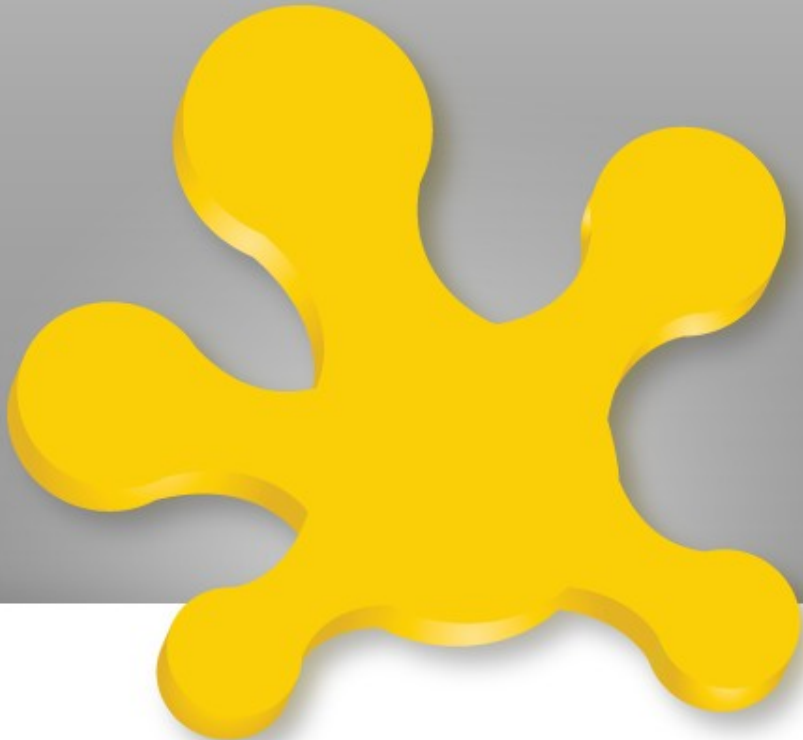


Bruno Bossola

Object Oriented for nonbelievers



About me

- C developer since 1988
- Java developer since 1996
- XP coach during 2000-2001
- Coordinator and co-founder of JUG Torino
- Sun Java Champion since 2005
- Speaker at Javapolis, Jazoon, Geecon!
- Running his own company!



Agenda

- Admit that we don't really know what Object Oriented is about
- Admit that we don't really know how to execute it
- Try to understand what really Object Oriented is!

Let's start with a problem!

- Create a payroll system
- Should have a web interface
- Should connect to a relational DB

Let's start with some simple requirements!

Requirements!

- *some employees are payed every two weeks, some every month*
- *sellers are payed by a fixed amount plus a percentage on sales*
- *some employees are paid based on the hours worked, the others receive a fixed rate*
- *employees may choose to be payed by cheque or bank money transfer*

But at the end of the day...

...who cares!

We need to think about the architecture!

- how do we manage web access?
- how do we manage persistence?

...isn't always like that??

Web access

- Servlet?

Web access

- ~~Servlet~~
- JSP+Taglibs?

Web access

- ~~Servlet~~
- ~~JSP+Taglibs~~
- Velocity?

Web access

- ~~Servlet~~
- ~~JSP+Taglibs~~
- ~~Velocity~~
- Struts2?

Web access

- ~~Servlet~~
- ~~JSP + Taglibs~~
- ~~Velocity~~
- ~~Struts2~~
- JSF + Ajax?

Web access

- ~~Servlet~~
- ~~JSP + Taglibs~~
- ~~Velocity~~
- ~~Struts2~~
- ~~JSF + Ajax~~
- GWT! (cool!)

Persistence

- JDBC?

Persistence

- ~~JDBC~~
- DAO?

Persistence

- ~~JDBC~~
- ~~DAO~~
- Entity Beans?

Persistence

- ~~JDBC~~
- ~~DAO~~
- ~~Entity Beans~~
- Hibernate?

Persistence

- ~~JDBC~~
- ~~DAO~~
- ~~Entity Beans~~
- ~~Hibernate~~
- Ibatis?

Persistence

- ~~JDBC~~
- ~~DAO~~
- ~~Entity Beans~~
- ~~Hibernate~~
- ~~ibatis~~
- JPA!

A-ehm...

- Where exactly did we use Object Oriented?
- Ah, yeah, we are using Java
- So we are doing OO, aren't we?
- And we are also using those cool frameworks, how can it be wrong??

