Java Überblick

Ein Seminar für die Commerzbank

Java Runtime

- Java Virtual Machine
  - Hoch-optimiert, z.B. „Just in Time Compiler“, „HotSpot“
  - Interne Speicherverwaltung mit integrierter „Garbage Collection“
- Bytecode-Spezifikation
  - Plattform-Unabhängigkeit
  - Bytecode-Verifier

Programmiersprache Java

- Die Programmiersprache „Java“
  - Es gibt einen Compiler, der Java Quellcode in Java Bytecode umwandelt
  - aber genauso Scala, Groovy, JPython, ...
- Java-Compiler arbeitet
  - portabel
  - reproduzierbar
  - Exkurs: Was ist denn die aktuelle Java-Version?
    - Java 15 = Java **1**.15
    - Abwärtskompatibilität ist garantiert

Die Java Runtime im Detail

- Die Java Virtual Machine der “Standard-Edition”
  - Bytecode-Interpreter
  - Ausführen von Algorithmen
  - Ressourcen-Zugriffe
    - Dateisystem
    - Netzwerk
- Standard-Hersteller
  - Oracle
  - IBM/Eclipse
  - ...

Die Java Runtime im Detail

- Die Java Virtual Machine der “Enterprise-Edition”: Der JEE Application Server
  - beruht auf der Standard-Edition
  - Authentifizierung und Autorisierung
  - Transaktion-Manager
  - Deployment-Verfahren für Anwendungen
  - Überwachung und Monitoring
  - Unterstützung der Standard-Protokolle
    - http/https
    - Web Services (RESTful, SOAP)
    - Messaging
- ...

Application Server: Produkte und Hersteller

- WebSphere
  - IBM
  - Kostenpflichtig
  - 2GByte + X
- Apache Tomcat
  - Apache Group
  - Open Source
  - 100MB
- JBoss/Wildfly
  - Red Hat
  - OpenSource
  - 200MB
- ...

Java Card

JAVACREAM
Training
Consulting
Projectmanagement

Java Virtual Machine für Smart Cards etc.

Exkurs: Virtualisierung, Container, Cloud

Betrieb einer Anwendung: “Blech”

- Rechner
- + Betriebssystem
- + Infrastruktur
- + Anwendung installieren
- + Nachrangige Prozesse
  - Registrierung im Load Balancer
  - Registrierung in der Firewall
  - Registrierung in der System-Überwachung
- Anwendung starten und betreiben

Zeitaufwand
1 Tag

Notwendiges Detailwissen für die Anwendung
enorm

Betrieb einer Anwendung mit virtualisierten Rechnern

- Virtueller Rechner
- + Betriebssystem
- + Infrastruktur
- + Anwendung installieren
- + Nachrangige Prozesse
  - Registrierung im Load Balancer
  - Registrierung in der Firewall
  - Registrierung in der System-Überwachung
- Anwendung starten und betreiben

Zeitaufwand
1h

Notwendiges Detailwissen für die Anwendung
enorm

Betrieb einer Anwendung mit Images

- Virtueller Rechner
- + Betriebssystem
- + Virtual Machine Player, z.B. VMWare
- + Kopieren einer Image-Datei
- + Nachrangige Prozesse
  - Registrierung im Load Balancer
  - Registrierung in der Firewall
  - Registrierung in der System-Überwachung
- Image starten und betreiben

Zeitaufwand
wenige Minuten

Notwendiges Detailwissen für die Anwendung
kaum

Plattform-Unabhängigkeit ist gewährleistet

Betrieb einer Java-Anwendung

- Virtueller Rechner
- + Betriebssystem
- + Application Server
- + Infrastruktur
- + Kopieren von Java Bytecode
- + Nachrangige Prozesse
  - Registrierung im Load Balancer
  - Registrierung in der Firewall
  - Registrierung in der System-Überwachung
- Java-Anwendung deployen

Zeitaufwand
eine Minute

Notwendiges Detailwissen für die Anwendung
deutlich

Plattform-Unabhängigkeit ist gewährleistet

Betrieb einer Anwendung mit Containern

- Virtueller Rechner
- + Betriebssystem
- + Container-Runtime (Docker)
- + Kopieren Image-Dateien
  - Anwendung
  - Infrastruktur
- + Nachrangige Prozesse
  - Registrierung im Load Balancer
  - Registrierung in der Firewall
  - Registrierung in der System-Überwachung
- Container starten und betreiben

Zeitaufwand
wenige Sekunden

Notwendiges Detailwissen für die Anwendung
kaum

Plattform-Unabhängigkeit ist nicht gewährleistet

Betrieb einer Anwendung mit orchestrierten Containern

- Virtueller Rechner
- + Betriebssystem
- + Orchestrierungs-Software
  - (Docker Swarm)
  - Kubernetes
- + Container-Runtime (Docker)
- + Kopieren Image-Dateien
  - Anwendung
  - Infrastruktur
-

