Building a modern 3-tier application

Hogeschool Utrecht : Les 4

Quintor

Agenda

- Theorie
- Hand-on opdrachten
- Lunch + Evaluatie
- Theorie
- Hand-on opdrachten

Doel: zelf een 3-tier web applicatie ontwikkelen

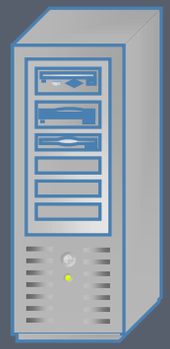
Hoe werkt een web applicatie?

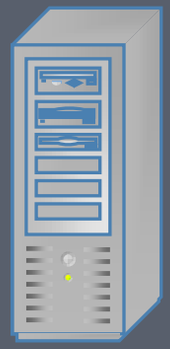
De hoog over architectuur:

Request HTML, CSS, JS, images, etc

Process Request

Send Response



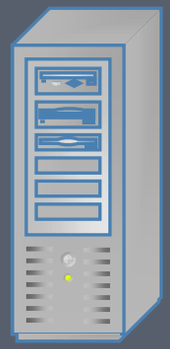


Send JSON Request (/api/login)

Forward Request naar tomcat

Process request

Send Response

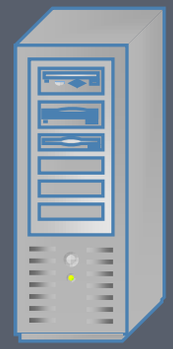


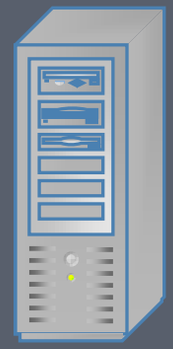
Send JSON Request (/api/login)

Forward Request naar tomcat

Process request

Send Response





Hoe werkt de java applicatie?

De software architectuur:

HTTP request
`/api/getAccount/6380423`

HTTP Servlet

Web controller

Spring service

JDBC
Repository

Database

Omzetten HTTP request naar java object
(javax.servlet.http.HTTPServletRequest)

Op basis van servlet configuratie (@RequestMapping)
de juiste java methode aanroepen

Parsen parameters

(rekeningnummer = 6380423)

JSON data mappen naar POJO's (plain old java objects)

Valideren request parameters

Aanroepen Service methode

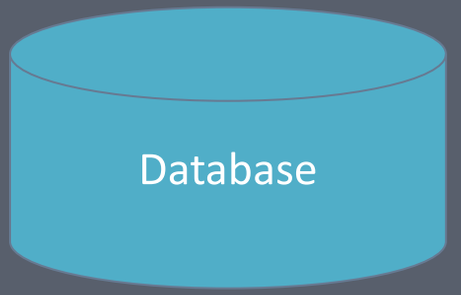
HTTP Servlet

Web controller

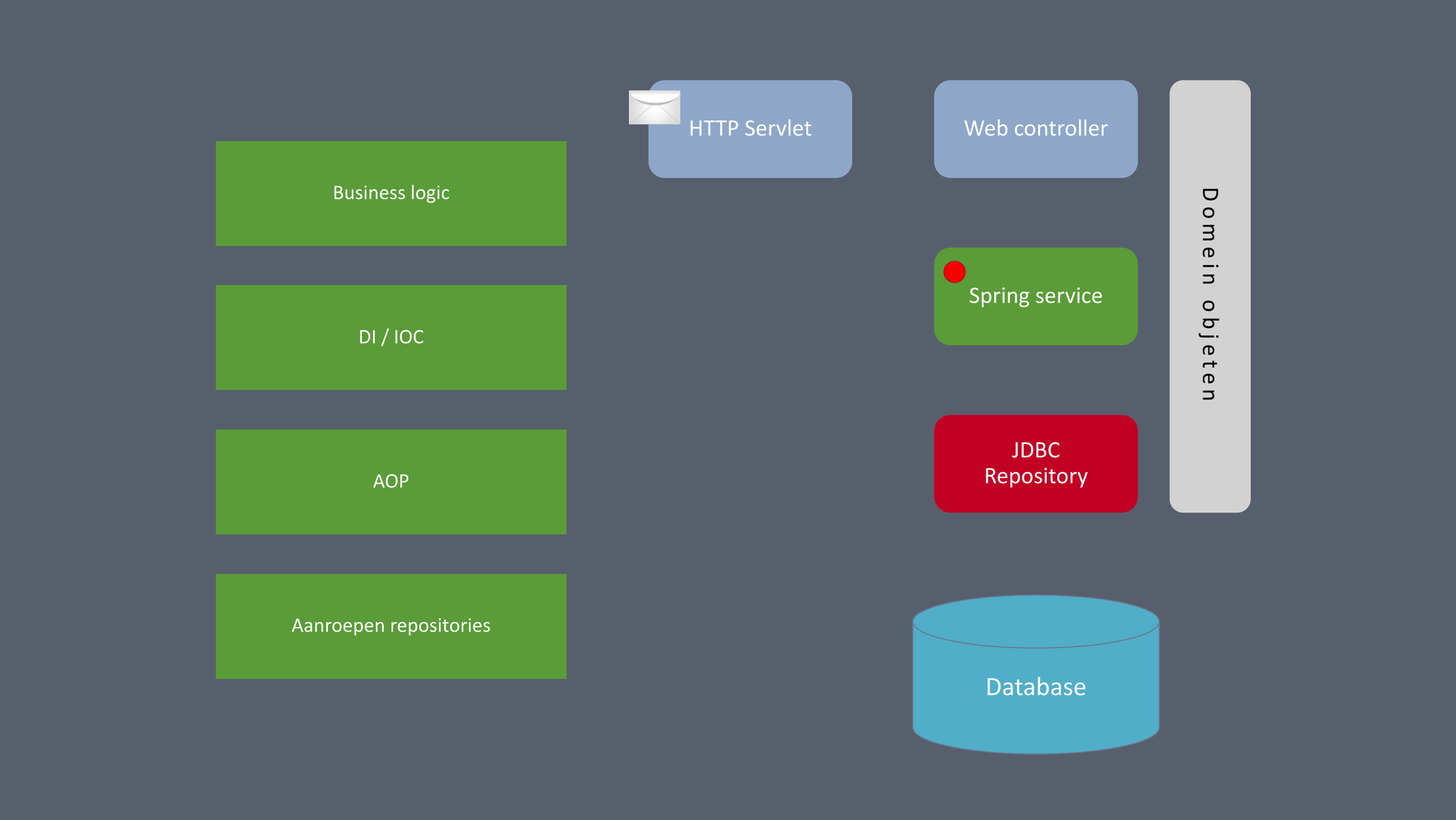
Spring service

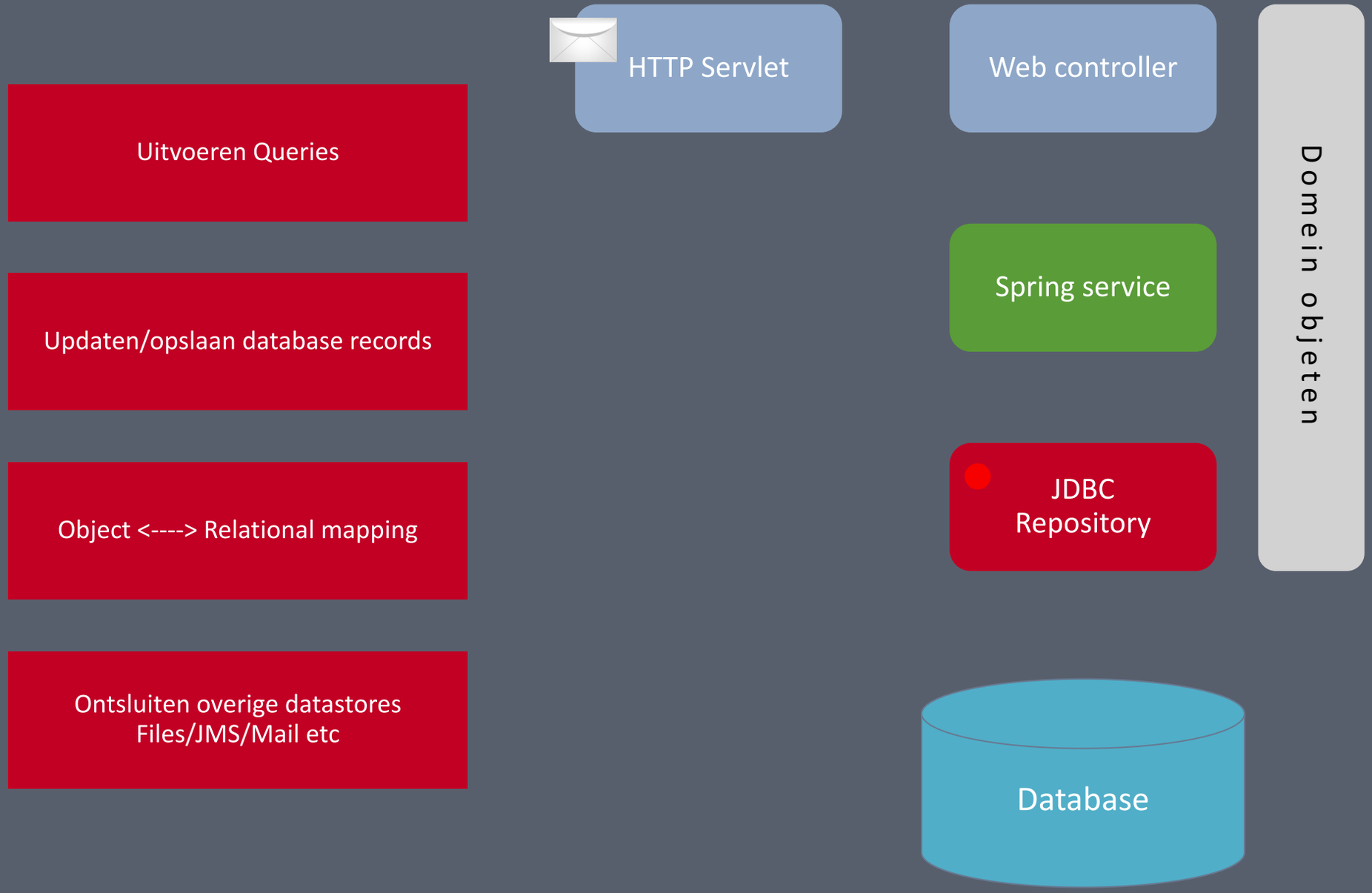
JDBC Repository

Domein objeten



Database


Business logic
DI / IOC
AOP
Aanroepen repositories
HTTP Servlet
Web controller
Spring service
JDBC
Repository
Domein objeten
Database