Sorry...

- ...nothing about OO here

...man, who cares!

Why OO was born?

- What exactly is it for?
- Does it give us real advantages?

Why OO was born?

- allows you to maintain traceability of requirements between the different models
 - analysis
 - design
 - implementation
- allows you to keep the complexity of your system low
- allows you to describe a complex system in simple terms

What are requirements?

- the desired behaviors of the system
- the entire set of requirements will define what the final system will do
- they may be expressed in different forms

What is analysis?

- the act of translating requirements in terms used to describe a software system
- from requirements you define the component that may execute the needed actions
- in OOA those are objects and their collaborations

So OOA is...

- defining abstractions
- defining high-level behaviors
- ...in short terms, is **what!**

And OOD is...?

- based on abstractions identified during OOA, you define how the high level behaviors are delivered
- it hides any concrete implementation
- ...in short terms, is **how!**

Let's restart!

- Better start *doing* something!
- Forget about the framework crap and let's do some plain old OOA and OOD!
- No, this is not going to be easy...

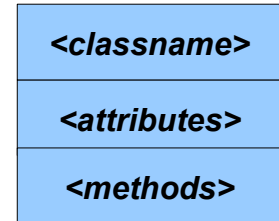
UML?

- Fancy UML anyone?
- Well, as a bonus, you will get used to it!
- UML is not difficult, it's just... a nice way to express some concept, great way of communicating
- As soon as you trash your UML diagrams, you're safe!

UML Basic crash course!

UML

- This is a class:



- This is an object



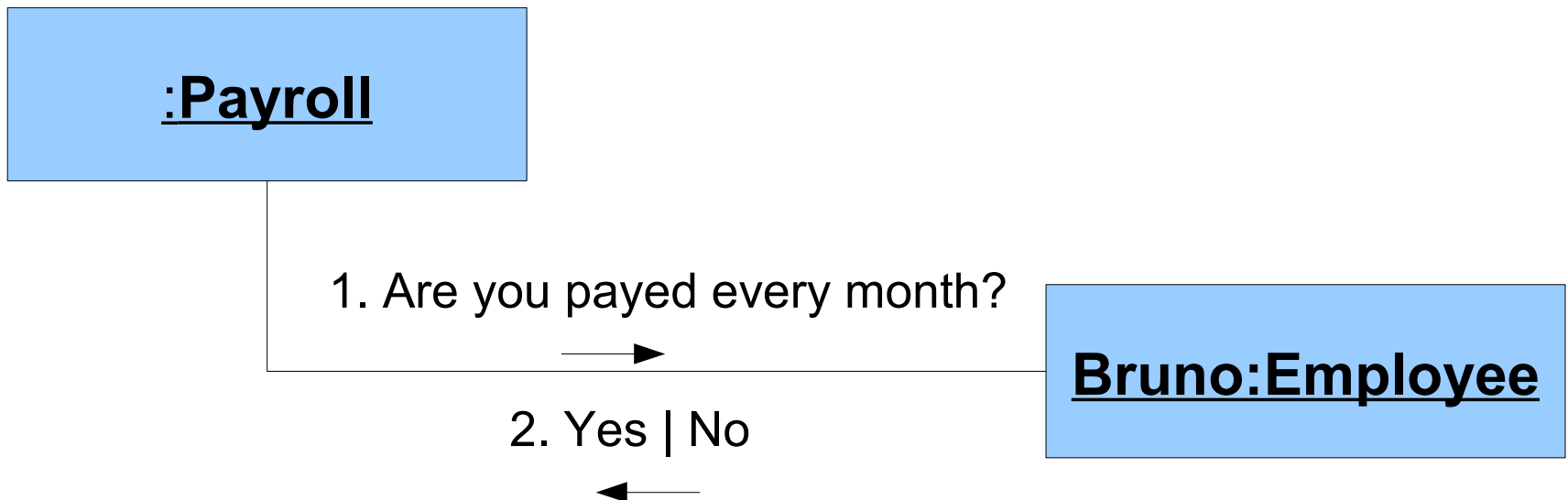
- This is a link



Thank you for attending the
UML Basic crash course!