Zeitaufwand
wenige Sekunden

Notwendiges Detailwissen für die Anwendung
kaum

Plattform-Unabhängigkeit ist nicht gewährleistet

Betrieb einer Anwendung mit orchestrierten Containern in der Cloud

- Kopiere eine Anwendung in die Cloud
  - Anwendungs-Format
    - Docker Image
    - Bytecode (Java-Welt!)
    - Quellcode
      - .java
      - JavaScript für Node.js
      - Python
      - Perl

Zeitaufwand
wenige Sekunden

Notwendiges Detailwissen für die Anwendung
kaum

Plattform-Unabhängigkeit ist nicht gewährleistet

Die Programmiersprache Java

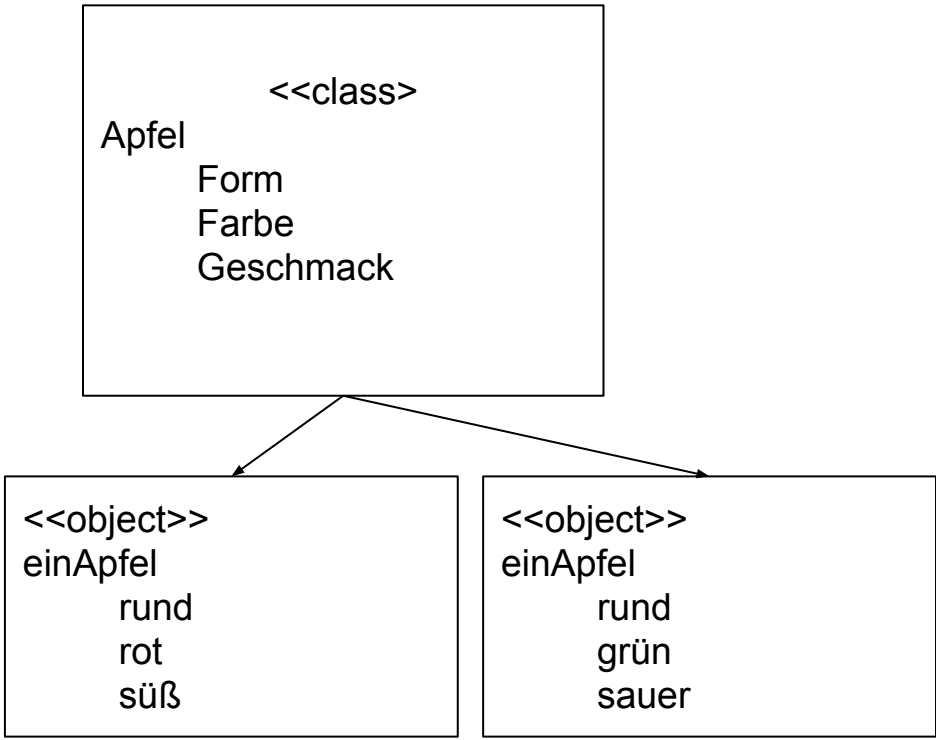
Features

- Java-Quellcode muss vom Java-Compiler in Bytecode umgewandelt werden
  - Dafür ist ein “Build-Prozess” notwendig
- Statisch typisiert
  - String message = “Hello”
  - //message = 42 -> falsch
  - Konsequenzen
    - Compiler kann damit frühzeitig Fehler erkennen
    - Java-Programme sind doch recht “verbose”/geschwätzig/enthalten überflüssige Tipparbeit
- Objekt-Orientiert
  - Java ist Klassen-orientiert
    - Eine Klasse ist ein abstrahiertes Template

