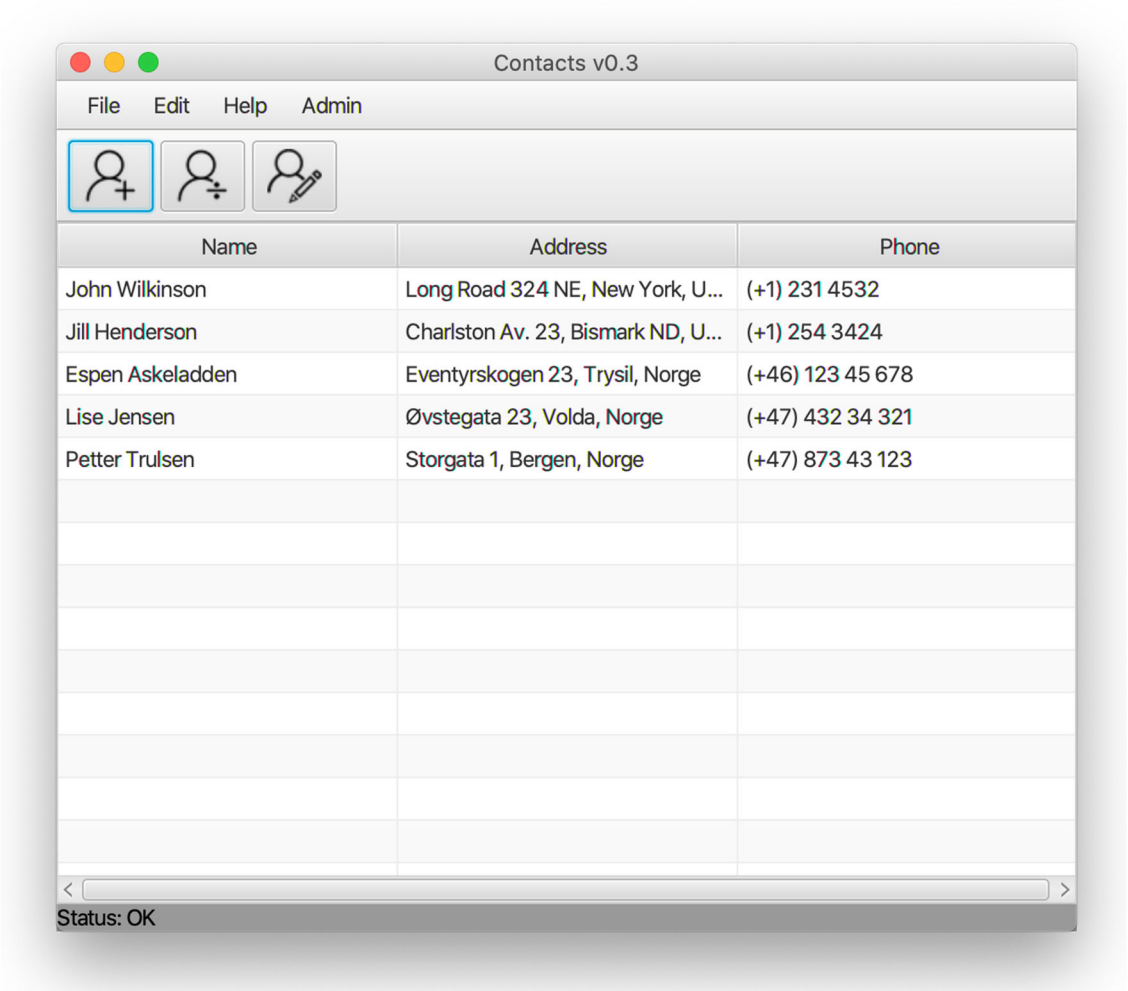
Java og Databaser (ORM/JPA)

I denne oppgaven skal vi jobbe med hvordan vi kobler på en database på en Java-applikasjon. Vi skal bruke rammeverket JPA, som er et ORM-rammeverk for Java.

Denne oppgaven bør sees i sammenheng med prosjektoppgaven i Systemutviklingsemnet.

For at du skal få jobbe mest mulig med selve database-delen, har vi laget en applikasjon som du skal utvide med databasestøtte. Prosjektet er utviklet i IntelliJ og gjøres tilgjengelig som en ZIP-fil.

