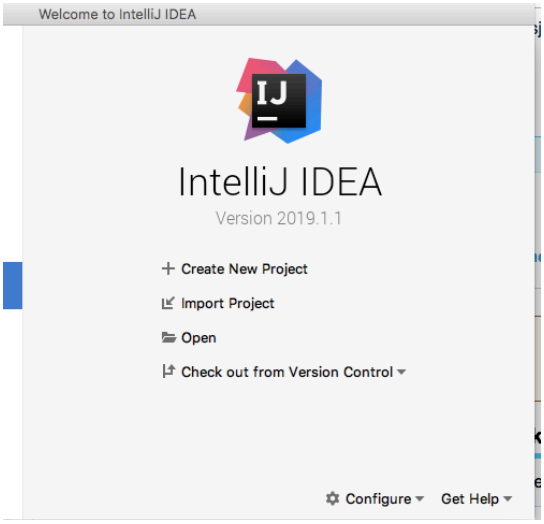
Vårt første program - 1

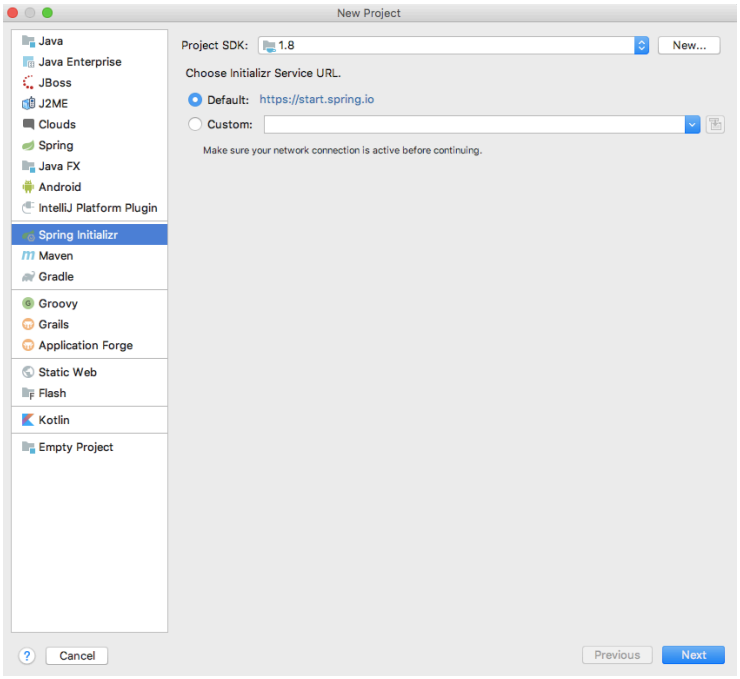
Her er hvordan vi oppretter vårt første prosjekt med IDE'en IntelliJ (Ultimate) :

Start IntelliJ (Ultimate) programmet.

