Projektarbete

Titel: Bokningssystem för däckbyte – Däckarn
Kurs: GIK299 – Objektorienterad programmering
Författare: Ariz Biswas Islam
E-post: h24ariis@du.se
Datum: 2025-04-02

Innehållsförteckning

Introduktion

I denna projektrapport redovisas mitt arbete med projektuppgiften i kursen Objektorienterad programmering. Syftet med detta projekt har varit att utveckla en konsolbaserad bokningsapplikation för däckbyten som uppfyller specifika användarkrav och tekniska krav.

Bokningssystem för däckbyte är en viktig tjänst för både företag och kunder som vill planera däckbyten smidigt och effektivt. Med hjälp av denna applikation kan administratören hantera kundbokningar, se dagens bokningar och kontrollera lediga tider. Programmet är designat för att vara enkelt och användarvänligt, med fokus på att förhindra dubbelbokningar och skapa en tydlig överblick över arbetsdagen.

Genom detta projekt har jag fått möjlighet att tillämpa mina kunskaper i objektorienterad programmering och använda programmeringsspråket C#. Rapporten beskriver de tolkningar och överväganden jag har gjort under arbetets gång samt de tekniska lösningar jag valde, och redovisar resultatet av det färdiga bokningssystemet.

Syfte

Syftet med projektet var att skapa en textbaserad konsolapplikation som simulerar ett bokningssystem för däckbyten hos företaget Däckarn. Programmet skulle innehålla en administratörspanel för att hantera bokningar, visa lediga tider och förhindra dubbelbokningar. Projektet syftade också till att ge praktisk träning i objektorienterad programmering, felhantering och användarvänligt gränssnitt.

Utifrån kursens instruktioner implementerades följande user stories:

1. Som företagsägare vill jag kunna hantera bokningar och lediga tider via en administratörspanel för att effektivt planera arbetsdagen.
2. Som företagsägare vill jag kunna lägga till, ta bort och ändra bokningar. Inga dubbelbokningar får ske.
3. Som företagsägare vill jag kunna söka efter lediga dagar och tider för bokningar.
4. Som företagsägare vill jag kunna få bekräftelse på genomförd bokning.
5. Som företagsägare vill jag få se dagens bokningar för att kunna förbereda mig för kundernas besök.

Egna user stories

1. Som administratör vill jag kunna söka efter en bokning via kundens namn.
2. Som administratör vill jag kunna visa alla bokningar för en specifik dag.
3. Som administratör vill jag kunna visa alla bokningar sorterade efter datum.
4. Som administratör vill jag kunna rensa alla bokningar om jag vill börja om.

Egna tolkningar, övervägande och val

Under arbetet med projektet gjorde jag flera tolkningar och val utifrån instruktionen och användarberättelserna.

En av de viktigaste tolkningarna var hur dubbelbokningskontrollen skulle fungera. I instruktionen stod det att ingen dubbelbokning får ske, men det specificerades inte exakt vad som räknas som dubbelbokning. Jag valde att tolka det som att det inte får finnas två bokningar med samma datum och tid, oavsett kund. Därför lade jag in en kontroll som söker igenom alla bokningar och kontrollerar om den tid och dag som administratören försöker boka redan är upptagen.

Jag valde också att lägga till ett fält för registreringsnummer i varje bokning. Det nämndes kort i instruktionen, men det stod inte tydligt hur det skulle hanteras. Jag bedömde att registreringsnumret är viktigt för att kunna identifiera kundens bil och därför gjorde jag det till en obligatorisk del av bokningen.

När det gällde sökfunktionen via kundnamn (en av mina egna user stories) gjorde jag tolkningen att administratören skulle kunna skriva in en del av ett namn och få upp alla bokningar som matchade. Detta för att göra sökningen mer flexibel.

Jag valde också att visa bokningar för en specifik dag och att visa alla bokningar sorterade efter datum, eftersom det inte specificerades exakt hur administratören skulle få översikt över bokningarna. Jag bedömde att det var praktiskt att kunna få en samlad vy över bokningar, både för en dag och för hela perioden.

För funktionen att rensa alla bokningar (egen user story) antog jag att det kunde finnas ett behov av att snabbt ta bort alla bokningar om något gått fel, men jag lade in en säkerhetsfråga innan bokningarna tas bort, för att undvika att administratören gör det av misstag.

Sammanfattningsvis gjorde jag alla dessa val för att skapa ett program som är så användarvänligt och logiskt som möjligt, utifrån de krav och instruktioner som fanns i projektbeskrivningen.