Requirement #1

- *“some employees are payed every two weeks, some every month”*

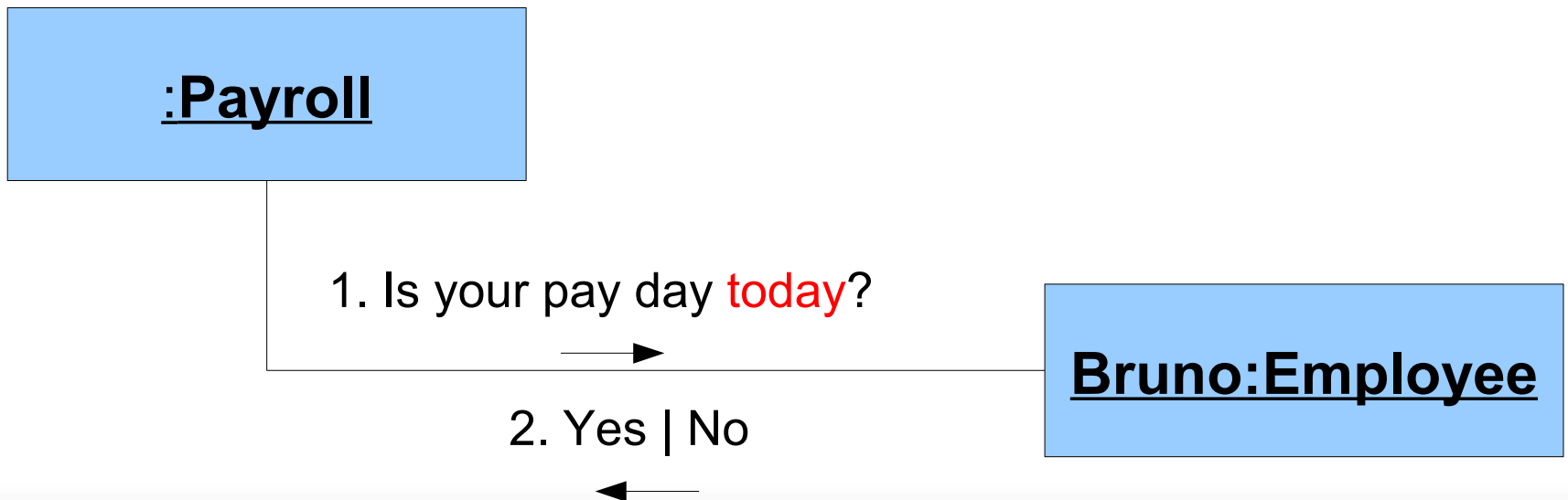


How do we do Analysis?

- That seems a bit over simplified :)
 - We need to understand the ratio behind the requirement!
- We need to find abstractions and behaviors
- My favourite way is to identify those things:
 - **what will not change?**
 - **what will probably change?**

Analysis #1

- What will not change: *every employee is payed based on some schedule*
- What will change: *the schedule*



How do we do Design?

- Okay, we cannot write it like *that*
- We need to “implement” the analysis model
- My favourite way is:
 - **decouple any responsibility**
 - (...or apply DIP continuously, but this is another story!)

UML Break!

UML Break #2

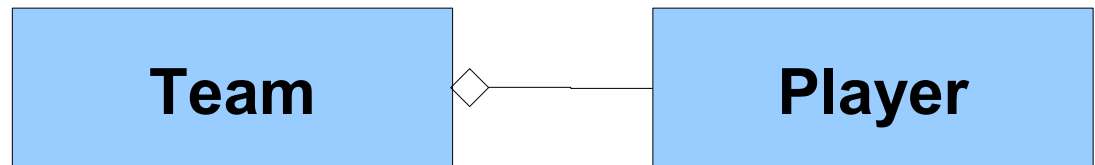
horizontal relationships

- dependency *nobody cares anymore :)*

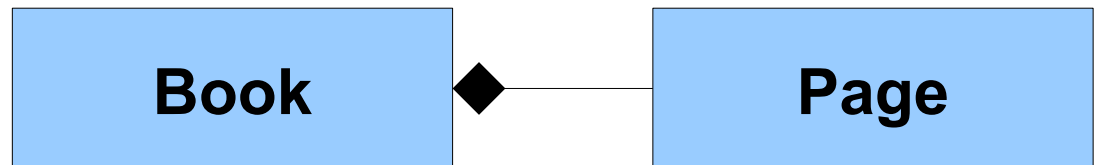
- association



- aggregation



- composition



UML Break #2

horizontal relationships

- Identify relationships types may be tricky...
- But let me give you a word of consolation: do you know what UML authors said about aggregation?

