TNG TECHNOLOGY CONSULTING

OOP is dead – long live Object Orientation!

Franz Thoma

3/14/2025

BobKonf 2025, Berlin

TNG TECHNOLOGY CONSULTING

About me

- Principal Consultant @ TNG
- github.com/fmthoma
- linkedin.com/in/fmthoma
- Organizer of MuniHac
  - Save the date: 12.–14.9.2025

Franz Thoma
Principal Consultant
franz.thoma@tngtech.com

My first Java `Hello World`


```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

My first Java `Hello World`


```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Very classy!

- A class around `main`
- Objects (`out`)
- Method calls (`out.println`)

My first Java `Hello World`


```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Very classy?

- Static methods
- Public properties (`System.out`)
- Classes are nothing more than namespaces

What's the definition for OOP?

- Polymorphism
- Inheritance
- Encapsulation

What's the definition for OOP?

- **Subtyping** Polymorphism
- Inheritance
- Encapsulation

Polymorphism

- **Ad-hoc polymorphism** (Overloading)
- **Parametric polymorphism** (Generics)
- **Subtyping polymorphism**
- **Row polymorphism**

What's the definition for OOP?

- Subtyping Polymorphism
- Inheritance
- Class Encapsulation ← ignored?

Encapsulation

- Module encapsulation
- Library encapsulation
- Class encapsulation
- Abstract data types

Getters and Setters

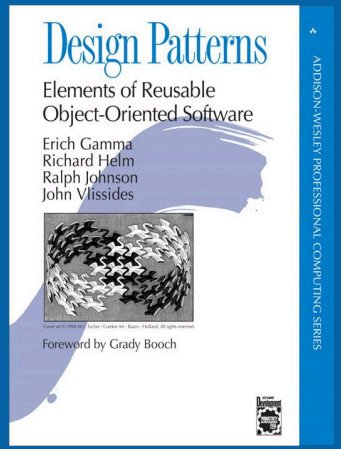
```
class Box {
  private Object thing;

  public Object getThing() { return thing; }
  public void setThing(Object newThing) { this.thing = newThing; }
}
```

What's the definition for OOP?

TNG TECHNOLOGY CONSULTING

- **Subtyping** Polymorphism
- ~~Inheritance~~ ← deprecated?
- **Class** Encapsulation ← ignored?

Inheritance



[...] our second principle of object-oriented design: *Favor object composition over class inheritance.*

Design Patterns, 1994, Chapter 1

What's the definition for OOP?

- **Subtyping** Polymorphism
- ~~Inheritance~~ ← deprecated?
- **Class** Encapsulation ← ignored?

Table of Contents

TNG TECHNOLOGY CONSULTING

1 | Problems with Inheritance & Subtyping

```
class InternalFrameInternalFrameTitlePaneInternalFrameTitlePaneMaximizeButtonWindowNotFocusedState extends State {
    InternalFrameInternalFrameTitlePaneInternalFrameTitlePaneMaximizeButtonWindowNotFocusedState() {
        super("WindowNotFocused");
    }

    @Override protected boolean isInState(JComponent c) {
        Component parent = c;
        while (parent.getParent() ≠ null) {
            if (parent instanceof JInternalFrame) {
                break;
            }
            parent = parent.getParent();
        }
        if (parent instanceof JInternalFrame) {
            return !(((JInternalFrame)parent).isSelected());
        }
        return false;
```

Inheritance

Code re-use you should not use

- Breaks encapsulation
- Tight coupling between parent and child
- Non-locality
- Overriding vs. Shadowing

Inheritance

Favor object composition over class inheritance.

Design Patterns, 1994, Chapter 1

Subtyping Polymorphism

The Casting Conundrum