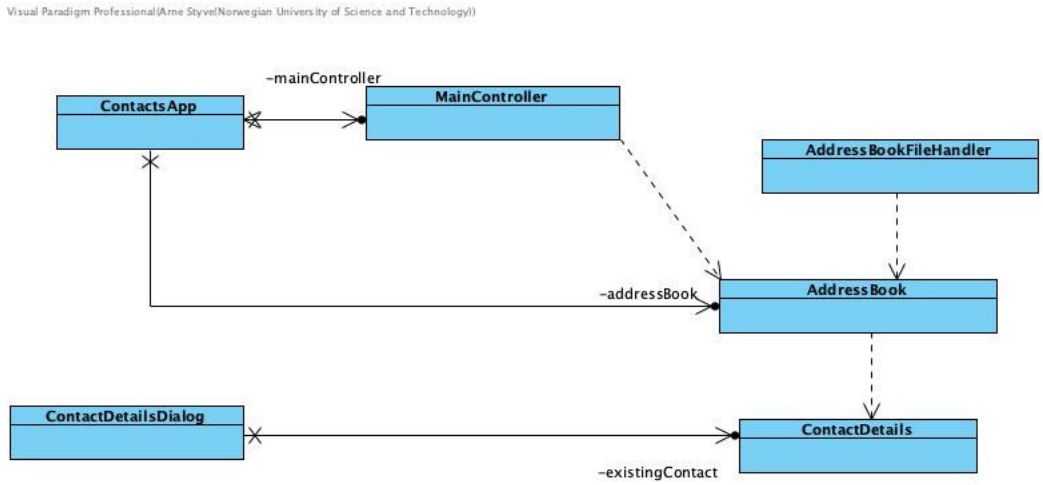


Figur 1 Contacts Applikasjon - en Adressebok

Denne applikasjonen er i utgangspunktet laget med TreeMap<> som arkiv for kontaktene. Det er lagt inn støtte for import og eksport fra/til CSV-fil (tekstbasert), i tillegg til objekt-serialisering ved oppstart og avslutning (slik at kontakter bli værende i kontaktlisten mellom hver gang applikasjonen kjøres). Se gjerne på hvordan filhåndtering er implementert, for å repetere filhåndtering.

Applikasjonen er laget med JDK 8 (forenkler en del i forhold til JavaFX og også i forhold til DB).

Klassediagrammet under viser hvordan klassene er organisert.



Figur 2 Klassediagram

Noen ord om de sentrale klassene:

Klasse	Ansvar/oppgave
ContactDetails	Entitetsklassen som holder informasjon om en kontakt. En kontakt registreres med navn, adresse og telefonnummer.
AddressBook	Adresseboken som holder på alle kontaktene.
AddressBookFileHandler	Ansvarlig for alt av filhåndtering i forhold til adresseboken: import/eksport fra/til CSV-fil samt objektserialisering av hele adresseboken.
ContactsApp	Selve applikasjonen, hovedvinduet laget i JavaFX
MainController	Ansvarlig for å **utføre** alle operasjoner initiert av brukergrensesnittet. M.a.o. all kode som skal utføres som følge av et klikk med mus, et rykk på en knapp osv. i GUI, ligger i denne klassen (iht MVC-prinsippet).
ContactDetailsDialog	En JavaFX dialog som benyttes til å vise kontaktdetaljer, redigere eksisterende kontakt og opprette ny kontakt.

Oppgave 1 - Refaktorering for tilpassing til DB

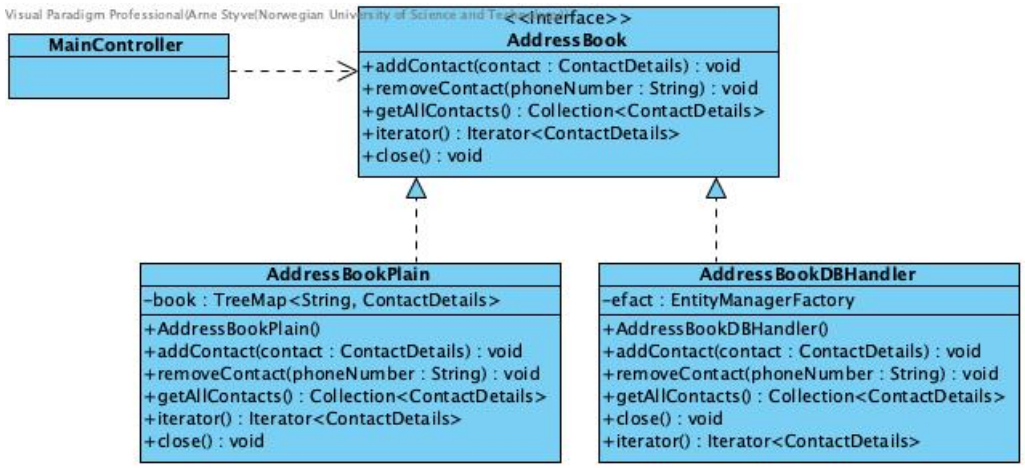
Når vi skal implementere støtte for DB i applikasjonen, vil det være fornuftig å beholde eksisterende **AddressBook** klasse parallelt med at vi legger til ny kode for DB støtte, slik at du alltid har en mulighet til å gå tilbake til en løsning uten DB dersom du skulle støte på problemer.