HTTP Servlet
Web controller
Domein objeten
Uitvoeren Queries
Spring service
Updaten/opslaan database records
JDBC
Repository
Object <----> Relational mapping
Ontsluiten overige datastores
Files/JMS/Mail etc
Database

POJO’s’teruggeven aan service laag

Eventueel andere repository raadplegen

Business logic uitvoeren -> POJO’s naar web controller

POJO’s omzetten naar response format

HTTP Servlet

Web controller

Spring service

JDBC Repository

Domein objeten

Database

REST

Representation State Transfer

Roy Fielding (http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm)

Architectuur stijl voor webservices

Gebruik het web zoals het bedoeld is, met wat extra afspraken

REST constraints

REST constraints

Client-Server

Stateless

Interface Standarisatie

REST constraints

Client-Server

Stateless

Interface Standarisatie

Uniform interface

Welke resources

Bewerken resources via representatie

Zelfbeschrijvende berichten

Uniform interface

Uniform interface - data elements

Table 5-1: REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

REST and HTTP

- REST is een *extensie* van het Web
- HTTP 1.1 is ontworpen om te voldoen aan REST
- De HTTP methodes zijn voldoende om alles te doen
- HTTP is het meest RESTful protocol

Toepassing REST: eenvoudig?

Toepassing REST – eenvoudig?

- GET: /getAccount
- GET: /getAllAccounts
- GET: /searchAccounts
- PUT: /updateAccountNumber?number=1234
- PUT: /updateAccountBalance?balance=200
- PUT: /updateAccountEmail?email=rest@gmail.com

URL bestaande uit zelfstandige naamwoorden is goed, werkwoorden (meestal) niet!

Toepassing REST: hoe het wel moet

Toepassing REST: hoe het wel moet

- GET: /Accounts
- GET: /Accounts/12345
- PUT: /Accounts/12345
- POST: /Accounts
- DELETE: /Accounts/12345

URL bestaande uit zelfstandige naamwoorden is goed, werkwoorden (meestal) niet!

Gebruik standaard methoden

POST	Create
GET	Retrieve (safe, cacheable, idempotent)
PUT	Update (idempotent)
DELETE	Delete (idempotent)
PATCH	Partial Update(idempotent)

Alleen ophalen gegevens
Geen side-effects

Niet steeds zelfde gegevens ophalen

Meerdere keren aanroepen, met zelfde resultaat mogelijk

POST voor Create

POST /klanten
{
voornaam: John,
achternaam: Do,
email: john@gmail.com,
}

Response:
201 created
Location: /klanten/12345

PUT voor Update

Volledige update!

```
PUT /klant/12345
{
voornaam: John,
achternaam: Do,
email: john@gmail.com,
}
```

Idempotent!

PATCH

Deels update

```
PUT /klant/12345
{
voornaam: Jane
}
```

HTTP Body

- HTTP header Content-Type -> meta data body
- Voor REST is JSON niet verplicht

JSON conventies

camelCase	HuisNummer i.p.v. huis_nummer
datums	ISO 8601 (2010-01-01T12:00:00+01:00)
quotes	Gebruik dubbele quotes, “voornaam”
Lege tags	Voorkom lege tags of null waardes

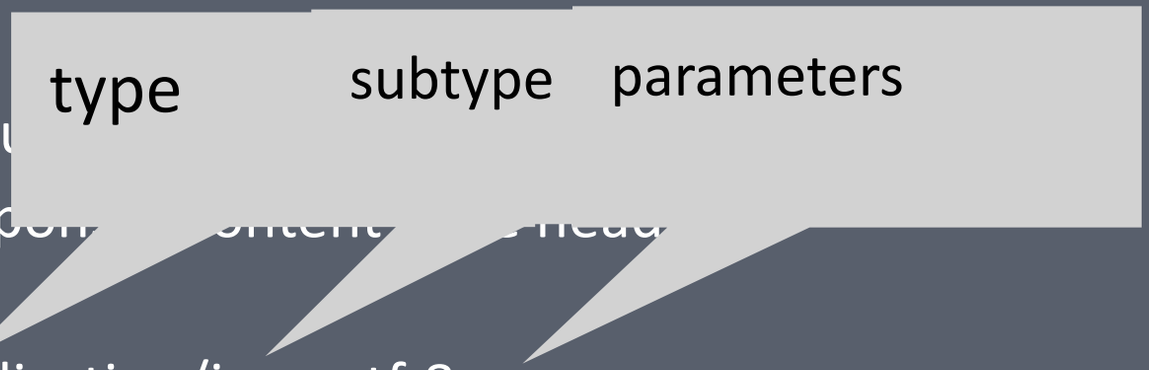
JSON

(JavaScript Object Notation)

```
{
   “persoon": {
      “voornaam": “John",
      “achternaam": “Do”,
      “adressen": [
            {
                “straat” : “Ubbo Emmiussingel”,
                “huisnummer” : “112”
            },
                “straat” : “Maanlander”,
                “huisnummer” : “14M”
            }
       ]
    } //einde persoon
} //einde object
```

Media types

- Opmaak specificatie voor resources
- Onderdeel van contract tussen client en server
- Content Negotiation



- Requ
- Respon ontent -head

- application/json;utf-8
- application/foo+json
- application/xml

REST en Java

- Jersey - JAX-RS – Reference Implementation
- Restlet
- RestEasy
- RestExpress

Veel nieuwe Microservices frameworks:

DropWizard

Ratpack

Ninja

Spring – RestController RequestMapping

```java
@RestController
public class MusicController {
    @Autowired
    AlbumRepository albumRepository;

    @RequestMapping(value = "/album", method = RequestMethod.GET)
    public Iterable<Album> getAllAlbums() {
        return albumRepository.findAll();
    }

    @RequestMapping(value = "/album", method = RequestMethod.POST)
    public Album createAlbum(@RequestBody Album album) {
        album.setId(null);
        return albumRepository.save(album);
    }
```

Spring – RestController
PathVariable & RequestBody

```
@RequestMapping(value = "/album/{id}", method = RequestMethod.GET)
public Album getAlbum(@PathVariable Long id) {
    return albumRepository.findOne(id);
}

@RequestMapping(value = "/album/{id}", method = RequestMethod.PUT)
public Album updateAlbum(@PathVariable Long id, @RequestBody Album album) {
    album.setId(id);
    return albumRepository.save(album);
}
```

Spring – RestController
RequestParam & defaultValue

```
    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                String.format(template, name));
    }
}
```

Opdracht 1

- Maak een Spring Boot hello world REST API
  - In een controller package
- Initiële maven project beschikbaar
- Aditionele informatie is hier te vinden:
  - https://spring.io/guides/gs/rest-service/
- POSTMAN Test script => TDD!
  - Zorg dat de Step 1 testen slagen!

Object Relational Mapping (ORM)

Waarom database?

- opslaan grote hoeveelheden data efficient
- gestructureerd
- snel en makkelijk data vinden
- makkelijk data toevoegen, aanpassen of verwijderen
- uitgebreide zoekfunctionaliteit inc. pagineren en sorteren
- exporteren en importeren van data
- multi-access van uit meerdere personen/applicaties
- secure

Databases

Voldemort

VortexDB

IBM DB2

dBase

Oracle

MySQL

MariaDB

VelocityDB

Cassandra

Altibase

SQLite

eXtremeDB

Progress

ProstgreSQL

FoxPro

Microsoft Access

SAP

CouchDB

RaptorDB

Terrastore

HSQLDB

Microsoft SQL Server

LibreOffice Base

Graphbase

SimpleDB

MongoDB

Elasticsearch

Relationele Databases

- schema
- tabel, rijen en kolommen
- sleutels en relaties, primary en foreign keys
- data types: alphanumeric, numeric, date/time
- indexen
- stored procedure
- views

Structured Query Language (SQL)

Insert	insert into gebruiker (loginnaam, password) values ('jan', 'pwd');
Delete	delete from gebruiker where loginnaam = 'jan';
Update	update gebruiker set password = 'secret' where loginnaam = 'jan';
Select	select password from gebruiker where loginnaam = 'jan';
Functions	AVG() - Returns the average value COUNT() - Returns the number of rows FIRST() - Returns the first value LAST() - Returns the last value MAX() - Returns the largest value MIN() - Returns the smallest value SUM() - Returns the sum

JDBC

Connect naar data source

```
Connection con =
DriverManager.getConnection(
      "jdbc:myDriver:myDatabase",
      username,
      password);
```

Query/update

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT
a, b, c FROM Table1");
```

Verwerk resultaat