```
@Override protected boolean isInState(JComponent c) {
    Component parent = c;
    while (parent.getParent() != null) {
        if (parent instanceof JInternalFrame) {
            break;
        }
        parent = parent.getParent();
    }
    if (parent instanceof JInternalFrame) {
        return !(((JInternalFrame)parent).isSelected());
    }
    return false;
}
```

Subtyping Polymorphism

The Casting Conundrum

Subtyping:

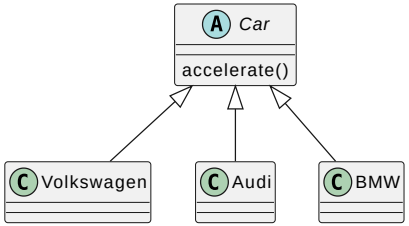
- Taxonomy that corresponds to real life
- Handle objects as generically as possible

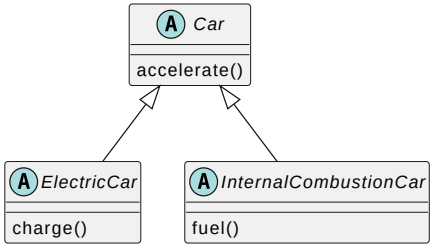
Subtyping Polymorphism

The Casting Conundrum

Subtyping:

- Taxonomy that corresponds to real life
- Handle objects as generically as possible





However:

- Real-life: Rarely hierarchical
  - Diamond problem!
- We often care about the specific sub-type
- Information loss when passing a specific thing to generic code

Subtyping Polymorphism

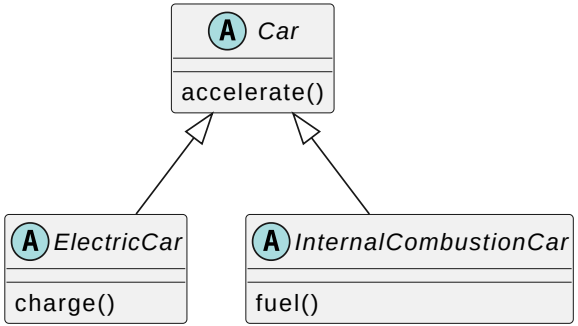
The Casting Conundrum

```
interface Garage {
  Ticket park(Car car);
  Car retrieve(Ticket ticket);
}
```

```
ElectricCar eTron = …;
Ticket ticket = garage.park(eTron);
```

```
Car eTron = garage.retrieve(ticket);
eTron.charge(); // ← Type error!
```

```
ElectricCar eTron = (ElectricCar) garage.retrieve(ticket);
eTron.charge();
```

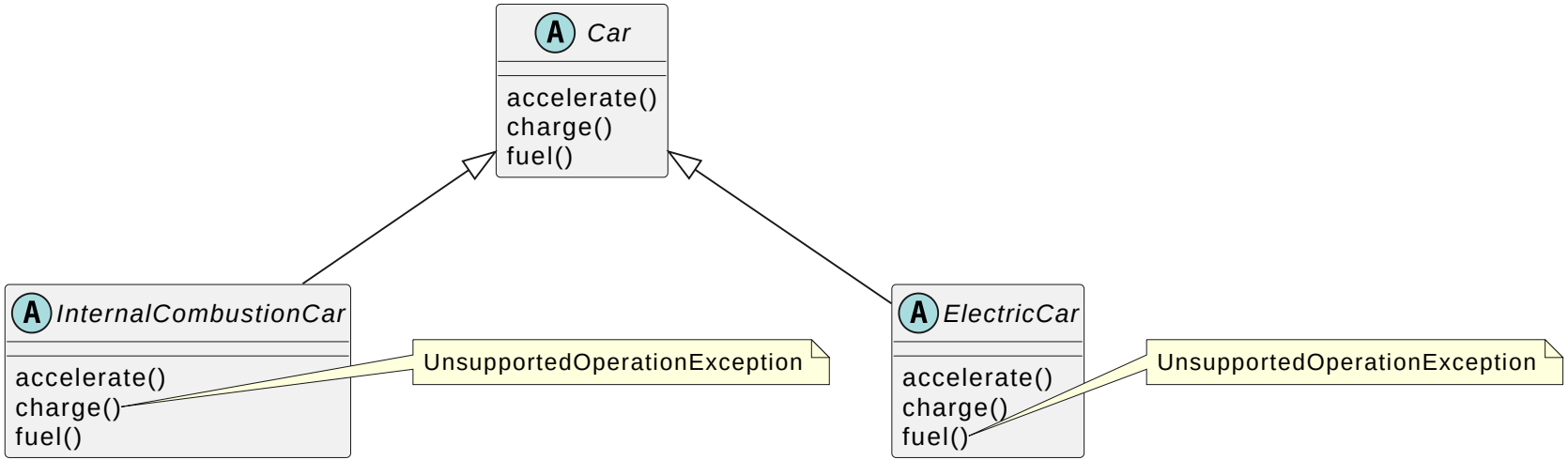


Subtyping Polymorphism

Whenever you cast, you've already given up on type safety.

Subtyping Polymorphism

To Liskov or not to Liskov?



Subtyping Polymorphism

To Liskov or not to Liskov?

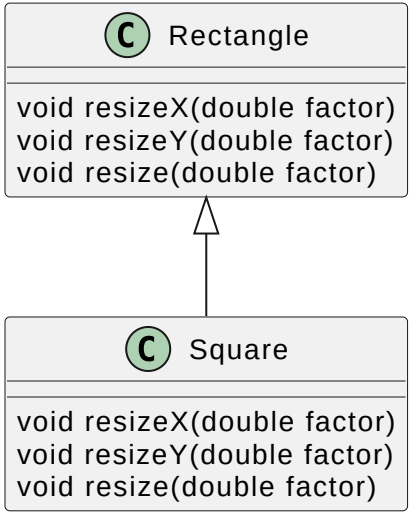
```
package java.util;