UML Break #2

horizontal relationships

“...if you don’t understand don’t use it.”

*talking about “aggregation”,
from UML 0.8 official docs*

Back to reality!

Design #1

- It seems to me that the Employee knows about some kind of schedule
- We need to split the responsibility



- But wait... what about code here?

Design #1 (attempt)

```
public class PaySchedule {
    public static int MONTHLY = 0;
    public static int BIWEEKLY = 1;

    public int schedule;
    public PaySchedule (int aSchedule) {...}

    public boolean isPayDay(Date today) {
        if (schedule == MONTHLY && today...)
            return true;
        else if (schedule == BIWEEKLY && today...)
            return true;
        else
            return false;    // doh!
    }
}
```

Design #1 (attempt)

- Okay, I probably choose the worst method
- But I could have made it cool!
 - make schedule an inner class
 - make schedule an Enum
 - introduce an external parser in Python (WTF??)
- What's the problem in this design?

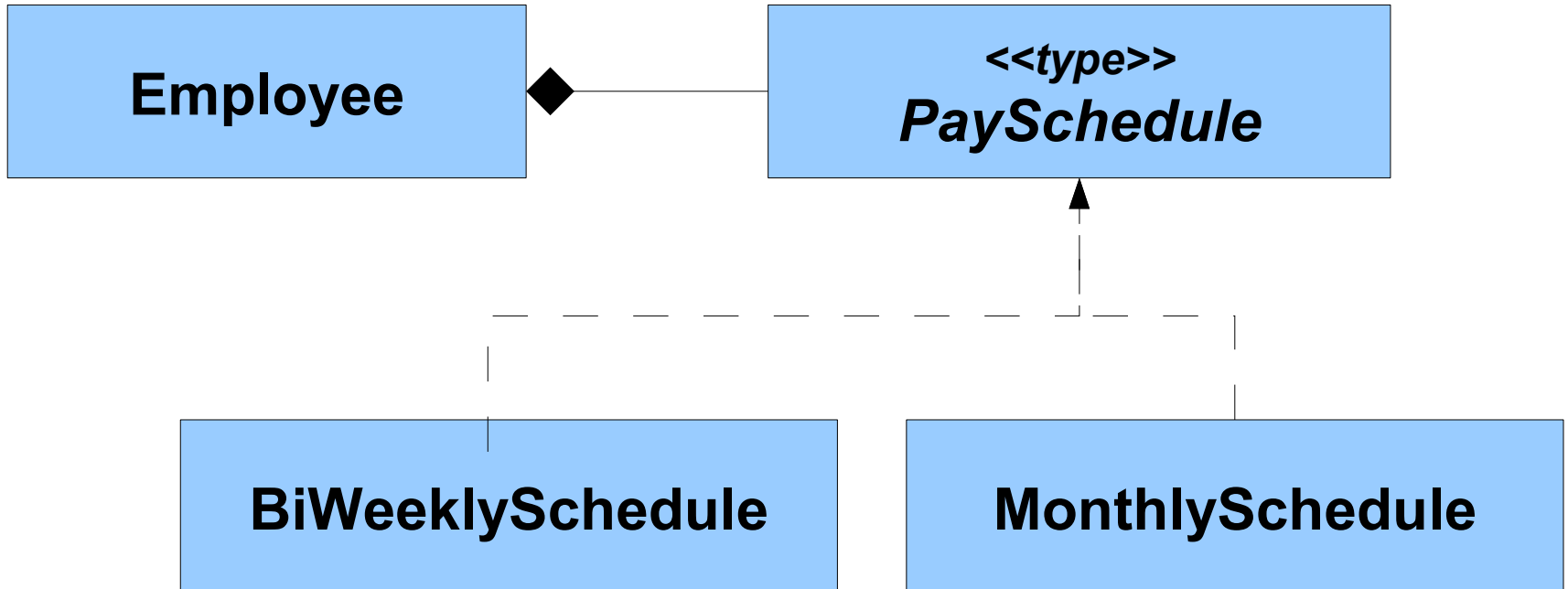
Design #1 (attempt)

- The s**t that I wrote is fully respecting the analysis model
- But it's **inflexible**! What if I need to add another schedule? As we agreed in analysis, the schedule may change!
- You can definitely have a brilliant analysis model with a really poor design!