```
  while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        float f = rs.getFloat("c");
  }
```

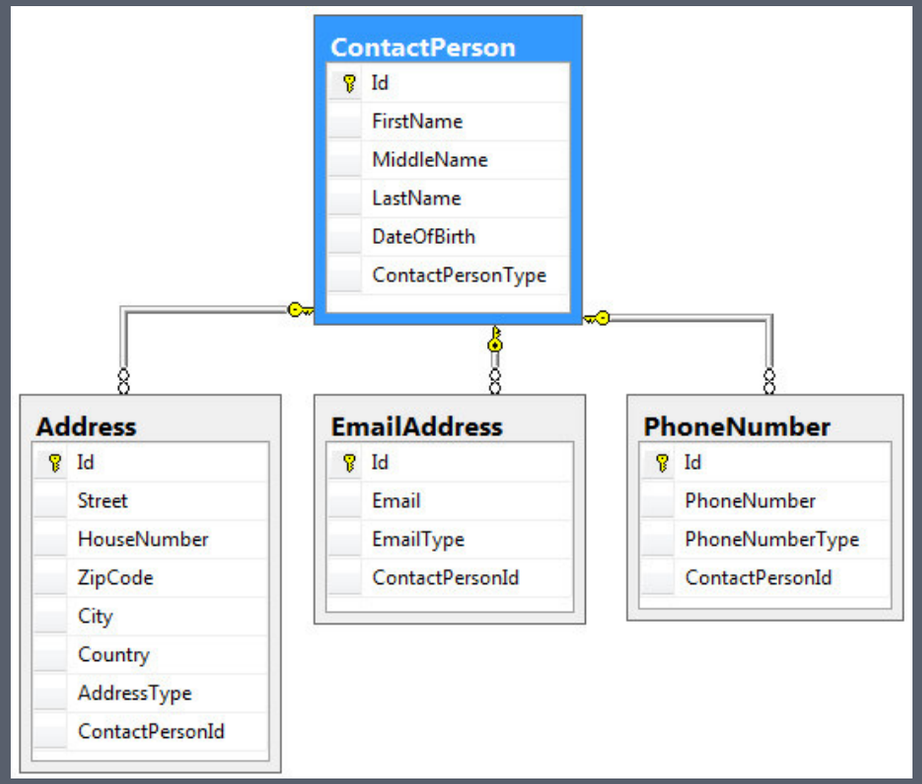
Waarom ORM?

- minder 'handmatige' mapping
- minder bijhouden persistence state van objectbomen
- 1 object model
- geen of minder low level sql code
- database onafhankelijk
- oplossingen/patterns voor meest voorkomende problemen (bijv. locking, caching)

Object/Relational Mapping (ORM)

Persistence

we willen graag dat de state van (sommige) objecten gepersisteerd blijft buiten de scope van de JVM zodat dezelfde informatie later beschikbaar is.



Object/Relational Mapping (ORM)

The Object-Relational Mismatch

Granulariteit

Subtypes (overerving)

Identity

Relaties

Navigeren over gegevens

Wat is JPA?

package javax.persistence

JPQL

object/relational metadata

Entity class

@Entity	Specificeert dat de klasse een entiteit is. Elementen: name (Optioneel) De naam van de entiteit
@Table	Specificeert de tabel voor de geannoteerde entiteit Elementen: • name (Optioneel) De naam van de tabel • catalog (Optioneel) De 'catalogus' van de tabel • schema (Optioneel) Het schema van de tabel

```
@Entity
@Table(name="CUST", schema="RECORDS")
public class Customer {...}
```

Columns

@Column

Wordt gebruikt om een mapped kolom te specificeren voor een persistent veld of eigenschap.

Elementen:

- name (Optioneel) De naam van de kolom
- unique (Optioneel) Of de kolom een unieke waarde moet bevatten

```
@Column(name="DESC", nullable=false, length=512)
private String description;


@Column(name = "COST", columnDefinition="Decimal(10,2) default '100.00'")
public BigDecimal getCost() { return cost; }
```

Identity

@Id	Specificeert de primary key van een entiteit
@GeneratedValue	Specificeert de strategie om primary keys te genereren Elementen: • strategy (Optioneel) De strategie om primary keys te genereren • generator (Optioneel) De naam van de primary key generator(SequenceGenerator of TableGenerator)

```
@Id
@GeneratedValue( strategy = GenerationType.IDENTITY )
private Long id;
```

Id generation

Identity

Voorbeeld van een database identity kolom

```
CREATE TABLE USERS(
     ID BIGINT NOT NULL

AUTO_INCREMENT,

     ……..
     PRIMARY KEY(ID)
);
```

Sequence

Voorbeeld van een database sequence

```
CREATE SEQUENCE
USERS_SEQ
START WITH 1 INCREMENT
BY 1;
```

Table

Voorbeeld van een database table

```
CREATE TABLE
APP_SEQ_STORE(
  APP_SEQ_NAME
VARCHAR(255)
       NOT NULL,
  APP_SEQ_VALUE BIGINT
       NOT NULL,
PRIMARY
KEY(APP_SEQ_NAME));
```

Transient

@Transient

Specificeert dat de eigenschap of veld niet gepersisteerd hoeft te worden

```
@Entity
public Person {

    private Gender g;

    @Transient
    private String temp;
}
```

Enum

@Enumerated

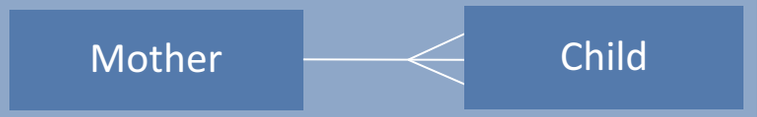
Specificeert de mapping van een veld van type Enum
Value: ORDINAL, STRING

```
@Entity
public Person {


    @Enumerated(EnumType.STRING)
    private Gender gender;


}
```

Relations

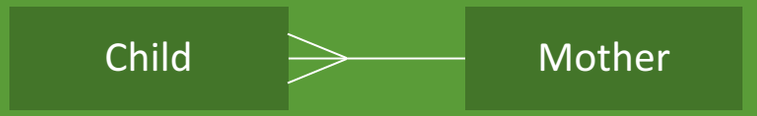
@OneToMany

```
@Entity
public class Mother {
    private List<Child> children;
}
```



@ManyToOne

```
@Entity
public class Child {
    private Mother mother;
}
```

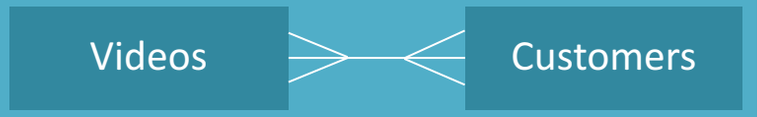


@OneToOne

```
@Entity
public class Husband {
    private Wife wife;
}
```

@ManyToMany

```
@Entity
public class Video {
    private List<Customer> customers;
}
```



Owning Side and Inverse Side

Bidirectioneel	Bij een bidirectionele relatie hebben beide objecten een referentie naar elkaar
Unidirectioneel	Bij een unidirectionele relatie heeft alleen de eigenaar een referentie naar het andere object.

```
public class Mother {
  private List<Child> children;
}
public class Child {
  private Mother mom;
}
```

```
public class Mother {
  private List<Child> children;
}
public class Child {
}
```

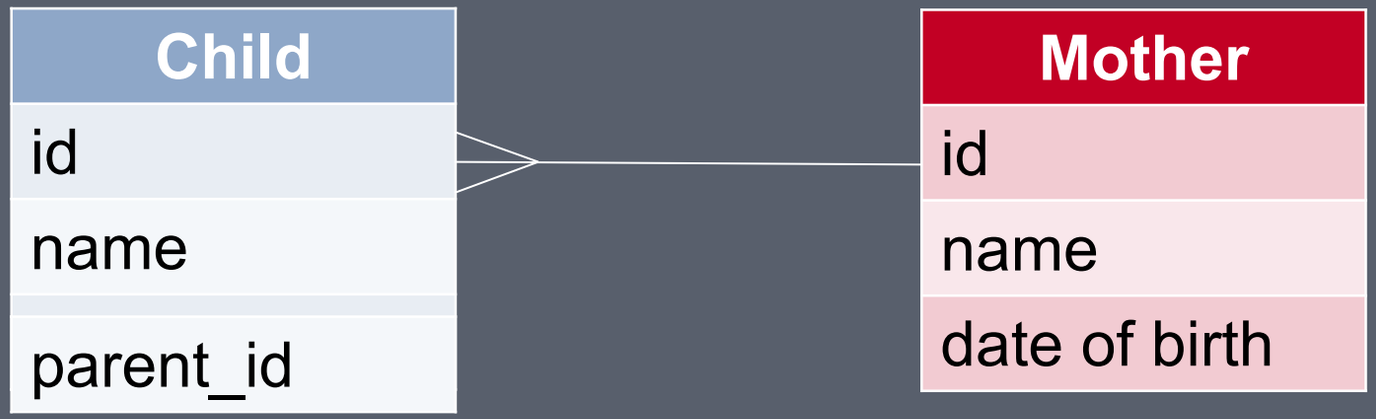
Bidirectioneel vs Unidirectioneel

Bidirectioneel	• beide kanten op navigeren • minder queries
Unidirectioneel	• bij veel many’s • minder foutgevoelig

ManyToOne

@ManyToOne

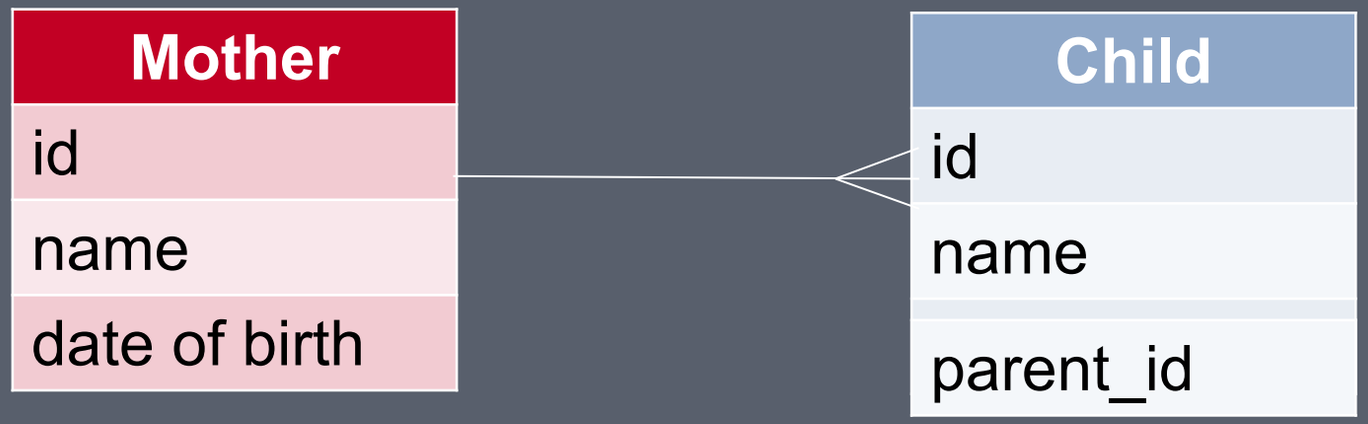
```
@Entity
public class Child {
    ...
    @ManyToOne
    @JoinColumn(name="parent_id")
    private Mother mom;
}
```



OneToMany

@OneToMany

