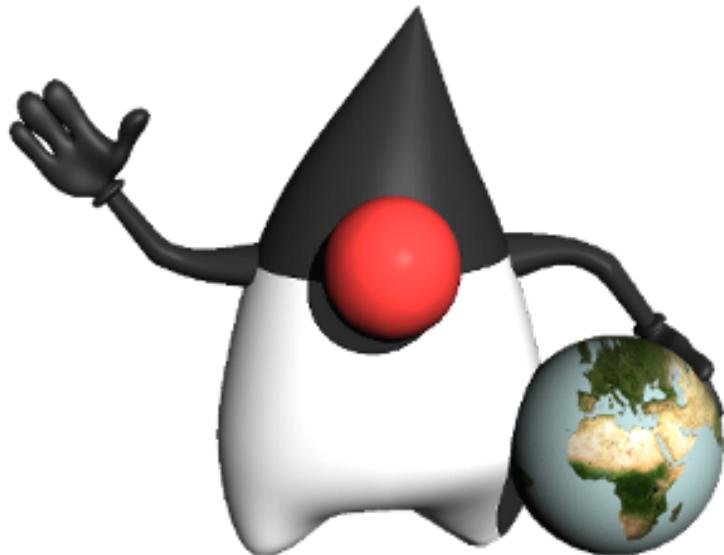


# PG6100

# Enterprise Programmering 2



# Om meg



**Jarle Hansen**

- [jarle.hansen@systek.no](mailto:jarle.hansen@systek.no)
- <https://github.com/jarlehansen>



# Gjennomføring

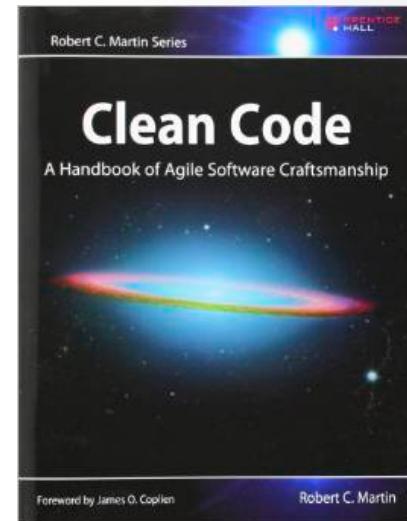
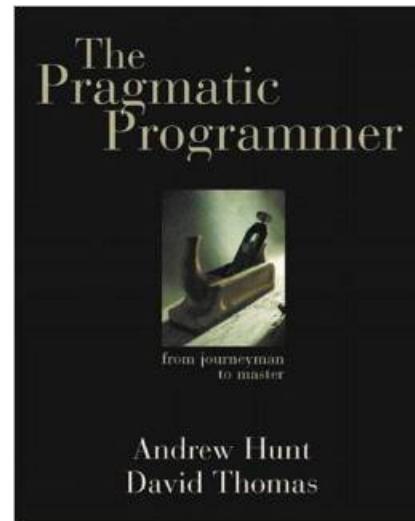
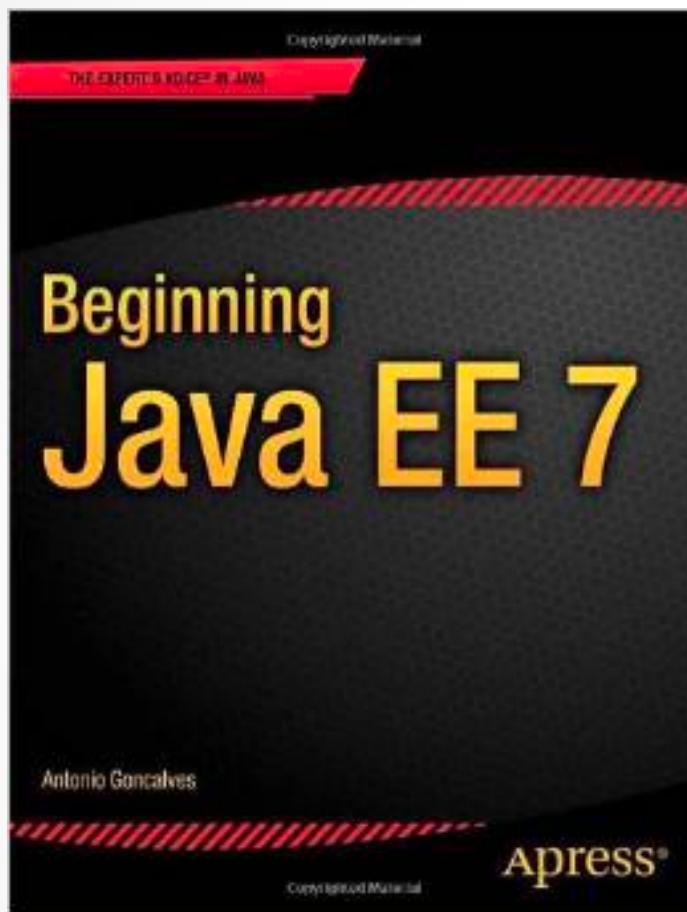
- Mandager 08:15-12:15
  - Forelesning
  - Egentrenering
  - Gruppearbeid
- Hvordan ønsker dere å bruke tiden?
  - 2 timer forelesning / 2 timer øving?
- Gjesteforelesning
- Forelesningsplan

