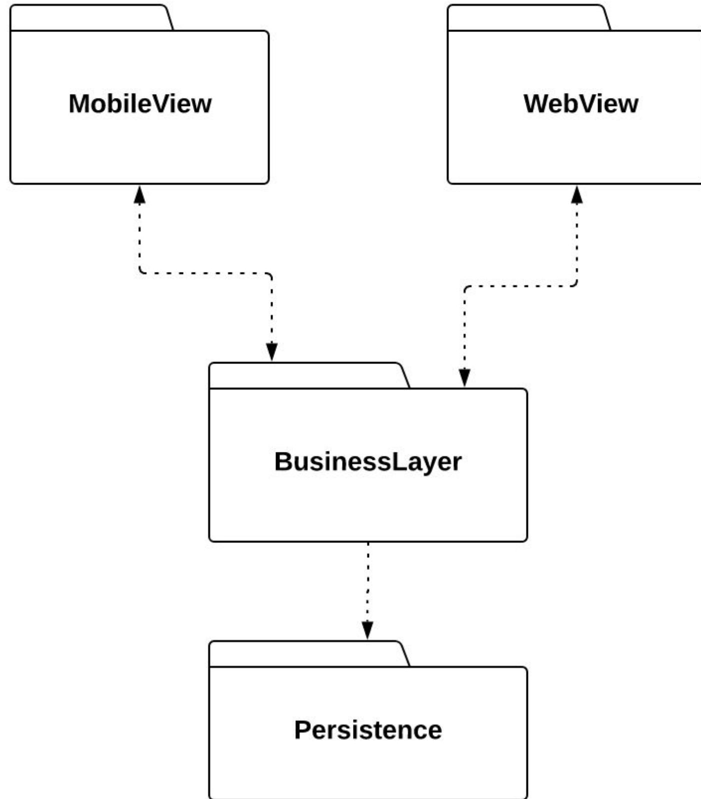
(b)

```
class HelloFrame extends JFrame {
   public HelloFrame() {
     super("Hello!");
   }
   public static void main(String[] args) {
     HelloFrame frame = new HelloFrame();
     frame.setVisible(true);
   }
}
```