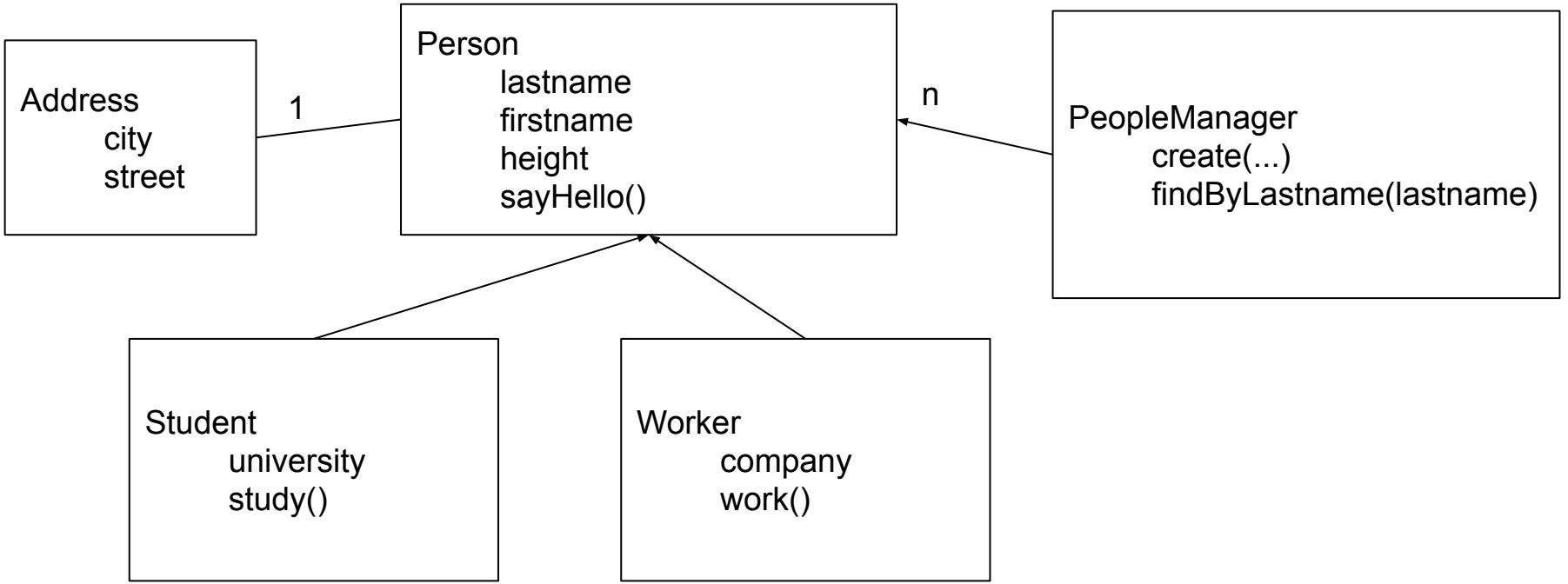
Exkurs: Klassen und Objekte

Objekt “ein Apfel”
rund
rot/grün/gelb
süß

Objekt “ein anderer Apfel”
rund
rot/grün/gelb
sauer
Kleine Delle rechts unten



Anwendungsmodellierung als Klassendiagramm



Spezifikation des Verhaltens

JAVACREAM
Training
Consulting
Projectmanagement

- Erzeugen einer Person muss Nachname und Vorname gesetzt sein
  - Mindestens 2 Zeichen
- Suche nach einer Person: nachname muss gesetzt sein
  - Falls kein Treffer: Fehler
- Körpergröße: Größer 0

Umsetzung in Java

- Entwicklungsumgebung, IDE
  - Eclipse (evtl. STS)
  - IntelliJ
- Features
  - Automatisches Compilieren
  - Code Assist
  - Quick Fix
  - Einfaches Formulieren und Ausführen von Spezifikations-Tests
    - “JUnit”
- Team-Zusammenarbeit durch Verwendung eines gemeinsamen Source Code Managements
  - Hier: GitHub