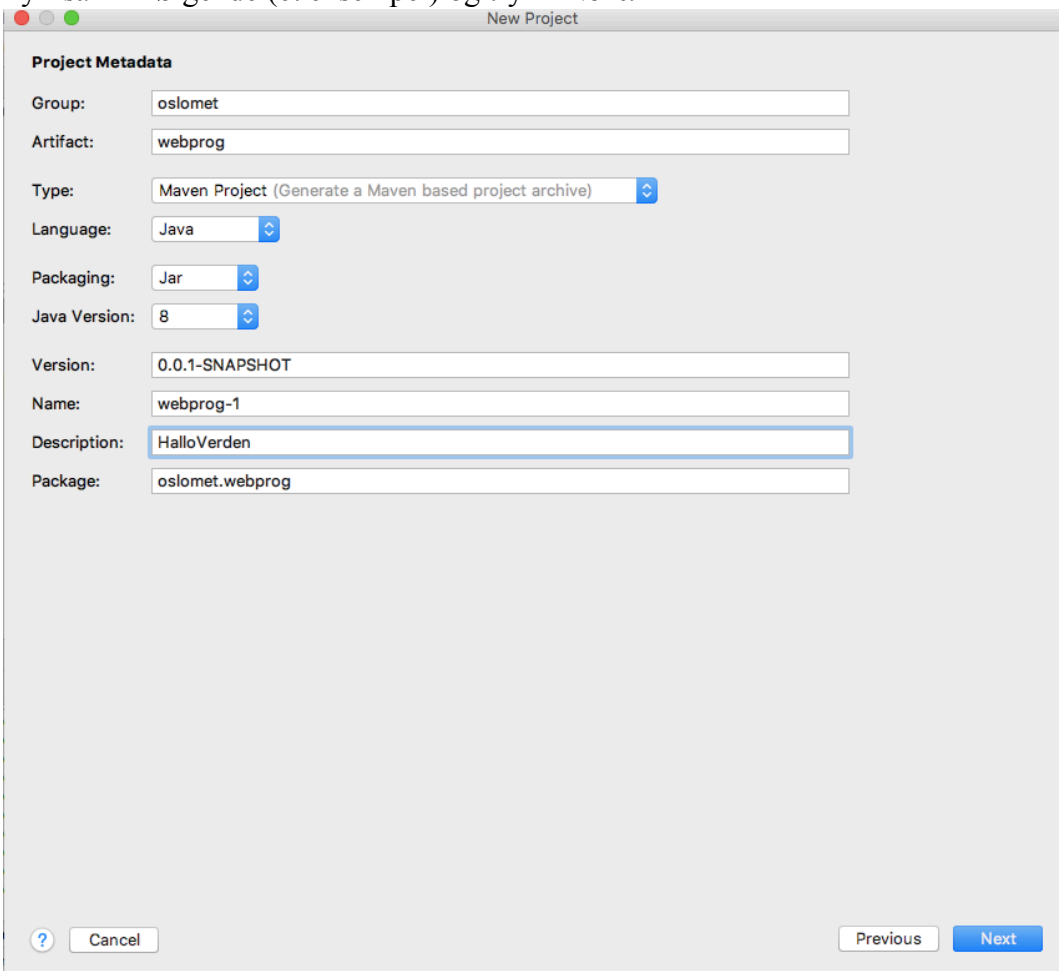
Velg så + **Create New Project** fra skjermbildet under:



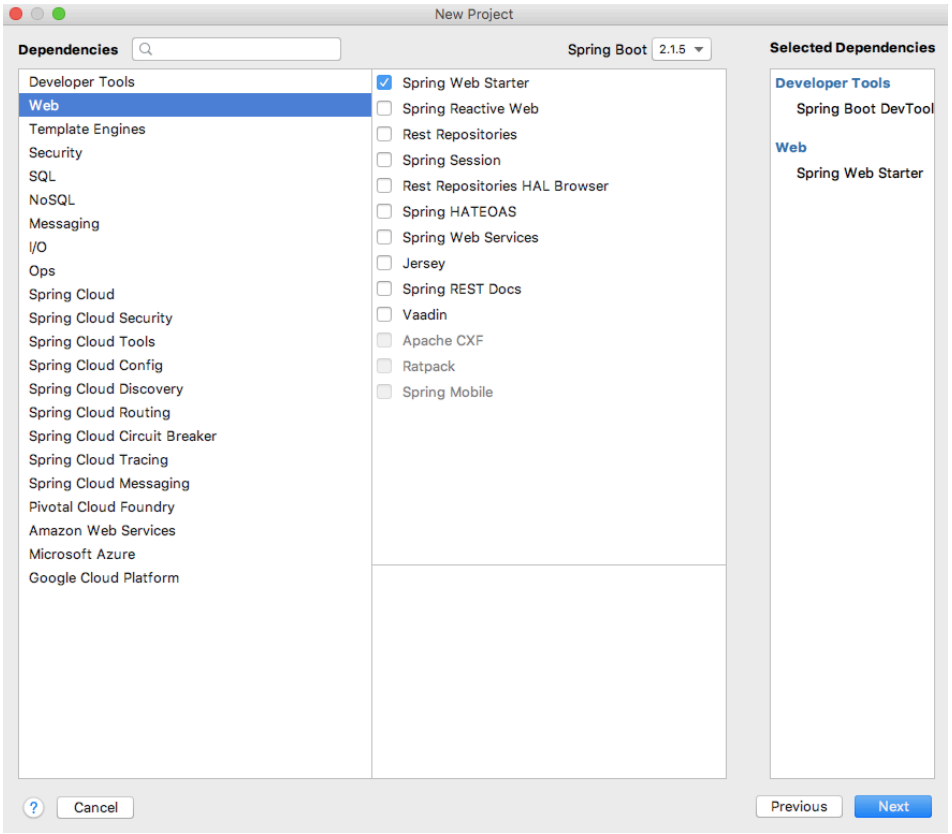
Så velges **Spring Initializer** og **Next**:



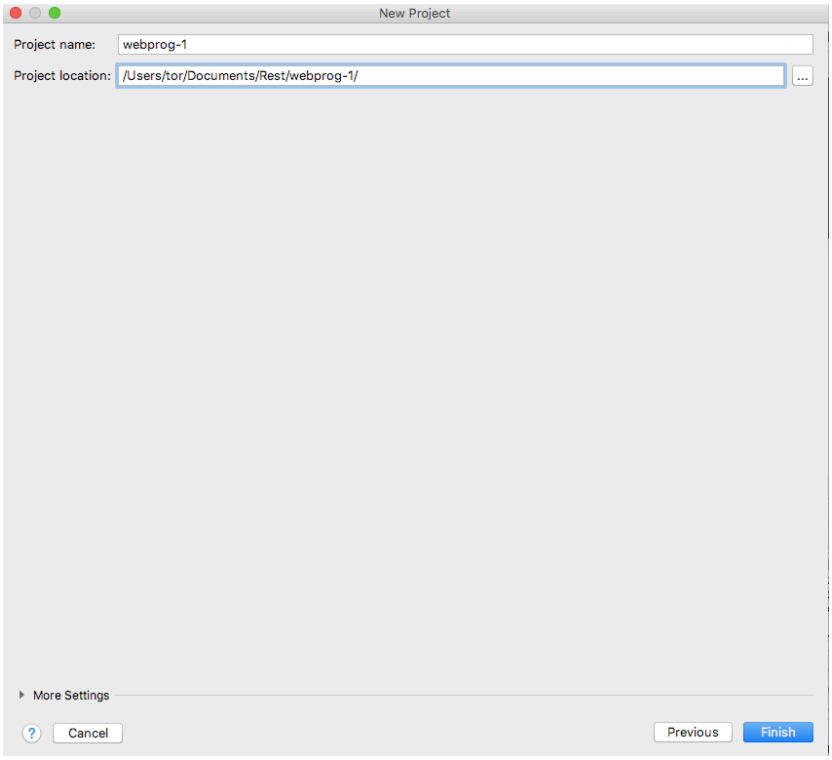
Fyll så inn følgende (et eksempel) og trykk **Next**:



Velg **Developer Tools ->Spring Boot DevTools** og **Web->Spring Web Starter** slik at du får dette, og trykk **Next**:



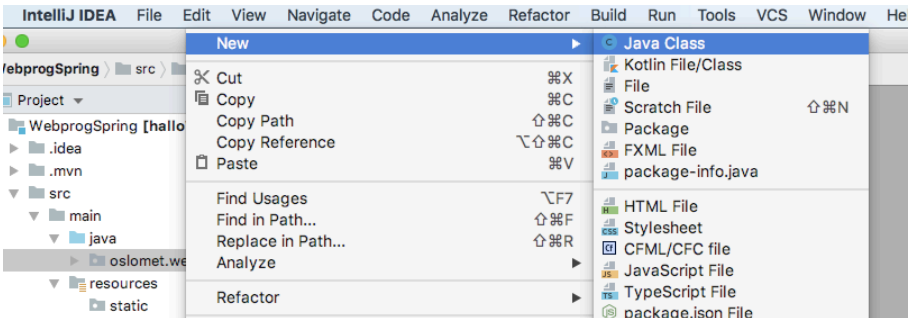
Skriv så inn prosjektnavnet (det prosjektet vil hete på disk) og legg det på en katalog (valgfritt), trykk **Finish**:



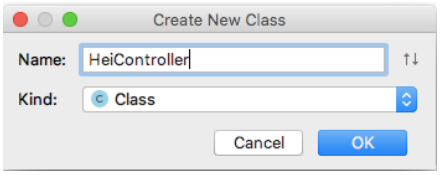
Etter en stund kommer løsningen opp (se nederst til høyre for fremdrift av "utpakkingen").