public abstract class AbstractList<E> extends AbstractCollection<E> implements List<E> {

    public void add(int index, E element) {
        throw new UnsupportedOperationException();
    }

      public void remove(int index) {
        throw new UnsupportedOperationException();
    }
}
```

Subtyping Polymorphism

Liskov Substitution Principle

If `A` extends or implements `B`, then:

- Preconditions of `B` should be weaker,
- Postconditions of `B` should be stronger,
- and invariants of `B` should be the same as for `A`.



Subtyping Polymorphism

To Liskov or not to Liskov?

```
package java.util;

public interface Collection<E> extends Iterable<E> {

    /**
     * […]
     *
     * @param e element whose presence in this collection is to be ensured
     * @return {@code true} if this collection changed as a result of the call
     * @throws UnsupportedOperationException if the {@code add} operation is not supported by this collection
     *
     * […]
     */
    boolean add(E e);
```

Subtyping Polymorphism

`instanceof`

```
Car car = garage.retrieve(ticket);
if (car instanceof ElectricCar eTron) {
  eTron.charge();
}
```

Subtyping Polymorphism

`instanceof`

- Objects are abstract (Encapsulation!)
  - Execution is driven from inside the objects, not from the outside
  - ⇒ Class of an object does not matter, only behaviour
- Unlike Functional Programming
  - In FP, execution is driven by Pattern Matching
  - ⇒ Identity of a constructor is the driving factor in FP

`instanceof`

- Decision making based on an object's class

```
interface Shape {
  double getArea();
}

class Square implements Shape {
  double getArea() {
    return Math.pow(a, 2);
  }
}

class Circle implements Shape {
  double getArea() {
    return Math.PI * Math.pow(r, 2);
  }
}
```