# Hensikten med faget

- Bygger videre på arbeidet gjort i PG5100
- Videregående kunnskap om distribuerte systemer og komponentbasert utvikling
  - Java Enterprise Edition med tilhørende applikasjonsarkitektur
- Kunnskap om
  - Web Services
  - Meldinger
  - Sikkerhet
  - Transaksjoner
  - Brukerstyring
- Kompetanse mål

# Vurdering

- **Deleksamen 1**, skriftlig (25%) – uke 10
  - Ingen hjelpebidrifter
  - 3 timer
- **Deleksamen 2**, hjemmeeksamen (75%) – uke 22
- Karakteren settes på bakgrunn av
  - Teknisk nivå på innlevert arbeid
  - Grad av måloppnåelse av kompetansemål
  - Svar på teorioppgaver



## Filer tilgjengelig på ITL:

- Unit Testing Checklist
- Git Cheat Sheet
- Maven: The Complete Reference
- Pro Git

# Forelesningsplan

- [Tilgjengelig på google docs](#)
- Linket fra ITL

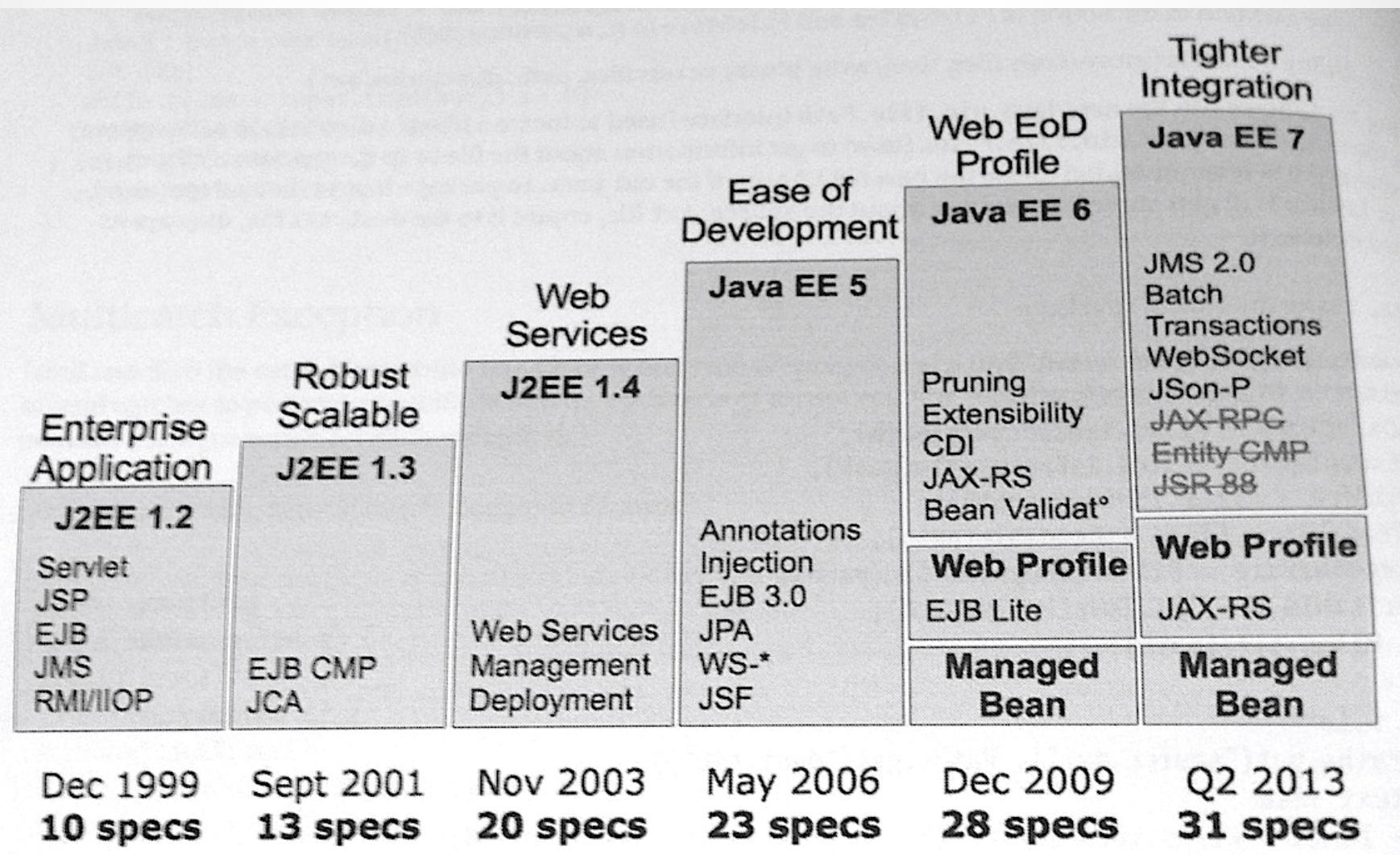
# Java quiz ☺

- <http://kahoot.it>

# Java Enterprise Edition

- Hva er forskjellen mellom JEE og JSE?
- Hvilke fordeler får vi ved bruk av JEE?

# Java Enterprise Edition



# Java Enterprise Edition

- Laget på toppen av Java SE
  - Alle Java SE APIer kan brukes fra Java EE
  - Fokuserer på å forenkle (POJO basert programering)
- Tilbyr ekstra funksjonalitet, som f.eks.
  - Transaksjoner
  - Sikkerhet
  - Interoperabilitet
  - Distribuert miljø

# Java Enterprise Edition

- Når det er behov for aksessering av data, anvende forretningslogikk, bruke presentasjonslag og kommunisere med eksterne systemer
- Standard-baserte metoder for å imøtekomme disse behovene og løse problemene på en generell måte