Användning av ChatGPT

Under projektets gång stötte jag på flera moment som jag hade svårt att förstå eller komma vidare med. Framför allt hade jag problem med att få dubbelbokningskontrollen att fungera korrekt och att hantera felaktiga inmatningar från användaren. Jag hade även svårt att strukturera mina klasser och metoder så att de hängde ihop logiskt.

För att få hjälp med att förstå dessa delar använde jag ChatGPT som ett verktyg. Jag använde AI:n enbart för att få enklare förklaringar och exempel på hur man kan lösa liknande problem inom C#, samt för att förstå hur menystrukturer och klassindelning kan byggas upp på ett tydligt sätt.

Jag använde inte ChatGPT för att skriva färdiga lösningar, utan som ett stöd för att bättre förstå hur jag själv kunde lösa olika problem. När jag fick förklaringar eller exempel testade jag alltid att skriva koden själv i min egen utvecklingsmiljö och anpassade den till mitt projekt.

Fördelar och nackdelar

Att använda ChatGPT gav mig en bättre förståelse för svåra koncept som listor, datumhantering och felhantering. Samtidigt insåg jag att det finns en risk att bli beroende av verktyget om man inte testar och försöker själv först. Därför valde jag att alltid först försöka lösa problemet själv innan jag sökte stöd hos AI:n.

Slutsats om AI-användningen

ChatGPT fungerade som ett pedagogiskt stöd som hjälpte mig att förstå objektorienterad programmering bättre. Jag ser verktyget som en hjälp för att lära mig och utvecklas, men allt kodande, testande och dokumentation har jag gjort själv.

Metod

Design

Jag började projektet med att fundera över strukturen och vilka klasser som behövdes. Jag valde att skapa tre huvudklasser:

- **Booking** – för att hantera information om bokningar.
- **Customer** – för att representera kundens namn och registreringsnummer.
- **ServiceType** – för att lista tillgängliga tjänster.

I Program.cs skapade jag ett menysystem som körs i en loop tills användaren väljer att avsluta. Menyn är uppdelad i olika val som hanterar bokningar, lediga tider och visning av information.

Samarbete

Jag arbetade ensam med projektet.

Testning

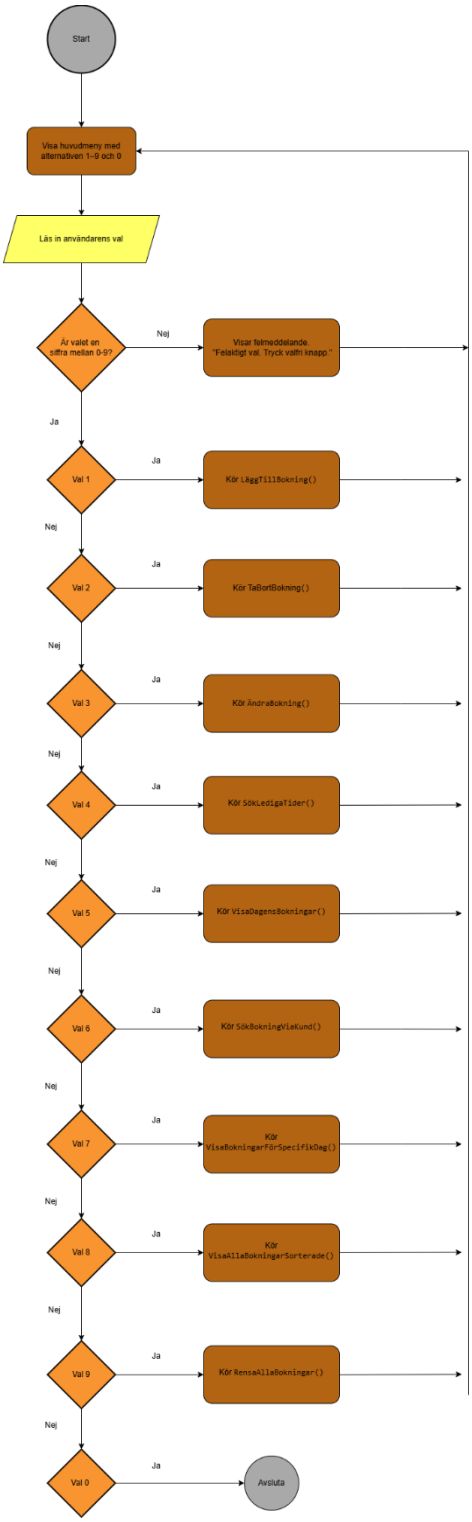
Jag testade programmet på tre sätt:

1. Genom att mata in korrekt data och säkerställa att alla funktioner fungerade som de skulle.
2. Genom att medvetet mata in felaktiga värden, som ogiltigt datumformat, tomma namn eller fel menyval, för att se att felmeddelanden visades och att programmet hanterade dessa situationer.
3. Jag bad en vän testa programmet för att se om det var lätt att förstå menyvalen och flödet. Under detta test upptäckte jag små förbättringsområden som jag justerade.