```
area :: Shape → Double
area (Square a) = a^2
area (Circle r) = pi * r^2
```

Subtyping Polymorphism

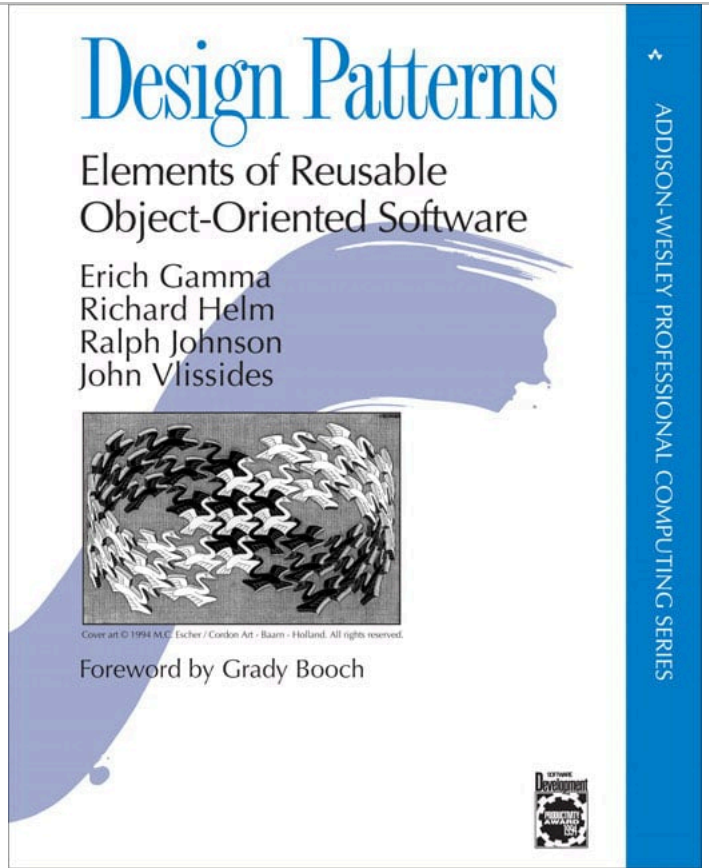
Whenever you're using `instanceof`, you've already given up on OOP.

Subtyping Polymorphism

Whenever you're using `instanceof`, you've already given up on OOP.

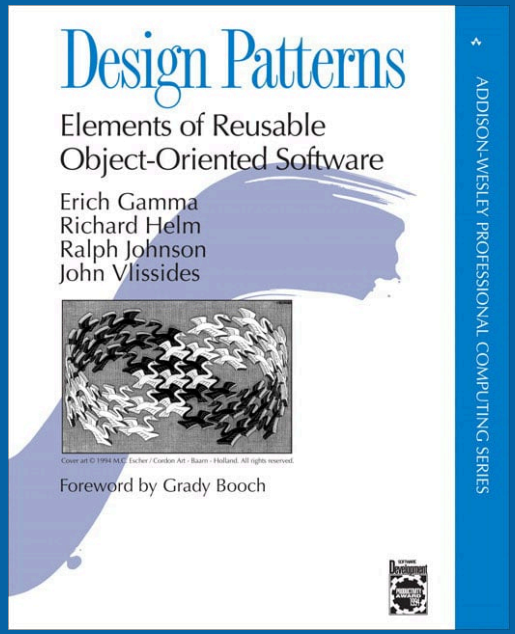
You've bought into a poor man's version of Pattern Matching.

Design Patterns



Design Patterns

OOP:	FP:
Visitor Pattern	Pattern Matching
Interpreter Pattern	Functions & Pattern Matching
Strategy Pattern	Higher-Order Functions
Command Pattern	Functions as values
Memento Pattern	Immutability



2 | A better definition for Object-Orientation

What's the definition for OOP?

- **Subtyping** Polymorphism
- ~~Inheritance~~ ← deprecated?
- **Class** Encapsulation ← ignored?

Dr. *Alan Kay* on the meaning of “object-oriented programming”

Dr. *Alan Kay* was so kind as to answer my questions about the term “object-oriented programming”.

(To link to this page, please use the above *PURL-URI* only, because any other *URI* is only temporary.)

Clarification of "object-oriented" [E-Mail]

```
Date: Wed, 23 Jul 2003 09:33:31 -0800
To: Stefan Ram [removed for privacy]
From Alan Kay [removed for privacy]
Subject: Re: Clarification of "object-oriented"
[some header lines removed for privacy]
Content-Type: text/plain; charset="us-ascii" ; format="flowed"
Content-Length: 4965
Lines: 117

