Peer review

Detta är en kollegial granskning av applikationen StoreIT. Gruppens kod och design kommer att granskas för att hitta saker som fungerar bra men också för att upptäcka sådant som behöver förbättras.

Do the design and implementation follow design principles?

- SRP: Följs delvis. Deras main klass (StorageSystem) beror på både vy och modell. IBorrower har inga metoder som ansvarar för att låna saker. Det end den har är getId och equals som inte har något med att låna att göra.
- OCP: Följs. Ifall de t.ex. skulle vilja lägga till en ny klass “premiumItem”, som skulle fungera likt item, så skulle de bara behöva implementera IReservable.
- LSP: Finns inga arv och kan därmed inte följas.
- ISP: Följs. Inga klasser implementerar ett interface med metoder som de ej använder.
  Klassen User implementerar IBorrower men använder inte metoden boolean equals(). Däremot finns metoden i superklassen Object. Men den avsedda metoden i IBorrower används inte på det sätt som är avsett(den ska jämföra ID).
- DIP: Följs. Det finns exempelvis ett interface (IReservable) som klassen Item implementerar i modellen. LoginPageController använder IReservable och därmed följs DIP då LoginPageController ej beror direkt på klassen Item eller vice versa. Med andra ord så beror LoginPageController på en abstraktion av Item.

Is the code documented?
Klasserna är väldokumenterade med hjälp av JavaDoc. Däremot är inte alla publika metoder dokumenterade.

Are proper names used?
Namn är tydliga och beskrivande. Dock är skillnaden mellan team och organisation otydligt. Inte uppenbart vad meningen är med teams.

Is the design modular? Are there any unnecessary dependencies?
De eftersträvar en modulär design och har en tydlig uppdelning mellan paket. Kontroller beror på modellen och modellen beror ej på kontroller. Däremot finns det ett onödigt beroende till javafx.scene.image.Image i modellen (i klassen Item och Location). Detta beroende ska finnas i paketet med vyn och inte i modellen. Även klassen StorageSystem (som är deras applikation/main-klass) har onödiga beroende. Det finns starka beroende från StorageSystem till modellen då den har instansvariabler av typer som härstammar från modellen. Exempelvis så finns det en lista med organisationer som upprättar beroendet storagesystem. Organization och en variabel av typen “User” som ger upphov till beroendet storagesystem.model.*. En lösning hade varit att ha ett “aggregate object” i modellen dvs. en klass som ansvarar för logiken bakom currentUser mm. Ett liknande exempel är att ha en klass “Monopol” som ansvarar för spelregler, logik och en “currentUser” när man har ett helt program som representerar monopol.

Ett annat problem är att klasserna LoginPageController, SettingsController och ItemListController beror på main-klassen StorageSystem. Main-klassen ska vara separerat från allt annat och endast ansvara för att köra applikationen.

Does the code use proper abstractions?

-

Is the code well tested?

Endast 51% av metoderna i modellen är testade. 44 otestade metoder. Endast väldigt simpel funktionalitet som setters och getters samt add och remove har testats.

Are there any security problems, are there any performance issues?

Lösenord är inte dolda när de skrivs in, borde visa stjärnor (*). Man kan registrera en användare och logga in utan att ha fyllt i några uppgifter.

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

Det finns en tydlig struktur som visar att MVC används i koden. Model, View och Controller delas in i tre separata paket och behandlar sina respektive arbetsområden. Model beror huvudsakligen inte på Controller eller View. Dock finns det Images i bland annat klassen “Item” och “Location”, vilket ska undvikas i modellen och påpekades tidigare. Det kan göras några förändringar för att göra koden ännu enklare att förstå. Ett exempel på detta togs upp ovan då skillnaden mellan team och organisation inte var jättetydlig. De är inte så tydligt att de är kopplade till kommitteer respektive sektioner.

Can the design or code be improved? Are there better solutions?

- Ett förslag för att hantera listan med föremål/produkter(som visas när man är inloggad) skulle vara att utnyttja en HashMap istället för en ArrayList. Fördelen med en HashMap är att man fortare kan hitta ett specifikt föremål för att ta bort det eller uppdatera.(Detta kanske inte gäller eftersom det är svårt att tolka hur ni har tänkt implementera era metoder för borttagning och redigering.)
- Användning av interfaces bör ses över. I alla tre interfaces i modellen finns metoderna “boolean equals (Obect o);” samt “int getID();” vilket känns onödigt. För att undvika duplicering av kod och istället dra nytta av återanvändning av kod föreslår vi att ett nytt/två nya separata interfaces skapas som har dessa metoder. Därmed kan IBorrower delas upp i två interfaces.