Media Shop
Av: Fabian Fröding

1. Antaganden

- Kassa eller lager-vyer avgörs av användaren när programmet öppnas.
- I klassen Product representeras varunumret av fältet ”id”.
- Data sparas i .txt-format.
- Funktionaliteten för att ta bort en typ av produkt kan ändast utföras i lagervyn.
- Diagrammet och programmets struktur kan komma att ändras allt eftersomprogrammet utvecklas.

2. Översikt

Vid programstart öppnas inloggningsvyn och användaren får då välja om den vill öppna kassa eller lager-vy.

Programmet består av tre huvudsakliga vyer:

- Inloggningsvy
- Lagervy
- Kassavy

Lagervyn hanterar tillägning och bortagning av produkter, samt uppdatering av lagerstatus på produkter. Kassavyn hanterar försäljning av produkter. Båda vyerna har funktionalitet att se alla produkter och dess relaterade information.

3. Detaljerad beskrivning

Programmet använder sig av en MVC-liknande arkitektur.

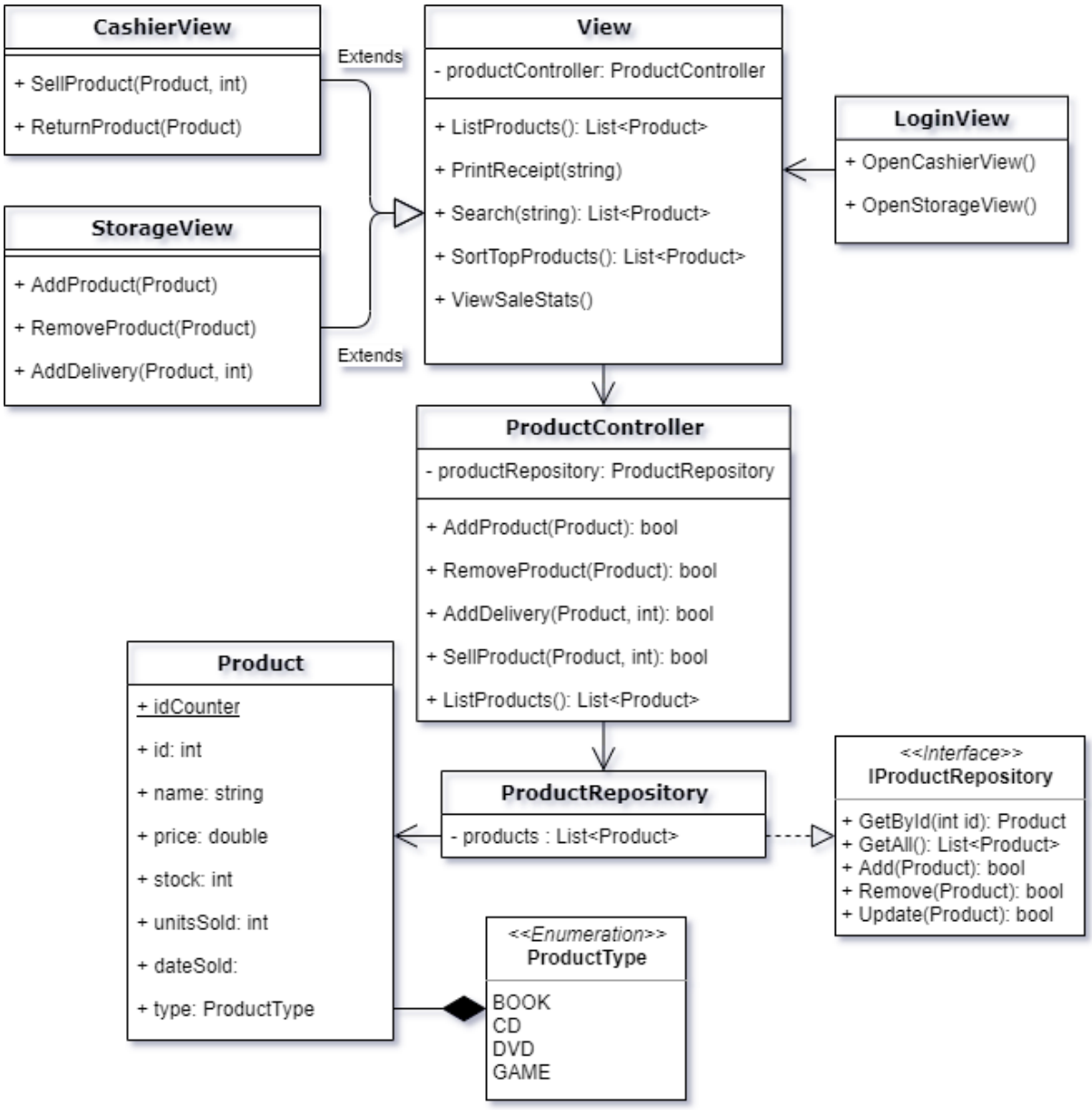
- Models består av domän-enheter som t.ex. Product.
- Views består av Windows Forms, i.e. “interfaces”.
- Controller-lagret är uppdelat i två ytterligare lager:
    - Controllers: Hanterar vilken data som skcikas vidare till vyerna från Repositories och hanterar även en del logik mellan dessa klasser.
    - Repositories: Hanterar lagring och hämtning av data från .txt-filer.

Programmet består av följande klasser:

- Domän-enheter:
    - Product: representationen av en fysisk produkt i lagret. Förutom grundläggande egenskaper som id, namn etc. så har en product även en ”type” som berättar vad för typ av produkt det är.
        - ProductType: En enum för att avgöra vilken kategori en Product tillhör.
- Datalagring:
    - ”Repository”-klasser är det understa lagret som hanterar sparande och hämtning av data. Repository-klassens ansvar och funktionalitet är baserat på CRUD-operationer (Create, read, update, delete).
    - ”Controller”-klasser använder datan som skickas från Repository-lagret för att sedan skicka den vidare till vyerna som visas för användaren.
    - Repositories är baserade på ”Interfaces” för att bevara det grundläggande CRUD-ansvaret som klassen har.
    - Data sparas i en extern textfil.
- Vyer:
    - Login: tillåter användare att välja vilken vy de vill använda.
    - Cashier View: Hanterar funktionalitet relaterat till kassa-ärenden.
    - Storage View: Hanterar funktionalitet relaterat till lager-ärenden.

- Grundläggande funktioner som getters/setters och liknande är inte inkluderat i klassdiagrammet. Detta för att hålla diagrammet minimalistiskt och begripligt.
- Metoder som ärvs från parent-klasser eller interfaces är inte representerade i den ärvande klassen. Detta för att reducera redundans.
- Domän-enheter (t.ex. Product) är POCOs (Plain Old C-Object), som är klasser som endast håller information.
- Somliga metoder som återger en bool gör detta för att indikera om operationen lyckades eller inte. T.ex. om metoden ”Add(Product)” återger *true* indikerar det att tilläget av produkten lyckades medans *false* hade indikerat att tilläget misslyckades. Denna sorts funktionalitet kan komma till nytta senare i utvecklingen.

Klassdiagram



4. Problem

Från början var det tänkt att ha ytterligare ett ”Service”-lager mellan Controller och Repository, men detta togs bort för att undvika over-engineering då lagret inte kändes så användbart (service-layers är till för att öka separation of concerns mellan Controller och Repository).

Det var också tänkt att ha en domän-enhet User, och funktionalitet för registrering och inloggning för olika användar-konton. Men efter planering insåg jag att detta inte är nödvändigt för att uppfylla MVP-beskrivningen (minimal viable product) som anges i instruktionerna.