```
@Entity
public class Mother {
     …
    @OneToMany(mappedBy = “mom”)
    private List<Child> children;
}
```



OneToOne

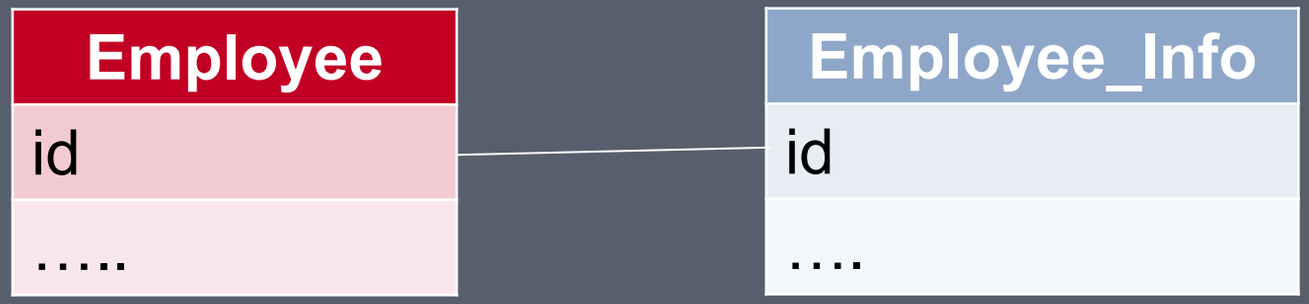
@OneToOne

```
@Entity
public class Husband {

    @OneToOne
    @JoinColumn(name = "partner_id")
    private Wife wife;
}
```

Deel dezelfde primary key waarde

@OneToOne

```
@Entity
public class Employee {

    @OneToOne
    @MapsId
    private EmployeeInfo info;
}
```



ManyToMany

@ManyToMany

```
@Entity
public class Video {

    @ManyToMany
    @JoinTable(name="Vid_Cust",
      joinColumn =
        @JoinColumn(name="vid_id",
              referencedColumnName="id")
      inverseJoinColumns =
        @JoinColumn(name="cust_id",
              referencedColumnName="id"))
    private List<Customers> customers;
}
```

Opdracht 2

- Maak een Album entity aan (id, name, artist, year)
  - In een model package
- Maak een AlbumRepository o.b.v. CrudRepository interface aan
  - In een dao package
- Voeg een Music RestControler toe (GET, POST, PUT, DELETE)
- Initiële maven project eventueel beschikbaar
- Aditionele informatie is hier te vinden:
  - https://spring.io/guides/gs/accessing-data-jpa/
- POSTMAN Test script => TDD!
  - Zorg dat de Step 2 testen slagen!

HTTP Headers

Request headers

Host: www.quintor.nl
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: nl-NL,nl;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: JSESSIONID=4ryvruxwpyaymh98yumd1yc2

Response headers

Set-Cookie: JSESSIONID=yrm5092tl40718xpggc2sm81h;Path=/module3
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Server: Jetty(9.1.0.v20131115)

HTTP return codes

1XX: informational

2XX: Success

3XX: Redirection

4XX: Client Error

5XX: Server Error

100: Continue

101: Switching Protocols

102: Processing (WebDAV)

HTTP return codes

1XX: informational

2XX: Success

3XX: Redirection

4XX: Client Error

5XX: Server Error

200: OK

201: Created

204: No Content

206: Partial content

HTTP return codes

1XX: informational

2XX: Success

3XX: Redirection

4XX: Client Error

5XX: Server Error

301: Moved Permanently

302: Found

304: Not Modified

307: Temporary redirect

308: Permanent redirect

HTTP return codes

1XX: informational

2XX: Success

3XX: Redirection

4XX: Client Error

5XX: Server Error

400: Bad Request

403: forbidden

404: Not Found

405: Method not Allowed

408: Request Timeout

HTTP return codes

1XX: informational

2XX: Success

3XX: Redirection

4XX: Client Error

5XX: Server Error

500: Internal Server Error

501: Not Implemented

502: Bad Gateway

505: HTTP Version not Supported

Opdracht 3

- Voeg fout afhandeling toe aan de Album REST API
- Voeg Location Header tot aan create response
- Initiële maven project eventueel beschikbaar
- POSTMAN Test script => TDD!
  - Zorg dat de Step 3 testen slagen!

Business laag

Functionaliteit

- Hart van het systeem
- Belangrijkste functionaliteit
- Implementeren use case beschrijvingen
- Gebruiken van data uit andere systemen

Transactie management

- Business laag bepaalt scope van de transactie
- Coördineert transacties over meerdere bronnen

Security

- Authenticatie
- Autorisatie
- Wie mag wat doen?
- RBAC
- ACL

Caching

- Caching van resultaten service laag (berekeningen etc)
- Zorgt voor minder calls naar achterliggende systemen

Spring framework

Alternatief voor J(2)EE en EJB

Geen applicatieserver (Glassfish, Websphere) nodig, alleen servlet container (Jetty, tomcat)

Heeft gezorgd voor nieuwe manier van ontwikkelen, met name testbaarheid en patterns

Begonnen als klein/lightweight framework
Inmiddels enorm uitgebreide codebase

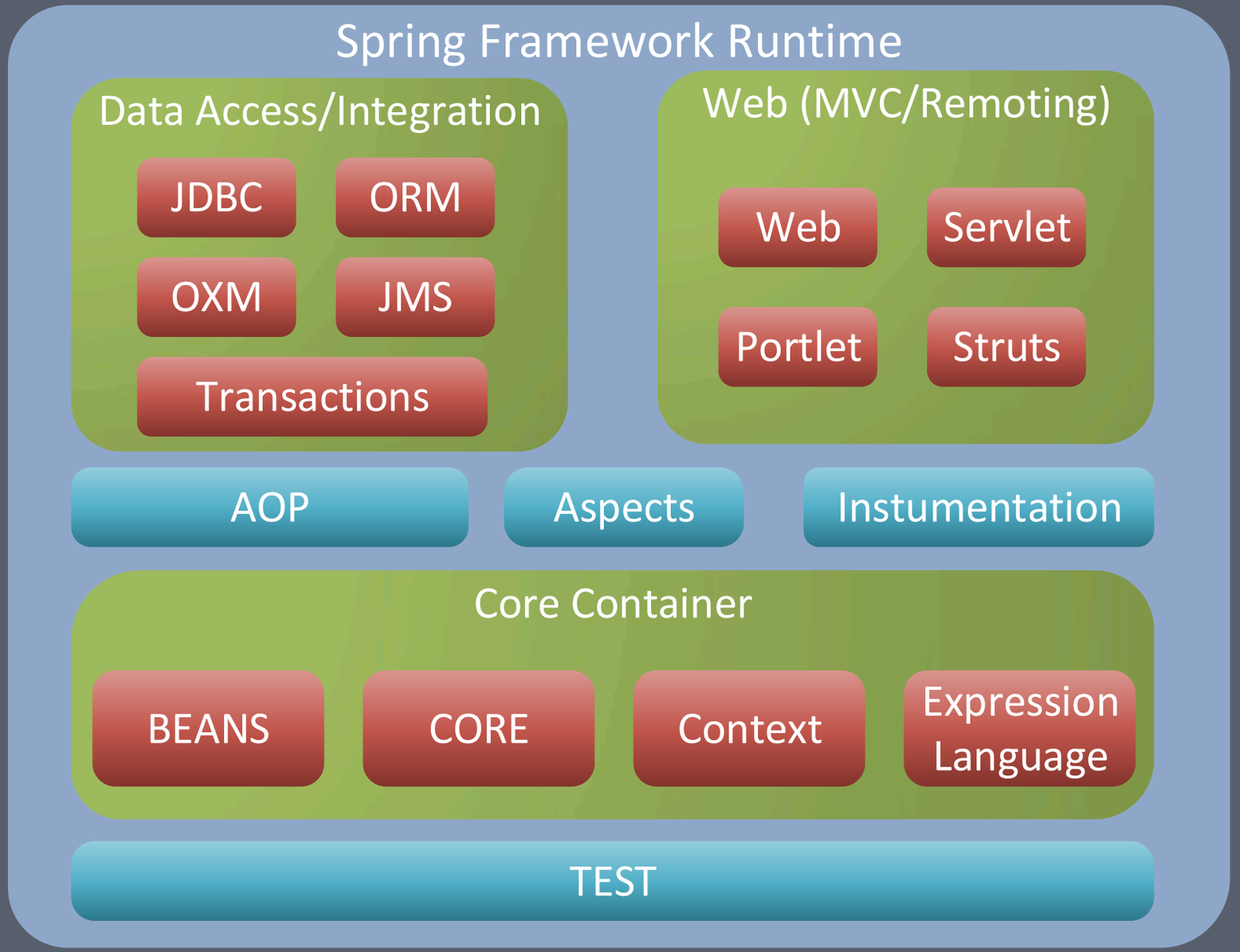
POJO voorbeeld