Package Diagrams

Package Diagrams

Package Diagrams

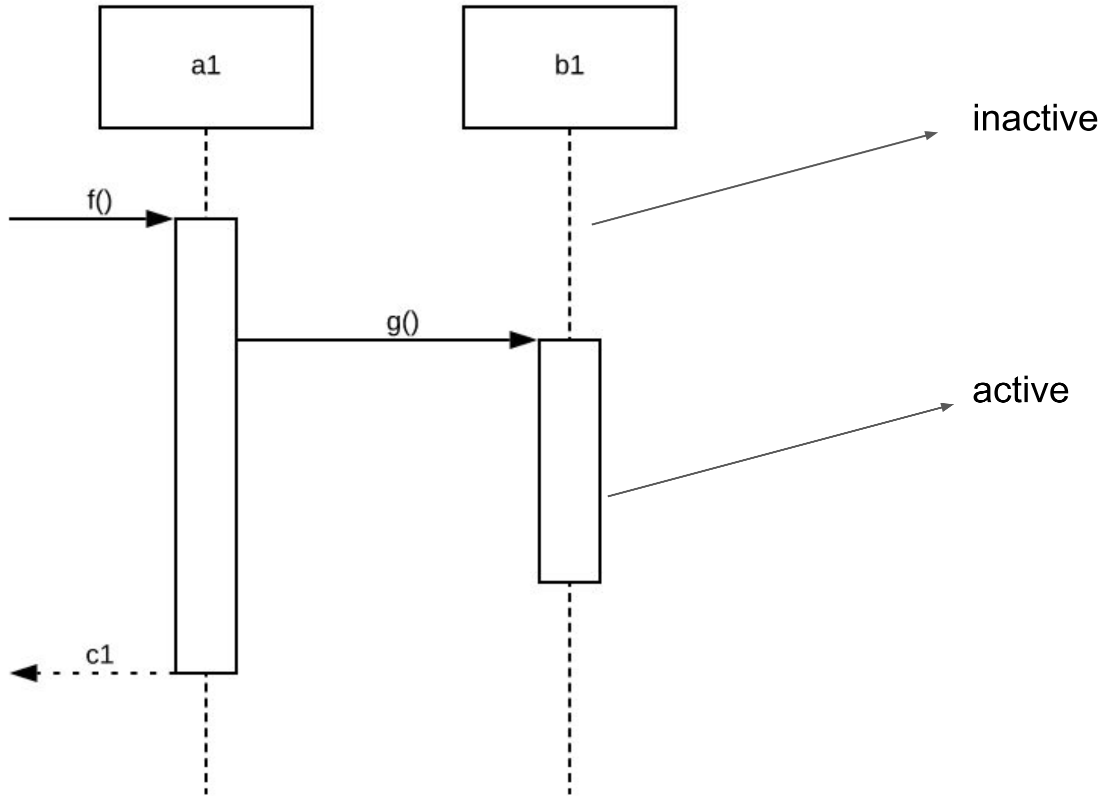


Sequence Diagrams

Sequence Diagrams

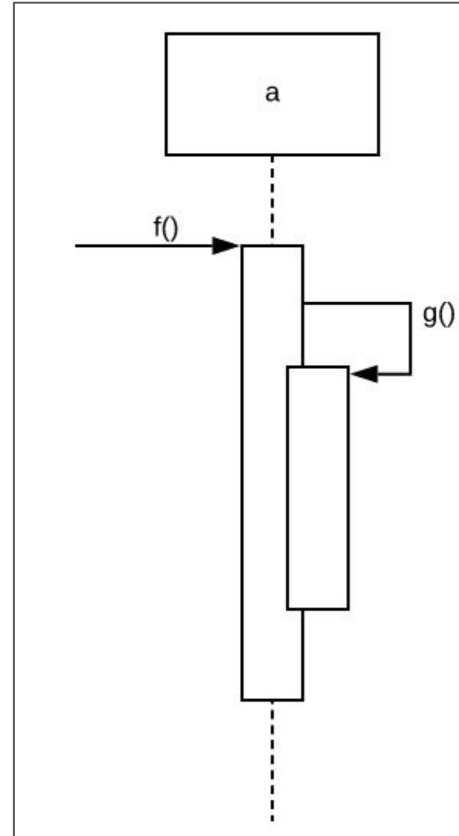
- Behavioral or dynamic diagrams that model:
  - Some objects of a system
  - Methods they execute

Example 1



Example 2

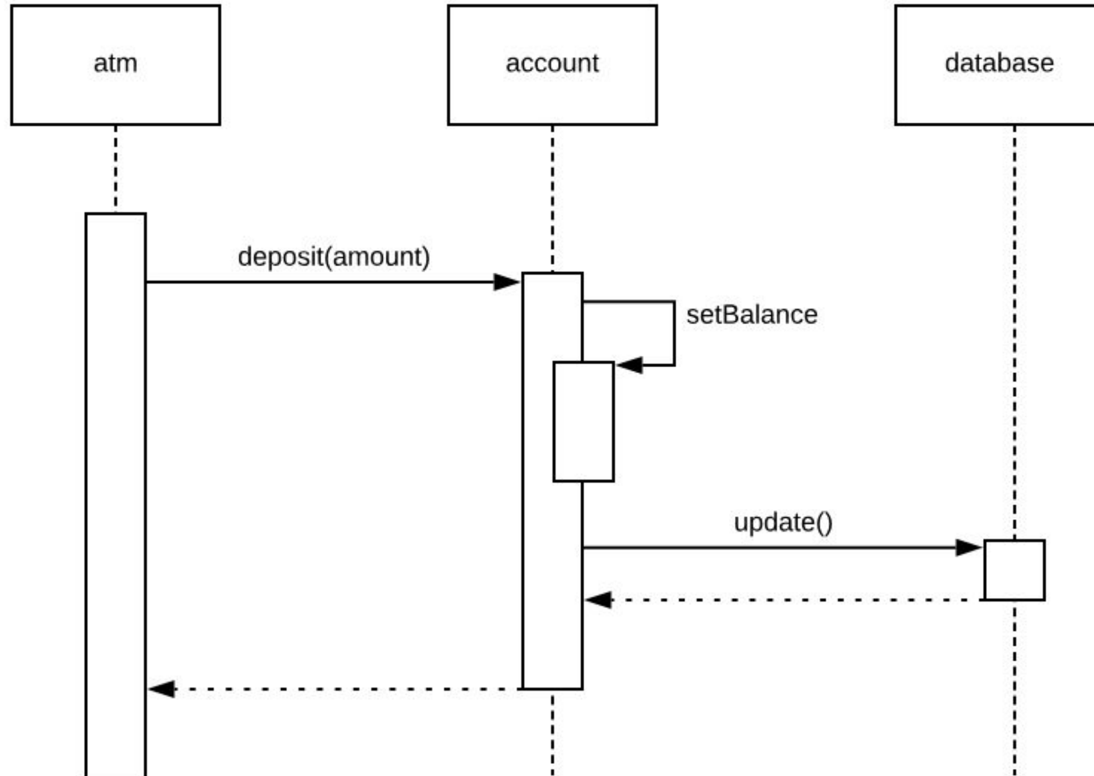
```
class A {

  void g() {
    ...
  }

  void f() {
    ...
    g();
    ...
  }

  main() {
    A a = new A();
    a.f();
  }
}
```