# Java Enterprise Edition

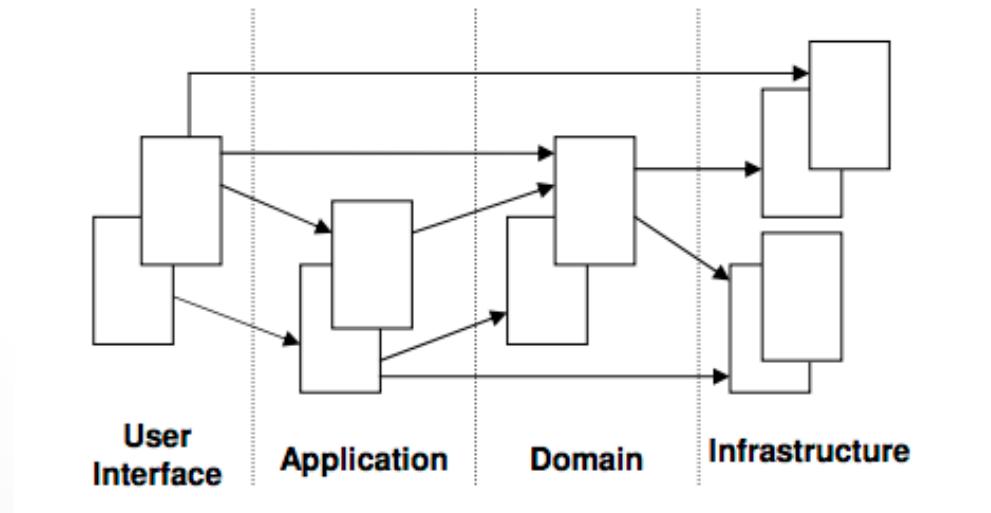
- Mange krav som må oppfylles
  - Stabilitet
  - Tilgjengelighet
  - Skalerbarhet
  - Sikkerhet
  - Integrasjon med eksterne systemer
- Utvikling
  - Smidig utvikling
  - Krav til systemet endres ofte
  - Vedlikehold / videreutvikling
  - Bruk av åpne standarder

# Enterprise utvikling

- Fokus på lettvekts metoder og rammeverk
  - Spring vs EJB 2.1
  - Stor interesse for open source
  - Rask tilbakemelding i utviklingen (med hjelp av verktøy og metoder)
- Standard API
  - JPA
- Automatiserte tester (TDD)
- Continuous integration
  - Detaljert gjennomgang senere forelesning
  - TDD (enhetstester)
  - Integrasjonstester

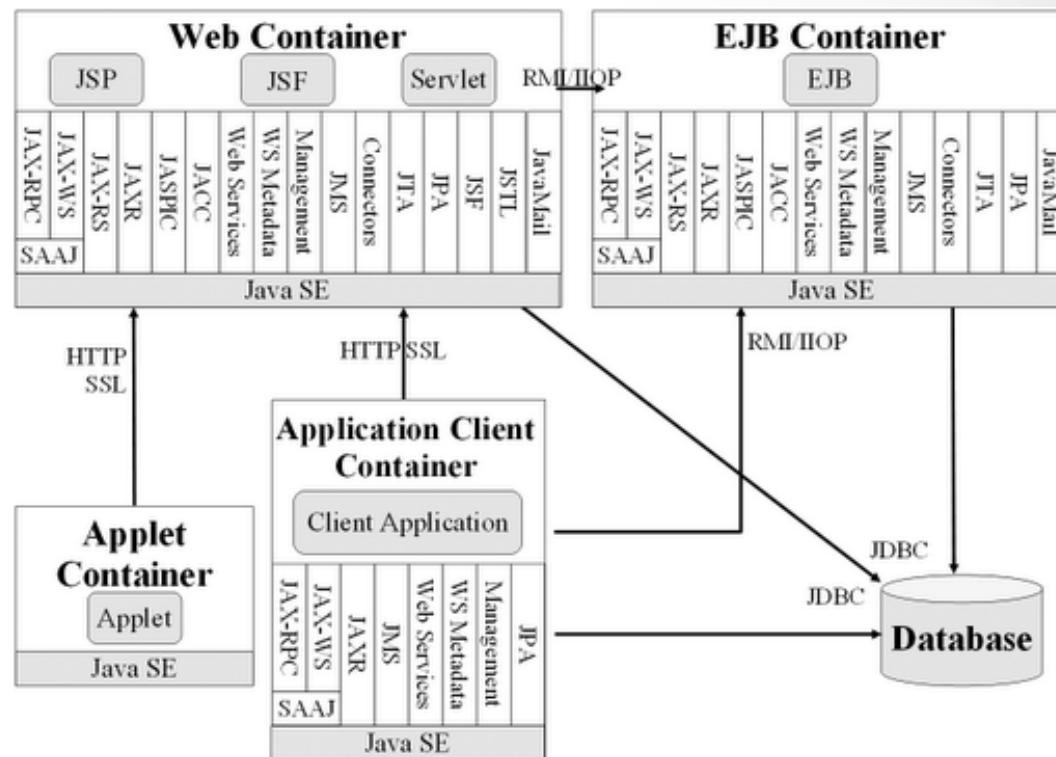
# Lagdeling

- Del opp et komplekst program i flere lag
- Hvert lag skal bestå av sammenhengende elementer med som bare er avhengig av lagene lenger ned