```
package nl.quintor.spring;

public class HelloWorldBean {
	public String sayHello(){
		return "HelloWorld";
	}
}
```

Spring strategieën

Lightweight, minimal invasive, POJO's	Simpel te gebruiken. Geen verplichtingen.
Loose coupling, Dependency Injection, Interfaces	Zo weinig mogelijk raakvlakken met Spring en ander frameworks
Declarative programming through aspects and common conventions	AOP en automatische configuratie
Boilerplate reduction through aspects and templates	Geen knippen en plakken van code

Spring modules



Spring framework artfiacts (300+)

spring-parent
spring-full
beandoc
spring-batch-parent
spring-framework-bom
spring-boot-archetypes
spring-plugin
spring-boot-build
spring-boot-tools
spring-boot-starters
spring-boot-dependencies
spring-boot-parent
spring-data-releasetrain
spring-ws
spring-ws-parent
spring-amqp-parent
spring-flex-parent
spring-flex
spring
spring-ldap-parent
spring-commons-parent
spring-integration-parent
spring-osgi
spring-jdo
spring-support
spring-jpa
spring-jmx
spring-mock
spring-hibernate3
spring-hibernate2
spring-toplink
spring-dao
spring-ibatis
spring-jca
spring-portlet
spring-remoting
spring-hibernate
spring-ojb
spring-js-resources
spring-integration-bom
spring-security-bom
spring-data-rest-parent
spring-data-rest-distribution
spring-data-cassandra-distribution
spring-data-cassandra-parent
spring-data-neo4j-parent
spring-data-neo4j-distribution
spring-data-mongodb-parent
spring-data-mongodb-distribution
spring-data-build
spring-data-build-resources
spring-data-parent
spring-messaging
spring-instrument
spring-core
spring-jdbc
spring-context
spring-jms
spring-orm
spring-tx
spring-expression
spring-websocket
spring-beans
spring-webmvc
spring-oxm
spring-test
spring-aspects
spring-web
spring-aop
spring-struts
spring-security-oauth-parent
spring-boot-sample-xml-archetype
spring-boot-sample-websocket-archetype
spring-boot-sample-traditional-archetype
spring-boot-sample-tomcat-archetype
spring-boot-sample-simple-archetype
spring-boot-sample-servlet-archetype
spring-boot-sample-secure-archetype
spring-boot-sample-profile-archetype
spring-boot-sample-jetty-archetype
spring-boot-sample-integration-archetype
spring-boot-sample-batch-archetype
spring-boot-sample-aop-archetype
spring-boot-sample-amqp-archetype
spring-boot-sample-actuator-archetype
springloade
spring-boot
spring-boot
spring-boot
spring-boot
spring-ws-a
spring-data-
spring-data-commons-dist
spring-data-commons-parent
spring-asm
spring-batch-admin-parent
spring-data-neo4j-dist
spring-security-parent
spring-integration-httpinvoker
spring-agent
spring-webmvc-struts
spring-tomcat-weaver
spring-security-samples-cas
spring-security-samples
spring-data-graph-dist
spring-data-graph-parent
spring-ldap-integration-tests
spring-ldap-integration-tests-parent
spring-ldap-ldif
spring-ldap-parent-tiger
spring-integration-adapter
spring-osgi-annotation
spring-osgi-core
spring-osgi-extender
spring-osgi-mock
spring-osgi-test
spring-osgi-web
spring-osgi-io
spring-batch
spring-integration-samples
spring-binding
spring-webflow
maven-spring-skin
spring-batch-docs
simple-cli-archetype
spring-batch-archetypes
spring-batch-samples
spring-instrument-tomcat
spring-context-support
spring-cloud-heroku-connector
spring-cloud-cloudfoundry-connector
spring-cloud-spring-service-connector
spring-cloud-core
spring-binding
spring-faces
spring-webflow
spring-js
spring-integration-jdbc
spring-integration-xmpp
spring-integration-http
spring-integration-security
spring-integration-jmx
spring-integration-mail
spring-integration-mongodb
spring-integration-ftp
spring-integration-amqp
spring-integration-groovy
spring-integration-scripting
spring-integration-test
spring-integration-twitter
spring-integration-gemfire
spring-integration-redis
spring-integration-mqtt
spring-integration-feed
spring-integration-rmi
spring-integration-xml
spring-integration-core
spring-integration-event
spring-integration-syslog
spring-integration-stream
spring-integration-sftp
spring-social-google
spring-batch-integration