Hi Stefan
```

Inspiration for OOP

I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages [...].

Alan Kay, in an email to Stefan Ram, 2003
http://www.purl.org/stefan_ram/pub/doc_kay_oop_de

On Polymorphism

- ~~Polymorphism~~
- Inheritance ← deprecated?
- **Class** Encapsulation ← ignored?

My math background made me realize that each object could have several algebras associated with it [...]. The term "polymorphism" was imposed much later (I think by Peter Wegner) and it isn't quite valid [...]. I made up a term "genericity" for dealing with generic behaviors in a quasi-algebraic form.

Alan Kay, in an email to Stefan Ram, 2003
http://www.purl.org/stefan_ram/pub/doc_kay_oop_de

On Inheritance

- ~~Polymorphism~~
- ~~Inheritance~~
- Class Encapsulation ← ignored?

I didn't like the way Simula I or Simula 67 did inheritance [...]. So I decided to leave out inheritance as a built-in feature until I understood it better.

Alan Kay, in an email to Stefan Ram, 2003
http://www.purl.org/stefan_ram/pub/doc_kay_oop_de

What's the definition for OOP?

TNG TECHNOLOGY CONSULTING

- ~~Polymorphism~~
- ~~Inheritance~~
- Only Messaging
- Encapsulation
- Late Binding

OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things.

Alan Kay, in an email to Stefan Ram
http://www.purl.org/stefan_ram/pub/doc_kay_oop_de

What's the definition for OOP?

- Only messaging
- Encapsulation local retention, protection and hiding
- Late binding

What's the definition for OOP?

"Only messaging"

- Objects communicate with each other using messages
- cf. Actor models (Erlang, Akka)
- Not just method calls!

What's the definition for OOP?

"Local retention and protection and hiding of state-process"

- Objects keep their own local state
  - retention = they can have local state
  - protection = it's not accessible to the outside
  - hiding = it's not even visible to the outside, except for message passing
- cf. Encapsulation!

What's the definition for OOP?

"Extreme late-binding of all things"

- If one object stops functioning, messages are not consumed any more, but the rest still keeps running
- Individual objects can be exchanged/updated at runtime, without having to shut down the entire system
- cf. dynamic dispatch of object methods
- Downside: Runtime errors (`ClassNotFoundException`, `NoSuchMethodException`)

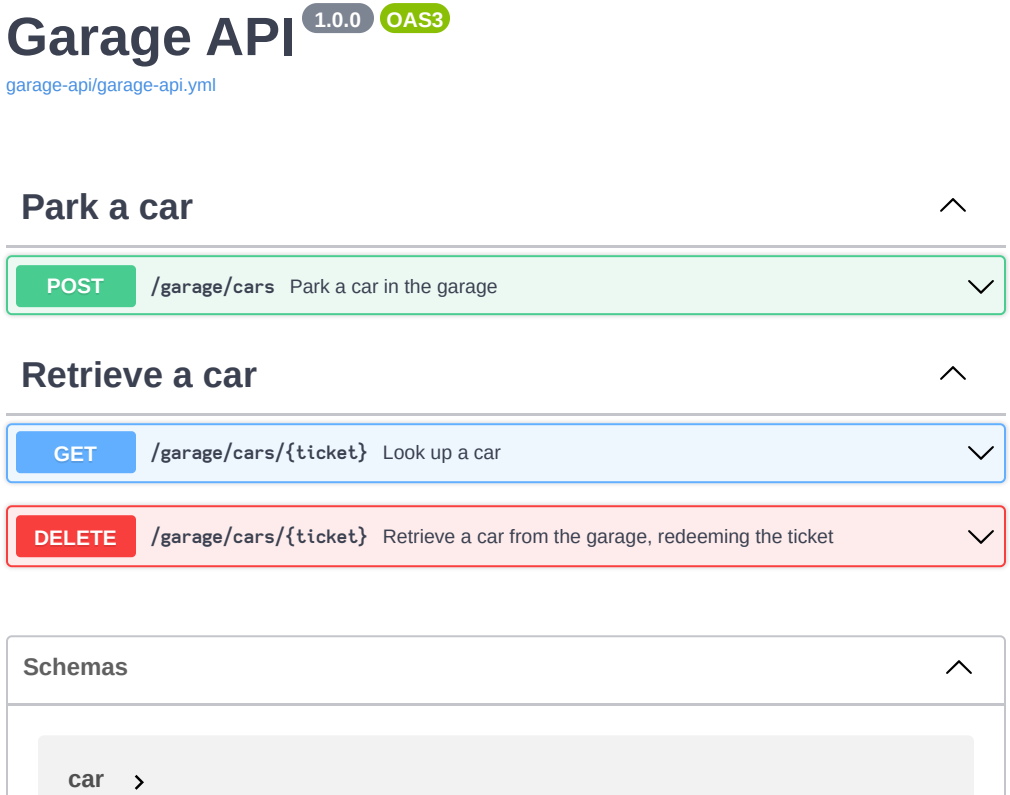
3 | Putting OOP to good use

Putting OOP to good use


