**Media Shop**

Av: Fabian Fröding

# 1. Antaganden

Ange alla antaganden ni gjort under projektets gång. Antaganden är sådant som inte står i labbspecen men ändå måste bestämmas, och inte saker som "användaren har datorvana". Dessa bör skrivas upp kontinuerligt eftersom de annars lätt glöms bort. Exempelvis om specen anger att data ska sparas till fil så kan ett antagande vara hur datan sparas (format på text, binärt o.s.v...)

* I klassen Product representeras varunumret av fältet ”Id”.
* Data sparas i .txt-format.

# 2. Översikt

En beskrivning av programmet från ett användarperspektiv. Beskriv vad programmet kan göra och hur det används. Exempelvis för ett spel bör dess regler beskrivas.

## Exempel:

Programmet består av ett fönster som visar den nuvarande månaden samt har en knapp för att byta till nästa månad. Programmet justerar automatiskt månad vid årsbyte.

(Mycket kortfattat exempel p.g.a. mycket enkelt program.)

Programmet startar med ett inloggningsfönster. Inloggningfönstret kräver att användaren anger sin mail och väljer ”department”. Användarens department avgör vilket fönster som öppnas härnäst. Om användaren arbetar i lager så öppnas ett fönster för funktioner inom lagerhantering och om användaren arbetar i kassa så öppnas ett fönster för funktioner inom kassahantering.

# 3. Detaljerad beskrivning

En beskrivning av programmet från ett implementationsperspektiv (kod). Beskriv programmets struktur och funktionalitet. Vilka klasser finns, vad representerar de, vad innehåller de (data och funktioner/metoder), och hur relaterar de till varandra. Beskriv kortfattat vad variabler lagrar för data samt vad funktionerna gör. Använd diagram för att illustrera programmets struktur (klassdiagram och eventuellt flödesdiagram eller liknande).

## Exempel:

Programmet är uppbyggt enligt MVC mönstret och består av klasserna Model, View och Controller (se klassdiagram).

[KLASSDIAGRAM HÄR]

### Klass Model:

En klass för hanteringen av en kalenders logik. Lagrar den nuvarande månaden och tillhandahåller metoder för att hämta och ändra månad.

Variabler:

* int current\_month – Ett heltal som representerar den nuvarande månaden

Metoder:

* public Model(int month) – Konstruktor. Sätter den nuvarande månaden i programmet till *month*. Giltiga värden: 1 <= month <= 12
* public int getMonth() – Returnerar den nuvarande månaden.
* public void nextMonth() – Byter till nästa månad (med hänsyn till årsbyte).

### Klass View:

...

o.s.v.

Programmet använder sig av MVC-arkitekturen.

* Models består av domän-enheter som t.ex. Product, User.
* Views består av Windows Forms, i.e. “interfaces”.
* Controller-lagret är uppdelat i två ytterligare lager:
  + <Model>Controller: Hanterar vilka vyer som ska visas baserat på vilka enheter som efterfrågas.
  + <Model>Repository: Hanterar lagring och hämtning av data.

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.
  + User: representerar en användare. Förutom grundläggande egenskaper som id, mail etc. så har även användare en ”department” för att hjälpa systemet avgöra vilka funktionaliteter som ska finnas tillgängliga för användaren.
* Datalagring:
  + ”Repository”-klasser är det understa lagret som hanterar sparande och hämtning av data. Repositiy-klassernas 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 ansvaret som klasserna har.
  + Data sparas i en extern textfil. En textfil för varje domän-enhet.
* Vyer:
  + Login: tillåter användare att logga in i systemet och ange sin roll.
  + Cashier View: visas efter login och när department ”SALES” valts.
  + Storage View visas efter login och när department ”STORAGE” valts.
* 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 reduca redundans.
* Varje domän-enhet (t.ex. Product och User) 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.

# 4. Problem

Beskriv problem som uppstått under projektets gång. Beskriv hur ni har löst dem och vad ni lärt er av dem. Anteckna kontinuerligt eftersom det kan vara svårt att komma ihåg varför man gjorde på ett visst sätt. Problem avser övergripande svårigheter, inte mindre misstag i koden.

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).

# 5. Sammanfattning

Sammanfatta ert projekt med hur det har gått och vad ni har lärt er. Beskriv alternativa lösningar samt fördelar och/eller nackdelar med dessa och varför ni valt den lösning ni har. Ta också med ungefärlig tidsåtgång för de olika momenten.

# 6. Referenslista

Författare, titel, förlag och år. Endast nödvändig om du använt andra böcker än kurslitteraturen.

# Bilagor

Innehållet är beroende av vilken kurs och vilken labb som utförs men innehåller ofta programkod och diagram.