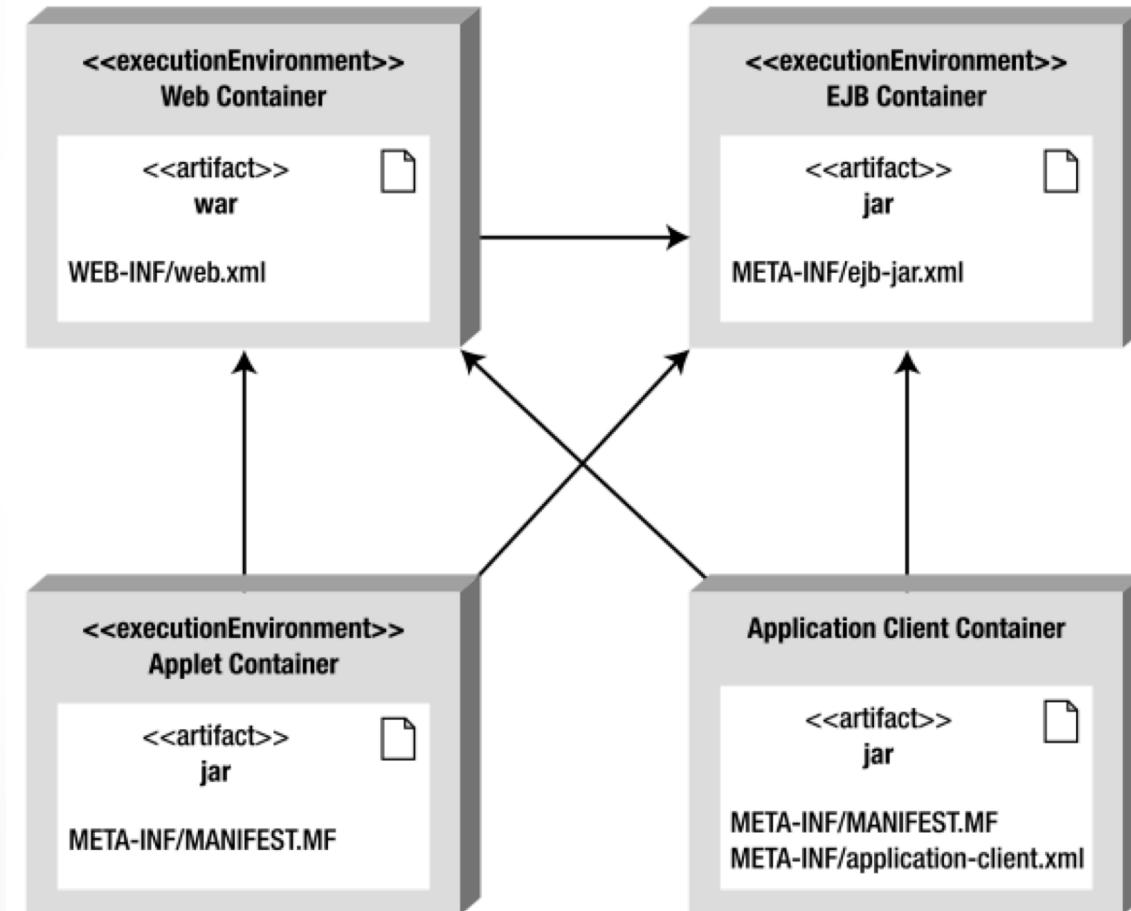
# Tjenester

- Java Persistence API (JPA)
- Validation (Bean validation)
- Dependency Injection
- Management
- Deployment
- Java Naming and Directory Interface (JNDI)
- **Java Transaction API (JTA)**
- **Java Message Service (JMS)**
- **Web services**
- **Security services (JAAS)**
- Java EE Connector Architecture (JCA)
- JavaMail
- JavaBeans Activation Framework (JAF)
- **XML Processing**
- **JSON processing**