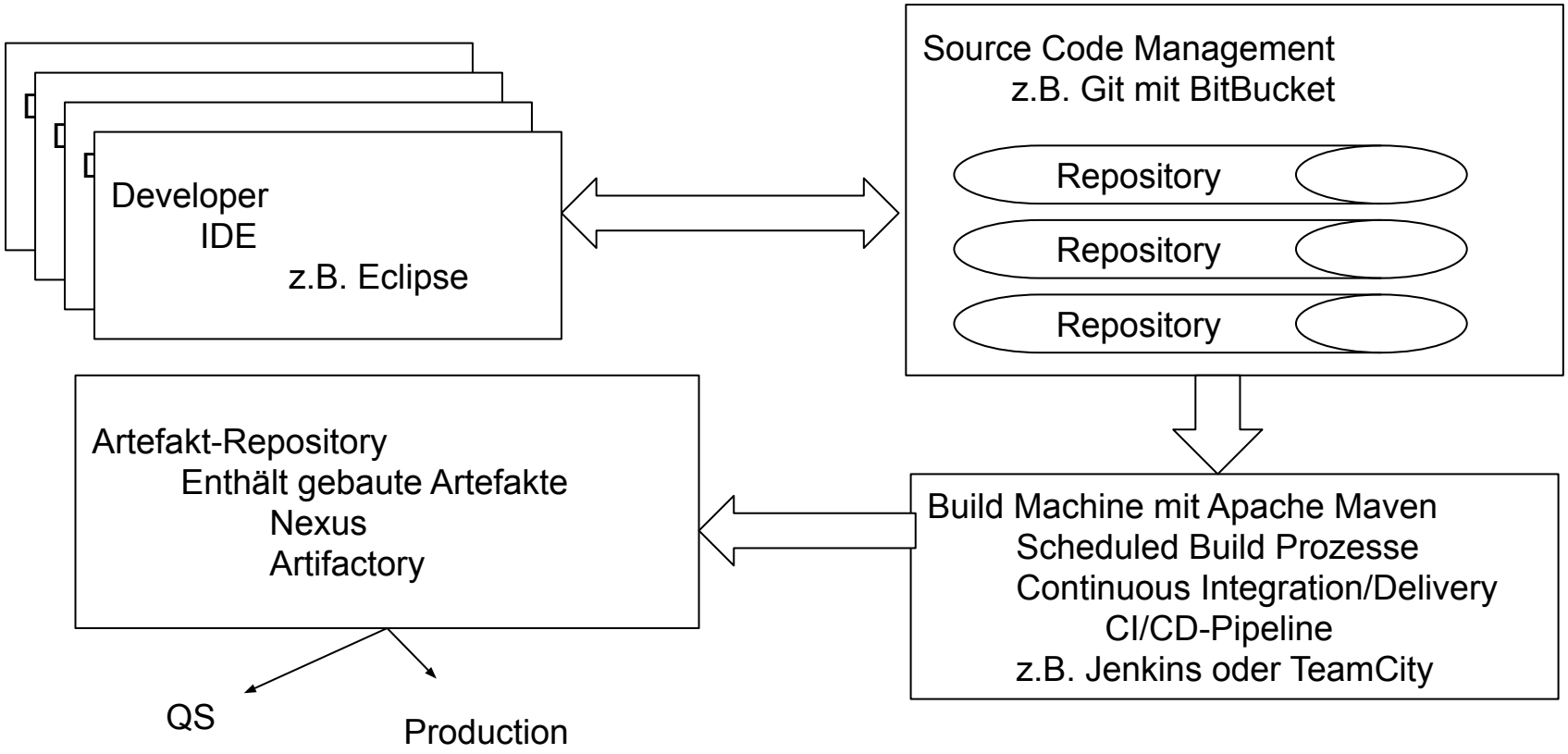
Ergebnis

- https://github.com/Javacream/org.javacream.training.java.overview/commit/015871961e7ccfae05043956a56ba50f0ce5ba18
-

Beteiligte Rollen

- “Fachabteilung”
  - Erstellt eine “Prosa-Dokument”
- Technisches Design
  - Klassendiagramme
  - Spezifikation
  - Architektur-Entscheidung
    - Realisierung in Java
- Anwendungsentwicklung
  - Implementierung der Fachlogik
  - Testen der Implementierung