spring-batch-core
spring-batch-infrastructure
spring-batch-test
spring-ws-support
spring-ws-core
spring-ws-security
spring-ws-test
spring-xml
spring-security-acl
spring-security-cas
spring-security-crypto
spring-security-samples-messages-jc
spring-security-core
spring-security-config
spring-security-web
spring-security-aspects
spring-security-openid
spring-security-remoting
spring-security-taglibs
spring-security-ldap
spring-ldap-core-tiger
spring-ldap-ldif-batch
spring-ldap-ldif-core
spring-ldap-test
spring-ldap-core
spring-ldap-odm
spring-data-rest-core
spring-data-rest-webmvc
spring-data-gemfire
spring-data-elasticsearch
spring-cql
spring-data-cassandra
spring-data-mongodb-cross-store
spring-data-mongodb-log4j
spring-data-mongodb
spring-data-jpa
spring-data-commons
spring-data-java8-stub
spring-hateoas
spring-security-oauth2
spring-security-oauth
spring-boot-sample-web-ui-archetype
spring-boot-sample-web-static-archetype
spring-boot-sample-web-secure-archetype
spring-boot-sample-web-method-security-archetype
spring-boot-sample-web-jsp-archetype
spring-boot-sample-data-rest-archetype
spring-boot-sample-data-redis-archetype
spring-boot-sample-data-mongodb-archetype
spring-boot-sample-data-jpa-archetype
spring-boot-sample-actuator-ui-archetype
spring-boot-sample-actuator-noweb-archetype
spring-boot-sample-actuator-log4j-archetype
spring-retry
spring-social-facebook
spring-social-facebook-web
spring-social-linkedin
spring-plugin-core
spring-plugin-metadata
spring-boot-starter-redis
spring-boot-starter-websocket
spring-boot-starter-integration
spring-boot-starter-test
spring-boot-starter-logging
spring-boot-maven-plugin
spring-boot-starter-jetty
spring-boot-loader-tools
spring-boot-starter-batch
spring-boot-starter-tomcat
spring-boot-starter-thymeleaf
spring-boot-loader
spring-boot-starter-data-rest
spring-boot-autoconfigure
spring-boot-starter-actuator
spring-boot-starter-amqp
spring-boot-starter-remote-shell
spring-boot-starter-mobile
spring-boot-starter-data-mongodb
spring-boot-dependency-tools
spring-boot-starter-web
spring-boot-starter-aop
spring-boot-starter-data-jpa
spring-boot-starter-jdbc
spring-boot-gradle-plugin
spring-boot-starter-log4j
spring-boot-starter-security
spring-boot-actuator
spring-boot-cli
spring-boot
spring-social-twitter
spring-social-config
spring-social-core
spring-social-web
spring-social-security
spring-security-jwt
spring-mobile-device
aws-maven
spring-plugin-integration
spring-data-hadoop
spring-social-test
spring-data-commons-core
spring-batch-admin-manager
spring-batch-admin-resources
spring-social-yammer
spring-android-auth
spring-android-rest-template
spring-android-core
spring-data-oracle
spring-data-jdbc-core
spring-data-envers
spring-data-rest-repository
spring-security-portlet
spring-security-ntlm
spring-security-core-tiger
spring-security-cas-client
spring-shell
spring-flex-core
spring-security-samples-tutorial
spring-security-samples-aspectj
spring-security-samples-cas-client
spring-security-samples-dms
spring-security-samples-ldap
spring-security-samples-contacts
spring-security-samples-openid
spring-security-samples-cas-server
spring-security-samples-preauth
spring-oxm-tiger
spring-oxm
spring-ws-core-tiger
spring-gemfire
spring-data-graph-core
spring-ldap
spring-osgi-bundle-archetype
spring-osgi-web-extender
spring-security-samples-portlet
spring-batch-infrastructure-tests
spring-security-adapters
spring-security-jetty
spring-security-jboss
spring-security-resin
spring-security-catalina

http://spring.io

Inversion Of Control (IOC)

Een concept voor applicatie ontwikkeling

“Don’t call us, we’ll call you”

Implementatie middels Dependency Injection (DI)

Minder code nodig

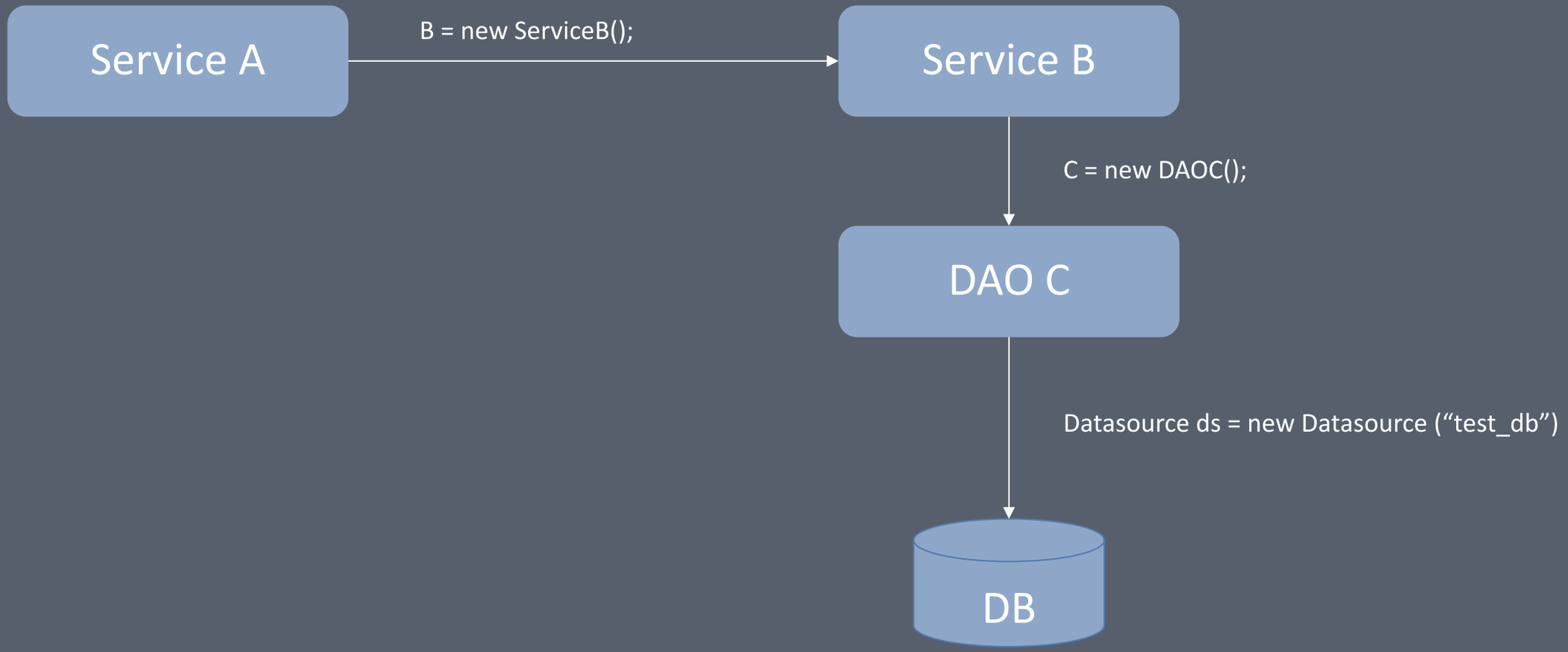
Code is beter testbaar

Code structuur wordt beter

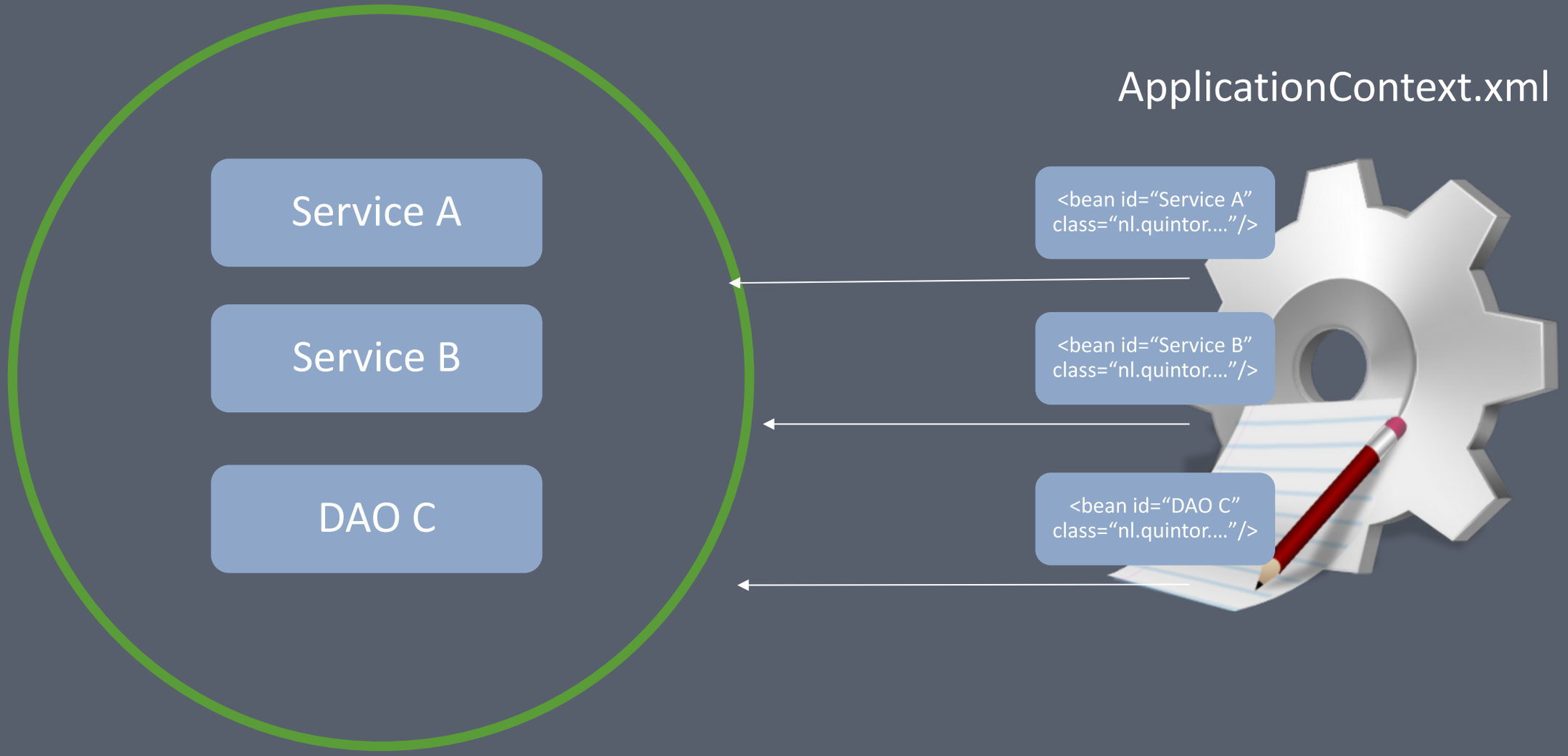
Zorgt voor loose coupling

Controle over object lifecycle

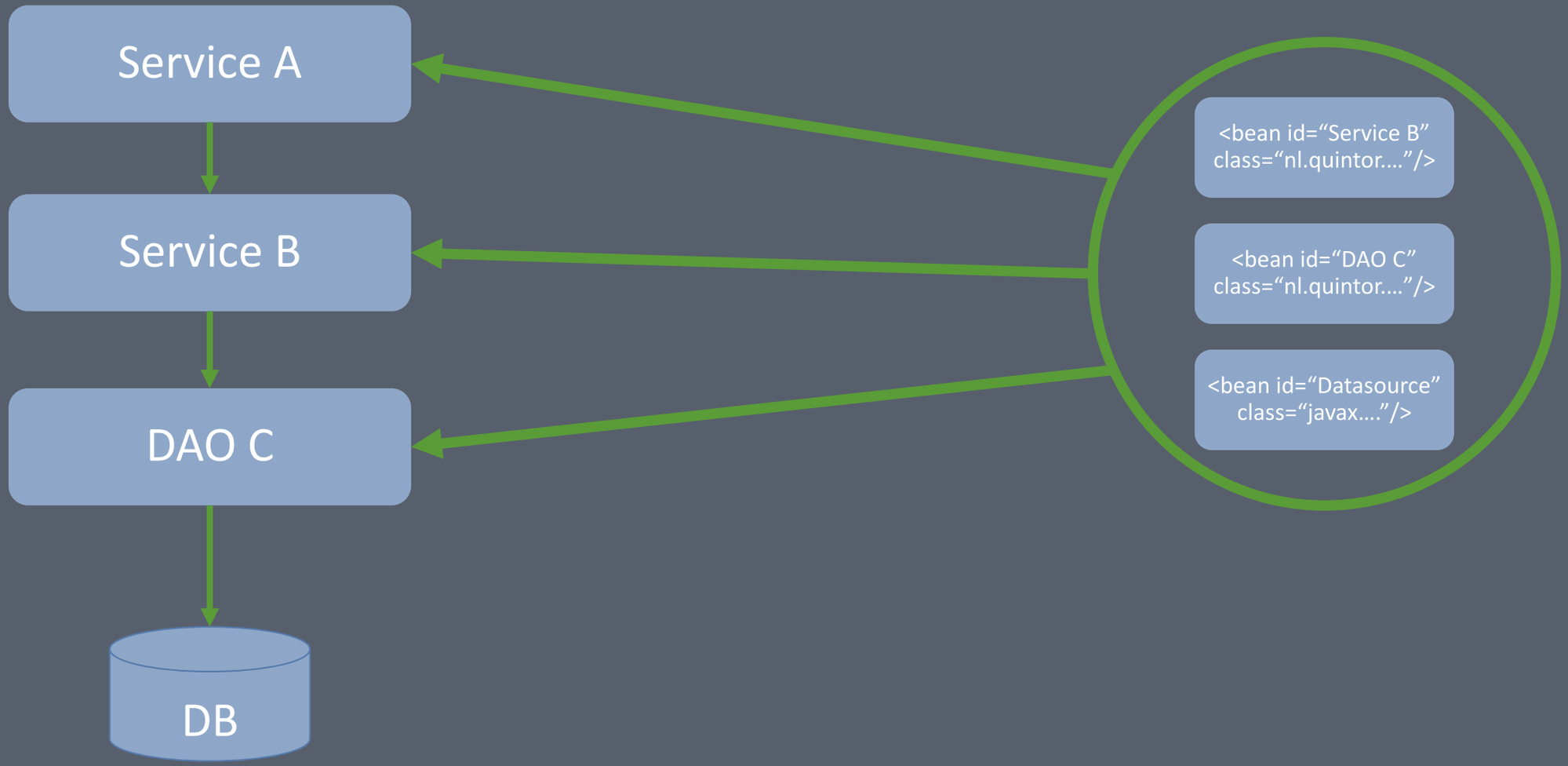
Dependency Injection



Dependency Injection



Dependency injection



Injection types

Constructor injection