Design #1 (attempt)

- Suggestions?

Design #1



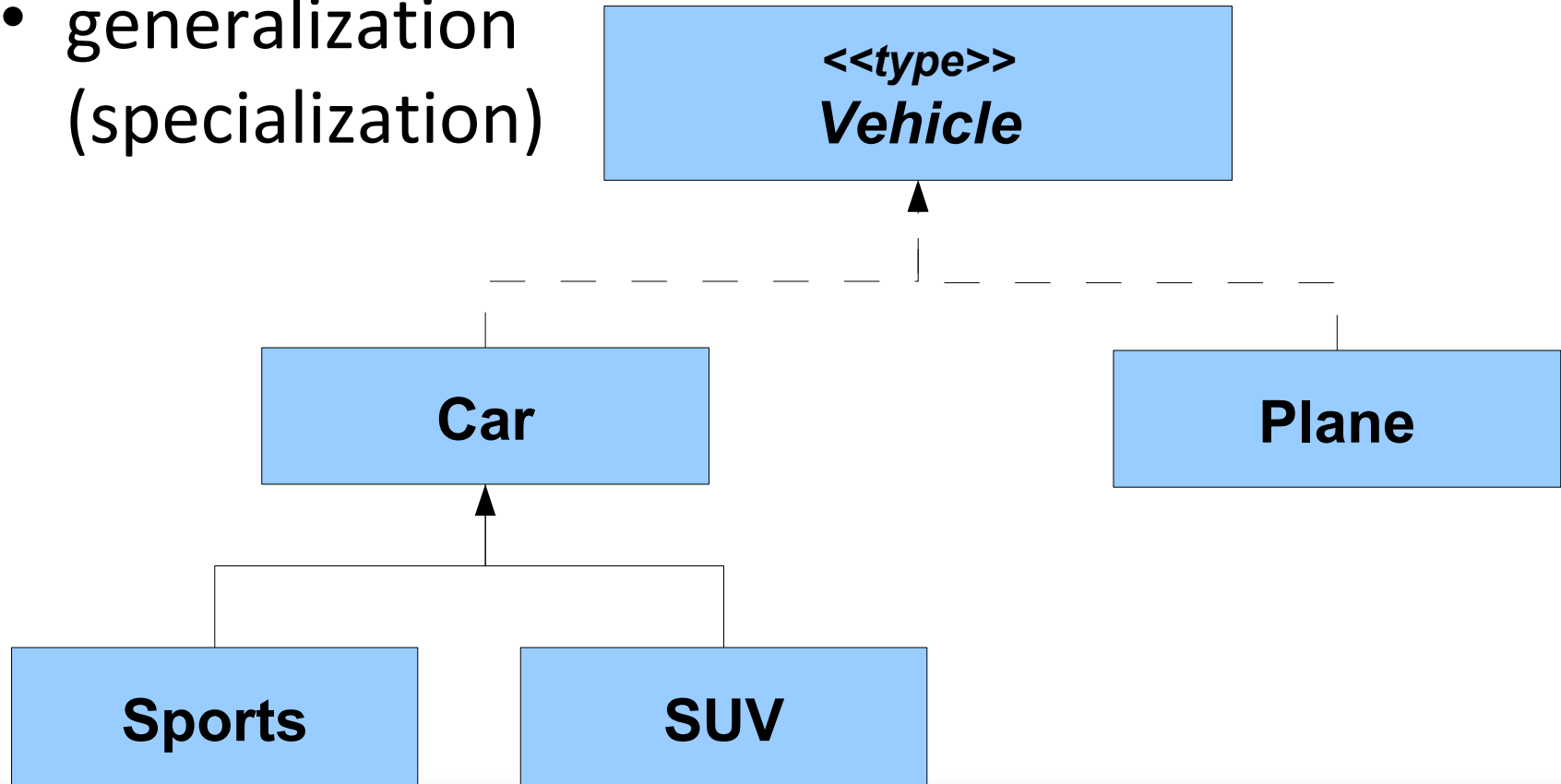
- No if, just polymorphism!
- This design is hiding complexity!

UML Break!