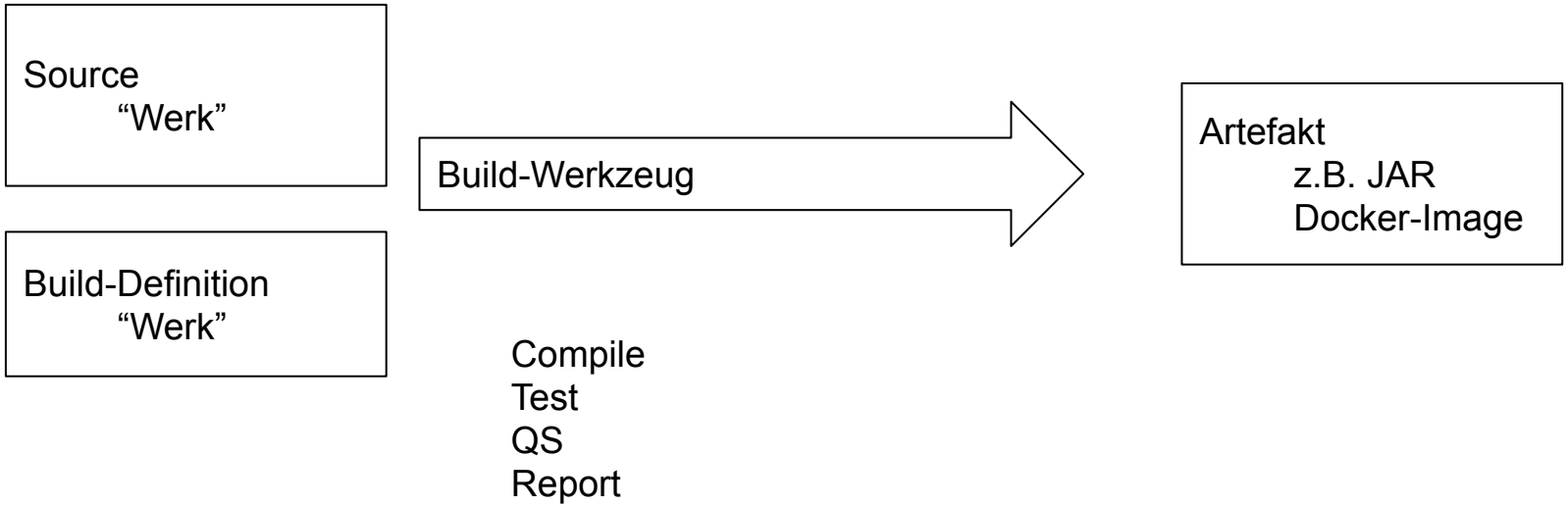
Toolchain


Developer
IDE
z.B. Eclipse
Source Code Management
z.B. Git mit BitBucket
Repository
Repository
Repository
Artefakt-Repository
Enthält gebaute Artefakte
Nexus
Artifactory
Build Machine mit Apache Maven
Scheduled Build Prozesse
Continuous Integration/Delivery
CI/CD-Pipeline
z.B. Jenkins oder TeamCity
QS
Production

Das Repository

- Enthält die Java-Quellcodes
- Weitere Ressourcen
  - Konfigurationsdateien
  - Definition und Konfiguration des Build-Prozesses
- Bytecode ist ein ”Artefakt”
  - Wird durch einen eindeutigen, portablen und reproduzierbaren Build-Prozess aus den Quellcodes erzeugt
    - und ist damit NICHT im SCM enthalten

Aufgaben des Build-Prozesses



Build-Werkzeuge für Java

- Apache Maven
  - pom.xml
- Gradle
  - build.gradle
- (Apache ANT)
  - build.xml