```
interface Garage {
  Ticket park(Car car);
  Car retrieve(Ticket ticket);
}
```

Putting OOP to good use

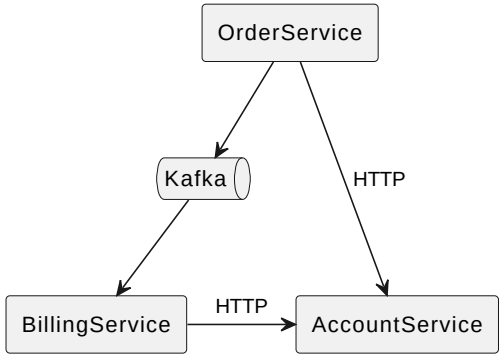

```
interface Garage {
  Ticket park(Car car);
  Car retrieve(Ticket ticket);
}
```



Putting OOP to good use

"Only messaging"

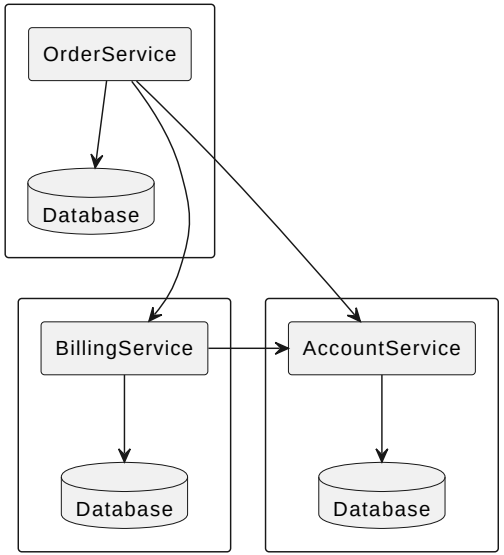
- Service communicate with each other using (actual!) messages
  - Synchronously (REST) or asynchronously (Kafka etc.)
- Not just method calls!



Putting OOP to good use

"Local retention and protection and hiding of state-process"

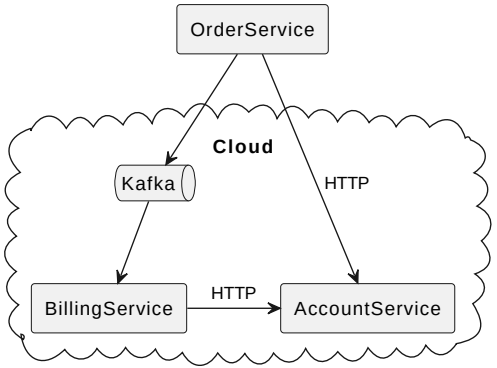
- Services keep their own local state
  - retention = service (although stateless) has a database
  - protection = only this service is allowed to access it
  - hiding = outside does not need to know the data model, only the API
- cf. Encapsulation!



What's the definition for OOP?

"Extreme late-binding of all things"

- If one service stops functioning, messages are not consumed any more, but the rest still keeps running.
- Individual services can be exchanged/updated at runtime, independent from each other, without having to shut down the entire system
- Services can even be enabled/disabled depending on load (auto-scaling, serverless)



Programming vs. Architectural Paradigms

Programming paradigms

- *Single team*
- *Code level*

Architectural paradigms

- *Across teams*
- *Organizational level*

Programming vs. Architectural Paradigms

Programming paradigms

- Static Predictability
- Cohesion
- Fault prevention

Architectural paradigms

- Runtime Flexibility
- Loose coupling
- Fault tolerance

On types

I'm not against types, but I don't know of any type systems that aren't a complete pain, so I still like dynamic typing.

Alan Kay, in an email to Stefan Ram, 2003
http://www.purl.org/stefan_ram/pub/doc_kay_oop_de

On types

Language	Polymorphism
TypeScript	**Structural Typing**, **Parametric**, Subtyping
Rust	**Parametric**, **Ad-hoc**
Golang	**Structural Typing**, Parametric

4 | Conclusion

Conclusion

1. The common definition of OOP in terms of “polymorphism, inheritance and encapsulation” misses the original point.
2. Polymorphism and inheritance are a disadvantage for programming paradigms.
3. "Biological cells that communicate via message passing" ⇿ Microservice Architectures.
4. The principles of OOP are better suited for architectural than programming paradigms.

TNG TECHNOLOGY CONSULTING

Thank you for your attention

Any questions?

Franz Thoma
Principal Consultant
franz.thoma@tngtech.com

TNG TECHNOLOGY CONSULTING

Franz Thoma
Principal Consultant
franz.thoma@tngtech.com