UML Break #3

vertical relationships

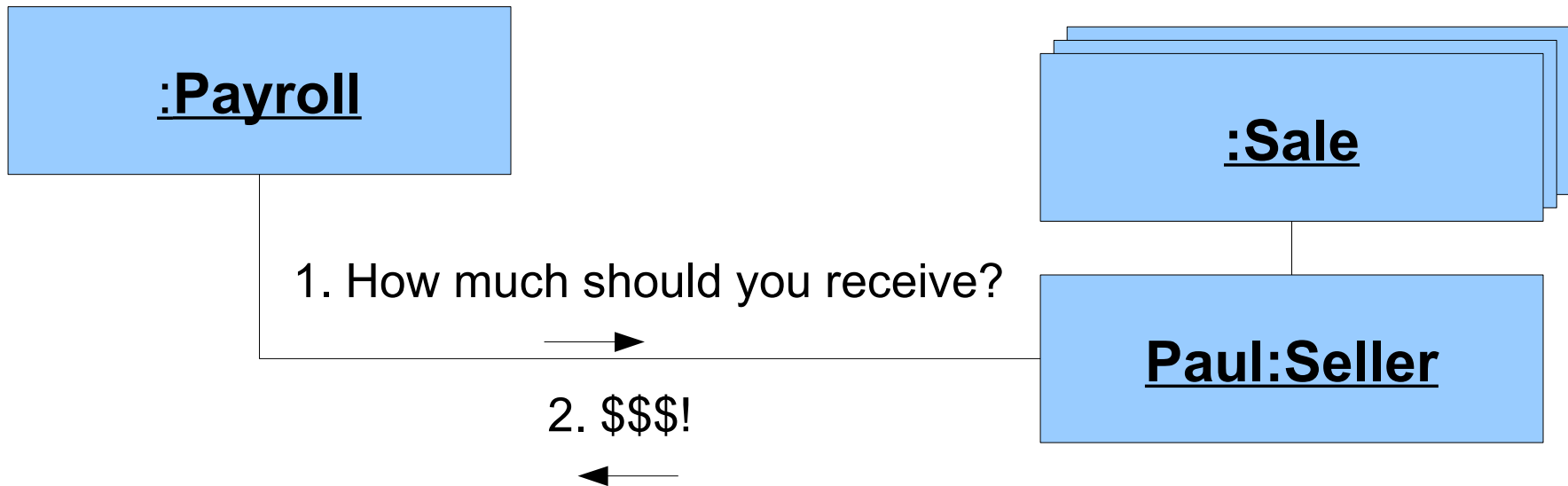
- generalization
(specialization)



Back to reality!

Requirement #2

- *“sellers are payed by a fixed amount plus a percentage on sales”*



- Trick: anything missing here?

Analysis #2 (attempt)

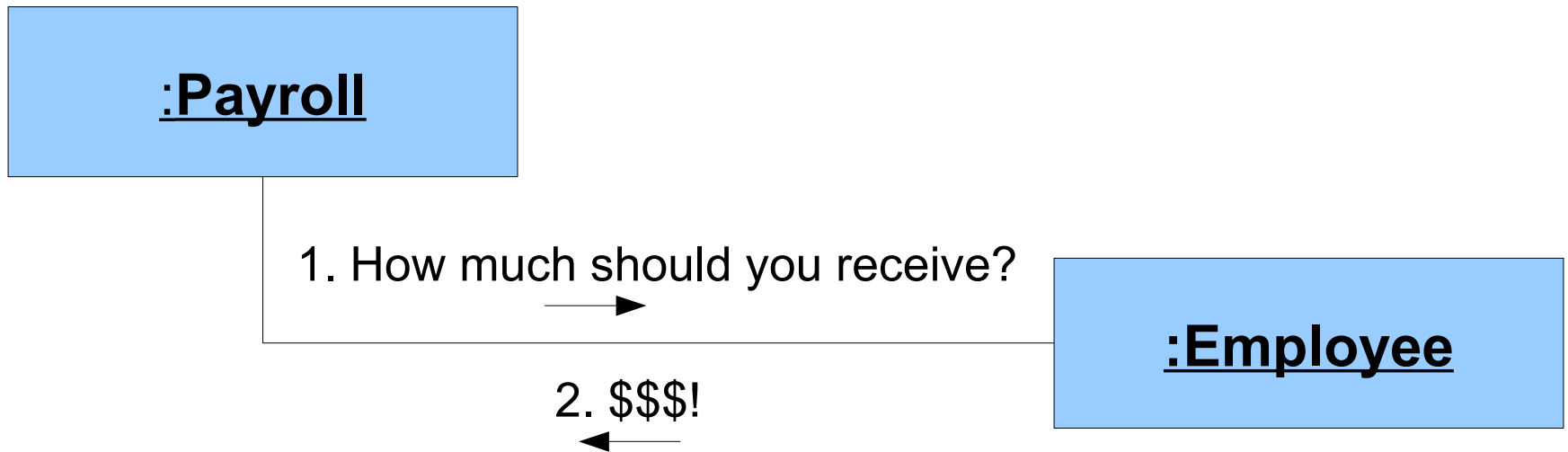
- How about the next requirement?
- *“some employees are paid based on the hours worked, the others receive a fixed rate”*
- *Ah-ha!* That means that we have at least three different ways to get the salary amount!
- This is what a BA should do