Stil der Java-Programmierung

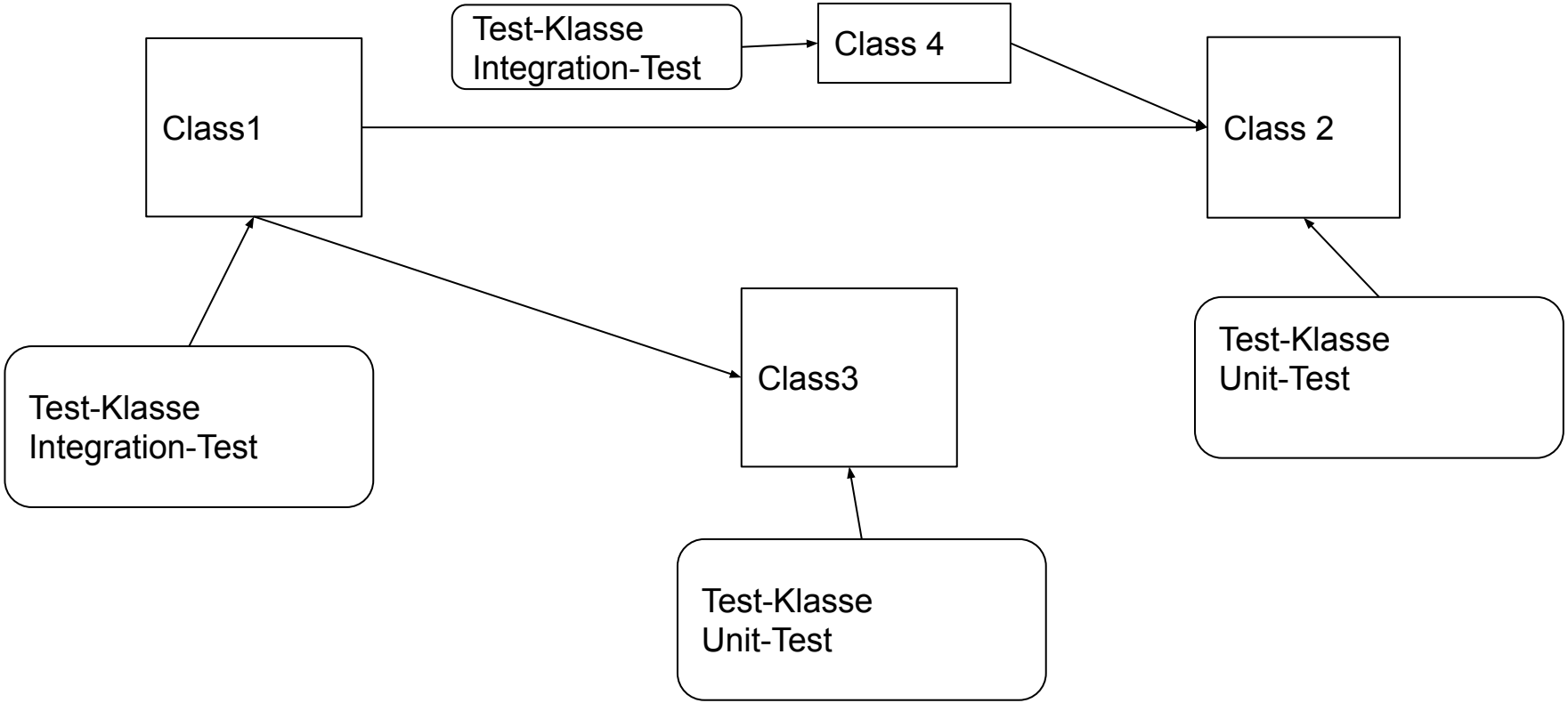
Objekt-orientiert

- Modellierung mit Klassendiagrammen
  - Trend geht hin zu sehr fein-granularen Klassen
  - “Goldene Regel der OOP-Modellierung”: Jede Klasse übernimmt ausschließlich eine Rolle/Aufgabe
- Ziel
  - Wartbare, wiederverwendbare, testbare Programme
  - Dieses Ziel kann nur dynamisch erreicht werden, bei erkannten Fehern wird “Refactoring” durchgeführt
- Konsequenz daraus: Beträchtliche Menge von Klassen mit Abhängigkeiten
  - Selbst eine einfach Anwendung wird aus dutzenden von Klassen bestehen

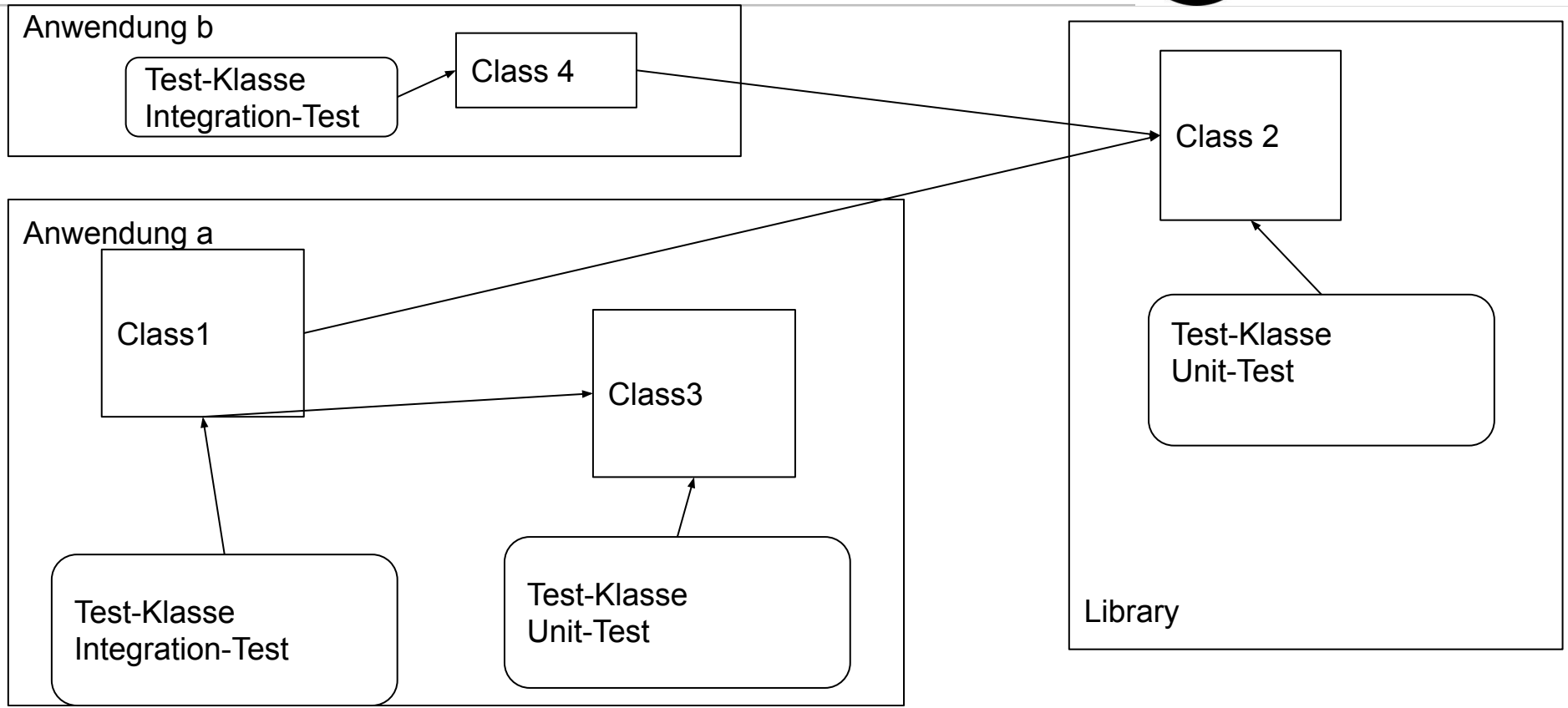
Exkurs: Klassendiagramme

- Teil der “Unified Modelling **Language**”, UML
  - Äußerst mächtig
  - sehr formal
  - Hoch kompliziert
  - Standard-Werkzeug für UML
    - “Enterprise Architect” (EA)
      - 4 Wochen Einarbeitungszeit in UML/EA ist typisch
- Pragmatic UML
  - Whiteboard-friendly
  - Visio
  - Umlet