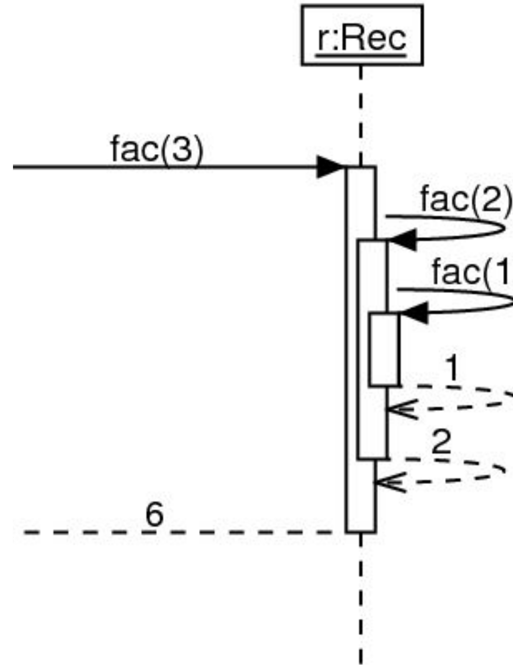
Return arrows

- They can be omitted when:
  - The return is not important
  - The method does not return any value (void)
- Fowler: "Some people use returns for all calls, but I prefer to use them only where they add information; otherwise, they simply clutter things".