```
public class KlantService {
        public KlantService(KlantRepository kr){
                this.klantRepository =kr;
        }
```

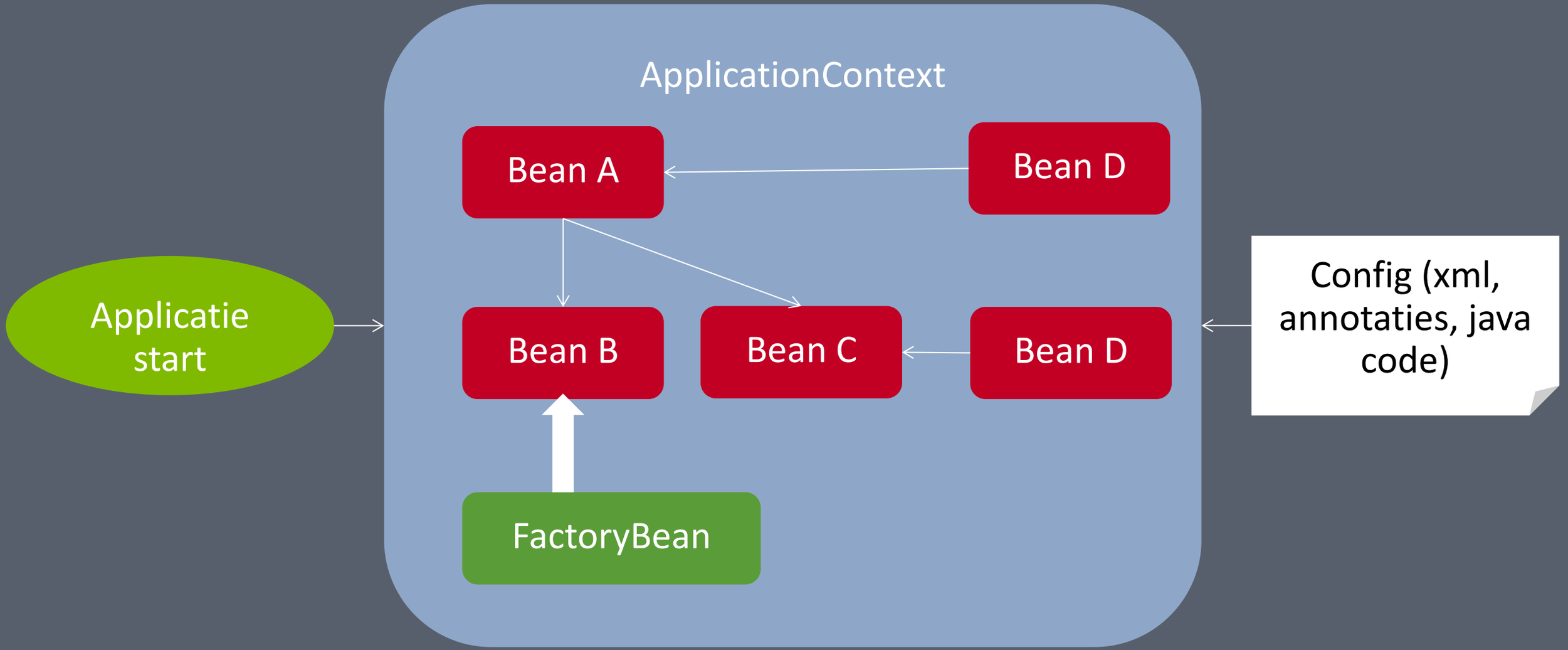
Setter injection