Analysis #2 / #3

- So let's go back to the plain old way...
 - **what will not change?**
 - **what will probably change?**

Analysis #2 / #3

- What will not change: *every employee is payed*
- What will change: *the amount*

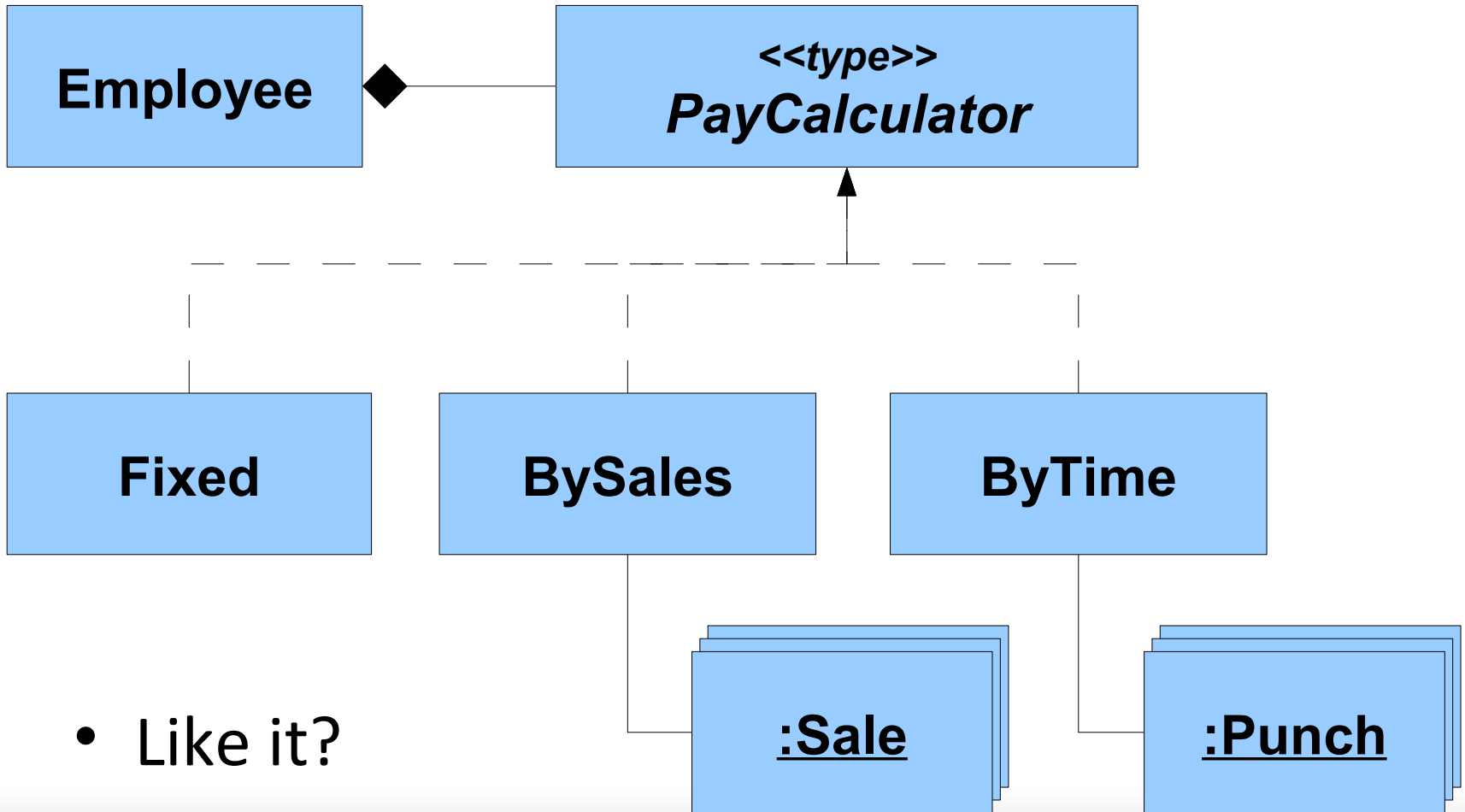


A bit of abstraction!