# Pakking

Bruk av maven plugins (maven-war-plugin, wildfly-maven-plugin)

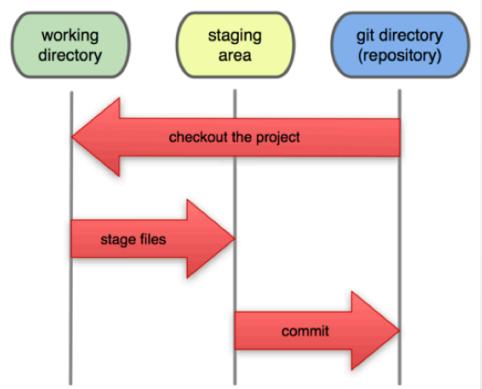
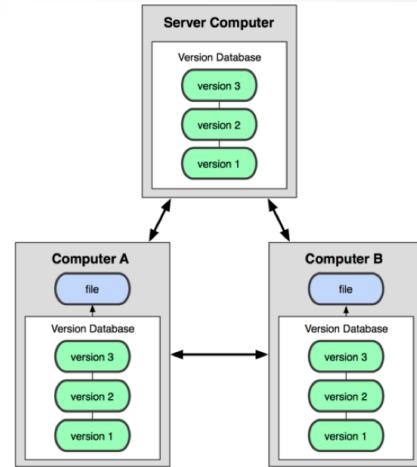


# Verktøy



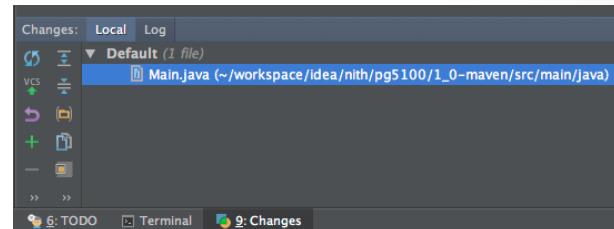


- Distributed Version Control System
  - Sjekker ut en kopi av hele *repository*
  - De fleste operasjoner skjer lokalt (f.eks. commit)
- 3 tilstander
  - **Modified** – lokale endringer som ikke er committed
  - **Staged** – endringene er satt klare for commit
  - **Committed** – endringene er lagret lokalt





- Vanlige kommandoer
  - git init
  - git clone <repo-url>
  - git status
  - git add
  - git commit -m "<commit melding>"
  - git checkout -b <branch-navn>
  - git stash (git stash pop)
- Bruk verktøy!
  - git er godt støttet i alle populære IDEer
- Hvorfor bruker vi versjonskontroll?
- Erfaringer med git?
  - Fordeler/ulemper
- Alternativer?
- Git demo
- For mer info se **Pro Git & Git Cheat Sheet**





- Byggssystem
- Convention over configuration
- Verktøy integrasjon
  - Godt støttet i IntelliJ, Eclipse og Netbeans



- <http://wildfly.org>
- Tidligere JBoss
- Utviklet av Red Hat
- Skrevet i Java
- Open source
- Implementerer Java EE 7 spesifikasjonen
- Getting started guide:
  - <https://docs.jboss.org/author/display/WFLY8/Getting+Started+Guide>



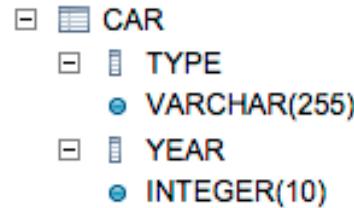
- Java SQL database
  - Open source
  - JDBC API
  - In-memory / filbasert
  - Rask og enkel
  - Og ikke minst, er integrert i Wildfly
- Dokumentasjon: <http://www.h2database.com/html/tutorial.html>

# Forslag?

- Andre verktøy vi burde se nærmere på i faget?