```
public class KlantService {
        public setKlantRepository(KlantRepository kr){
                this.klantRepository =kr;
        }
```

Interface injection

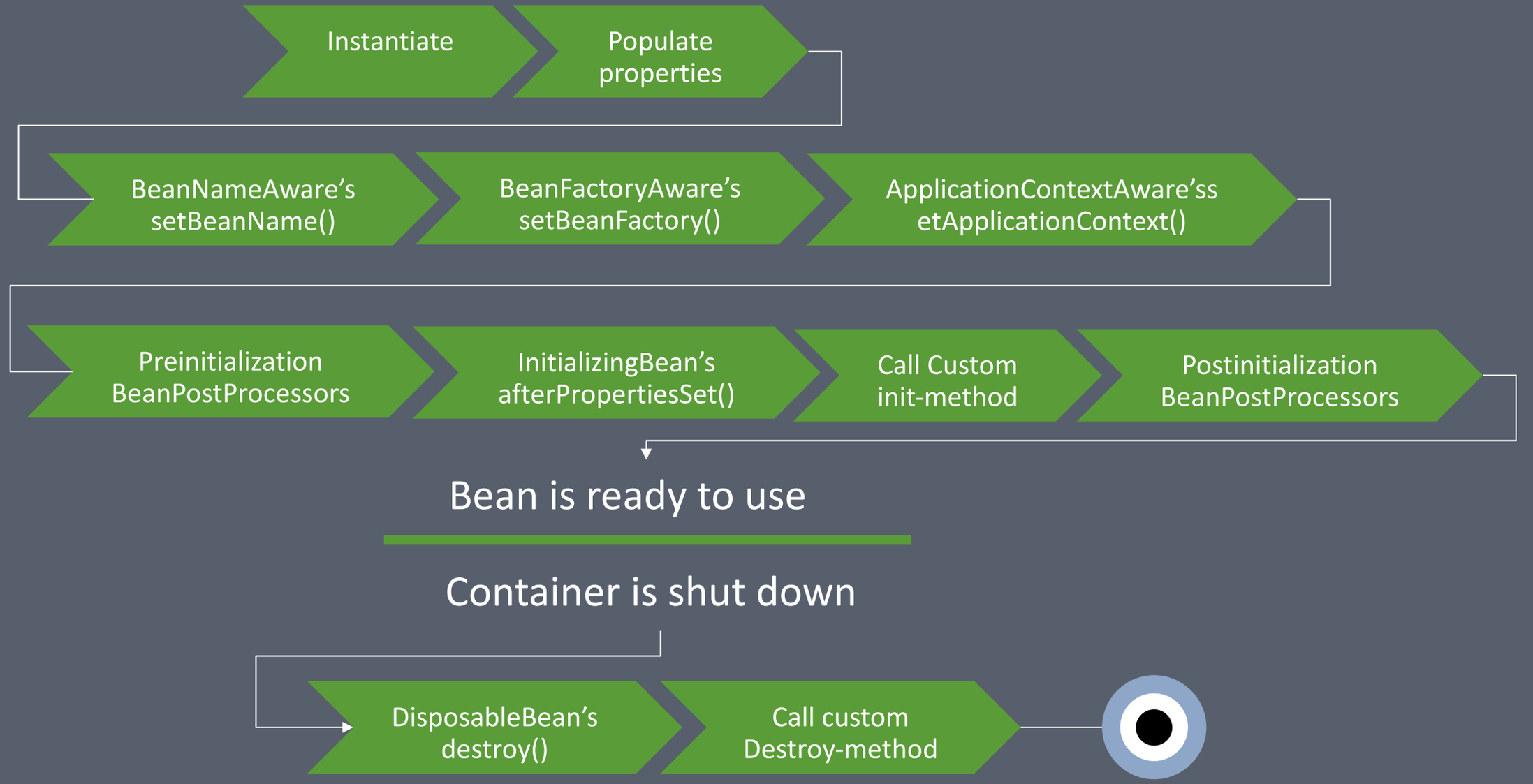
```
public class KlantService implements ApplicationContextAware {
        public setApplicationContext(ApplicationContext ctx){
                this.klantRepository = ctx.getBean(“klantRepo”);
        }
```

Core en Beans



Application lifecycle

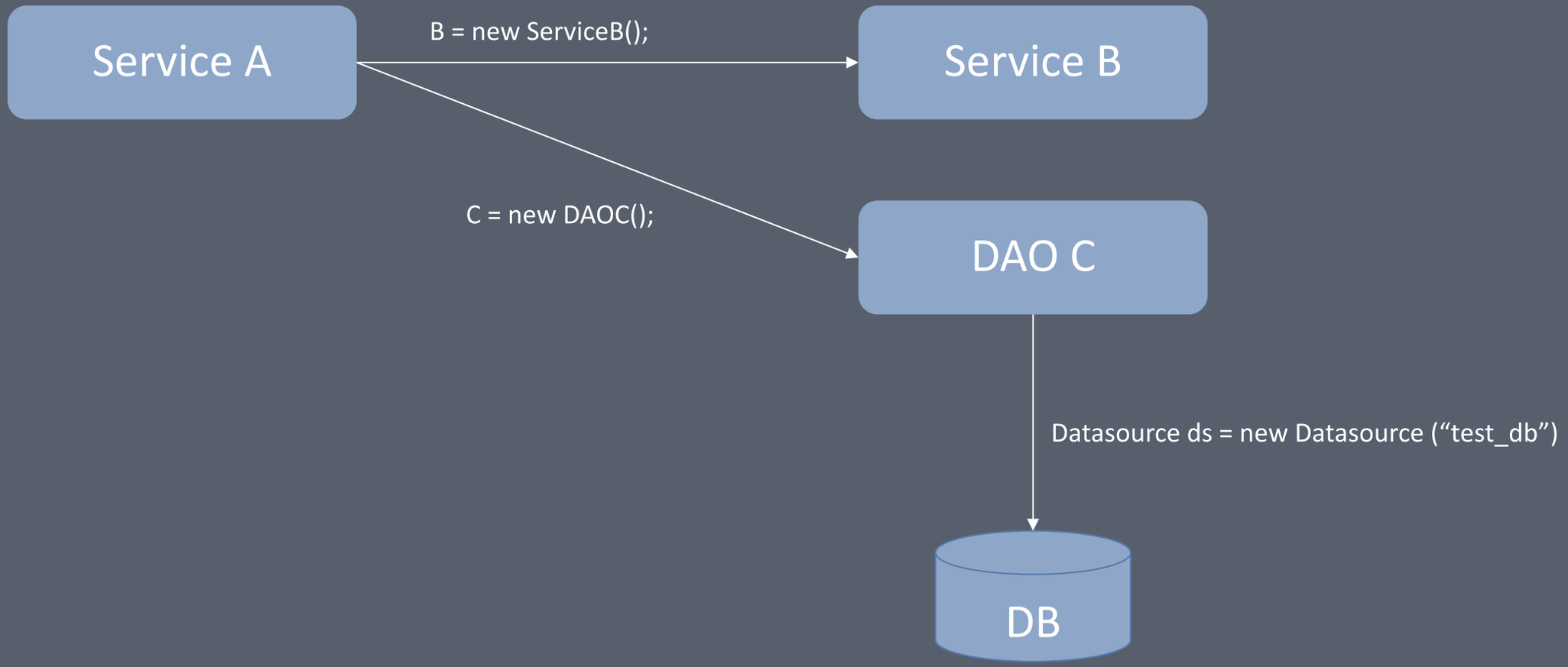
Bean lifecycle



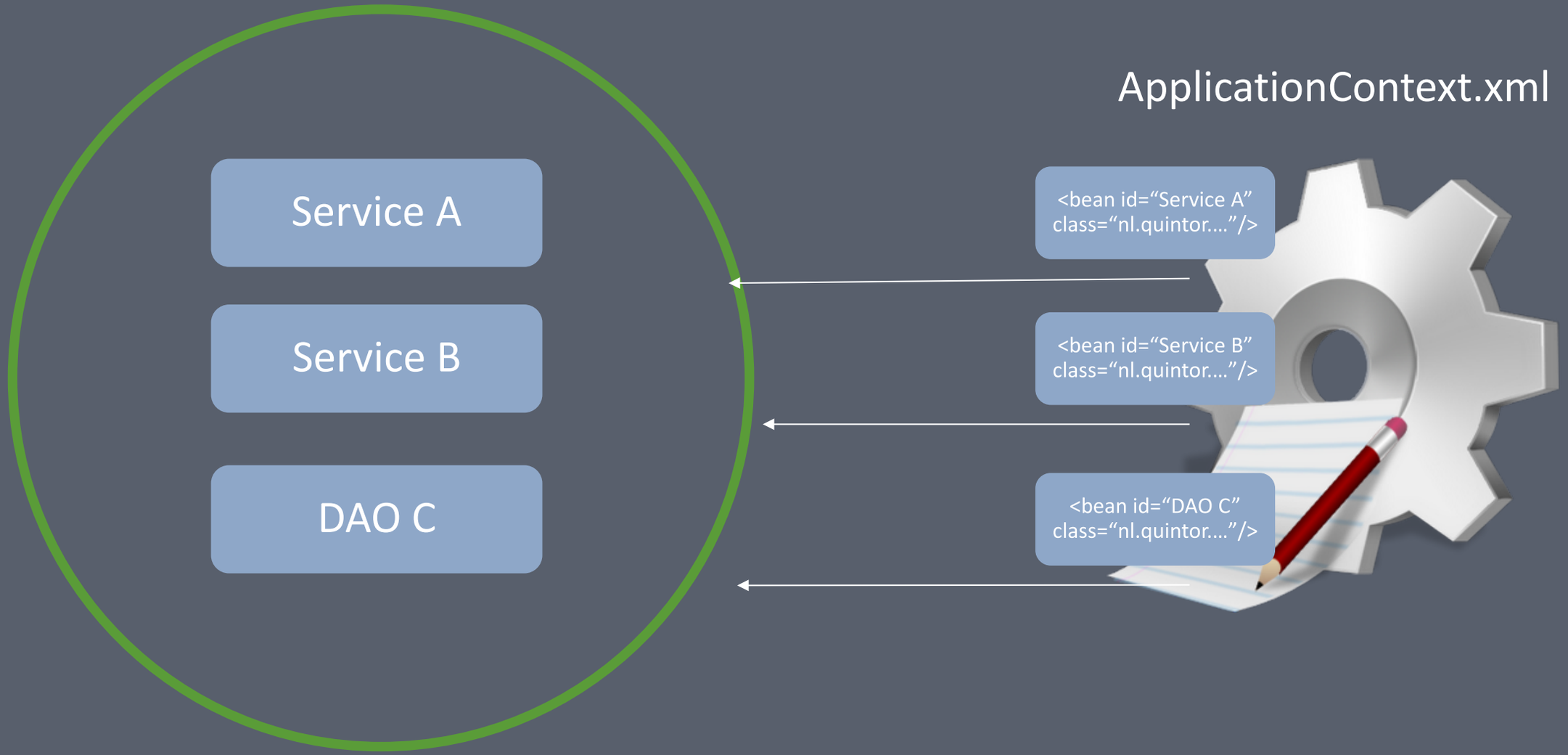
Bean scopes

Singleton	Één bean instantie wordt gedeeld door de gehele IoC container
Prototype	Enkele bean definitie heeft meerdere instanties (per aanroep)
Request/Session	Web application context only: Creëert een bean instantie per HTTP request/session
GlobalSession	Web portlet only: Creëert een bean instantie per globaal HTTP request

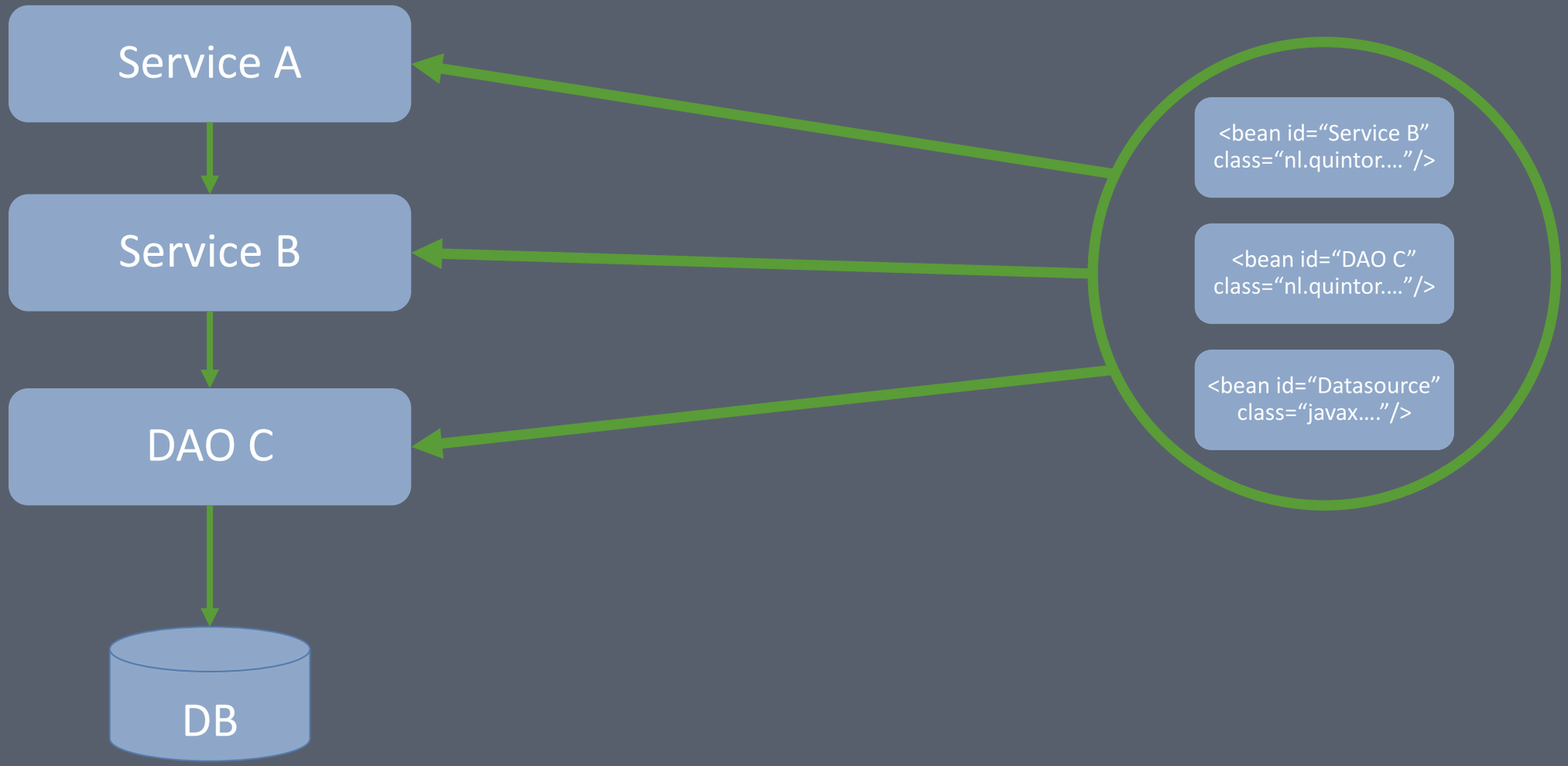
Dependency Injection



Dependency Injection



Dependency injection



Opdracht 4

- Voeg fout afhandeling toe aan de Album REST API
- Voeg Location Header tot aan create response
- Initiële maven project eventueel beschikbaar
- POSTMAN Test script => TDD!
  - Zorg dat de Step 3 testen slagen!