Laboration - Java I/O

Beskrivning

Denna laboration går ut på att ni, i grupper om fyra, ska bygga ett datalager för att hantera en enkel webbshop. Ni kommer under senare kurser att bygga vidare på det ni bygger nu. Datalagrets uppgift är att hantera kommunikationen mot den lagringsmodell som används, i detta fall kommer lagringsmodellen att vara serialisering. Nedan följer en design som ni förväntas följa.

Förklaring

ECommerceService - har hand om att erbjuda all funktionalitet i webbshoppen. Denna klass arbetar i sin tur mot olika s.k. *repository* för att komma åt lagringen av sin data. Ansvaret denna klass har är att ha hand om affärslogik, rättighetskontroll, mm samt dölja vilken lagringsmodell som används.

XxxRepository - olika interface som exponerar metoder för s.k. CRUD-operationer (Create, Read, Update, Delete) för de domänobjekt (User, Product, Order, etc) som finns i applikationen.

FileXxxRepository - olika klasser som implementerar de olika XxxRepository-interfacen.

Model-klasserna - dessa klasser representerar olika begrepp i webbshoppen som ex. *User, Product, Order, OrderItem, etc*. Du kan se dessa klasser som bärare av data

Vad är skillnaden mellan ECommerceService och de olika Repository-implementationerna?

ECommerceService arbetar inte direkt mot den lagringsteknologi som används utan arbetar istället mot sina repositories för att utföra CRUD-operationer. Repository-implementationerna arbetar däremot direkt mot den lagringsteknologi som används (i detta fall serialisering till disk).

Varför är de olika repositories som finns exponerade som interface istället för klasser?

Detta är för att man ska kunna byta den bakomliggande lagringsteknologin utan att påverka andra klasser, i detta fall ECommerceService. I denna laboration kommer lagringsteknologin vara serialisering men detta kan ganska enkelt bytas mot exempelvis databaslagring eller minneslagring genom att skicka in en annan implementation av önskad repository-interface till ECommerceService.

ECommerceService känns ganska onödig, varför inte arbeta direkt mot repository-implementationerna?

ECommerceServicen ansvarar för validering av data innan den skickar vidare data som ska sparas till respektive repository. Detta kan innefatta att vissa regler uppfylls innan exempelvis en order sparas. Repository-implementationerna har inte det ansvaret och saknar därför sådan validering.

Tillvägagångssätt

1) Börjar med att ta fram de modell-objekt som ska finnas i applikationen. Detta innefattar exempelvis *User, Order, OrderRow, Product, etc*. Lägg till de datamedlemmar som varje klass behöver.

2) Skapa repository-interfacen och ECommerceService

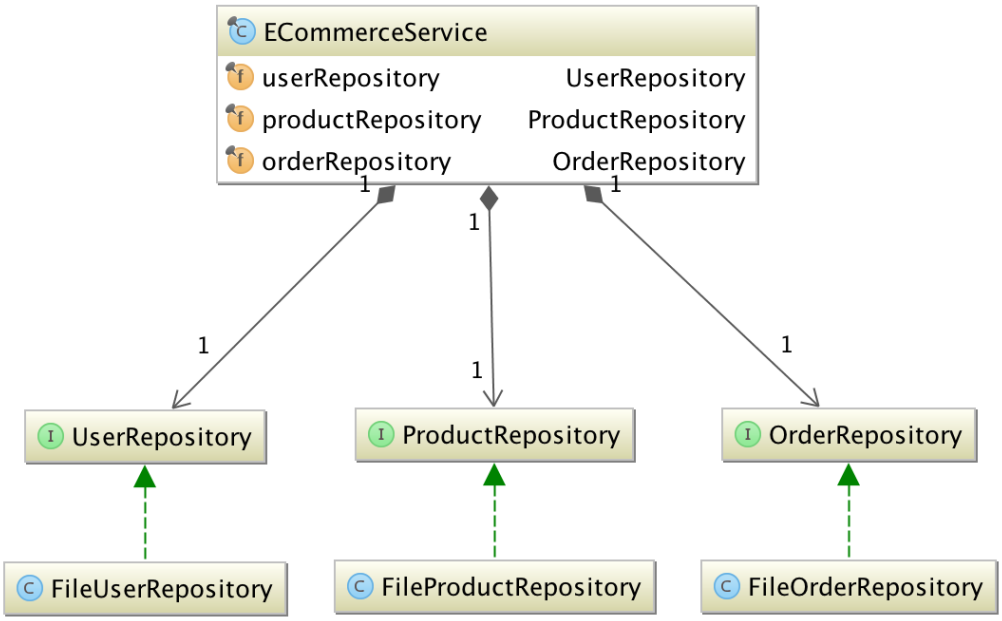
3) Lägg till de metoder som behöver finnas på ECommerceService och repository-interfacen. Var noga med att ge dessa passande namn

4) Implementera en metod i ett av repository-interfacen och testa att denna fungerar. Ta sedan nästa metod osv

5) Avsluta med att bygga upp din main-klass så att den testar hela ECommerceServicen

Generellt

Håll er design enkel. Lägg inte till för många datamedlemmar till dina modell-klasser. Det går alltid att lägga till flera senare om det skulle behövas. Ta också hjälp av papper och penna och rita upp hur klasserna hänger ihop. Då detta är en laboration som kommer att betygsättas kommer den hjälp ni får vara i form av vägledning, inte direkta svar.
Viktigt: koden måste vara korrekt formaterad. System.out.println får bara finnas i main-klassen**.**



Lycka till,
Anders och Pablo