Åpne katalogen **src->main->java->oslomet.webprog**
Alle java-filer skal plasseres under denne java-katalogen. Nye java-pakker kan opprettes.

Opprett så en Java-klasse under pakken **oslomet.webprog**:
Høyre-kilkk på mappen (oslomet.webprog) og velg **New->Java Class**



Kall den **HeiController:**



Legg så på en **@RestController** dekorasjon på det opprettede klassen slik:

```
package oslomet.webprog;

import org.springframework.stereotype.Controller;

@RestController
public class HeiController {


}
```

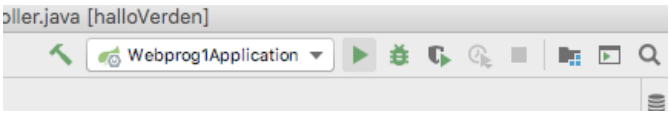
Lag en metode kalt **hei** med to dekoratører; hele filen blir nå slik:

```
package oslomet.webprog;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HeiController {
    @GetMapping("/")
    public String hei(String navn){
        return "Hei verden "+navn;
    }
}
```

Kjør løsningen ved å trykke på grønn pil øverst til høyre eller trykk **<ctrl>+r:**



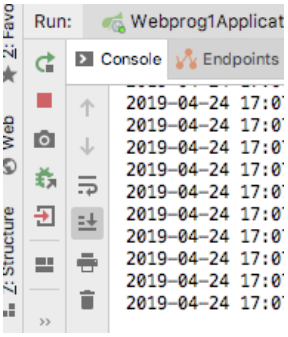
Gå til en nettleser og skriv inn:

http://localhost:8080/?navn=Per

Resultatet blir :

Hei verden Per

Stopp løsningen med å trykke rød knapp nede til venstre:



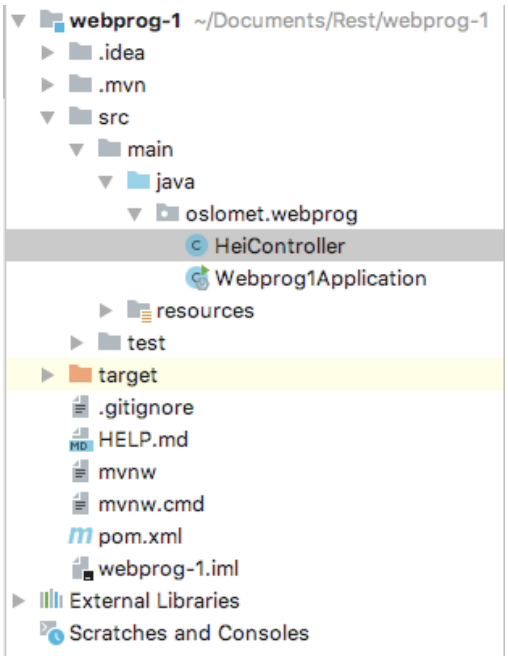
Virkemåten til vårt første program

1. Når vi skriver inn url'en **localhost:8080** i nettleseren leter Spring etter en metode dekorert (merket) med **@GetMapping("/")** i en controller dekorert med **@RestController.**
2. Det som står etter / i url'en er en såkalt get-streng **(?navn=Per**). Denne strengen kalles også spørrestreng eller query string. Det betyr at vi kan sende inn data til den kallende metoden via ? og et variabelnavn og en verdi. Det vil si at **navn** er variabelnavnet og **Per** er verdien til variabelen. Denne verdien kan altså hentes fra parameterlisten til metoden **heiVerden(String navn)**. For at dette skal virke må variabelnavnet i get-strengen være det samme som i parameterlisten til metoden, i dette tilfelle **navn**.
3. Vi har også en dekoratør til på metoden; **@ResponsBody**. Denne er ikke veldig mye brukt, men brukt her for å forenkle det første programmet. Denne dekoratøren bestemmer at det som returneres fra metoden skal overføres til html-meldingen som skal sendes tilbake inne i **<body>** taggen.
4. Dersom det ikke oppgis noe etter **localhost:8080/** får vi naturlig nok:
   Hei verden null; da verdien av variabelen ikke har noe verdi (er null).