Example 3



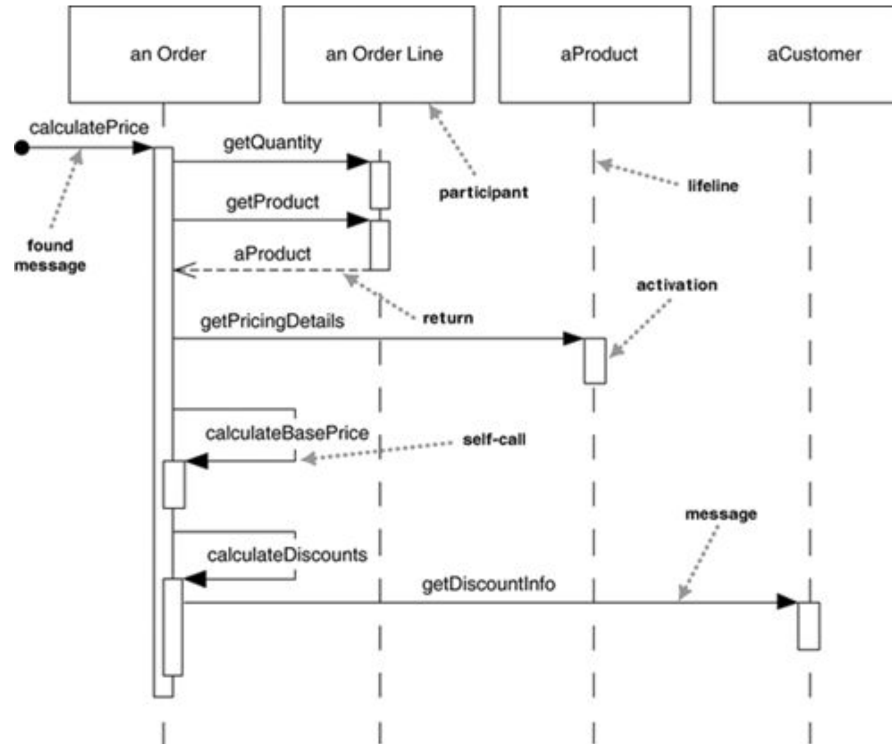
Example 4



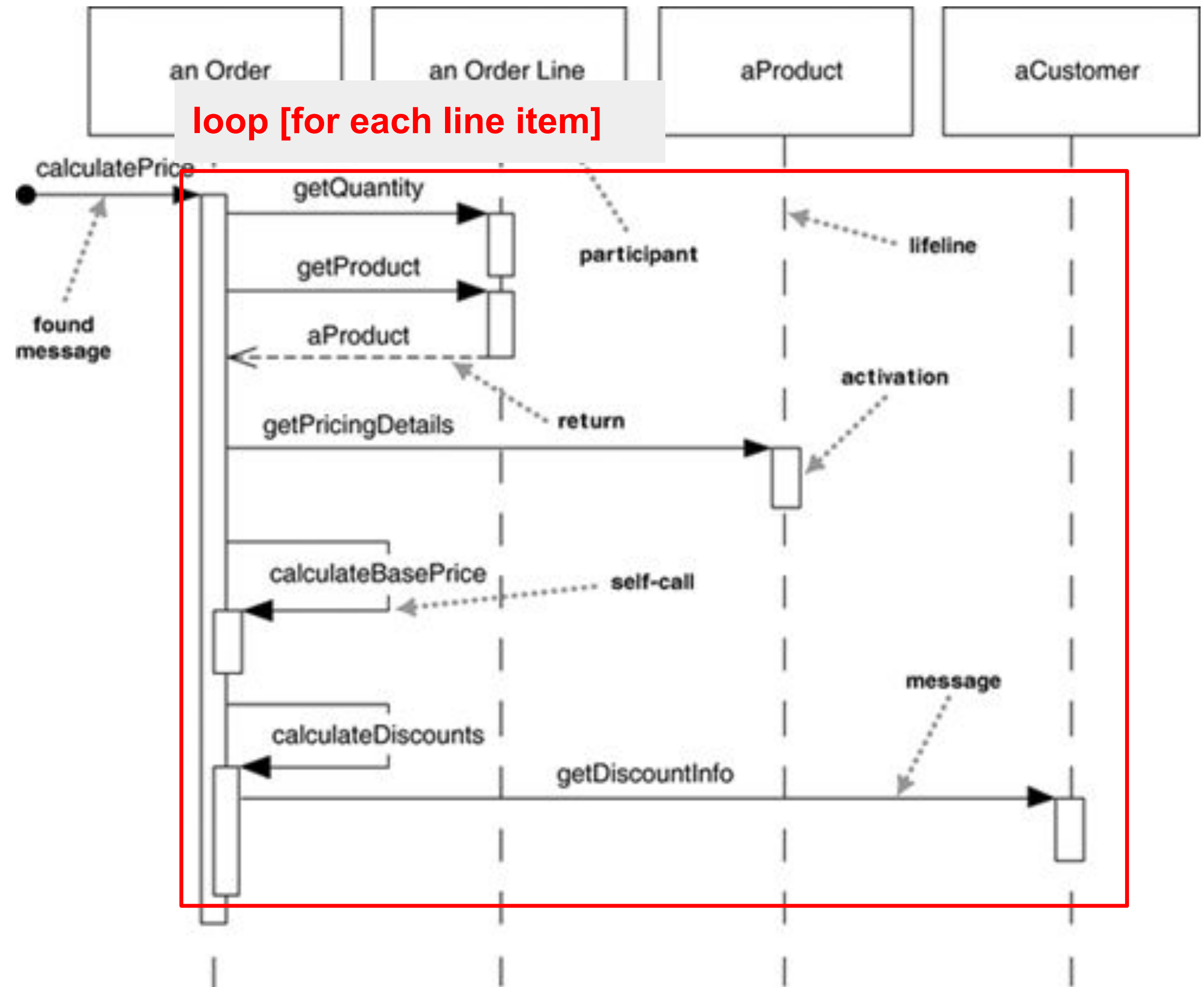
This is <u>not</u> an interesting use of sequence diagrams

Exercises

This sequence diagram should represent the method calls required to calculate the total value of an Order, comprising multiple Order Lines, each linked to a Product along with a quantity. However, why does the diagram fail to accurately represent this process?



