Rapport - Ticket Sales

• Beskriva programmet och vald fördjupning om en sådan är utförd.

(Fördjupning: “Kunna spara och hitta specifika transaktioner”)

Programmet är ett biljettsystem, där en kassör ska kunna registrera in transaktioner med biljetter som säljs. En transaktion har ett ordernummer som fungerar som ett ID, ifall att kunden vill dra tillbaka köpet. Transaktionen har även information om hur många biljetter av varje åldersort (vuxen, barn eller pensionär) som är registrerade på köpet, vilket gör det möjligt för en grupp att köpa enstaka eller flera biljetter per transaktion. Denna information skrivs in/definieras av kassören vid registrering.

Efter att en transaktion har skapats registreras den automatiskt in i en textfil som fungerar som ett register med alla transaktioner. När en kund vill ha pengarna tillbaka för ett köp, skrivs transaktionens ordernummer in, och transaktionen ändrar sin status till “återbetald”. Detta innebär att den inte längre räknas med i den totala summan av sålda biljetter, eller i den totala summan intjänade pengar. Dessa kalkylationer finns som metoder i programmet, och är även utskrivna i ett separat “bokföringsregister”, en .txt vid namn “bokföring”.
Registret (.txt filen med namn register) skrivs innan programmet avslutas, och läses in vid programmets uppstart, då streamreader läser in existerande transaktioner och skriver in de i listan så att de alltid finns.
Detta innebär att en transaktion aldrig kommer att försvinna från registret, även när det “ändrar status” till återbetald, så länge man inte skriver om i register-filen (vilket går, men kan skapa exekveringsproblem). Anledning: Uppgiften specificerar inte att man skall kunna ta bort transaktioner. I ett verkligt scenario, ja, och jag skulle kunna fixa det relativt lätt, men för uppgiften, inte nödvändigt.

• Beskriva utvecklingsprocessen och eventuella problem som uppstod.

Jag började med att skissa upp för mig vad jag ville att mitt program skulle göra, med en checklista över funktionella krav. Detta gjorde att jag sedan kunde spåna ideer om hur programmet skulle kunna ser ut. En klass “Transactions”, ville jag jobba med, vars instanser har sina egna värden för tex. antal biljetter och dess totala pris. Det tillsammans med en lista “transactions” gav mig en bra bas för att börja bygga mitt program på. Jag märkte att min metod som skapar transaktioner började bli fylld med kod av svår läsbarhet, och bestämde mig för att dela upp den metoden lite för att förbättra läsbarhet. Därefter gick det mesta av kodskrivandet bra, med endast lätt överkommerliga svårigheter. Jag återanvände mycket av koden från föregående projekt: Myrstacken, och använde den för referens, då de två programmen i grund och botten har en liknande struktur.

När det kom till att skapa i/o-strömmen blev det lite mer klurigt, mest för att jag inte hade erfarenhet med StreamWriter/StreamReader sedan innan, vilket också gjorde att jag till slut

missade inlämningsdagen. Jag hade hoppats på att kunna lära mig om i/o strömmen tillräckligt snabbt för att hinna med innan deadline, men det tog mer tid än jag trodde att få allt att fungera som jag ville. Men till slut lyckades jag researcha mig fram till en lösning som jag är nöjd med.

• Beskriva hur arbetet gått och reflektera över huruvida arbetet hade kunnat förbättras eller effektiviseras.

Jag känner att arbetet för det mesta har gått bra, men det var inte så effektivt. Det största problemet med mitt arbete var hur sent jag kom igång. Tidspressen mot slutet av projektet, när jag faktiskt började med arbetet gjorde att jag inte presterade med en helt finslipad och felfri lösning. Jag hade till exempel velat fokusera mer på att refaktorera min kod för effektivisering och läsbarhet. Dessutom hade jag velat implementera en ytterligare fördjupning i min kod, men som sagt stoppades jag från allt detta på grund av dålig användning av projektets längd, alltså att jag inte började i tid. Arbetet skulle ha kunnat förbättras genom att jag snabbare hade börjat med kodskrivandet. Jag fick ganska snabbt till min planering med checklista, och jag visste vad jag ville att programmet skulle göra från första början, men hade för lite motivation att börja skriva på koden i tid. Jag kände att jag kom igång rejält när jag väl började, så om det hade varit tidigare under projektet hade jag varit golden.

• Vad du lärt dig.

Jag har lärt mig mycket om hur man är effektiv med den tid man har. Jag kom igång med projektet alldeles för sent, men med rätt fokus kunde jag effektivisera mitt arbete, genom att till exempel använda Myrstacken så mycket som möjligt för referens.

Jag har även lärt mig om I/O-strömmar, och hur en sådan kan vara användbar/nödvändig i sammanhang där man vill spara data mellan användningar av programmet.

Jag har fått en bättre förståelse för klassers struktur, deras konstruktor och hur Get och Set-metoder egentligen fungerar och används.

Har lärt mig mycket under projektet som jag kanske inte kan komma på just nu, men jag känner till exempel att min förmåga att skriva och förstå c# basics förbättras för varje gång.

• Vad du skulle vilja lära dig för att förbättra programmet i framtiden eller vad du vill lära dig generellt.

Jag hade originellt satt “Lagra och hitta information med hjälp av en hashtabell, ett treeset eller någon annan datastruktur” som en av mina fördjupningspunkter. Jag kom aldrig så långt att jag kunde researcha och prova att implementera någon av dessa i TicketSales, men det verkar som något som skulle vara intressant att lära mig mer om, hur man optimerar lagring av data utifrån ett scenario.

Även ett grafiskt användargränssnitt hade definitivt också förbättrat programmet. Jag tror att, trots att det inte påverkar huruvida programmet fungerar eller ej, är det en av de viktigaste komponenterna för ett program, användbarhet. Användaren måste känna att det är stressfritt att, i detta fall, registrera in biljetter osv. Det funkar ofta inte optimalt i en terminal som den i visual studio.

Jag skulle också gärna bli mer effektiv och mer konsekvent när det kommer till c# basics, som användning av allt från loopar, till klasser, variabler och metoder.