Arkitektur/Design

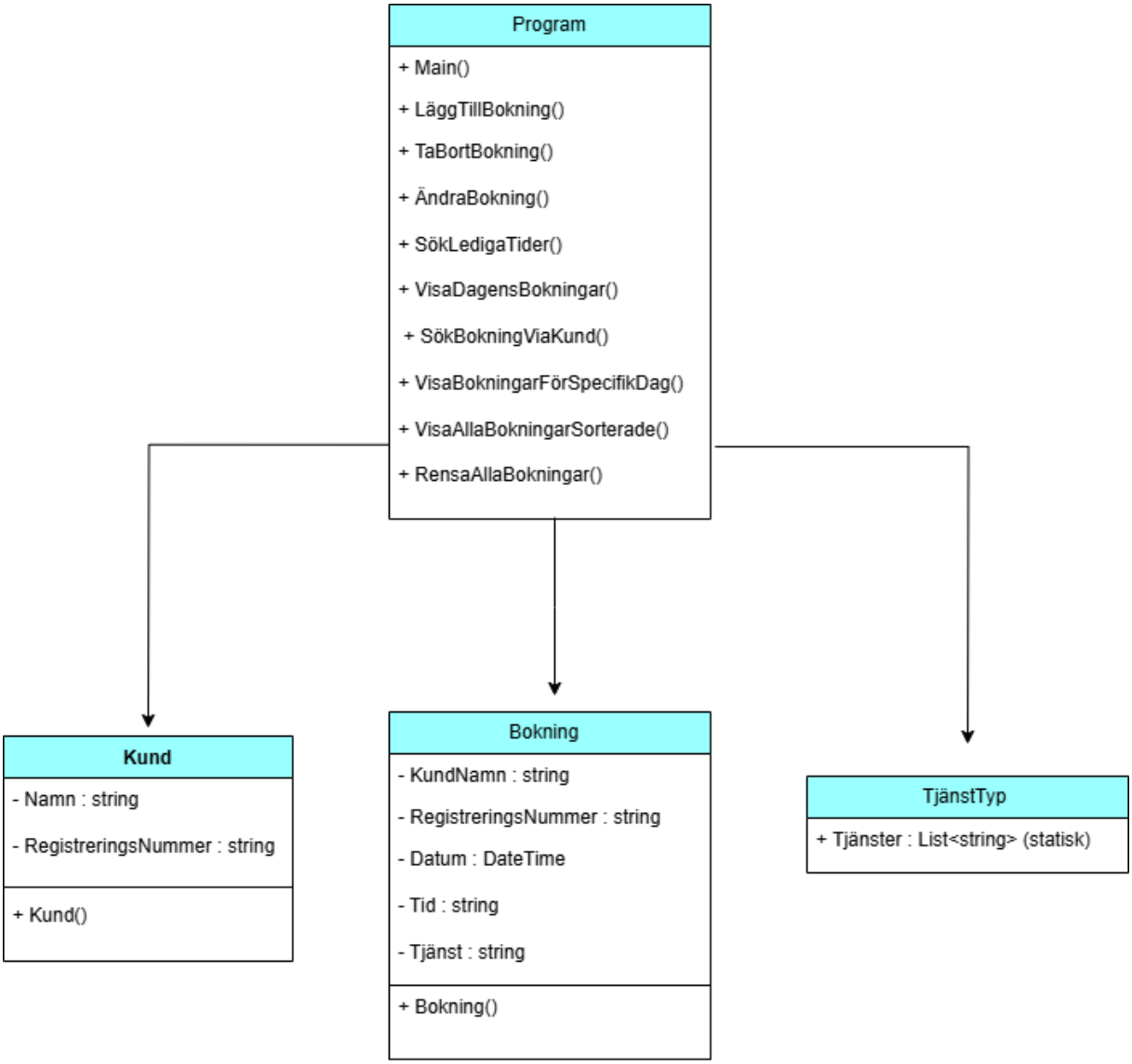
Programmet är uppbyggt med tre klasser som representerar bokningar, kunder och tjänster. I Program-klassen finns en huvudloop som visar en meny där administratören kan välja olika alternativ. Bokningarna lagras i en lista som uppdateras varje gång användaren lägger till, ändrar eller tar bort en bokning.