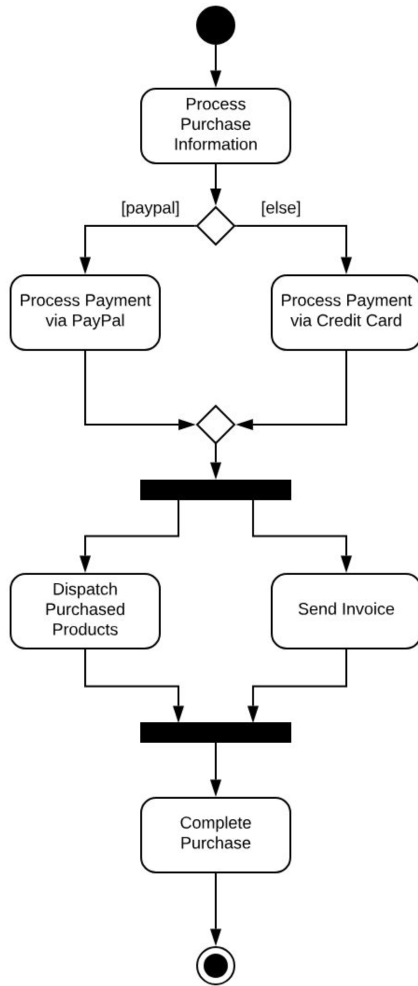
Fixing the diagram



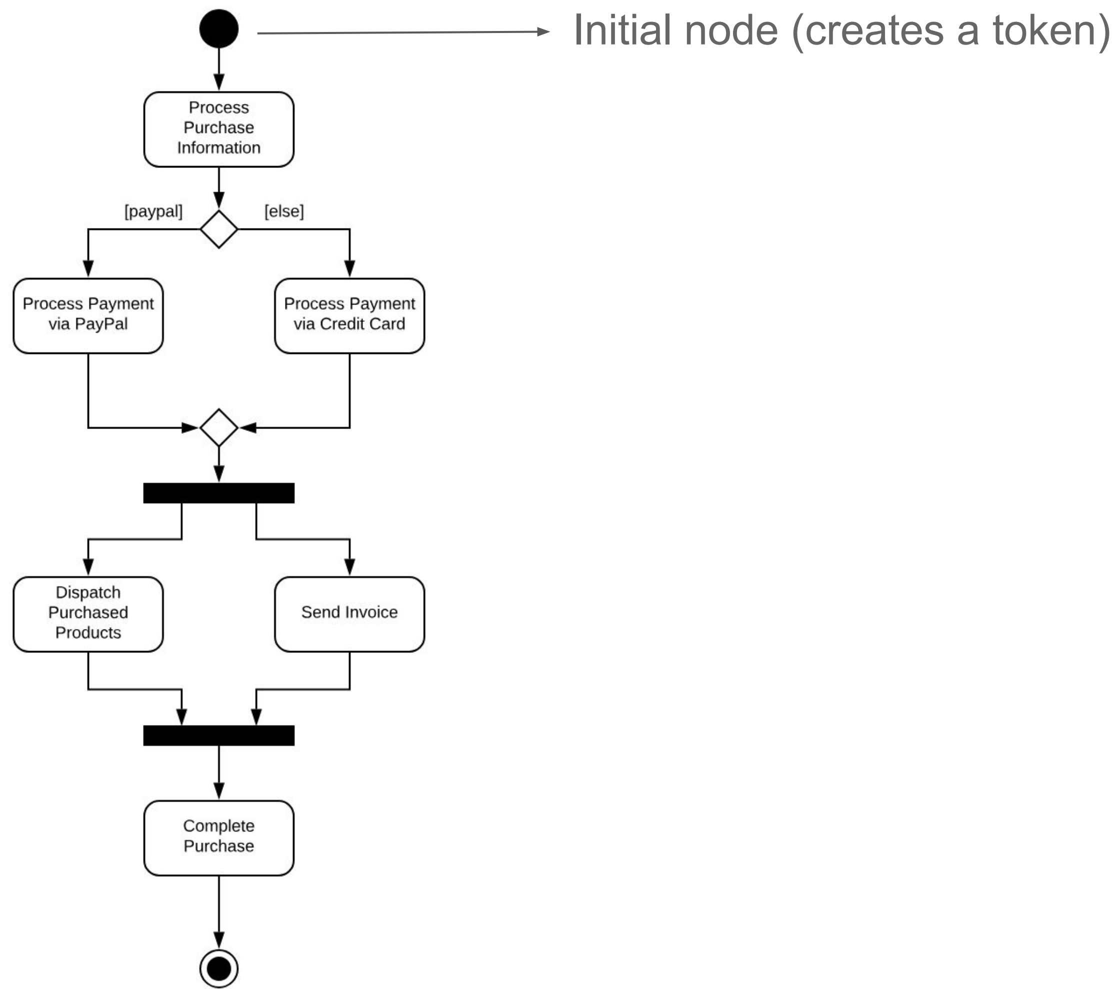
Activity Diagrams

Activity Diagrams

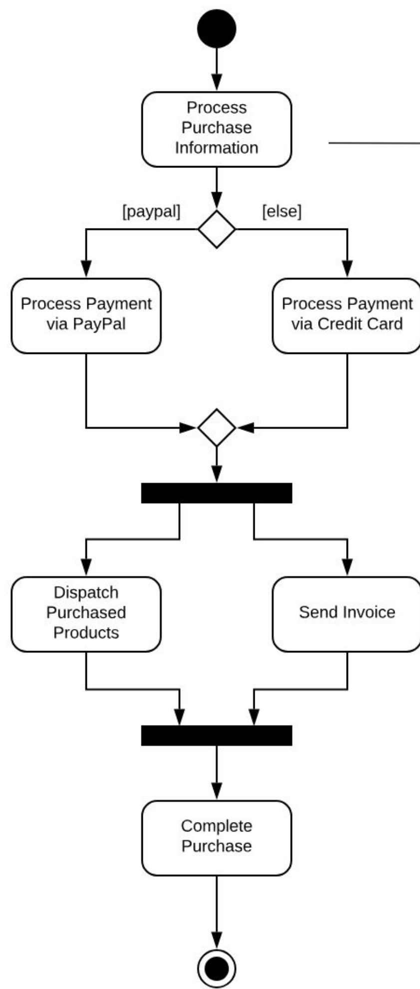
- Behavioral or dynamic diagrams
- Model a business process or workflow