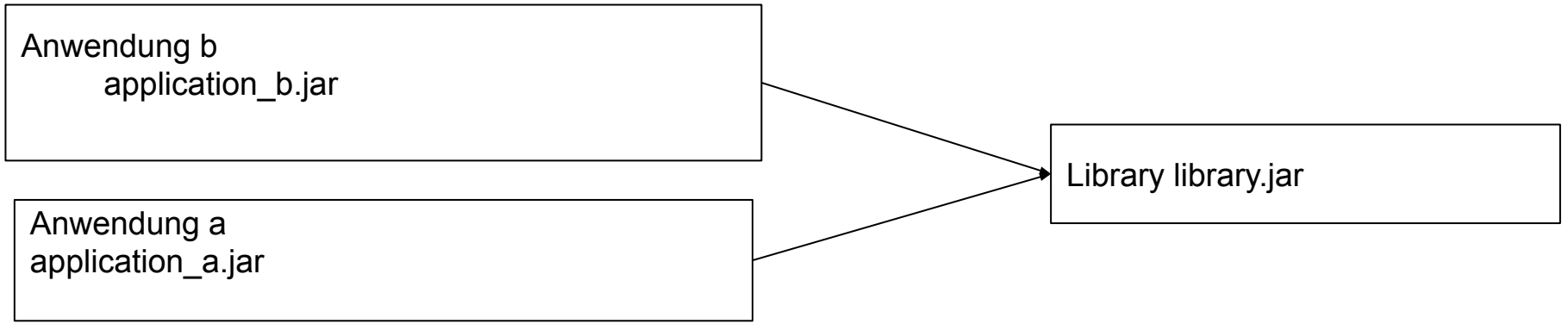
“Reales” Klassendiagramm



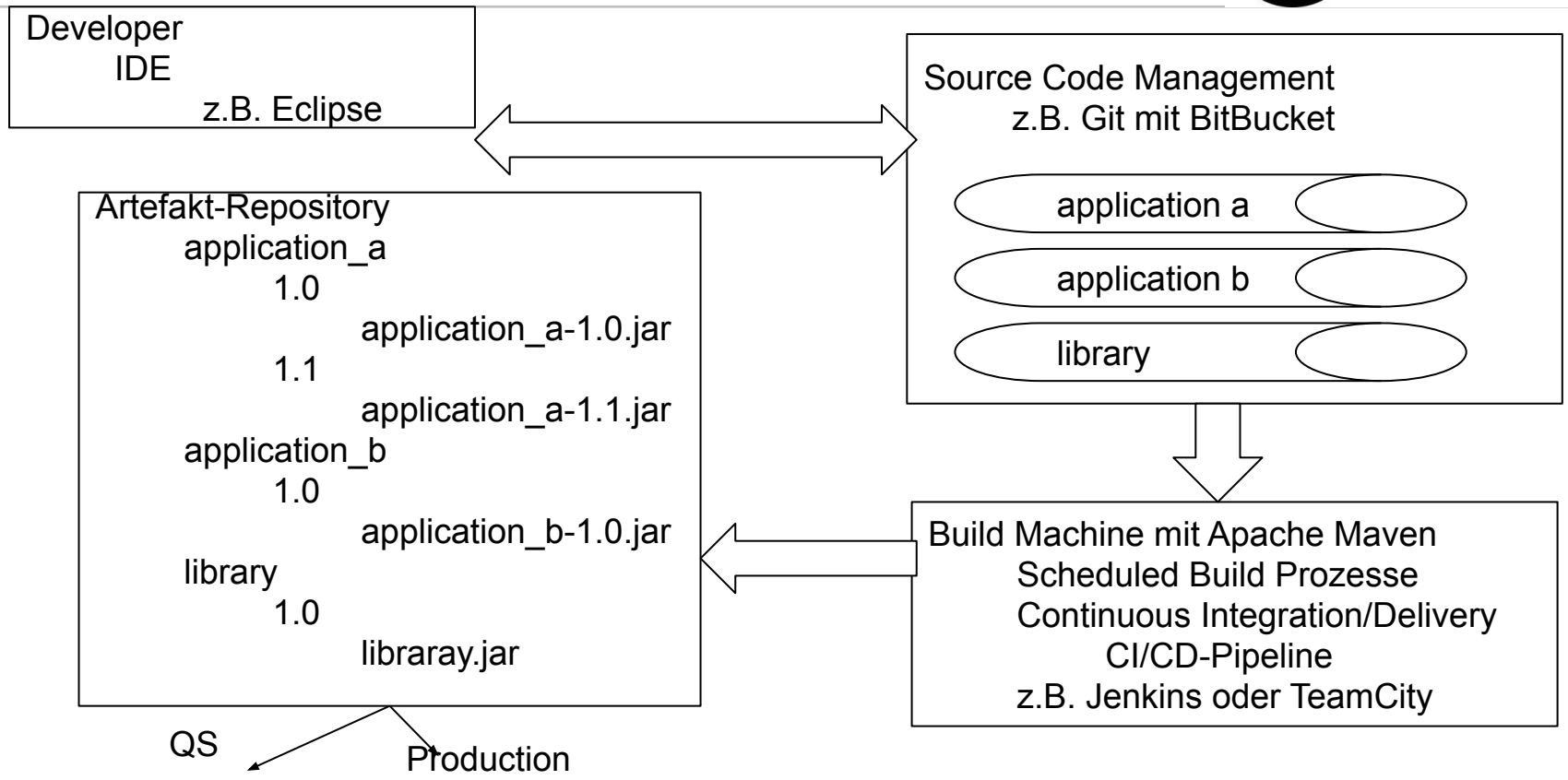
“Reale” Anwendung



“Reale” Java-Anwendungen



Artefakte


Developer
IDE
z.B. Eclipse
Source Code Management
z.B. Git mit BitBucket
application a
application b
library
Artefakt-Repository
application_a
1.0
application_a-1.0.jar
1.1
application_a-1.1.jar
application_b
1.0
application_b-1.0.jar
library
1.0
libraray.jar
Build Machine mit Apache Maven
Scheduled Build Prozesse
Continuous Integration/Delivery
CI/CD-Pipeline
z.B. Jenkins oder TeamCity
QS
Production

Reale Anwendung

- Ein kompletter Web Server
  - http-Protokoll
- Zugriff auf Personen-Informationen
- Ablage erfolgt in einer relationalen Datenbank
- https://github.com/Javacream/org.javacream.training.java.overview/tree/commerzbank_20.8.2020/org.javacream.training.rest.people.server
-

Umsetzung

- Java Standard Edition
  - 50.000 Lines of Code
- Enterprise Edition mit Applikationsserver, z.B. Apache Tomcat
  - 500 Lines of Code
  - Zusätzlich notwendig:
    - Installation des Applikationsservers
    - Konfiguration
    - Deployment der Anwendung
    - Installation eines Datenbank-Servers
- Verwendung des Spring-Frameworks mit Spring Boot
  - 100 Lines of Code
    - Rest der Installation und Konfiguration ist optional

Die Anwendung in Aktion

JAVACREAM
Training
Consulting
Projectmanagement

- http://10.12.7.1:8080/people

Maven “revisited”

- Bisher:
  - Werke werden durch einen Build-Prozess in Artefakte umgewandelt
- Nun:
  - Dependency Management
    - Direkt benutze Dependencies werden im pom.xml deklariert