Flödesschema:



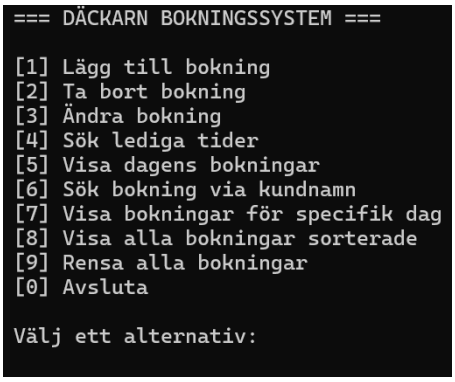
UML- klassdiagram

Jag har skapat en UML- klassdiagram över hela programmet:

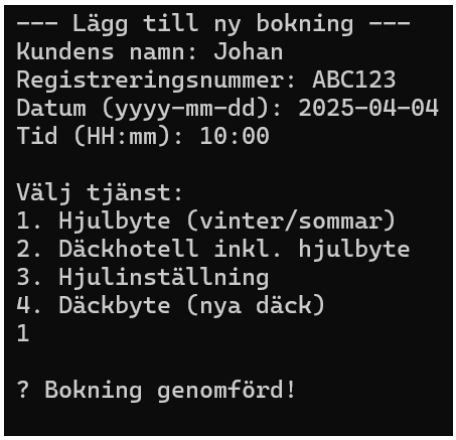


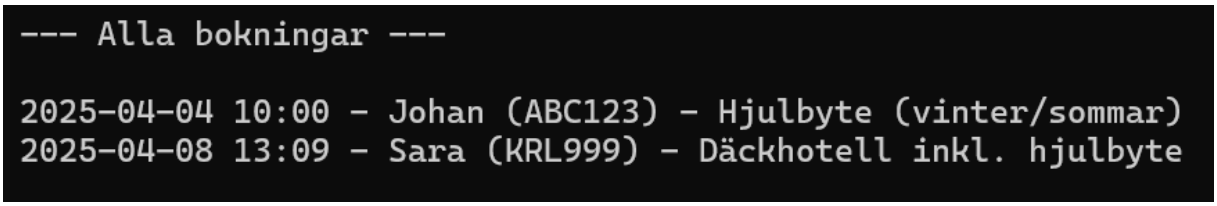
Resultat

Programmet fungerar som tänkt och alla krav är uppfyllda. Här är exempel på programkörning:



Exempel på bokning:





Slutsats

Det här projektet har varit mycket utmanande och lärorikt för mig. Jag har haft svårt att komma igång och fastnade flera gånger under arbetet. Flera gånger kände jag mig stressad och frustrerad när koden inte fungerade som jag ville.

Det som tog mest tid och energi var att förstå hur jag skulle bygga upp klasserna och hur jag skulle få dubbelbokningskontrollen att fungera. Jag fick göra om stora delar av koden flera gånger när jag märkte att mina lösningar inte höll eller när jag kört fast.

Samtidigt har projektet gett mig mycket kunskap. Jag har lärt mig grunderna i objektorienterad programmering, hur man hanterar listor, felinmatningar och hur viktigt det är att testa sin kod noggrant. Jag har även lärt mig att ibland behöver man söka information, be om hjälp och att det är okej att göra misstag.

Projektet har varit ett bevis för mig själv att jag kan klara av svåra saker om jag fortsätter kämpa.

Appendix

Par-programmeringslogg

Datum	Tid i timmar	Ensam eller par	% fördelat i att sitta vid tangentbordet
2025-03-26	4	Ensam	100%
2025-03-28	8	Ensam	100%
2025-03-29	5	Ensam	100%
2025-03-31	6	Ensam	100%
2025-04-01	10	Ensam	100%
2025-04-02	7	Ensam	100%
2025-04-03	8	Ensam	100%

Referenser

Land, R. (2024). Objektorienterad programmering [Föreläsning]. 15

• Microsoft. (n.d.). .NET documentation. https://learn.microsoft.com/en-us/dotnet/