Det er også god design å lage løsningen slik at GUI-delen (view og controller) trenger minimalt med endringer for å gå fra en løsning basert på TreeMap<> og fil, til DB.

Du bør derfor beholde **grensesnittet** mellom GUI-klassene (ContactsApp, MainController og ContactDetailsDialog) og «modell»-klassene (forretningslogikken).

Endre koden ved å lage et **grensesnitt/interface** basert på klassen AddressBook. Den gamle AddressBook klassen kan endres til AddressBookPlain f.eks., slik at det nye grensesnittet får navnet AddressBook. TIPS: Her kan du bruke *Refactor*-verktøyet i IntelliJ (Refactor->Extract->Interface, og velg «Rename..:»).

Lag deretter en **ny klasse**, og kall denne for **AddressBookDBHandler** som implementerer grensesnittet AddressBook, slik at du ender opp med følgende:



Figur 3 AddressBook-klasser etter refaktorering

Oppgave 2 – Implementer JPA ved hjelp av EclipseLink

Last ned EclipseLink (https://www.eclipse.org/eclipselink/). Opprett en ny mappe i prosjektet ditt kalt **"lib"** og plasser følgende JAR-filer fra EclipseLink i mappen:

- eclipselink.jar
- jakarta.persistence_2.2.3.jar.

Oppdater **entitetsklassen** ContactDetails med nødvendige **annotasjoner** og eventuelle tilleggsmetoder (getter/setter samt default konstruktør) for å oppfylle kravet til en **JavaBean**.

Implementer klassen **AddressBookDBHandler** ved bruk av JPA (EntityManager, EntityManagerFactory, Query osv.). Du bør muligens utvide AddressBook interfacet med en *close()*-metode for å lukke EM og EMF etter bruk.

Oppdater til slutt ContactsApp-klassen slik at den oppretter en instans av AddressBookDBHandler og ikke AddressBookPlain ved oppstart. Utover dette er det behov for minimalt med endringer i denne og MainCotroller klassen.

Oppgave 3 – Sett opp for embedded Apache Derby DB

Last ned Apache Derby RDBMS (http://db.apache.org/derby/). Kopier JAR-filen **derby.jar** til **lib-folderen** i prosjektet.

Opprett filen **persistence.xml** med følgende innstillinger:

```
<persistence-unit name="contacts-pu" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>

    <properties>
        <property name="javax.persistence.jdbc.driver"
                  value="org.apache.derby.jdbc.EmbeddedDriver"/>
        <property name="javax.persistence.jdbc.url"
                  value="jdbc:derby:contactsdb;create=true"/>
        <property name="javax.persistence.jdbc.user" value="app"/>
        <property name="javax.persistence.jdbc.password" value="app"/>

        <property name="eclipselink.target-database" value="Derby"/>

        <property name="eclipselink.ddl-generation" value="create-tables"/>

        <property name="eclipselink.logging.level" value="FINE"/>

    </properties>
</persistence-unit>
```

Du kan velge et annet PU-navn enn «contacts-pu» om du ønsker det.

Lagre filen i mappen **META-INF** i src-katalogen.

Bygg, kjør og test applikasjonen.

Endre **logging level** til **ALL**, **OFF** osv. og legg merke til hva som kommer ut av loggen ved kjøring.

Oppgave 3 – lokal DB server

Det er mulig å ha flere persistens-enheter (persistency unit) i samme persistence.xml fil. Legg inn en ny persistency enhet i eksisterende persistence.xml-fil for å koble applikasjonen opp mot en lokal Apache Derby Server.

Du må da gi denne nye persistens enheten et nytt unikt navn, f.eks. «contacts-pu-local-db».

Legg merke til at du ikke trenger endre noe av Java-koden.

Oppgave 4 – MySQL-server hos IDI

Til slutt utvider du persistence.xml med en tredje persistens enhet (PU), som kobler applikasjonen din opp mot databaseserveren hos IDI. Dette er en MySQL-database.

Innlevering

Pakk **hele prosjekt-mappen** i en ZIP-fil og lever i henhold til instrukser fra faglærer (Ålesund leverer som vanlig i Blackboard).