# JPA repetisjon

- Objekter / databaserader
  - Rader i CAR tabellen i motsetning til Car objektet



```
public class Car {  
    private String type;  
    private int year;  
  
    public boolean isActive() {  
        return year > 1970;  
    }  
}
```

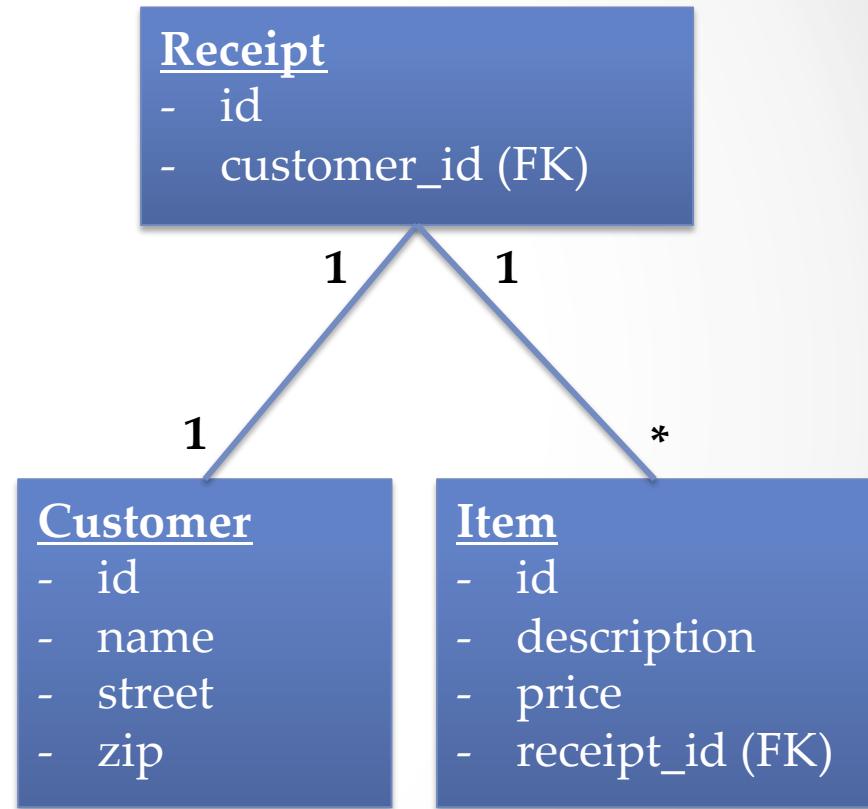
# Entitet

- Begreper i JPA:
  - Objekter er instanser som er tilgjengelig *in-memory*
  - Entiteter er objekter *in-memory* og persisteres i en database
- Vanlig objekt, kan inneholde metoder i tillegg til data
- Annotert som `@Entity`
- Må ha en public/protected default konstruktør
- Klassen kan ikke være final
  - Hvorfor ikke?

# JPA repetisjon

## Egentrenings

- Lag entitetene i pakken `no.nith.pg6100` i `src/main/java`
- `Customer` deles opp i
  - `Customer` objekt (`id, name`)
  - `Address` objekt (`street, zip`)
- En til en - `receipt` → `customer`
- En til mange – `receipt` → `item`
- Named query som henter ut alle entiteter
- Skriv integrasjonstester
  - Test verdier satt inn ved oppstart (`init.sql`)
  - Test opprettelse av nye instanser av entitetene
  - Test sletting av en entitet



[https://github.com/jarlehansen/pg6100/tree/master/1\\_0-jpa-repetisjon](https://github.com/jarlehansen/pg6100/tree/master/1_0-jpa-repetisjon)

# Transaksjoner

- Hva må vi passe på ved bruk av JPA utenfor container?
- Hvilke fordeler gir containeren oss?