    - Diese werden automatisch vom Artefakt-Repository geladen
      - inklusive aller transitiven Dependencies

Service Beschreibungen

Komplexeste Historie...

- ((IDL))
  - Interface Definition Language als Bestandteil von Corba
- ((Java RMI))
  - Remote Java Interfaces
- (SOAP basierte Web Services)
  - Web Service Description Language, WSDL
- REST

Grundlagen von REST

- Daten = Ressourcen = Dokumente werden eindeutig über einen Resource Locator identifiziert
- Dokumente können Beziehungen zu anderen Dokumenten enthalten, in dem der Resource Locator als “Link” angegeben wird
- Jedes Dokument hat einen so genannten “Media Type”, der sein Format beschreibt
  - text/plain
  - image/jpg
  - application/json

Service Beschreibungen: Part 1

- Struktur der Daten (wahrscheinlich JSON) müssen beschrieben werden

Grundlagen von REST: Operationen?

Book
- isbn (Id)
- title
- price

Person
- id (Id)
- lastname
- firstname

Invoice
- number (Id)
- amount
- date

REST-Operationen

- CRUD
  - Create
  - Read
  - Update
  - Delete
- Eindeutiger Hash für jede Ressource
- Cache Policy
- Authentifizierung
- Verschlüsselung
- Satz von Status

REST-Operationen mit http

- CRUD
  - Create -> POST, (PUT)
  - Read -> GET
  - Update -> (PUT, (POST)), PATCH
  - Delete -> DELETE
- Eindeutiger Hash für jede Ressource -> ETag
- Cache Policy -> Cache Policy
- Authentifizierung -> (BASIC, DIGEST, CERTIFICATE), Open ID
- Verschlüsselung -> Transport Layer Security, TLS,
  - fast identisch: Secure Socket Layer, SSL
- Satz von Status -> Http-Status, z.B., Not Found -> 404