Design #2 / #3

- Ok, so how each Employee will know about how much is payed?
- Decouple!
- There's something that is needed here!
 - an abstraction
 - some implementations
- Ideas?

Design #2 / #3



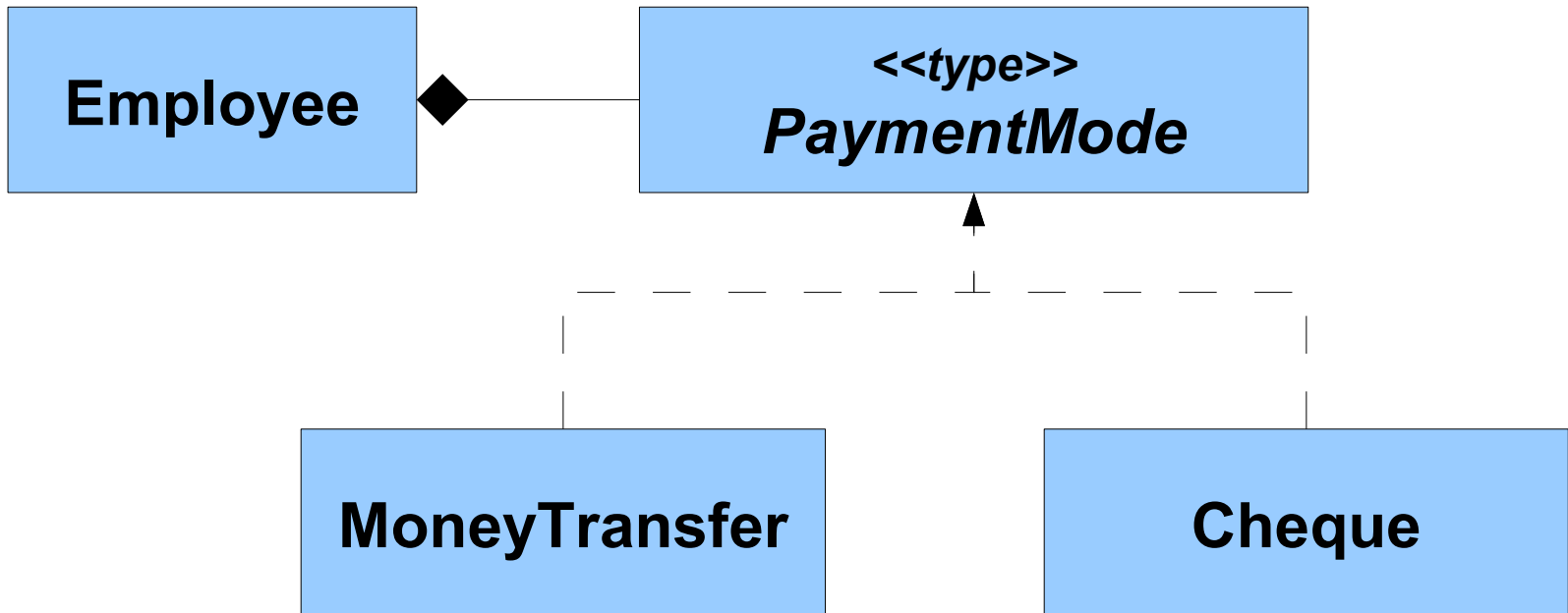
- Like it?

Analysis #4

- *“employees may choose to be payed by cheque or bank money transfer”*
- Invariant: each employee is payed with a method
- Variant: the method used

Design #4

- Piece of cake!



Let's move the Java world!

What about NFRs?

- Non functional requirement should not pollute the analysis model
- If possible they should not pollute the design model as well
- They should be handled at implementation level
- Trick: attach web and db later :)

The end...

Why “for nonbelievers”?

Why “for nonbelievers”?

- Because you're not going to believe this
- because I did not convince you
- tomorrow you will start selecting the framework as usual :)

- ...this how the world goes on! Peace :)

Q & A