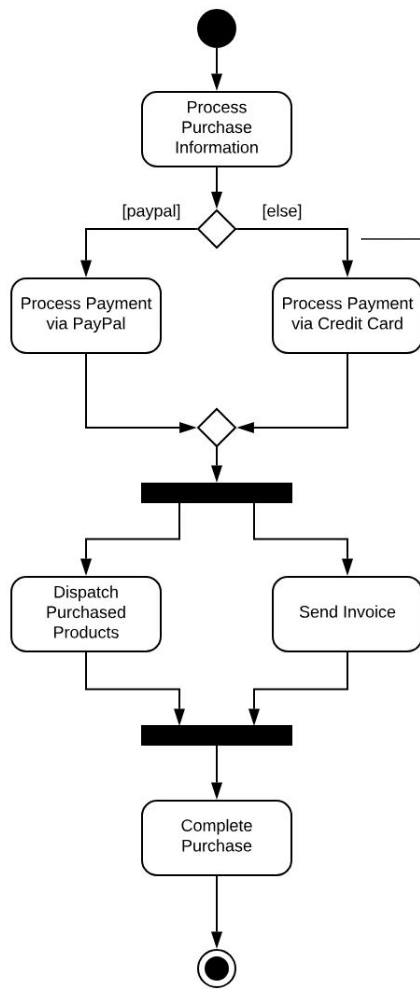
Assume that there is a token that moves through the nodes of the diagram.



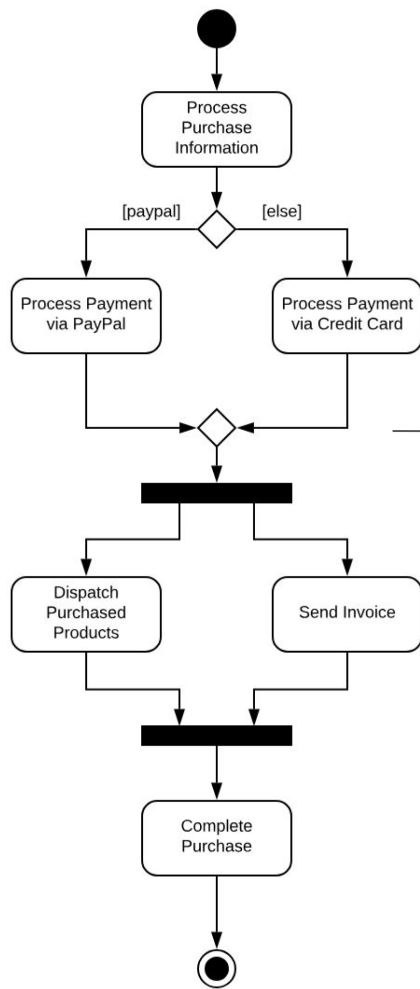
Initial node (creates a token)



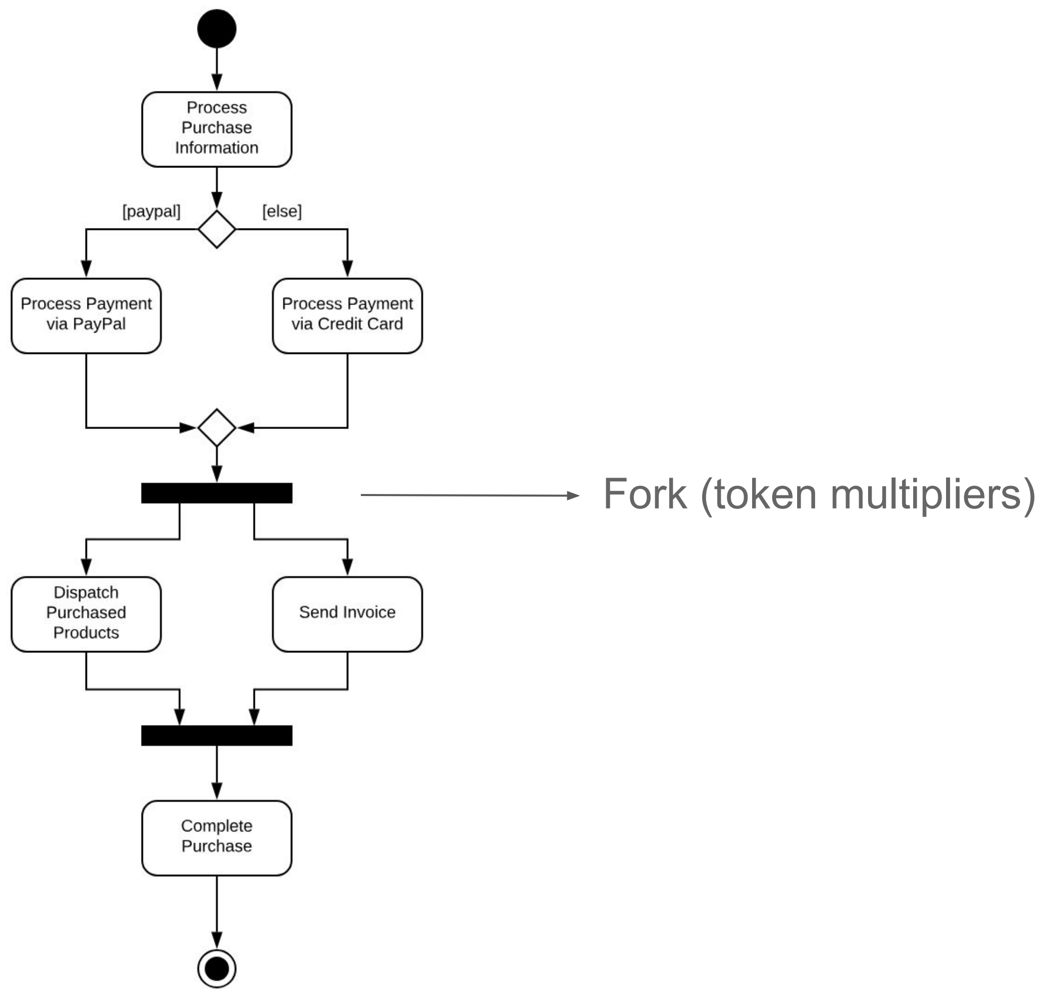
Action (passes token from input to output flow)



