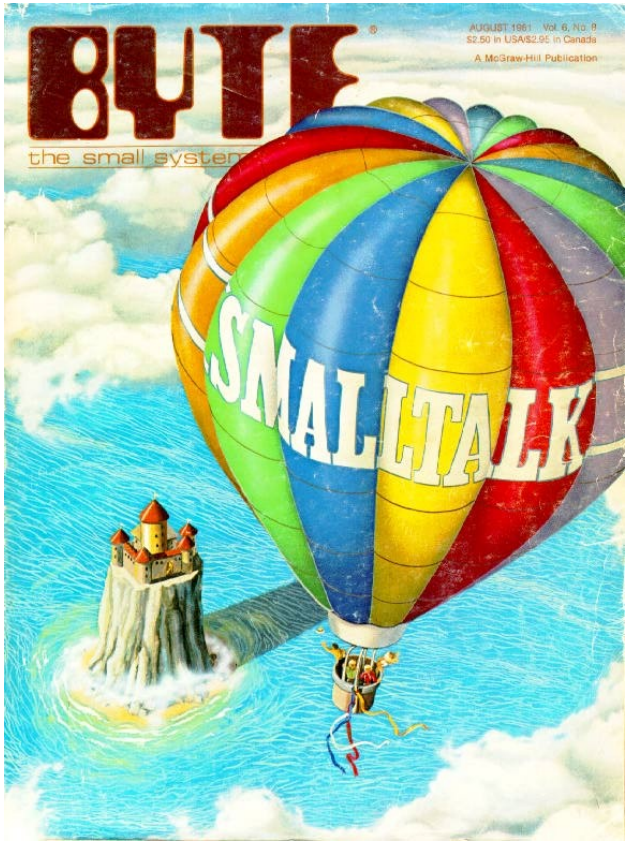
30 Jahre OO – 20 Jahre Java
The Good, The Bad, The Ugly



Wie hat Java die ursprünglichen
OO Konzepten umgesetzt?

Quellen:
Byte Magazin 1981,
Alan Kay: An Early History of Smalltalk

Andreas Tönne, Director R&D, Novatec
Java Forum Stuttgart - Pecha Kucha September 2012

Programmieren in natürlich-sprachlichen Konzepten

Zielsetzung von Anfang an!

Klassifikation und Abstraktion von Konzepten

Ein Mensch sollte das System als ganzes verstehen können

Grundlage: Die Objekt Idee

Simula 67: Objekte lagen in der Luft

Klasse = Softwaremodul mit Information-Hiding

Konzept Modellierung

Modularität → Information Hiding

Klassifikation → Klassen Typen

Factoring → Vererbung

Polymorphismus → Duck Typing und Interfaces

„Lauter kleine kommunizierende Computer“

Rekursive Modelle auf Basis eines gleichartigen Bausteins „Objekt“

Keine Unterscheidung von Datentypen, Modulen und größeren Strukturen

Kommunizierende Objekte

Herzstück der OO →

(Alan Kay) Eine mächtige uniforme Metapher der Berechnung für die ganze Sprache:

Objekte schicken sich Nachrichten

Selbstbezügliche Sprache

Sprache wird in sich definiert und erweitert

Steuerung des Programmablaufs nur durch Nachrichten

Methoden wie if-then-else benötigen Verhalten als Argument-Objekte

Virtuelle Maschine Pflicht

Zeitgeist: Fortgeschrittene Konzept wurden damals immer auf VMs implementiert

Plattform für das „Kommunizierende Objekte“ Konzept mit Garbage Collection

Garbage Collection notwendig für OO Modellierung

Externe Zerstörung von Objekten verletzt die OO Idee

Etwas das nicht mehr existiert muss auch in der Implementierung selbstständig verschwinden

Was hiervon Java umgesetzt hat

Objekt-Idee 

Konzept Modellierung 

Objekt Plattform 

Uniforme Bausteine 

Nur kommunizierende Objekte 

Konzept Modellierung gut

Statisches Typsystem hat Java nicht behindert

Interfaces und Implementierungsklassen völlig adäquat

Mit Traits wäre Java „rund“

Java VM Plattform der größte Erfolg der Informatik

(Alan Kay)

„Again, the whole point of OOP is ***not*** to have to ***worry*** about what is ***inside an object***.

Objects made on different machines and with different languages ***should be able to talk to each other***”

Das hat Java sehr gut hinbekommen

Keine Uniformen Bausteine

Nicht alles ist Objekt

Primitive Datentypen unnötiger Anachronismus

Rekursive Dekomposition in gleichberechtigte Bausteine verletzt

Keine Uniforme Metapher der Berechnung

Kontrollfluss sollte nur durch Nachrichten erfolgen

Stattdessen Kontroll-Statements in der Sprache

Externer Compiler statt Eigenschaft von Objekten

Warum? Es fehlen etwas wie Closures!

Selbstbezügliche Definition erfordert „Higher-Order Programmierung“

Ausdrücke (Verhalten) muss Objekt sein

Programm-Literale statt Reflection

Implementierung von Kontrollstrukturen

Objekt A sagt zu Objekt B: „Mache **Dies** mit mir wenn du willst“

true.ifThenElse(Ja, Nein) -> Ausdruck Ja

false.ifThenElse(Ja, Nein) -> Ausdruck Nein

Statements statt implementierte Kontrollstrukturen

```
ArrayList<String> result = new ArrayList<String>();
   for (String elem: object) {
       result.add(elem.trim())}
```

Fetter Text ist Verhalten „Map“ von object

Closure (Lambda) erlaubt Map

object.map((**String elem) -> elem.trim()**);

Sender übergibt Verhalten zur Ausführung an object

Java SE 8 wird Lambda bringen

20 Jahre für ein fundamentales OO Prinzip

Gerade noch rechtzeitig!

Ist das Problem damit behoben?

Unklar → Diskussion nach dem Talk

Java nicht perfekt aber ok

Konzepte meist gut ausdrückbar

Häufig technischer Zusatzcode

Metapher der kommunizierenden autarken Objekte funktioniert sensationell gut