Katalogstruktur

Etter at et Spring-prosjekt er opprettet er det veldig mange kataloger og underkataloger som blir opprettet. Dette kan virke noe overveldende i begynnelsen, men det er mange kataloger vi kan se bort i fra i dette kurset. Ikke prøv å slett noen kataloger selv om de er tomme, de trengs for å at rammeverket skal fungere.

Her er katalogstrukturen til **halloVerden** applikasjonen (de katalogene som betyr noe er ekspandert)



Under **java-mappen** skal all egenutviklet Java-kode ligge. Her kan man også opprette nye Java-pakker etter ønske. Det blir opprettet en java fil i den opprettede pakken kalt **Webprog1Application.java** her. Denne klassen inneholder **main**-metoden som starter applikasjonen. I **oslomet.webprog** ligger også den controlleren som er utviklet av oss.

Under **resources** er den en fil som heter **application.properties** og som brukes til å definere enkelte ting. Bla. navnet til den databasen som vi etterhvert skal bruke i vår applikasjon. I utgangspunktet er denne filen tom.

Videre under samme mappe er den en mappe som kalles **static** og som skal inneholde statiske html-filer. Dynamiske html-filer skal lagres under **templates** dette er såkalte views i Spring. Disse to katalogene er tomme da vi ikke har noe html-filer i løsningen. Mer om dette senere.

Til slutt har vi en viktig fil lenge nede: **pom.xml.** Dette er en fil som brukes for å definere hvilke komponenter vil ha med i løsningen vår. F.eks om løsningen skal ha view's og om løsningen skal ha en database. POM står for Project Object Model og er en XML-fil. Mer om denne filen under.

Introduksjon til Maven

Maven er et tillegg til Apache-webserver som håndterer konfigurasjonen til Spring via pom-filen, kjører kompileringer og starter løsningen i Apache. Dette er altså et verktøy som håndterer byggingen og kjøringen av Java-applikasjonen.

Project Object Model (pom.xml)

Denne filen definerer hvilke komponenter som skal inngå i akkurat dette prosjektet. Det titalls ulike komponenter som kan velges i et Spring-prosjekt. Disse kan velges når prosjektet opprettes eller legges til i etterkant ved å redigere filen (legge til en eller flere komponenter).

Filen fra vårt første program ser slik ut:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>OsloMet</groupId>
    <artifactId>webprog</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>webprog1</name>
    <description>halloVerden</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Her er det først en del informasjon om navn og versjoner etc. Under dependencies listes de komponentene vi vil ha med i løsningen. Dette er:

- spring-boot-starter-web
- spring-boot-devtools

Den første må være med i alle prosjekter da dette er "minimums-spring". Den andre er tatt med for å kunne gjøre utviklingen enklere ved ikke å behøve å restarte løsningen ved endringer. Så er det en mulighet for testing av løsningen, noe vi ikke skal se på i dette kurset. Til slutt er det spesifisert at en maven-plugin brukes for å holde rede på dette.

Forskjeller mellom Java og Javascript

Java og Javascript har absolutt ingen ting med hverandre å gjøre og er så forskjellige som to programmeringsspråk om mulig kan være. Før vi starter med kodingen av Javascript beskrives noen av de største forskjellene:

Java-koden må kompileres, mens Javascript-koden interpreteres. Med det menes at Javascript-koden tolkes der og da når den kjøres (i nettleseren). Videre betyr det at Java-koden er strengt typet (alle variabler har kun en type). Variabler i Javascript defineres ved implisitt tildeling (ingen spesifikk deklarering) og kan endre type underveis (ganske cool't, men også ganske uoversiktlig!).

I tillegg har Javascript kun 3 typer: String, Number og Object:

```
var lengde = 16,25;                                 // Number
var fornavn = "Per";                                // String
var navn = {fornavn:"Per", etternavn:"Hansen"};     // Object
```

Funksjoner i Javacript har noen likheter med metoder i Java da de tar inn parametere, og kan returnere en verdi.

Som regel inneholder Javascript-objekter bare attributter. Det vil si at et Javascript-objektet

```
var Kunde = { navn : "", adresse :"" }
```

tilsvarer Java objektet (klassen) med dens attributt-navn:

```
public class Kunde {
    private String navn;
    private String adresse;
```

Objekter i Javascript vil bli mye brukt videre.