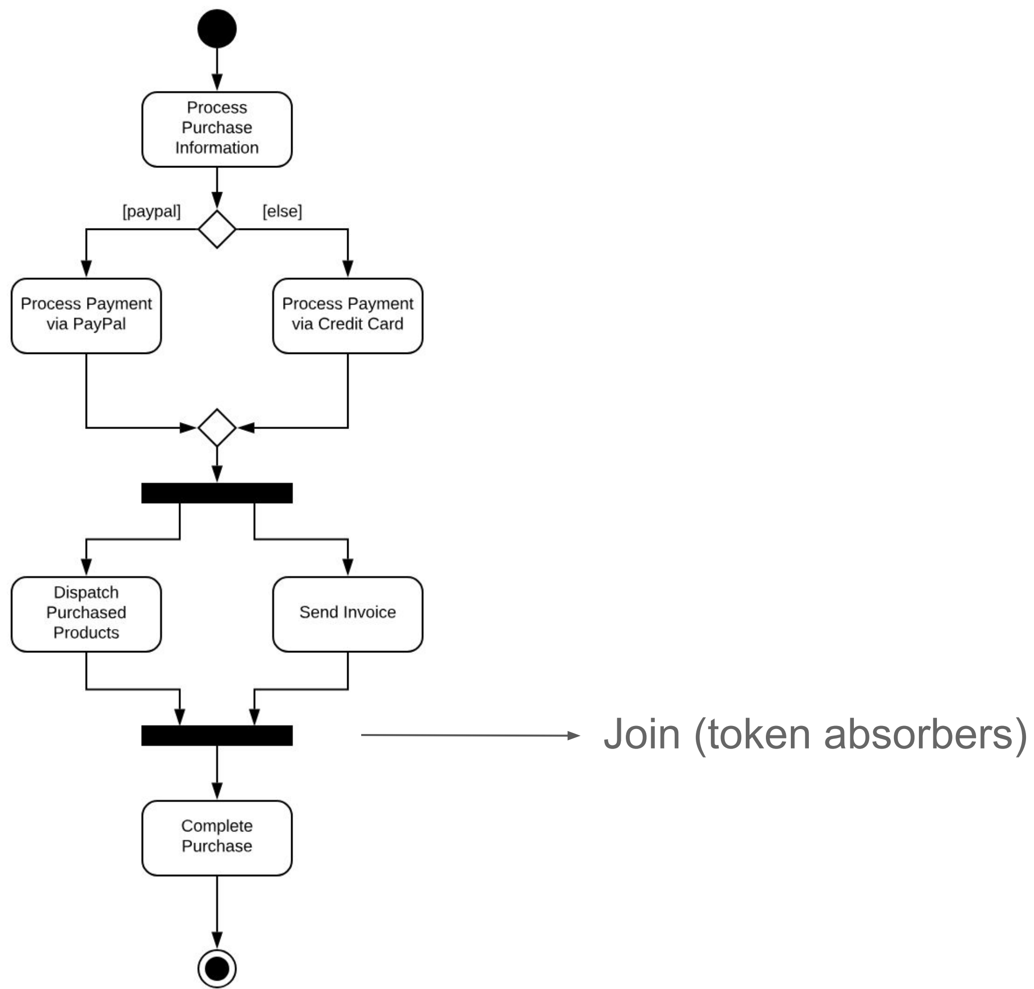
Decision (decides to which output flow it will pass the token)



