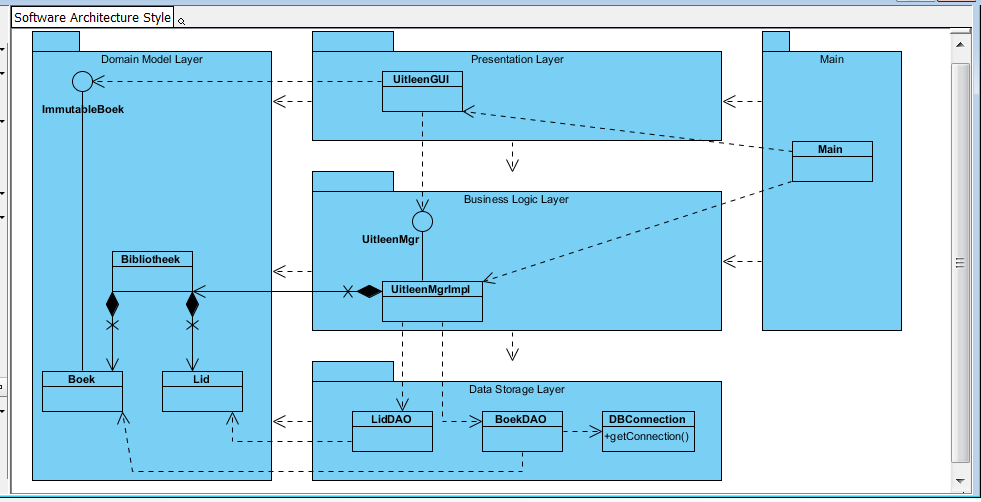
SW architectuurstijl 23IVP3/23IVP4 plus rationale  
Laatste aanpassing: 15 sept 2014

TODO:

* Extra entiteit exemplaar en uitlening nodig om te laten zien hoe je de state consistent houdt over verschillende entiteiten heen
* We hebben gekozen om de domeinobjecten ook door presentatielaag te laten gebruiken en zodoende dus geen gebruik te maken va DTO’s. Het lijkt erop dat dit niet lekker samengaat met het idee van de immutable interfaces, omdat het leidt tot casten tussen immutable domeinobjecten en mutable domeinobjecten.
* De GUI heeft in de code een member van het type ImmutableMember. In het klassediagram staat echter een “uses” relatie. Dit is in principe wel correct, maar zou als verwarrend kunnen worden ervaren door studenten.

# Voorgestelde architectuurstijl



Opmerkingen over de data storage layer:

* Verantwoordelijkheid van een DAO: leveren van CRUD operaties voor zijn geassocieerde entiteit. Er zijn twee opties die verschillen in de hoeveelheid delegatie die wordt toegepast door een manager klasse:
  + De MemberAdminManager gebruikt de MemberDAO, LoanDAO en ReservationDao om een Member domeinobject uit persistente storage te halen (weinig delegatie).
  + De MemberDAO gebruikt de LoanDAO en ReservationDao om een Member domeinobject uit persistente storage te halen. De MemberAdminManager gebruikt dus alleen de MemberDAO voor deze taak (veel delegatie).

We hebben gekozen voor de optie met weinig delegatie. In hindsight was het misschien beter geweest om juist de optie met veel delegatie te kiezen.

* We leggen er wel nadruk op dat DAO’s stateless classes zijn.
* We leggen er geen nadruk op dat DAO’s alleen static methodes mag hebben (al mag het wel). Reden: static wordt door studenten als moeilijk ervaren.
* Verantwoordelijkheid van de DBConnection klasse: maken van een verbinding met de database en *ondersteunende* methodes voor CRUD operaties op de database
* We leggen er geen nadruk op dat DBConnection een singleton is (al mag het wel)
* DBConnection. getConnection() is private en wordt aangeroepen door de *ondersteunende* methodes binnen DBConnection
* Database password met properties file wegwerken uit de broncode

Opmerkingen over de domain model layer:

* De voorgestelde architectuurstijl leidt ertoe dat entiteiten POJO’s zijn
* We gebruiken een rich domain model: entiteiten mogen business logica bevatten. Reden: strookt met object-georiënteerde aanpak
* We gebruiken geen data transfer objects (DTO) tussen business logic layer en presentation layer. In plaats daarvan gebruikt de presentation layer entities via een immutable interface.
* Aangezien entities door data storage layer, business logic layer en presentation layer worden gebruikt, tekenen we de domain model layer verticaal.
* Hoe kom ik aan mijn POJO’s? Er zijn twee opties die de moeite waard zijn om te noemen:
  + (eager approach) De Mgr creëert een Bibliotheek object (er is geen bibliotheek tabel in de database). Vervolgens worden er m.b.v. de DAO’s entiteiten uit de database gehaald en deze entiteiten worden in hiërarchische structuur gehangen onder het Bibliotheek object. Na het opbouwen van deze hiërarchische structuur worden use cases uitgevoerd door object navigatie.
  + (lazy approach) Een use case maakt gebruik van DAO’s om aan de benodigde entities te komen.
  + Omdat de database “onder water” door andere partijen veranderd kan worden, kiezen wij de lazy approach voor ons voorbeeld

Opmerkingen over de business logic layer:

* Loose coupling (onder meer voor testability) wordt verkregen door presentation layer en business logic layer te ontkoppelen door het toevoegen van Manager interfaces.
* Optioneel kunnen de Manager interfaces in een aparte laag, de “service layer” worden gezet.

Opmerkingen over de presentation layer:

* Action listeners worden geïmplementeerd in de presentation layer. De action listeners roepen methode uit de business logic layer aan. Reden: business logic layer blijft Swing onafhankelijk.

Opmerkingen over het Main package:

* Om meerdere GUI’s (e.g. lokale GUI en web GUI) toe te staan, is het niet handig als de GUI de Manager creëert. Daarom trekken we creatie verantwoordelijkheid uit GUI of Manager classes naar een apart Main package.
* Het Main package bevindt zich buiten de presentation layer en business logic layer.

Architecture principles:

* We sturen naar één consistente wijze van exception handling binnen elk subsysteem van de proftaak.

# Didactische aspecten

* De voorgestelde architectuurstijl lijkt complex. Echter omdat de architectuurstijl volledig gegeven en uitgewerkt is en er dus geen optie voor vaagheid is, is het waarschijnlijk makkelijker voor de studenten.
* Voor wat betreft architectuurstijl is het leerdoel als volgt: “de student is in staat software te realiseren volgens een gegeven architectuurstijl”. Pas in de hogere leerjaren gaat de student aan de slag met het zelf kiezen van een architectuurstijl.
* De 23IVP3 docenten leveren een volledig uitgewerkt, maar klein, voorbeeld aan. Studenten moeten het PoC realiseren volgens de architectuurstijl en mogen daarbij het uitgewerkte voorbeeld bij gebruiken. Dit zetten ze door in de HH realisatie tijdens 23IVP4.
* Interfaces en exception handling kunnen nog niet in 23IVP3; DAO en initialisatie m.b.v. main doen we wél in 23IVP3
* Om tijd te besparen kan voor het voorbeeld eventueel uitgegaan worden van de VP4I\_1213D klas C software.