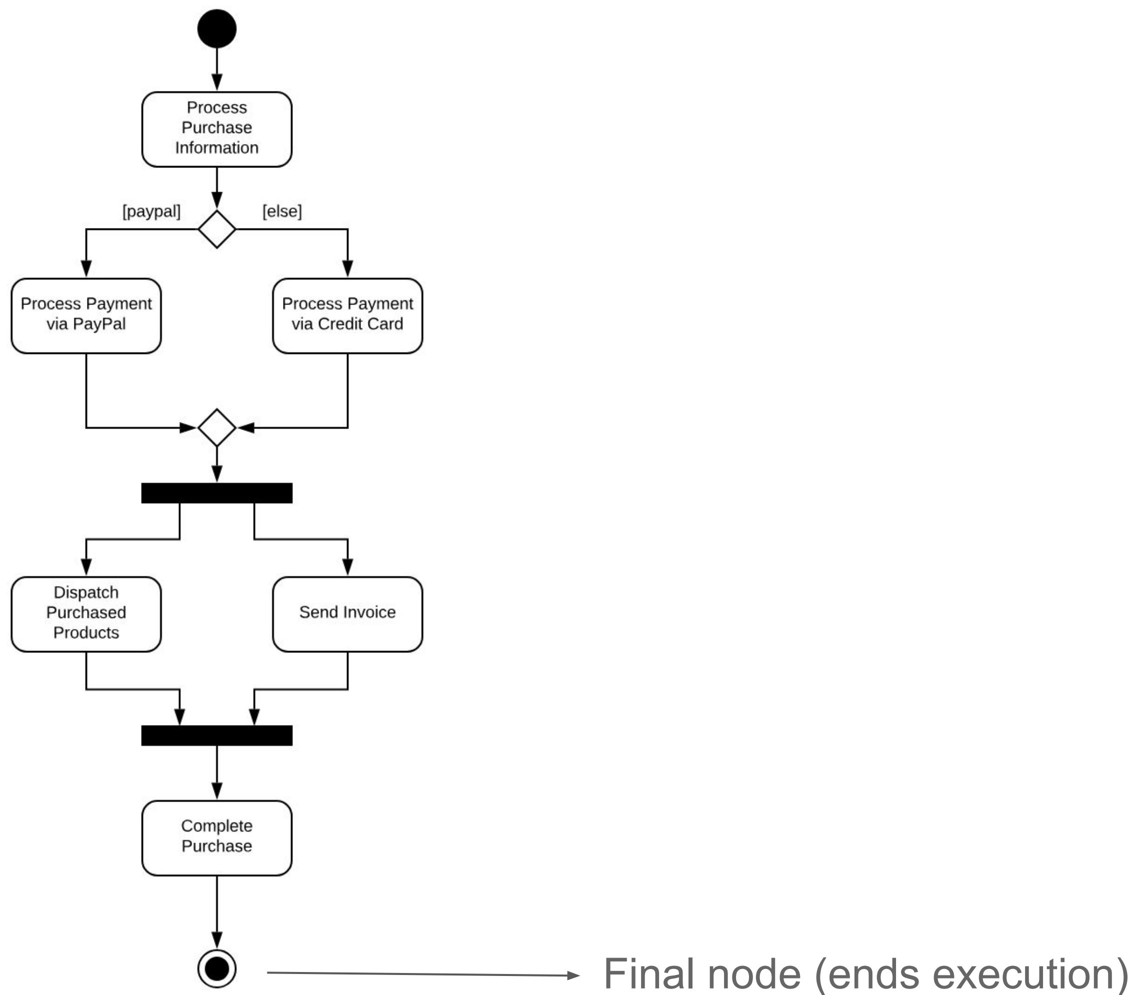
Merge (when token arrives at one of the inputs, passes it to the output)



Fork (token multipliers)



Join (token absorbers)



Final node (ends execution)

Exercises

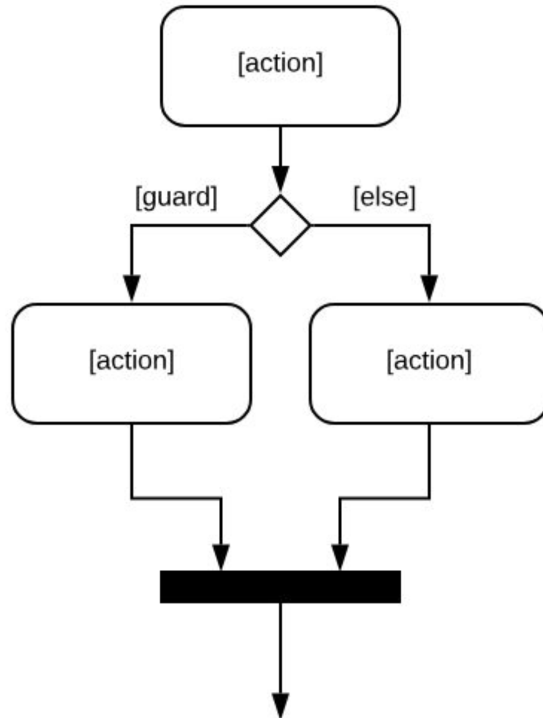
1. Model in UML using a class diagram.

```
class Computer {
    ...
    private List<Keyboard> keybord;
    ...
}
```

```
class Keyboard {
    ...
}
```

Note: `Keyboard` does not have a reference to `Computer`. However, in our system, we know that any `Keyboard` is always connected to exactly one `Computer`.

2. What is the error in the following activity diagram? Change the diagram to fix this error and to reflect the intention of the software designer.



End